MODULE 1

INTRODUCTION TO OPERATING SYSTEM

What is an Operating System?

An operating system is system software that acts as an intermediary between a user of a computer and the computer hardware. It is software that manages the computer hardware and allows the user to execute programs in a convenient and efficient manner.

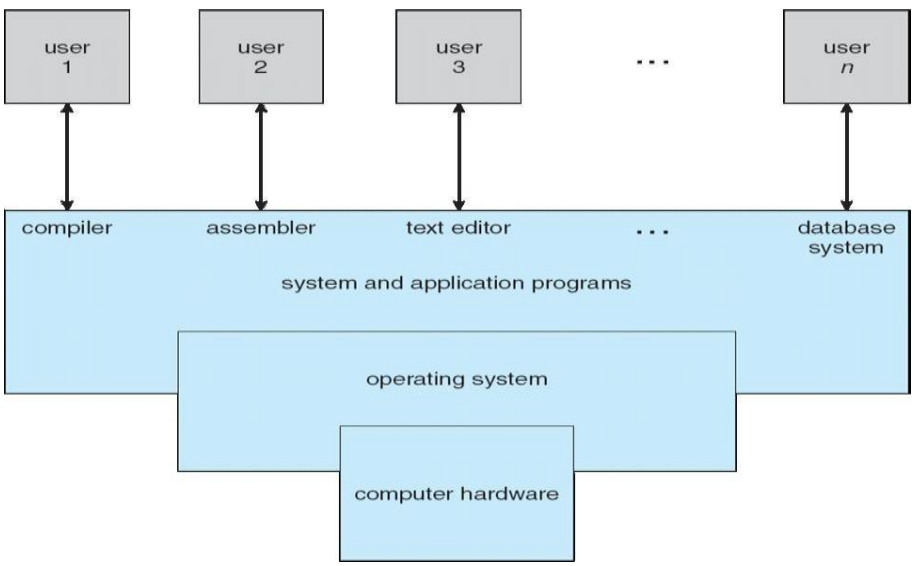
Operating system goals:

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

Computer System Structure (Components of Computer System)

Computer system mainly consists of four components-

- ***Hardware –*** provides basic computing resources CPU, memory, I/O devices
- ***Operating system*** **-** Controls and coordinates use of hardware among various applications and users
- ***Application programs*** – define the ways in which the system resources are used to solve the computing problems of the users, Word processors, compilers, web browsers, database systems, video games
- ***Users*** **-** People, machines, other computers



List out the User Views and System views of OS

Operating System can be viewed from two viewpoints– User views & System views

User Views: -*T*he user's view of the operating system depends on the type of user.

- If the user is using **standalone** system, then OS is designed for ease of use and high performances. Here resource utilization is not given importance.

- If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to maximize resource utilization. OS is designed such that the CPU time, memory and i/o are used efficiently and no single user takes more than the resource allotted to them.

- If the users are in **workstations**, connected to networks and servers, then the user have a system unit of their own and shares resources and files with other systems. Here the OS is designed for both ease of use and resource availability (files).

- Other systems like embedded systems used in home device (like washing m/c) & automobiles do not have any user interaction. There are some LEDs to show the status of its work

- Users of **hand-held** systems, expects the OS to be designed for ease of use and performance per amount of battery life

System Views*:* - Operating system can be viewed as a **resource allocator** and **control program**.

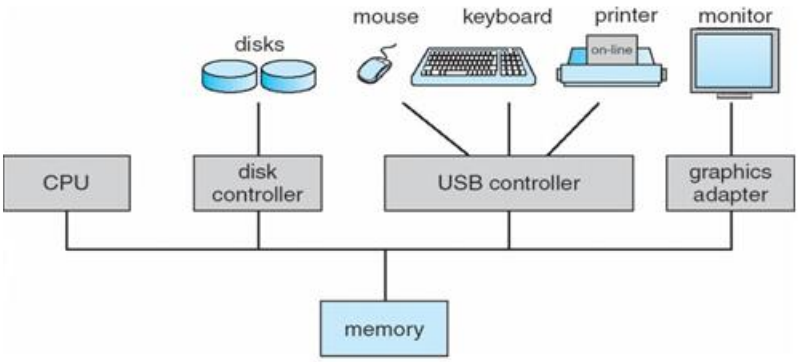
- **Resource allocator** – The OS acts as a manager of hardware and software resources. CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program. There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority.

- **Control Program** – The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer.

Computer System Organization

Computer - system operation

One or more CPUs, device controllers connect through common bus providing access to shared memory. Each device controller is in-charge of a specific type of device. To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory. The CPU and other devices execute concurrently competing for memory cycles. Concurrent execution of CPUs and devices competing for memory cycles



- When system is switched on, '**Bootstrap'** program is executed. It is the initial program to run in the system. This program is stored in read-only memory (ROM) or in electrically erasable programmable read-only memory (EEPROM).
- It initializes the CPU registers, memory, device controllers and other initial setups. The program also locates and loads, the OS kernel to the memory. Then the OS starts with the first process to be executed (ie. 'init' process) and then wait for the interrupt from the user.

Switch on ⟶ 'Bootstrap' program

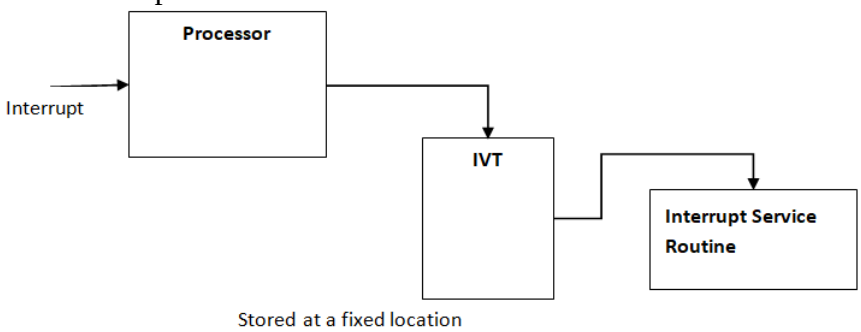
- Initializes the registers, memory and I/O devices
- Locates & loads kernel into memory
- Starts with 'init' process
- Waits for interrupt from user.

Interrupt handling –

- The occurrence of an event is usually signaled by an interrupt. The interrupt can either be from the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU. Software triggers an interrupt by executing a special operation called a system call (also called a monitor call).

- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location (Interrupt Vector Table) contains the starting address where the service routine for the interrupt is located. After the execution of interrupt service routine, the CPU resumes the interrupted computation.

- Interrupts are an important part of computer architecture. Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine



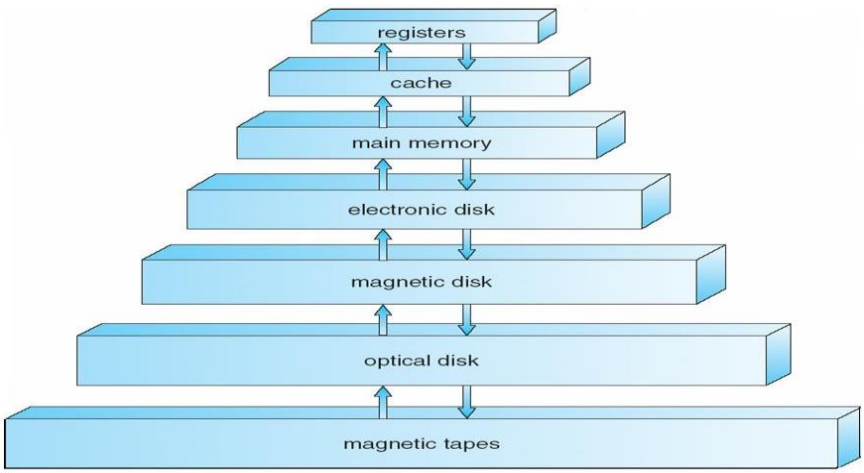
Storage Structure

- Computer programs must be in main memory (**RAM)** to be executed. Main memory is the large memory that the processor can access directly. It commonly is implemented in a semiconductor technology called **dynamic random-access memory (DRAM).** Computers provide Read Only Memory (ROM), whose data cannot be changed.
- All forms of memory provide an array of memory words. Each word has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses.
- A typical instruction-execution cycle, as executed on a system with a **Von Neumann** architecture, first fetches an instruction from memory and stores that instruction in the **instruction register.** The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory.
- Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:

  1. Main memory is usually too small to store all needed programs and data permanently.
  2. Main memory is a *volatile* storage device that loses its contents when power is turned off.

- Thus, most computer systems provide **secondary storage** as an extension of main memory. The main requirement for secondary storage is that it will be able to hold large quantities of data permanently.

- The most common secondary-storage device is a **magnetic disk,** which provides storage for both programs and data. Most programs are stored on a disk until they are loaded into memory. Many programs then use the disk as both a source and a destination of the information for their processing.



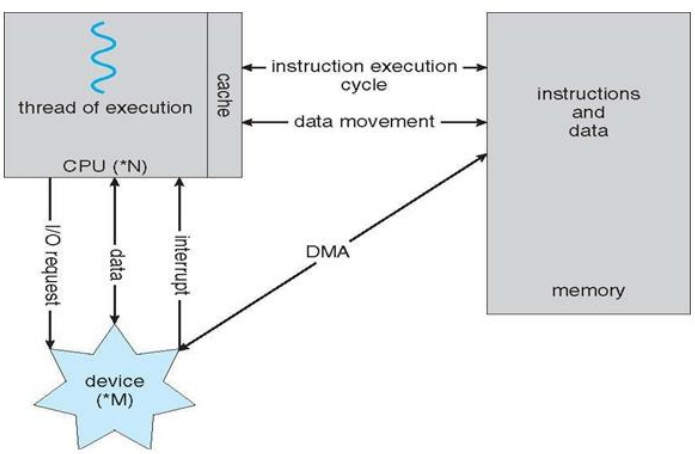
- The wide variety of storage systems in a computer system can be organized in a hierarchy as shown in the figure, according to speed, cost and capacity. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time and the capacity of storage generally increases.
- In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile. V**olatile storage** loses its contents when the power to the device is

removed. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage** for safekeeping. In the hierarchy shown in figure, the storage systems above the electronic disk are volatile, whereas those below are nonvolatile.

- An **electronic disk** can be designed to be either volatile or nonvolatile. During normal operation, the electronic disk stores data in a large DRAM array, which is volatile. But many electronic-disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic-disk controller copies the data from RAM to the magnetic disk. Another form of electronic disk is flash memory.

I/O Structure

- A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.
- Every device has a device controller, maintains some local buffer and a set of special- purpose registers. The device controller is responsible for moving the data between the peripheral devices. The operating systems have a **device driver** for each device controller.
- Interrupt-driven I/O is well suited for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem, direct memory access (DMA) is used.
- After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed.



Computer System Architecture

Categorized roughly according to the number of general-purpose processors used.

Single-Processor Systems –

- The variety of single-processor systems range from PDAs through mainframes. On a single-processor system, there is one main CPU capable of executing instructions from user processes. It contains special-purpose processors, in the form of device-specific processors, for devices such as disk, keyboard, and graphics controllers.
- All special-purpose processors run limited instructions and do not run user processes. These are managed by the operating system; the operating system sends them information about their next task and monitors their status.
- For example, a disk-controller processor, implements its own disk queue and scheduling algorithm, thus reducing the task of main CPU. Special processors in the keyboard, converts the keystrokes into codes to be sent to the CPU.
- The use of special-purpose microprocessors is common and does not turn a single- processor system into a multiprocessor. If there is only one general-purpose CPU, then the system is a single-processor system.

Multi -Processor Systems **(parallel systems** or **tightly coupled systems)**

Systems that have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices are the multiprocessor systems.

Multiprocessor systems have three main advantages:

1. *Increased throughput* - In multiprocessor system, as there are multiple processors execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly and also due to the competition for the shared resources. The speed-up ratio with *N* processors is not *N*, rather, it is less than N. Thus the speed of the system is not has expected.

2. *Economy of scale* - Multiprocessor systems can cost less than equivalent number of many single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several processes are working on the same data, the data can also be shared among them.

3. *Increased reliability*- In multiprocessor systems functions are shared among several processors. If one processor fails, the system is not halted, it only slows down. The job of the failed processor is taken up, by other processors.

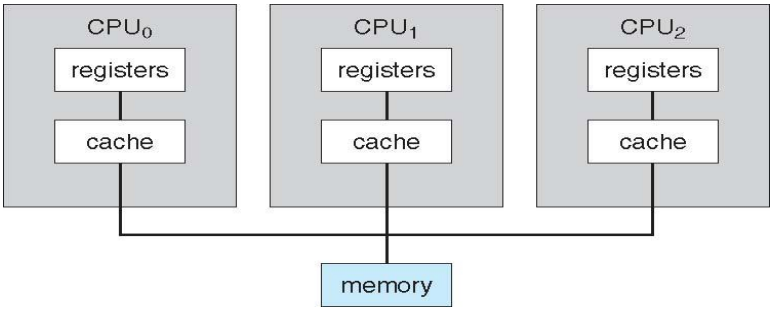
   Two techniques to maintain 'Increased Reliability' - graceful degradation & fault tolerant
   1. **Graceful degradation –** As there are multiple processors when one processor fails other process will take up its work and the system go down slowly.
   2. **Fault tolerant –** When one processor fails, its operations are stopped, the system failure is then detected, diagnosed, and corrected.

Different types of multiprocessor systems

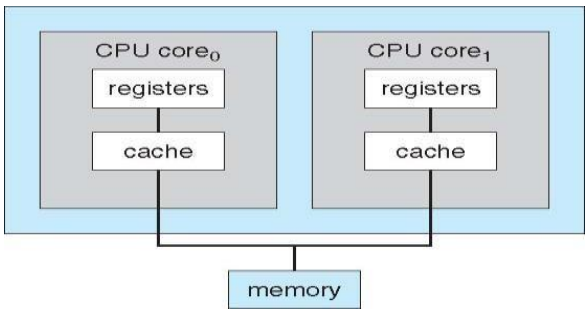
1. Asymmetric multiprocessing
2. Symmetric multiprocessing

1) **Asymmetric multiprocessing** – **(Master/Slave architecture)** Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors.

2) **Symmetric multiprocessing (SMP)** – All the processors are considered peers. There is no master-slave relationship. All the processors have their own registers and CPU, only memory is shared.



The benefit of this model is that many processes can run simultaneously. *N* processes can run if there are *N* CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP. A recent trend in CPU design is to include multiple compute **cores** on a single chip. The communication between processors within a chip is faster than communication between two single processors.



Clustered Systems

Clustered systems are two or more individual systems connected together via a network and sharing software resources. Clustering provides high availability of resources and services. The service will continue even if one or more systems in the cluster fail. High availability is generally obtained by storing a copy of files (s/w resources) in the system.

There are two types of Clustered systems – **asymmetric** and **symmetric**

1. **Asymmetric clustering** – one system is in **hot-standby mode** while the others are running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.

2. *Symmetric clustering* – two or more systems are running applications, and are monitoring each other. This mode is more efficient, as it uses all of the available hardware. If any system fails, its job is taken up by the monitoring system.

Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN). Parallel clusters allow multiple hosts to access the same data on the shared storage. Cluster technology is changing rapidly with the help of **SAN (storage-area networks)**. Using SAN resources can be shared with dozens of systems in a cluster, that are separated by miles.

Operating System Structure

Explain multiprogramming and multitasking systems.

Multiprogramming

One of the most important aspects of operating systems is the ability to multiprogram. A single user cannot keep either the CPU or the I/O devices busy at all times. **Multiprogramming** increases CPU utilization by organizing jobs, so that the CPU always has one to execute.

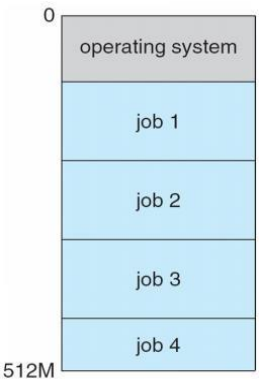
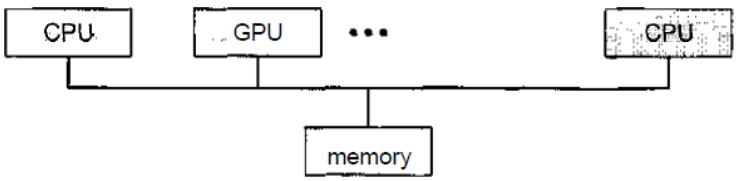


Fig - Memory layout for a multiprogramming system

- The operating system keeps several jobs in memory simultaneously as shown in figure. This set of jobs is a subset of the jobs kept in the job pool. Since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool (in secondary memory). The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some tasks, such as an I/O operation, to complete. In a non-multiprogram system, the CPU would sit idle.
- In a multiprogrammed system, the operating system simply switches to, and executes, another job. When *that* job needs to wait, the CPU is switched to *another* job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. Thus, the CPU is never idle.
- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

Multitasking Systems

- In **Time sharing** (or **multitasking) systems,** a single 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. The user feels that all the programs are being executed at the same time.
- Time sharing requires an **interactive** (or **hands-on) computer system,** which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the **response time** should be short—typically less than one second.
- A time-shared operating system allows many users to share the computer simultaneously. 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 only, even though it is being shared among many users.
- A **multiprocessor system** is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.



Operating-System Operations

Modern operating systems are **interrupt driven.** If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are signaled by the occurrence of an interrupt or a trap. A **trap (or** an **exception)** is a software-generated interrupt. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

Explain dual mode operation in operating system with a neat block diagram

Dual-Mode Operation

Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system.

The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.

The system can be assumed to work in two separate **modes** of operation:

1. User mode
2. Kernel mode (supervisor mode, system mode, or privileged mode).

- A hardware bit of the computer, called the **mode bit**, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.
- When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.

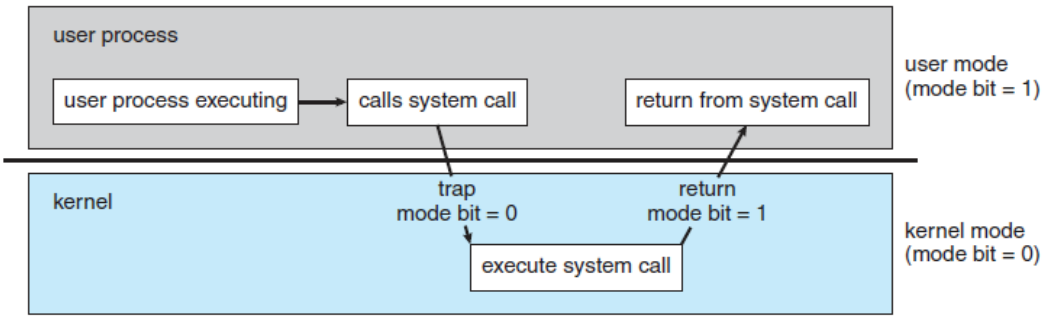


Figure Transition from user to kernel mode.

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 (that is, changes the mode bit from 1 to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.

- 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 user mode is an example of a privileged instruction.

- Initial control is within the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

Process Management

- A program under execution is a process. A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time. When the process terminates, the operating system reclaims the resources.

- The program stored on a disk is a **passive entity** and the program under execution is an **active entity**. A single-threaded process has one **program counter** specifying the next instruction to execute. The CPU executes one instruction of the process after another, until the process completes. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread.

- The operating system is responsible for the following activities in connection with process management:
    - Scheduling process and threads on the CPU
    - Creating and deleting both user and system processes
    - Suspending and resuming processes
    - Providing mechanisms for process synchronization
    - Providing mechanisms for process communication

Memory Management

Main memory is a large array of words or bytes. Each word or byte has its own address.
Main memory is the storage device which can be easily and directly accessed by the CPU. As the program executes, the central processor reads instructions and also reads and writes data from main memory.

To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

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

- Keeping track of which parts of memory are currently being used by user.
- Deciding which processes and data to move into and out of memory.
- Allocating and deallocating memory space as needed.

Storage Management

There are three types of storage management

i) File system management
ii) Mass-storage management
iii) Cache management.

File-System Management

- File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.
- A file is a collection of related information defined by its creator. Commonly, files represent programs and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).
- The operating system implements the abstract concept of a file by managing mass storage media. Files are normally organized into directories to make them easier to use. When multiple users have access to files, it may be desirable to control by whom and in what ways (read, write, execute) files may be accessed.

The operating system 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

- As the main memory is too small to accommodate all data and programs, and as the data that it holds are erased when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the storage medium for both programs and data.

- Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

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

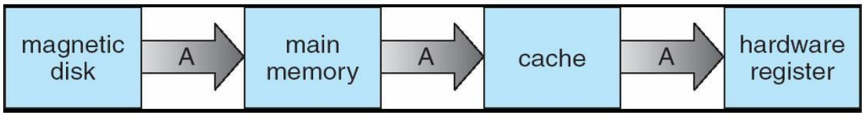
- Free-space management
- Storage allocation
- Disk scheduling

As the secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may depend on the speeds of the disk. Magnetic tape drives and their tapes, CD, DVD drives and platters are **tertiary storage** devices. The functions that operating systems provides include mounting and unmounting media in devices, allocating and freeing the devices for exclusive use by processes, and migrating data from secondary to tertiary storage.

Caching

- **Caching** is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system—the cache—as temporary data. When a particular piece of information is required, first we check whether it is in the cache. If it is, we use the information directly from the cache; if it is not in cache, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.
- Because caches have limited size, **cache management** is an important design problem. Careful selection of the cache size and page replacement policy can result in greatly increased performance.
- The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software. For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention. In contrast, transfer of data from disk to memory is usually controlled by the operating system.

- In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose to retrieve an integer A from magnetic disk to the processing program. The operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory. This operation is followed by copying A to the cache and to an internal register. Thus, the copy of A appears in several places: on the magnetic disk, in main memory, in the cache, and in an internal register.



- In a multiprocessor environment, in addition to maintaining internal registers, each of the CPUs also contains a local cache. In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute concurrently, any update done to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called **cache coherency,** and it is usually a hardware problem (handled below the operating-system level).

I/O Systems

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. The I/O subsystem consists of several components:

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

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

Protection and Security

- 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.
- For example, memory-addressing hardware ensures that a process can execute only within its own address space. The timer ensures that no process can gain control of the CPU for a long time. Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.
- **Protection** is a mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.
- Protection improves reliability. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage. A system can have adequate protection but still be prone to failure and allow inappropriate access.
- Consider a user whose authentication information is stolen. Her data could be copied or

deleted, even though file and memory protection are working. It is the job of **security** to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of service attacks etc.

- Protection and security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated **user identifiers (user IDs).** When a user logs in to the system, the authentication stage determines the appropriate user ID for the user.

Distributed Systems

- A distributed system is a collection of systems that are networked to provide the users with access to the various resources in the network. Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- A **network** is a communication path between two or more systems. Networks vary by the protocols used(TCP/IP,UDP,FTP etc.), the distances between nodes, and the transport media(copper wires, fiber-optic,wireless).
- TCP/IP is the most common network protocol. The operating systems support of protocols also varies. Most operating systems support TCP/IP, including the Windows and UNIX operating systems.
- Networks are characterized based on the distances between their nodes. A **local-area network (LAN)** connects computers within a room, a floor, or a building. A **wide-area network (WAN)** usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide. These networks may run one protocol or several protocols. A **metropolitan-area network** (MAN) connects buildings within a city. BlueTooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a **small-area network** such as might be found in a home.
- The transportation media to carry networks are also varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes, and radios. When computing devices are connected to cellular phones, they create a network.

Special Purpose Systems

Multimedia Systems

- Multimedia data consist of audio and video files as well as conventional files. These data differ from conventional data in that multimedia data—such as frames of video—must be delivered (streamed) according to certain time restrictions (for example, 30 frames per second).
- Multimedia describes a wide range of applications like audio files - MP3, DVD movies, video conferencing, and short video clips of movie previews or news. Multimedia applications may also include live webcasts of speeches or sporting events and even live webcams. Multimedia applications can be either audio or video or combination of both. For example, a movie may consist of separate audio and video tracks.

Handheld Systems

- Handheld systems include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones. Developers of these systems face many challenges, due to the limited memory, slow processors and small screens in such devices.
- The amount of physical memory in a handheld depends upon the device, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager when the memory is not being used. A second issue of concern to developers of handheld devices is the speed of the processor used in the devices. Processors for most handheld devices run at faster speed than the processor in a PC. Faster processors require more power and so, a larger battery is required. Another issue is the usage of I/O devices.
- Generally, the limitations in the functionality of PDAs are balanced by their convenience and portability. Their use continues to expand as network connections become more available and other options, such as digital cameras and MP3 players, expand their utility.

Computing Environments

The different computing environments are -

Traditional Computing

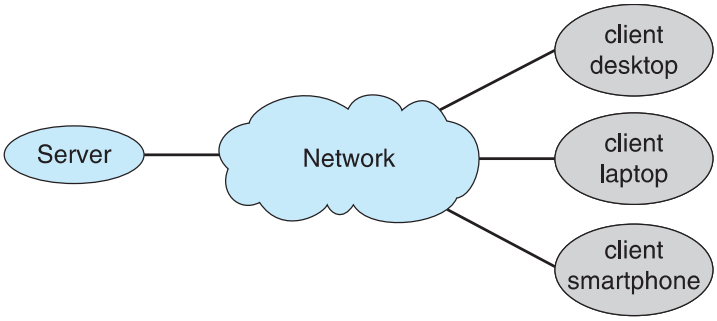
- The current trend is toward providing more ways to access these computing environments. Web technologies are stretching the boundaries of traditional computing. Companies establish **portals,** which provide web accessibility to their internal servers. **Network computers** are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. Handheld PDAs can also connect to **wireless networks** to use the company's web portal. The fast data connections are allowing home computers to serve up web pages and to use networks. Some homes even have **firewalls** to protect their networks.

- In the latter half of the previous century, computing resources were scarce. Years before, systems were either batch or interactive. Batch system processed jobs in bulk, with predetermined input (from files or other sources of data). 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 rapidly cycle processes through the CPU, giving each user a share of the resources.

- Today, traditional time-sharing systems are used everywhere. The same scheduling technique is still in use on workstations and servers, but frequently the processes are all owned by the same user (or a single user and the operating system). User processes, and system processes that provide services to the user, are managed so that each frequently gets a slice of computer time.

Client-Server Computing

Designers shifted away from centralized system architecture to - terminals connected to centralized systems. As a result, many of today's systems act as **server systems to** satisfy requests generated by **client systems.** This form of specialized distributed system, called **client- server** system.



General Structure of Client – Server System

Server systems can be broadly categorized as compute servers and file servers:

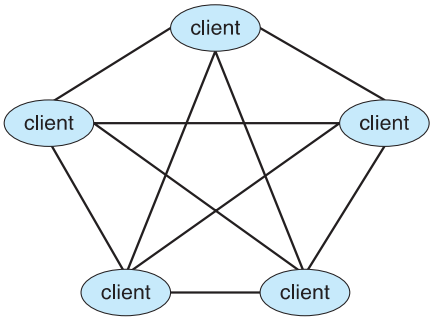
- The **compute-server system** provides an interface to which a client can send a request to perform an action (for example, read data); in response, the server executes the action and sends back results to the client. A server running a database that responds to client requests for data is an example of such a system.
- The **file-server system** provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running the web browsers.

Peer-to-Peer Computing

- In this model, clients and servers are not distinguished from one another; here, 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.
- In a client-server system, the server is a bottleneck, because all the services must be served by the server. 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—and requesting services from—other nodes in the network.

Determining what services are available is accomplished in one of two general ways:

- When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
- A peer acting as a client must know, which node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) 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.



Web-Based Computing

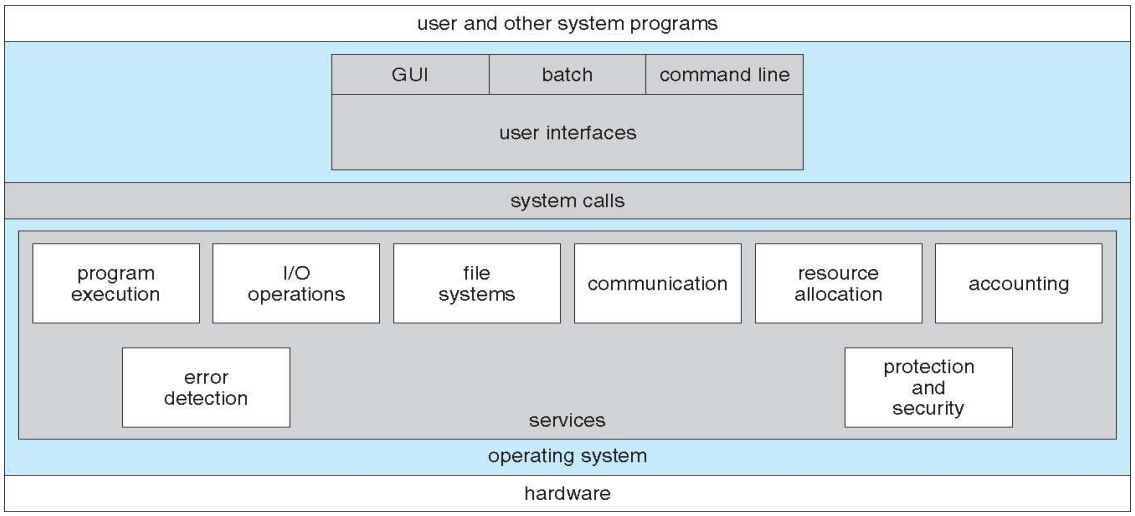
- Web computing has increased the importance on networking. Devices that were not previously networked now include wired or wireless access. Devices that were networked now have faster network connectivity.
- The implementation of web-based computing has given rise to new categories of devices, such as **load balancers,** which distribute network connections among a pool of similar servers. Operating systems like Windows 95, which acted as web clients, have evolved into Linux and Windows XP, which can act as web servers as well as clients. Generally, the Web has increased the complexity of devices, because their users require them to be web-enabled.
- The design of an operating system is a major task. It is important that the goals of the new system be well defined before the design of OS begins. These goals form the basis for choices among various algorithms and strategies.

OPERATING SYSTEM SERVICES

Operating-System Services

Q) List and explain the services provided by OS for the user and efficient operation of system.

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



OS provide services for the users of the system, including:

- **User Interfaces** - Means by which users can issue commands to the system. Depending on the operating system these may be a **command-line interface** (e.g. sh, csh, ksh, tcsh, etc.), a **Graphical User Interface** (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a **batch command systems**.
  In Command Line Interface (CLI)- commands are given to the system.
  In Batch interface – commands and directives to control these commands are put in a file and then the file is executed.
  In GUI systems- windows with pointing device to get inputs and keyboard to enter the text.
- **Program Execution** - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions are provided (device drivers) by OS.
- **File-System Manipulation** – Programs need to read and write files or directories. The services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS.

- **Communications** - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented by using the service of OS- like shared memory or message passing.
- **Error Detection** - Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the CPU and memory hardware (such as power failure and memory error), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location).

OS provide services for the efficient operation of the system, including:

- **Resource Allocation** – Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time.
- **Accounting** – There are services in OS to keep track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.
- **Protection and Security** – The owners of information (file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders must also be done, by means of a password.

User Operating-System Interface

There are several ways for users to interface with the operating system.

i) Command-line interface, or command interpreter, allows users to directly enter commands to be performed by the operating system.
ii) Graphical user interface (GUI), allows users to interface with the operating system using pointer device and menu system.

Command Interpreter

- Command Interpreters are used to give commands to the OS. There are multiple command interpreters known as shells. In UNIX and Linux systems, there are several different shells, like the *Bourne shell, C shell, Bourne-Again shell, Korn shell,* and others.
- The main function of the command interpreter is to get and execute the user-specified command. Many of the commands manipulate files: create, delete, list, print, copy, execute, and so on.

The commands can be implemented in two general ways-

i) The command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a particular section of its code that sets up the parameters and makes the appropriate system call.
ii) The code to implement the command is in a function in a separate file. The interpreter searches for the file and loads it into the memory and executes it by passing the parameter.

Thus by adding new functions new commands can be added easily to the interpreter without disturbing it.

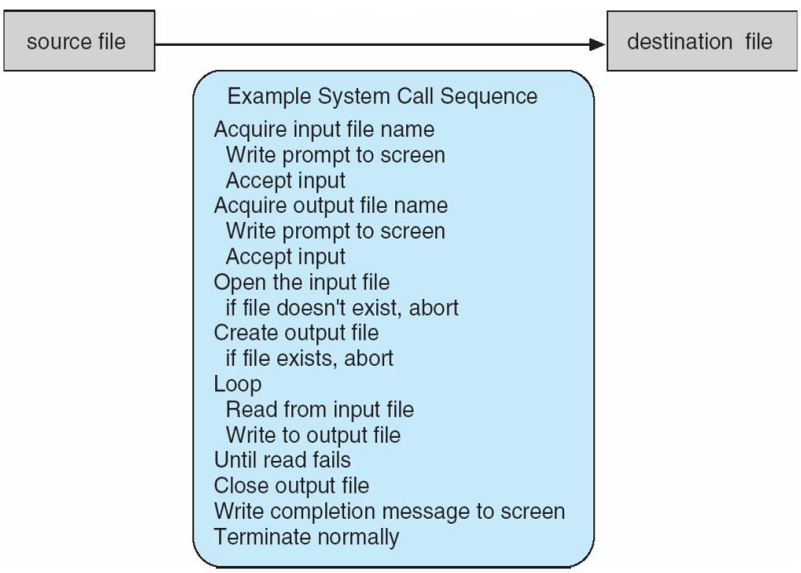
Graphical User Interfaces

- A second strategy for interfacing with the operating system is through a userfriendly graphical user interface or GUI. Rather than having users directly enter commands via a command-line interface, a GUI allows provides a mouse-based window-and-menu system as an interface.
- A GUI provides a **desktop** metaphor where the mouse is moved to position its pointer on images, or **icons,** on the screen (the desktop) that represent programs, files, directories, and system functions.
- Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a **folder—** or pull down a menu that contains commands.

System Calls

Q) What are system calls? Briefly point out its types.

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



An example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file

- There are number of system calls used to finish this task. The first system call is to write a message on the screen (monitor). Then to accept the input filename. Then another system call to write message on the screen, then to accept the output filename.

- When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another system call) and then terminate abnormally (another system call) and create a new one (another system call).
- Now that both the files are opened, we enter a loop that reads from the input file (another system call) and writes to output file (another system call).
- Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (system call), and finally terminate normally (final system call).

- Most programmers do not use the low-level system calls directly, but instead use an "Application Programming Interface", API.
- Instead of direct system calls provides for greater program portability between different systems. The API then makes the appropriate system calls through the system call interface, using a system call table to access specific numbered system calls.
- Each system call has a specific numbered system call. The system call table (consisting of system call number and address of the particular service) invokes a particular service routine for a specific system call.
- The caller need know nothing about how the system call is implemented or what it does during execution.

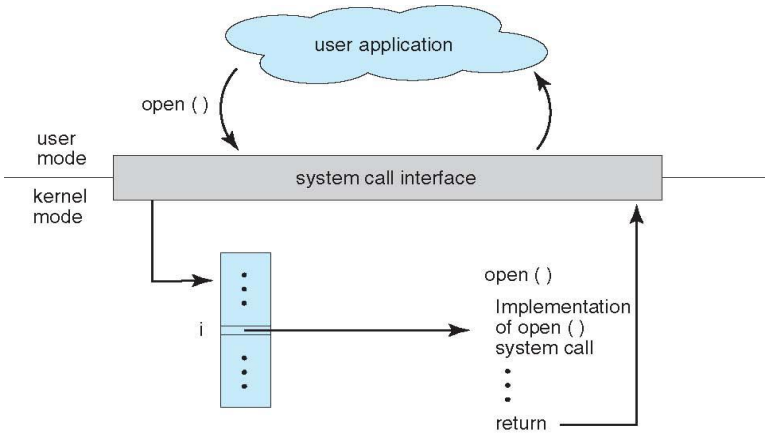


Figure: The handling of a user application invoking the open() system call.

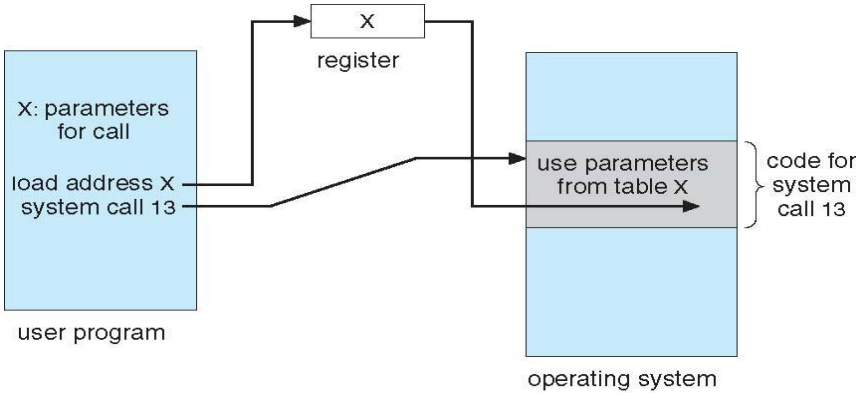


Figure: Passing of parameters as a table.

Three general methods used to pass parameters to OS are –

i) To pass parameters in registers
ii) If parameters are large blocks, address of block (where parameters are stored in memory) is sent to OS in the register. (Linux & Solaris).
iii) Parameters can be pushed onto the stack by program and popped off the stack by OS.

Types of System Calls

The system calls can be categorized into six major categories:

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

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure: Types of system calls

1. Process Control

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed, and eventually stopped.
- When one process pauses or stops, then another must be launched or resumed
- Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
- After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur (wait event). The process sends back a signal when the event has occurred (signal event)

2. File Management

The file management functions of OS are –

- File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
- After **creating** a file, the file is **opened**. Data is **read** or **written** to a file.
- The file pointer may need to be **repositioned** to a point.
- The file **attributes** like filename, file type, permissions, etc. are set and retrieved using system calls.
- These operations may also be supported for directories as well as ordinary files.

3. Device Management

- Device management system calls include **request device**, **release** device, **read**, **write**, **reposition**, **get/set** device attributes, and logically **attach** or **detach** devices.
- When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
- In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
- Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).

4. Information Maintenance

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- These system calls care used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.

5. Communication

- Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
- The **message passing** model must support calls to:
  - Identify a remote process and/or host with which to communicate.
  - Establish a connection between the two processes.
  - Open and close the connection as needed.
  - Transmit messages along the connection.
  - Wait for incoming messages, in either a blocking or non-blocking state.
  - Delete the connection when no longer needed.
- The **shared memory** model must support calls to:
  - Create and access memory that is shared amongst processes (and threads.)
  - Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared. This model is difficult to implement, and it consists of only few system calls.

6. Protection

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non- privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.

System Programs

Q) List and explain the different categories of system program?

A collection of programs that provide a convenient environment for program development and execution (other than OS) are called system programs or system utilities.

System programs may be divided into five categories:

1. **File management** - programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
2. **Status information** - Utilities to check on the date, time, number of users, processes running, data logging, etc. System registries are used to store and recall configuration information for particular applications.
3. **File modification** - e.g. text editors and other tools which can change file contents.
4. **Programming-language support** - E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.

5. **Program loading and execution** - loaders, dynamic loaders, overlay loaders, etc., as well as interactive debuggers.
6. **Communications** - Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.

Operating-System Design and Implementation

Design Goals

- The first problem in designing a system is to define goals and specifications. At the highest level, the design of the system will be affected by the choice of hardware and the type of system: batch, time shared, single user, multiuser, distributed, real time, or general purpose.
- Beyond this highest design level, the requirements may be much harder to specify. The requirements can, however, be divided into two basic groups
    1. User goals (User requirements)
    2. System goals (system requirements)

- **User requirements** are the features that user care about and understand like system should be convenient to use, easy to learn, reliable, safe and fast.
- **System requirements** are written for the developers, ie. People who design the OS. Their requirements are like easy to design, implement and maintain, flexible, reliable, error free and efficient.

Mechanisms and Policies

- Policies determine *what* is to be done. Mechanisms determine *how* it is to be implemented.
- Example: in timer- counter and decrementing counter is the mechanism and deciding how long the time has to be set is the policies.
- Policies change overtime. In the worst case, each change in policy would require a change in the underlying mechanism.
- If properly separated and implemented, policy changes can be easily adjusted without re-writing the code, just by adjusting parameters or possibly loading new data / configuration files.

Implementation

- Traditionally OS were written in assembly language.
- In recent years, OS are written in C, or C++. Critical sections of code are still written in assembly language.
- The first OS that was not written in assembly language, it was the Master Control Program (MCP).
- The advantages of using a higher-level language for implementing operating systems are: The code can be written faster, more compact, easy to port to other systems and is easier to understand and debug.
- The only disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.

Operating-System Structure

Simple Structure

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

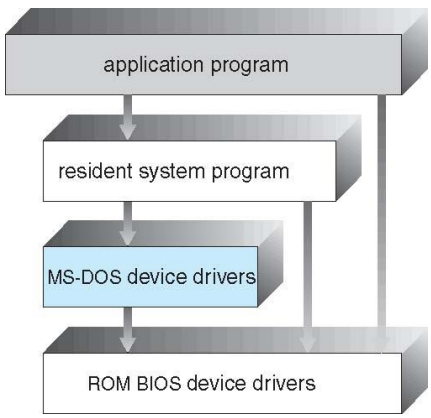


Figure: MS-DOS layer structure.

Layered Approach

- The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.
- Bottom layer (layer 0) is the hardware and the topmost layer is the user interface.
- A typical layer, consists of data structure and routines that can be invoked by higher-level layer.
- Advantage of layered approach is simplicity of construction and debugging.
- The layers are selected so that each uses functions and services of only lower-level layers. So simplifies debugging and system verification. The layers are debugged one by one from the lowest and if any layer doesn't work, then error is due to that layer only, as the lower layers are already debugged. Thus, the design and implementation are simplified.
- A layer need not know how its lower-level layers are implemented. Thus hides the operations from higher layers.

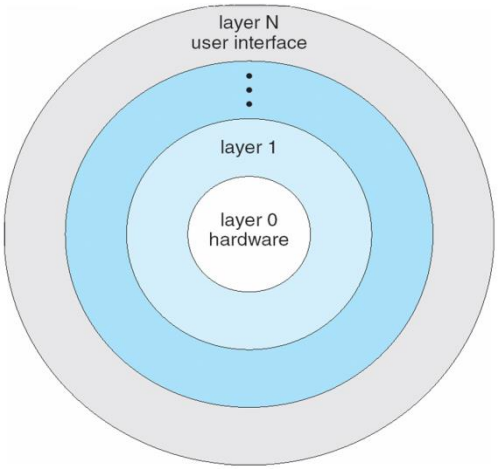


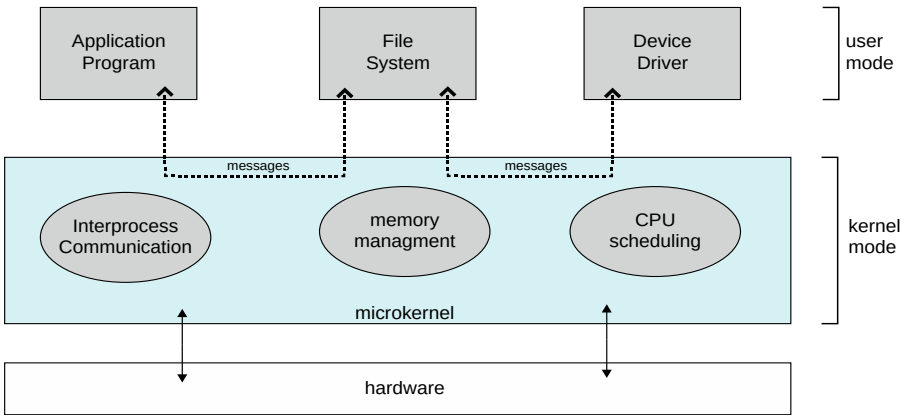
Figure: A layered Operating System

Disadvantages of layered approach:

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

Microkernels

- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs thus making the kernel as small and efficient as possible.
- The removed services are implemented as system applications.
- Most microkernels provide basic process and memory management, and message passing between other services.
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.



Benefit of microkernel –

- System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.

Disadvantage of Microkernel -

- Performance overhead of user space to kernel space communication

Modules

- Modern OS development is object-oriented, with a relatively small core kernel and a set of ***modules*** which can be linked in dynamically.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly. Eg: Solaris, Linux and MacOSX.

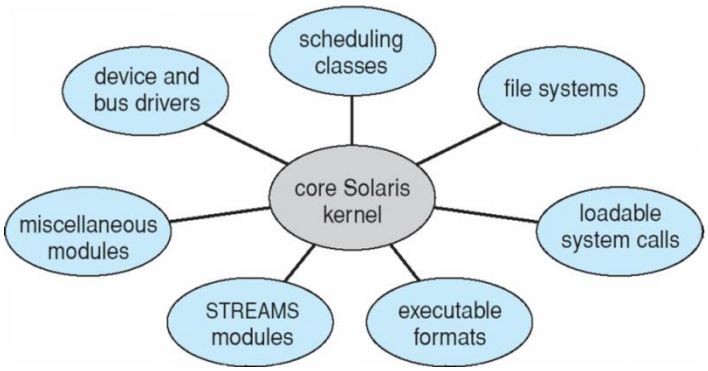


Figure: Solaris loadable modules

- The Max OSX architecture relies on the Mach microkernel for basic system management services, and the BSD kernel for additional services. Application services and dynamically loadable modules (kernel extensions) provide the rest of the OS functionality.
- Resembles layered system, but a module can call any other module.
- Resembles microkernel, the primary module has only core functions and the knowledge of how to load and communicate with other modules.

Virtual Machines

Q) Demonstrate the concept of virtual machine with an example

- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.
- Creates an illusion that a process has its own processor with its own memory.
- Host OS is the main OS installed in system and the other OS installed in the system are called guest OS.

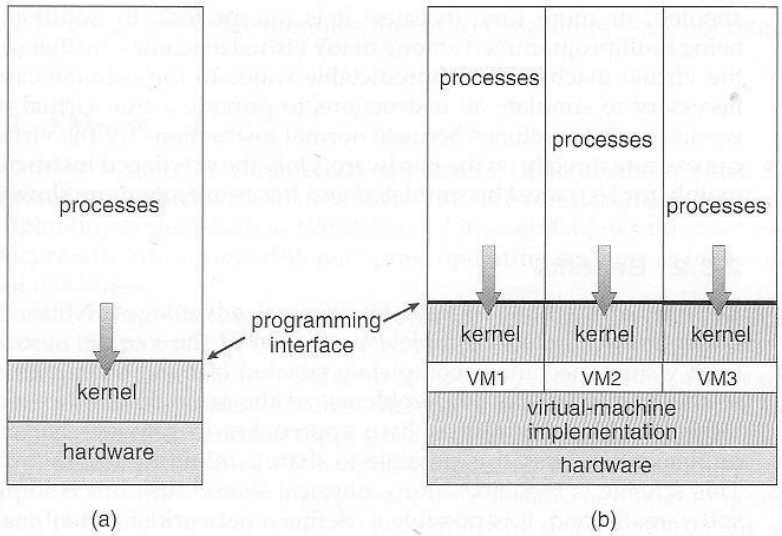


Figure: System modes. (A) Non-virtual machine (b) Virtual machine

Virtual machines first appeared as the VM Operating System for IBM mainframes in 1972.

Implementation

- The virtual-machine concept is useful, it is difficult to implement.
- Work is required to provide an exact duplicate of the underlying machine. Remember that the underlying machine has two modes: user mode and kernel mode.
- The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute in only user mode.

Benefits

- Able to share the same hardware and run several different execution environments (OS).
- Host system is protected from the virtual machines and the virtual machines are protected from one another. A virus in guest OS, will corrupt that OS but will not affect the other guest systems and host systems.
- Even though the virtual machines are separated from one another, software resources can be shared among them. Two ways of sharing s/w resource for communication are:
    - To share a file system volume (part of memory).
    - To develop a virtual communication network to communicate between the virtual machines.
- The operating system runs on and controls the entire machine. Therefore, the current system must be stopped and taken out of use while changes are made and tested. This period is commonly called *system development time.* In virtual machines such problem is eliminated. User programs are executed in one virtual machine and system development is done in another environment.
- Multiple OS can be running on the developer's system **concurrently**. This helps in rapid porting and testing of programmer's code in different environments.
- **System consolidation** – two or more systems are made to run in a single system.

Simulation –

Here the host system has one system architecture and the guest system is compiled in different architecture. The compiled guest system programs can be run in an emulator that translates each instructions of guest program into native instructions set of host system.

Para-Virtualization –

This presents the guest with a system that is similar but not identical to the guest's preferred system. The guest must be modified to run on the para-virtualized hardware.

Examples

VMware

- VMware is a popular commercial application that abstracts Intel 80X86 hardware into isolated virtual machines. The virtualization tool runs in the user-layer on top of the host OS. The virtual machines running in this tool believe they are running on bare hardware, but the fact is that it is running inside a user-level application.
- VMware runs as an application on a host operating system such as Windows or Linux and allows this host system to concurrently run several different **guest operating systems** as independent virtual machines.

In below scenario, Linux is running as the host operating system; FreeBSD, Windows NT, and Windows XP are running as guest operating systems. The virtualization layer is the heart of VMware, as it abstracts the physical hardware into isolated virtual machines running as guest operating systems. Each virtual machine has its own virtual CPU, memory, disk drives, network interfaces, and so forth.

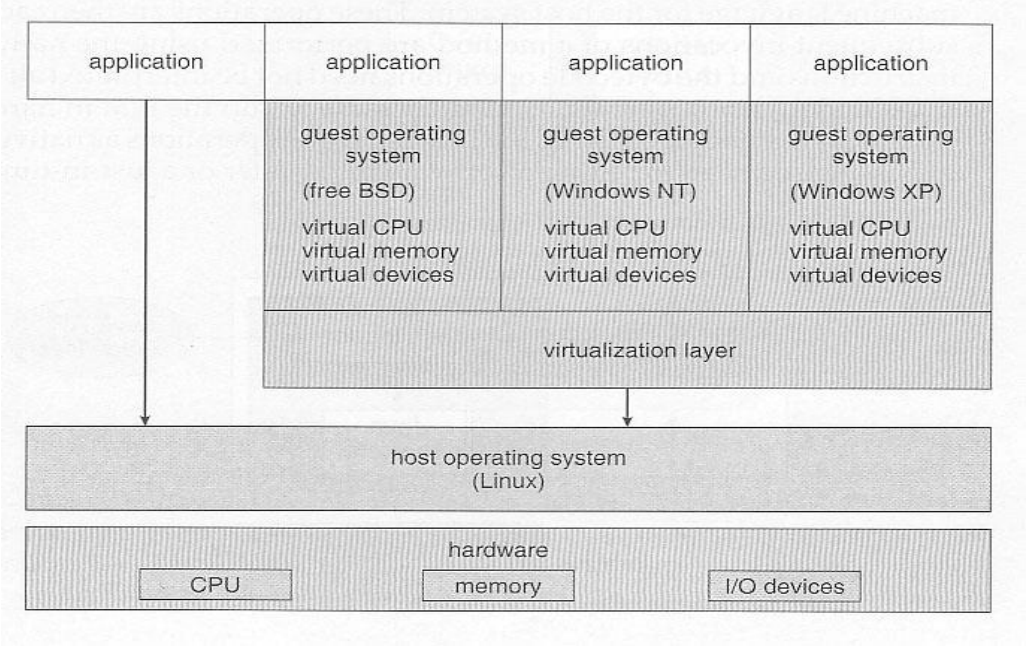


Figure: VMware architecture

The Java Virtual Machine

- Java was designed from the beginning to be platform independent, by running Java only on a Java Virtual Machine, JVM, of which different implementations have been developed for numerous different underlying HW platforms.
- Java source code is compiled into Java byte code in .class files. Java byte code is binary instructions that will run on the JVM.
- The JVM implements memory management and garbage collection.
- JVM consists of class loader and Java Interpreter. Class loader loads compiled .class files from both Java program and Java API for the execution of Java interpreter. Then it checks the .class file for validity.

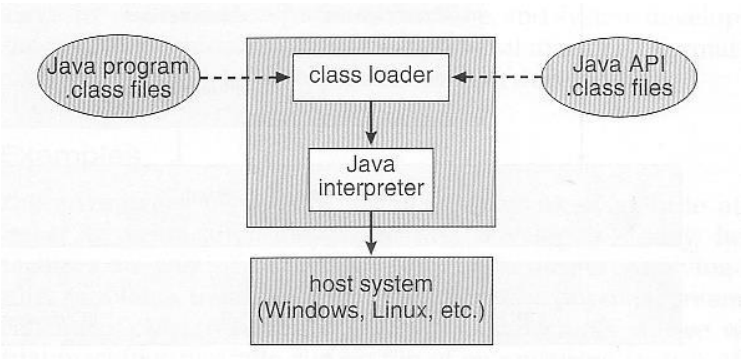


Figure: The JVM

Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
    - *Booting* – starting a computer by loading the kernel.
    - *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

System Boot

- Operating system must be made available to hardware so hardware can start it.
- Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it Sometimes two-step process where **boot block** at fixed location loads bootstrap loader.
- When power initialized on system, execution starts at a fixed memory location Firmware used to hold initial boot code

PROCESS MANAGEMENT

Process Concept

- A process is a program under execution.
- Its current activity is indicated by PC (Program Counter) and the contents of the processor's registers.

The Process

Process memory is divided into four sections as shown in the figure below:

- The **stack** is used to store temporary data such as local variables, function parameters, function return values, return address etc.
- The **heap** which is memory that is dynamically allocated during process run time
- The **data** section stores global variables.
- The **text** section comprises the compiled program code.
- Note that, there is a free space between the stack and the heap. When the stack is full, it grows downwards and when the heap is full, it grows upwards.

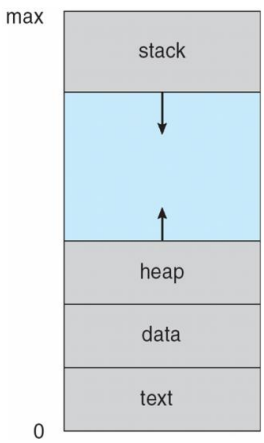


Figure: Process in memory.

Process State

Q) Illustrate with a neat sketch, the process states and process control block.

Process State

A Process has 5 states. Each process may be in one of the following states –

1. **New** - The process is in the stage of being created.
2. **Ready** - The process has all the resources it needs to run. It is waiting to be assigned to the processor.
3. **Running** – Instructions are being executed.
4. **Waiting** - The process is waiting for some event to occur. For example, the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
5. **Terminated** - The process has completed its execution.

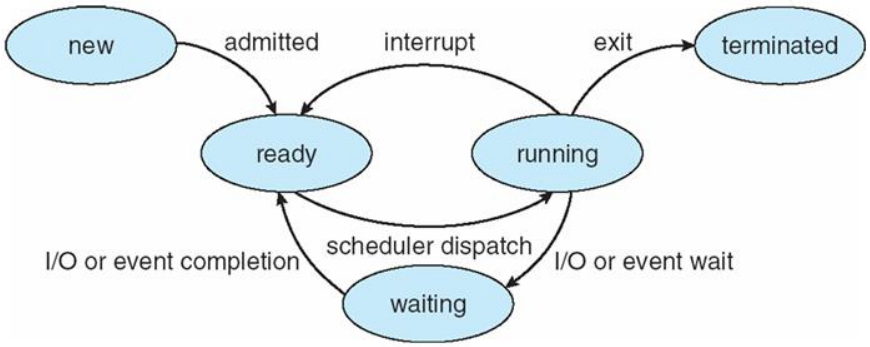


Figure: Diagram of process state

Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

- **Process State** – The state of the process may be new, ready, running, waiting, and so on.
- **Program counter –** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers** - The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
- **CPU scheduling information**- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information** – This includes information such as the value of the base and limit registers, the page tables, or the segment tables.
- **Accounting information** – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information –** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

The PCB simply serves as the repository for any information that may vary from process to process.

Figure: Process control block (PCB)

CPU Switch from Process to Process

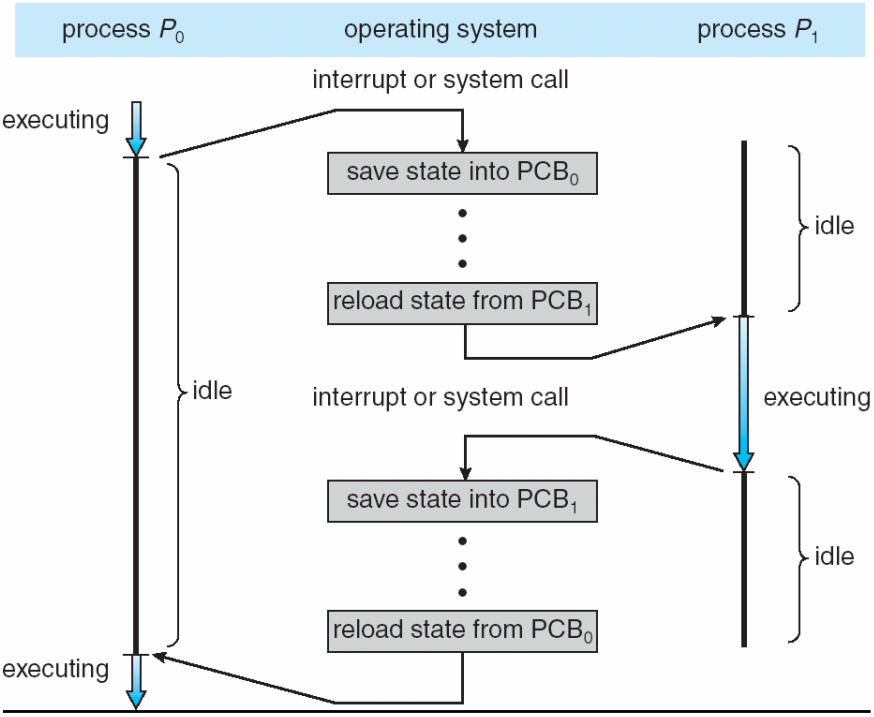


Figure: Diagram showing CPU switch from process to process.

Process Scheduling

Scheduling Queues

- As processes enter the system, they are put into a job queue, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list.
- A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

Ready Queue and Various I/O Device Queues

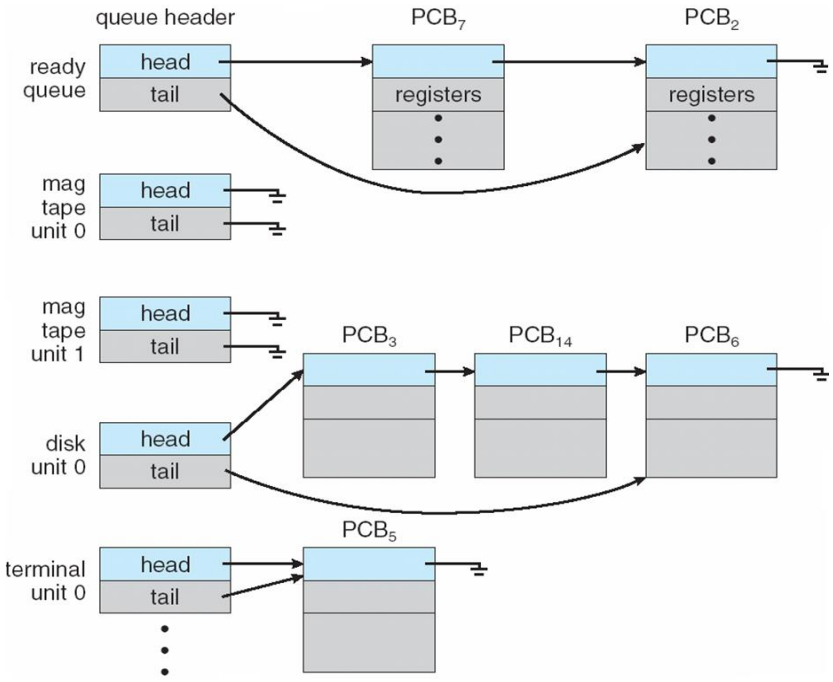


Figure: The ready queue and various I/O device queues

- A common representation of process scheduling is a ***queueing diagram***. Each rectangular box in the diagram represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution and is given the CPU. Once the process is allocated the CPU and is executing, one of several events could occur:
    - The process could issue an I/O request, and then be placed in an I/O queue.
    - The process could create a new subprocess and wait for its termination.
    - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues.

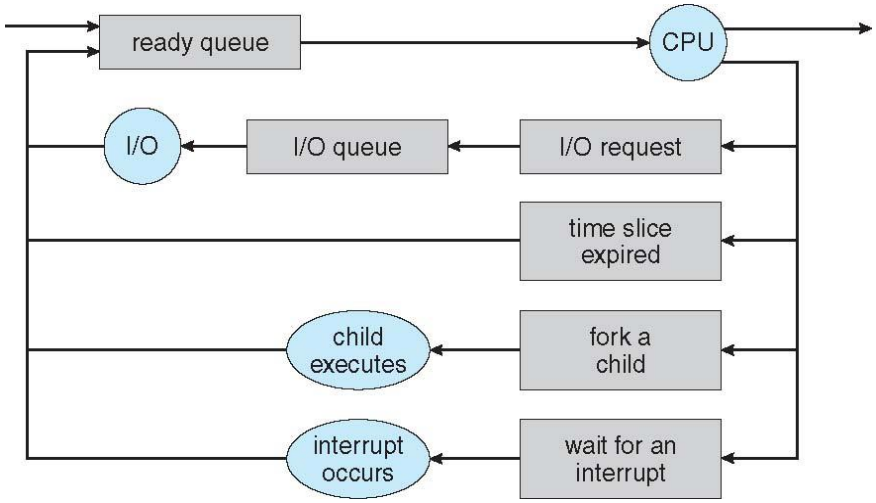


Figure: Queueing-diagram representation of process scheduling.

Schedulers

Schedulers are software which selects an available program to be assigned to CPU.

- A **long-term scheduler or Job scheduler** – selects jobs from the job pool (of secondary memory, disk) and loads them into the memory.
  If more processes are submitted, than that can be executed immediately, such processes will be in secondary memory. It runs infrequently, and can take time to select the next process.

- **The short-term scheduler, or CPU Scheduler** – selects job from memory and assigns the CPU to it. It must select the new process for CPU frequently.
- **The medium-term scheduler** - selects the process in ready queue and reintroduced into the memory.

Processes can be described as either:

- I/O-bound process – spends more time doing I/O than computations,
- CPU-bound process – spends more time doing computations and few I/O operations.

An efficient scheduling system will select a good mix of **CPU-bound** processes and **I/O bound** processes.

- If the scheduler selects **more I/O bound process**, then I/O queue will be full and ready queue will be empty.
- If the scheduler selects **more CPU bound process**, then ready queue will be full and I/O queue will be empty.

Time sharing systems employ a **medium-term scheduler**. It swaps out the process from ready queue and swap in the process to ready queue. When system loads get high, this scheduler will swap one or more processes out of the ready queue for a few seconds, in order to allow smaller faster jobs to finish up quickly and clear the system.

Advantages of medium-term scheduler –

- To remove process from memory and thus reduce the degree of multiprogramming (number of processes in memory).
- To make a proper mix of processes (CPU bound and I/O bound)

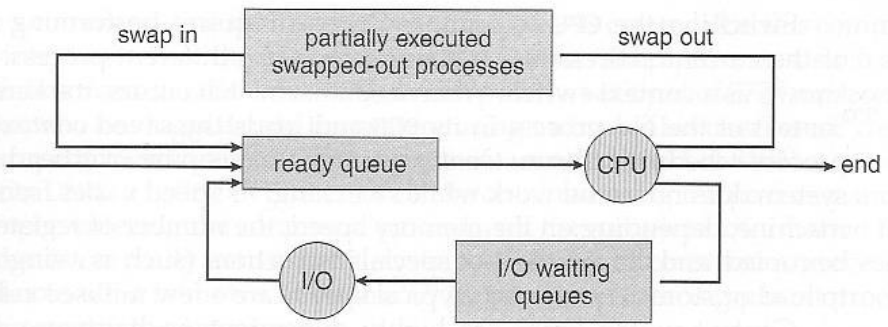


Figure 3.8 Addition of medium-term scheduling to the queueing diagram.

Context switching

- The task of switching a CPU from one process to another process is called context switching. Context-switch times are highly dependent on hardware support (Number of CPU registers).
- Whenever an interrupt occurs (hardware or software interrupt), the state of the currently running process is saved into the PCB and the state of another process is restored from the PCB to the CPU.
- Context switch time is an overhead, as the system does not do useful work while switching.

Operations on Processes

Q) Demonstrate the operations of process creation and process termination in UNIX

Process Creation

- A process may create several new processes. The creating process is called a **parent** process, and the new processes are called the **children** of that process. Each of these new processes may in turn create other processes. Every process has a unique process ID.
- On typical Solaris systems, the process at the top of the tree is the '**sched'** process with PID of 0. The **'sched'** process creates several children processes – **init**, **pageout** and **fsflush**. Pageout and fsflush are responsible for managing memory and file systems. The init process with a PID of 1, serves as a parent process for all user processes.

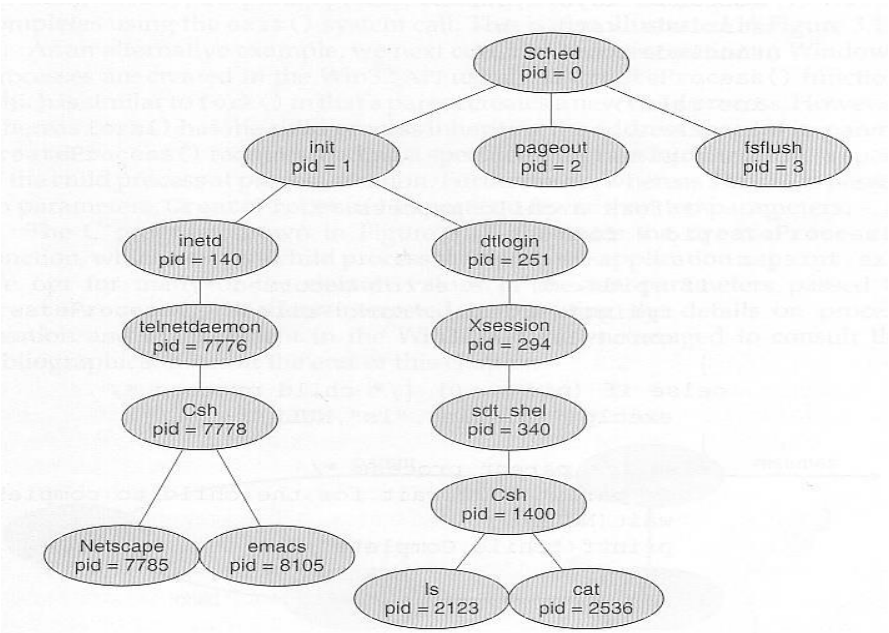


Figure 3.9 A tree of processes on a typical Solaris system.

A process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task. When a process creates a subprocess, the subprocess may be able to obtain its resources in two ways:

- directly from the operating system
- Subprocess may take the resources of the parent process.

  The resource can be taken from parent in two ways –
  - The parent may have to partition its resources among its children
  - Share the resources among several children.

There are two options for the parent process after creating the child:

- Wait for the child process to terminate and then continue execution. The parent makes a wait() system call.
- Run concurrently with the child, continuing to execute without waiting.

Two possibilities for the address space of the child relative to the parent:

- The child may be an exact duplicate of the parent, sharing the same program and data segments in memory. Each will have their own PCB, including program counter, registers, and PID. This is the behaviour of the **fork** system call in UNIX.
- The child process may have a new program loaded into its address space, with all new code and data segments. This is the behaviour of the **spawn** system calls in Windows.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) {/* error occurred */
       fprintf(stderr, "Fork Failed");
       exit(-1);
    }
    else if (pid == 0) {/* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else {/* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

Figure 3.10 C program forking a separate process.

In UNIX OS, a child process can be created by **fork()** system call. The **fork** system call, if successful, returns the PID of the child process to its parents and returns a zero to the child process. If failure, it returns -1 to the parent. Process IDs of current process or its direct parent can be accessed using the getpid( ) and getppid( ) system calls respectively.

The parent waits for the child process to complete with the wait() system call. When the child process completes, the parent process resumes and completes its execution.

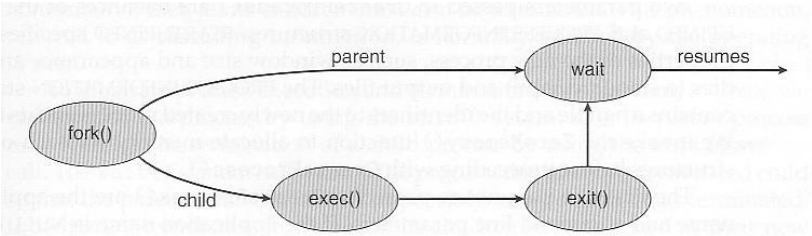


Figure 3.11 Process creation.

In windows the child process is created using the function **createprocess( )**. The createprocess( ) returns 1, if the child is created and returns 0, if the child is not created.

Process Termination

- A process terminates when it finishes executing its last statement and asks the operating system to delete it, by using the **exit** () system call. All of the resources assigned to the process like memory, open files, and I/O buffers, are deallocated by the operating system.
- A process can cause the termination of another process by using appropriate system call. The parent process can terminate its child processes by knowing of the PID of the child.
- A parent may terminate the execution of children for a variety of reasons, such as:
    - The child has exceeded its usage of the resources, it has been allocated.
    - The task assigned to the child is no longer required.
    - The parent is exiting, and the operating system terminates all the children. This is called **cascading termination**.

Interprocess Communication

Q) What is interprocess communication? Explain types of IPC.

Interprocess Communication- Processes executing may be either co-operative or independent processes.

- ***Independent Processes*** – processes that cannot affect other processes or be affected by other processes executing in the system.
- ***Cooperating Processes*** – processes that can affect other processes or be affected by other processes executing in the system.

Co-operation among processes are allowed for following reasons –

- **Information Sharing** - There may be several processes which need to access the same file. So the information must be accessible at the same time to all users.
- **Computation speedup** - Often a solution to a problem can be solved faster if the problem can be broken down into sub-tasks, which are solved simultaneously (particularly when multiple

processors are involved.)

- **Modularity** - A system can be divided into cooperating modules and executed by sending information among one another.
- **Convenience** - Even a single user can work on multiple tasks by information sharing.

Cooperating processes require some type of inter-process communication. This is allowed by two models:

1. Shared Memory systems
2. Message passing systems.

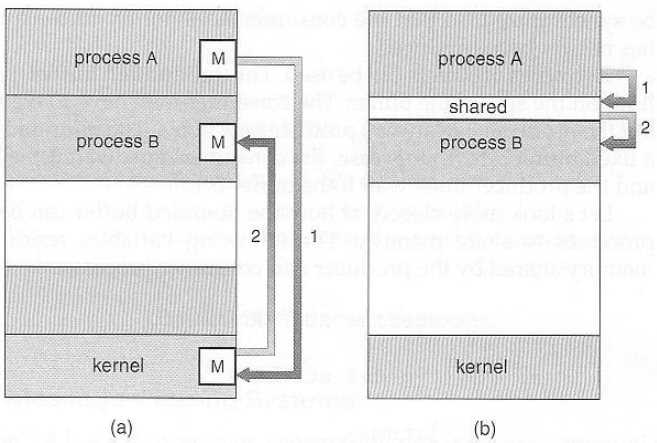


Figure 3.13 Communications models. (a) Message passing. (b) Shared memory.

Sl No	**Shared Memory**	**Message passing**
1.	A region of memory is shared by communicating processes, into which the information is written and read	Message exchange is done among the processes by using objects.
2.	Useful for sending large block of data	Useful for sending small data.
3.	System call is used only to create shared memory	System call is used during every read and write operation.
4.	Message is sent faster, as there are no system calls	Message is communicated slowly.

- **Shared Memory** is faster once it is set up, because no system calls are required and access occurs at normal memory speeds. Shared memory is generally preferable when large amounts of information must be shared quickly on the same computer.
- **Message Passing** requires system calls for every message transfer, and is therefore slower, but it is simpler to set up and works well across multiple computers. Message passing is generally preferable when the amount and/or frequency of data transfers is small.

Shared-Memory Systems

- A region of shared-memory is created within the address space of a process, which needs to communicate. Other process that needs to communicate uses this shared memory.
- The form of data and position of creating shared memory area is decided by the process. Generally, a few messages must be passed back and forth between the cooperating processes first in order to set up and coordinate the shared memory access.
- The process should take care that the two processes will not write the data to the shared memory at the same time.

Producer-Consumer Example Using Shared Memory

- This is a classic example, in which one process is producing data and another process is consuming the data.
- The data is passed via an intermediary buffer (shared memory). The producer puts the data to the buffer and the consumer takes out the data from the buffer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.
- There are two types of buffers into which information can be put –
    - Unbounded buffer
    - Bounded buffer

- **With Unbounded buffer**, there is no limit on the size of the buffer, and so on the data produced by producer. But the consumer may have to wait for new items.

- **With bounded-buffer** – As the buffer size is fixed. The producer has to wait if the buffer is full and the consumer has to wait if the buffer is empty.

This example uses shared memory as a circular queue. The **in** and **out** are two pointers to the array. Note in the code below that only the producer changes "in", and only the consumer changes "out".

- First the following data is set up in the shared memory area:

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
}item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- The producer process –
  Note that the buffer is full when [(in+1) % BUFFER_SIZE == out]

```
item nextProduced;

while (true) {
     /* produce an item in nextProduced */
     while (((in + 1) % BUFFER-SIZE) == out)
        ; /* do nothing */
     buffer[in] = nextProduced;
     in = (in + 1) % BUFFER_SIZE;
}
```

Figure The producer process.

- The consumer process –
  Note that the buffer is empty when [in == out]

```
item nextConsumed;

while (true) {
     while (in == out)
        ; //do nothing

     nextConsumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
     /* consume the item in nextConsumed */
}
```

Figure The consumer process.

Message-Passing Systems

A mechanism to allow process communication without sharing address space. It is used in distributed systems.

- Message passing systems uses system calls for "send message" and "receive message".
- A communication link must be established between the cooperating processes before messages can be sent.
- There are three methods of creating the link between the sender and the receiver-
  - Direct or indirect communication (naming)
  - Synchronous or asynchronous communication (Synchronization)
  - Automatic or explicit buffering.

1. Naming

Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication.

a) Direct communication the sender and receiver must explicitly know each other's name. The syntax for send() and receive() functions are as follows-

- **send** (*P*, *message*) – send a message to process P
- **receive**(*Q*, *message*) – receive a message from process Q

Properties of communication link:

- A link is established automatically between every pair of processes that wants to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly one pair of communicating processes
- Between each pair, there exists exactly one link.

Types of addressing in direct communication –

- Symmetric addressing – the above-described communication is symmetric communication. Here both the sender and the receiver processes have to name each other to communicate.
- Asymmetric addressing – Here only the sender's name is mentioned, but the receiving data can be from any system.

  send (P, message) --- Send a message to process ***P***
  receive (id, message). Receive a message from any process

Disadvantages of direct communication – any changes in the identifier of a process, may have to change the identifier in the whole system (sender and receiver), where the messages are sent and received.

b) Indirect communication uses shared mailboxes, or ports.

A mailbox or port is used to send and receive messages. Mailbox is an object into which messages can be sent and received. It has a unique ID. Using this identifier messages are sent and received.

Two processes can communicate only if they have a shared mailbox. The send and receive functions are –

- **send** (*A*, *message*) – send a message to mailbox A
- **receive** (*A*, *message*) – receive a message from mailbox A

Properties of communication link:

- A link is established between a pair of processes only if they have a shared mailbox

- A link may be associated with more than two processes
- Between each pair of communicating processes, there may be any number of links, each link is associated with one mailbox.
- A mail box can be owned by the operating system. It must take steps to –
  - create a new mailbox
  - send and receive messages from mailbox
  - delete mailboxes.

2. Synchronization

The send and receive messages can be implemented as either **blocking** or **non-blocking**.

- **Blocking (synchronous) send -** sending process is blocked (waits) until the message is received by receiving process or the mailbox.
- **Non-blocking (asynchronous) send** - sends the message and continues (does not wait)

- **Blocking (synchronous) receive -** The receiving process is blocked until a message is available
- **Non-blocking (asynchronous) receive** - receives the message without block. The received message may be a valid message or null.

3. Buffering

When messages are passed, a temporary queue is created. Such queue can be of three capacities:

- **Zero capacity** – The buffer size is zero (buffer does not exist). Messages are not stored in the queue. The senders must block until receivers accept the messages.
- **Bounded capacity**- The queue is of fixed size(n). Senders must block if the queue is full. After sending 'n' bytes the sender is blocked.
- Unbounded capacity **- The queue is of infinite capacity. The sender never blocks.**

Question Bank

1. What is operating system? Explain multiprogramming and time-sharing system.
2. Explain dual mode operating in operating system with a neat block diagram.
3. What are system calls? Briefly print out its types.
4. What is Interprocess communication? Explain direct and indirect communication with respect to message passing system.
5. Analyze modular kernel approach with layered approach with a neat sketch.
6. List and explain the services provided by OS for the user and efficient operation of system.
7. Illustrate with a neat sketch, the process states and process control block.
8. Discuss the methods to implement message passing IPC in detail.
9. With a neat diagram, explain the concept of virtual machines.
10. Define the following terms
    - Virtual Machine
    - CPU scheduler
    - System call
    - Context switch
11. What is Interprocess communication? Explain direct and indirect communication with respect to message passing system.
12. Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.
13. What are the tradeoffs inherent in handheld computers?
14. Distinguish between the client-server and peer-to-peer models of distributed systems.
15. Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.
16. What are the main differences between operating systems for mainframe computers and personal computers?
17. Identify several advantages and several disadvantages of open-source operating systems. Include the types of people who would find each aspect to be an advantage or a disadvantage.
18. How do clustered systems differ from multiprocessor systems? What is required for two machines belonging to a cluster to cooperate to provide a highly available service?
19. What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

MODULE 2

MULTITHREADED PROGRAMMING

- A thread is a basic unit of CPU utilization.
- It consists of
  - thread ID
  - PC
  - register-set and
  - stack.
- It shares with other threads belonging to the same process its code-section &data-section.
- A traditional (or heavy weight) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

such a process is called **multithreaded process**

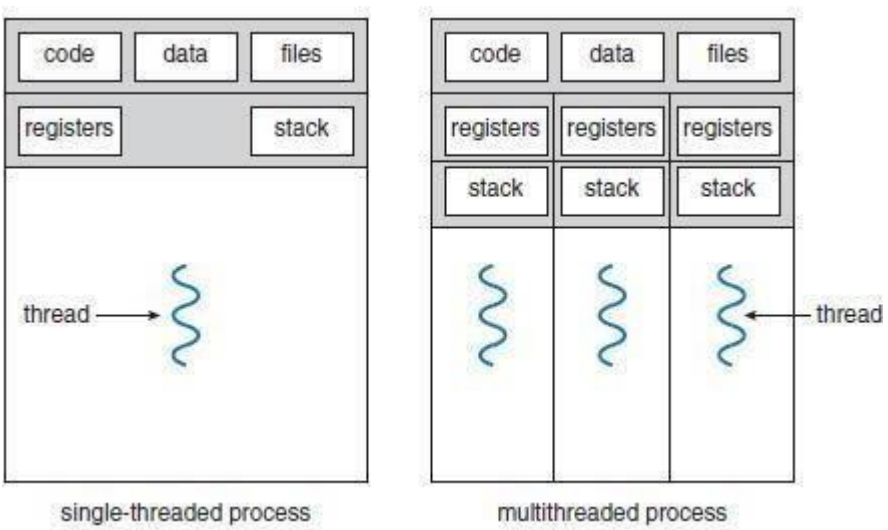


Fig: Single-threaded and multithreaded processes

Motivation for Multithreaded Programming

1. The software-packages that run on modern PCs are multithreaded. An application is implemented as a separate process with several threads of control. For ex: A word processor may have

- first thread for displaying graphics
- second thread for responding to key strokes and
- Third thread for performing grammar checking.

2. In some situations, a single application may be required to perform several similar tasks. For ex: A web-server may create a separate thread for each client requests. This allows the server to service several concurrent requests.

3. RPC servers are multithreaded.

- When a server receives a message, it services the message using separate concurrent threads.

4. Most OS kernels are multithreaded;

- Several threads operate in kernel, and each thread performs a specific task, such as managing devices or interrupt handling.

Benefits of Multithreaded Programming

- ***Responsiveness*** A program may be allowed to continue running even if part of it is blocked. Thus, increasing responsiveness to the user.

- ***Resource Sharing*** By default, threads share the memory (and resources) of the process to which they belong. Thus, an application is allowed to have several different threads of activity within the same address-space.
- ***Economy*** Allocating memory and resources for process-creation is costly. Thus, it is more economical to create and context-switch threads.
- ***Utilization of Multiprocessor Architectures*** In a multiprocessor architecture, threads may be running in parallel on different processors. Thus, parallelism will be increased.

MULTITHREADING MODELS

- Support for threads may be provided at either
  1. The user level, for **user threads** or
  2. By the kernel, for **kernel threads**.
- User-threads are supported above the kernel and are managed without kernel support. Kernel- threads are supported and managed directly by the OS.
- Three ways of establishing relationship between user-threads &kernel-threads:
  1. Many-to-one model
  2. One-to-one model and
  3. Many-to-many model.

Many-to-One Model

- Many user-level threads are mapped to one kernel thread.

  Advantages:

  - Thread management is done by the thread library in user space, so it is efficient.

  Disadvantages:

  - The entire process will block if a thread makes a blocking system-call.
  - Multiple threads are unable to run in parallel on multiprocessors.

- For example:
  - Solaris green threads
  - GNU portable threads.

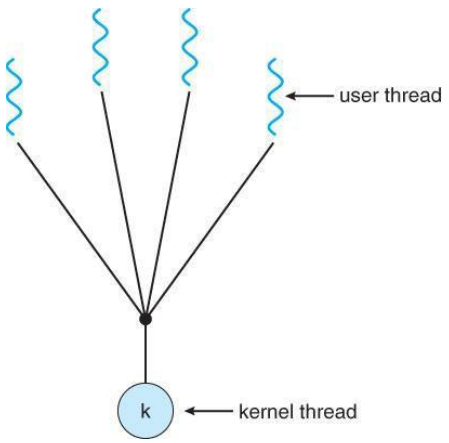


Fig: Many-to-one model

One-to-One Model

- Each user thread is mapped to a kernel thread.

  Advantages:

  - It provides more concurrency by allowing another thread to run when a thread makes a blocking system-call.
  - Multiple threads can run in parallel on multiprocessors.

  Disadvantage:

  - Creating a user thread requires creating the corresponding kernel thread.

- For example:
  - Windows NT/XP/2000, Linux

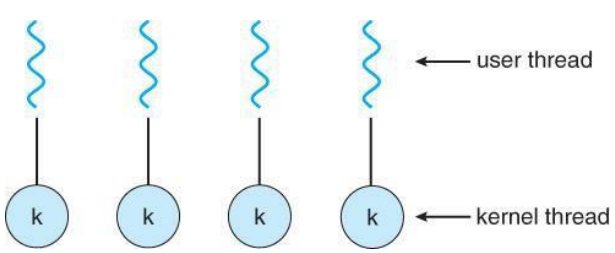


Fig: one-to-one model

Many-to-Many Model

- Many user-level threads are multiplexed to a smaller number of kernel threads.

  Advantages:

  - Developers can create as many user threads as necessary
  - The kernel threads can run in parallel on a multiprocessor.
  - When a thread performs a blocking system-call, kernel can schedule another thread for execution.

Two Level Model

- A variation on the many-to-many model is the two level-model
- Similar to M:N, except that it allows a user thread to be bound to kernel thread.
- For example:
  - HP-UX
  - Tru64 UNIX

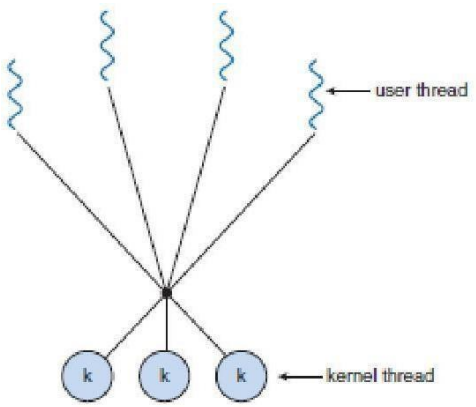


Fig: Many-to-many model

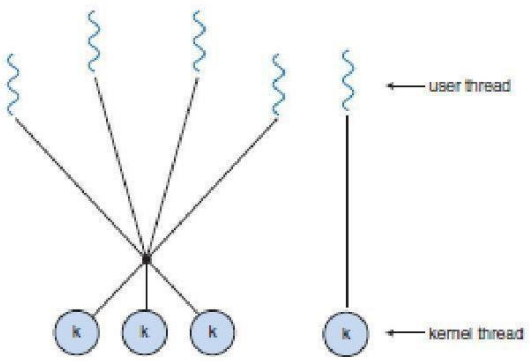


Fig: Two-level model

Thread Libraries

- It provides the programmer with an API for the creation and management of threads.

- Two ways of implementation:
  1. ***First Approach:***
     Provides a library entirely in user space with no kernel support. All code and data structures for the library exist in the user space.
  2. ***Second Approach***
     Provides a library entirely in user space with no kernel support. All code and data structures for the library exist in the user space.

Three main thread libraries:

1. POSIXP threads
2. Win32 and
3. Java.

P threads

- This is a POSIX standard API for thread creation and synchronization.
- This is a specification for thread-behavior, not an implementation.
- OS designers may implement the specification in any way they wish.
- Commonly used in: UNIX and Solaris.

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }

   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}
```

```
/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Win32 threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Java Threads

- Threads are the basic model of program-execution in
  - Java program and
  - Java language.
- The API provides a rich set of features for the creation and management of threads.
- All Java programs comprise at least a single thread of control.
- Two techniques for creating threads:
  1. Create a new class that is derived from the Thread class and override its run() method.
  2. Define a class that implements the Runnable interface. The Runnable interface is defined as follows:

```
public interface Runnable
{
    public abstract void run();
}
```

THREADING ISSUES

fork() and exec() System-calls

- fork() is used to create a separate, duplicate process.
- If one thread in a program calls fork(),then
    1. Some systems duplicates all threads and
    2. Other systems duplicate only the thread that invoked the forkO.
- If a thread invokes the exec(), the program specified in the parameter to exec() will replace the entire process including all threads.

Thread Cancellation

- This is the task of terminating a thread before it has completed.
- Target thread is the thread that is to be cancelled
- Thread cancellation occurs in two different cases:
    1. ***Asynchronous cancellation***: One thread immediately terminates the target thread.
    2. ***Deferred cancellation***: The target thread periodically checks whether it should be terminated.

Signal Handling

- In UNIX, a signal is used to notify a process that a particular event has occurred.
- All signals follow this pattern:
    1. A signal is generated by the occurrence of a certain event.
    2. A generated signal is delivered to a process.
    3. Once delivered, the signal must be handled.
- A signal handler is used to process signals.
- A signal may be received either synchronously or asynchronously, depending on the source.
    1. ***Synchronoussignals***
        - Delivered to the same process that performed the operation causing the signal.
        - E.g. illegal memory access and division by 0.
    2. ***Asynchronoussignals***
        - Generated by an event external to a running process.
        - E.g. user terminating a process with specific keystrokes<ctrl><c>.

- Every signal can be handled by one of two possible handlers:

  1. A Default Signal Handler

  - Run by the kernel when handling the signal.

  2. A User-defined Signal Handler

  - Overrides the default signal handler.
- In ***single-threaded programs***, delivering signals is simple (since signals are always delivered to a process).
- In ***multithreaded programs***, delivering signals is more complex. Then, the following options exist:
  1. Deliver the signal to the thread to which the signal applies.
  2. Deliver the signal to every thread in process
  3. Deliver the signal to certain threads in the process.
  4. Assign a specific thread to receive all signals for the process.

THREAD POOLS

- The basic idea is to
  - create a no. of threads at process-startup and
  - place the threads into a pool (where they sit and wait for work).
- Procedure:
  1. When a server receives a request, it awakens a thread from the pool.
  2. If any thread is available, the request is passed to it for service.
  3. Once the service is completed, the thread returns to the pool.
- Advantages:
  - Servicing a request with an existing thread is usually faster than waiting to create a thread.
  - The pool limits the no. of threads that exist at any one point.
- No. of threads in the pool can be based on actors such as
  - no. of CPUs
  - amount of memory and
  - expected no. of concurrent client-requests.

THREAD SPECIFIC DATA

- Threads belonging to a process share the data of the process.
- this sharing of data provides one of the benefits of multithreaded programming.
- In some circumstances, each thread might need its own copy of certain data. We will call such data **thread-specific data**.
- For example, in a transaction-processing system, we might service each transaction in a separatethread.

- Furthermore, each transaction may be assigned a unique identifier. To associate each thread with its unique identifier, we could use thread-specific data.

SCHEDULER ACTIVATIONS

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application.
- Scheduler activations provide **upcalls** a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads
- One scheme for communication between the user-thread library and the kernel is known as **scheduler activation.**

PROCESS SCHEDULING

Basic Concepts

- In a single-processor system,
  - Only one process may run at a time.
  - Other processes must wait until the CPU is rescheduled.
- Objective of multiprogramming:
  - To have some process running at all times, in order to maximize CPU utilization.

CPU-I/0 Burst Cycle

- Process execution consists of a cycle of
  - CPU execution and
  - I/O wait
- Process execution begins with a CPU burst, followed by an I/O burst, then another CPU burst, etc…
- Finally, a CPU burst ends with a request to terminate execution.
- An I/O-bound program typically has many short CPU bursts.
- A CPU-bound program might have a few long CPU bursts.

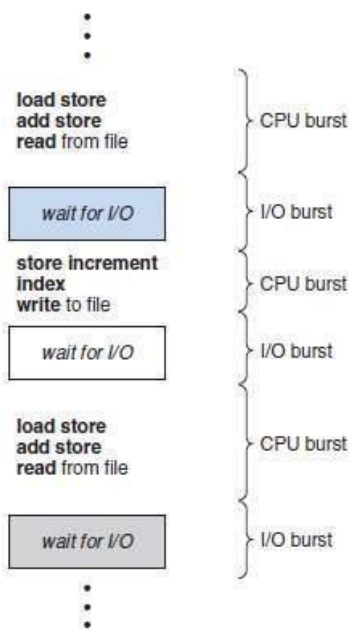


Fig Alternating sequence of CPU and I/O bursts

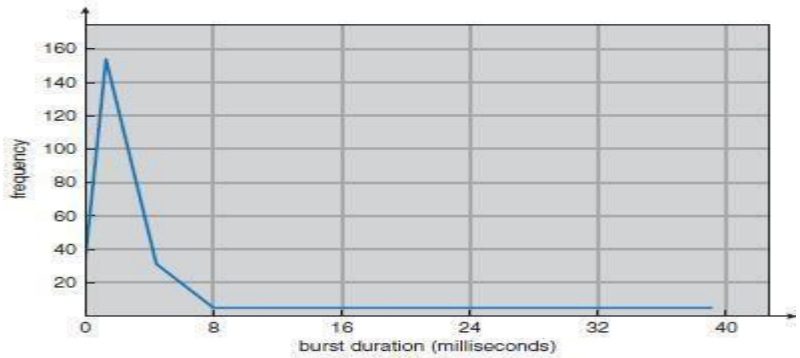


Fig: Histogram of CPU-burst durations

CPU Scheduler

- This scheduler
  - selects a waiting-process from the ready-queue and
  - allocates CPU to the waiting-process.
- The ready-queue could be a FIFO, priority queue, tree and list.
- The records in the queues are generally process control blocks (PCBs) of the processes.

CPU Scheduling

- Four situations under which CPU scheduling decisions take place:
  1. When a process switches from the running state to the waiting state. For ex; I/O request.
  2. When a process switches from the running state to the ready state. For ex: when an interrupt occurs.
  3. When a process switches from the waiting state to the ready state. For ex: completion of I/O.
  4. When a process terminates.
- Scheduling under 1 and 4 is non- preemptive. Scheduling under 2 and 3 is preemptive.

Non Preemptive Scheduling

- Once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either
  - by terminating or
  - by switching to the waiting state.

Preemptive Scheduling

- This is driven by the idea of prioritized computation.
- Processes that are runnable may be temporarily suspended
- Disadvantages:

Dispatcher

- It gives control of the CPU to the process selected by the short-term scheduler.
- The function involves:
    1. Switching context
    2. Switching to user mode&
    3. Jumping to the proper location in the user program to restart that program
- It should be as fast as possible, since it is invoked during every process switch.
- ***Dispatch latency*** means the time taken by the dispatcher to
    - stop one process and
    - start another running.

SCHEDULING CRITERIA:

In choosing which algorithm to use in a particular situation, depends upon the properties of the various algorithms. Many criteria have been suggested for comparing CPU-scheduling algorithms. The criteria include the following:

1. **CPU utilization:** We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
2. **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

3. **Turnaround time.** This is the important criterion which tells how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/0.
4. **Waiting time**: The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/0, it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.
5. **Response time:**In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is

SCHEDULING ALGORITHMS

- CPU scheduling deals with the problem of deciding which of the processes in the ready-queue is to be allocated the CPU.
- Following are some scheduling algorithms:
    1. FCFS scheduling (First Come First Served)
    2. Round Robin scheduling
    3. SJF scheduling (Shortest Job First)
    4. SRT scheduling
    5. Priority scheduling
    6. Multilevel Queue scheduling and
    7. Multilevel Feedback Queue scheduling

FCFS Scheduling

- The process that requests the CPU first is allocated the CPU first.
- The implementation is easily done using a FIFO queue.
- Procedure:
    1. When a process enters the ready-queue, its PCB is linked onto the tail of the queue.
    2. When the CPU is free, the CPU is allocated to the process at the queue's head.
    3. The running process is then removed from the queue.

- Advantage:
    1. Code is simple to write & understand.
- Disadvantages:
    1. **Convoy effect:** All other processes wait for one big process to get off the CPU.
    2. Non-preemptive (a process keeps the CPU until it releases it).
    3. Not good for time-sharing systems.
    4. The average waiting time is generally not minimal.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Example: Suppose that the processes arrive in the order P1, P2,P3.
- The Gantt Chart for the schedule is as follows:

- Waiting time for *P1* = 0; *P2* = 24; *P3* =27

  Average waiting time: (0 + 24 + 27)/3 = 17ms

- The Gantt chart for the schedule is as follows:

- Waiting time for P1 = 6;P2 = 0; P3 =3
- Average waiting time: (6 + 0 + 3)/3 = 3ms

SJF Scheduling

- The CPU is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length CPU burst, FCFS scheduling is used to break the tie.
- For long-term scheduling in a batch system, we can use the process time limit specified by the user, as the‘ length’
- SJF can't be implemented at the level of short-term scheduling, because there is no way to know the length of the next CPU burst
- Advantage:
    1. The SJF is optimal, i.e. it gives the minimum average waiting time for a given set of processes.
- Disadvantage:
    1. Determining the length of the next CPU burst.

- SJF algorithm may be either 1) non-preemptive or 2)preemptive.
    1. ***Non preemptive SJF***

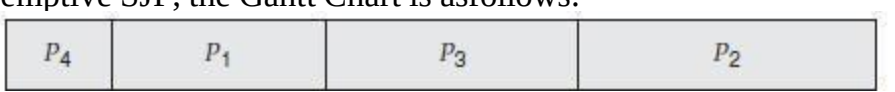
       The current process is allowed to finish its CPU burst.
    2. ***Preemptive SJF***

       If the new process has a shorter next CPU burst than what is left of the executing process, that process is preempted. It is also known as **SRTF** scheduling (Shortest-Remaining-Time-First).

- Example (for non-preemptive SJF): Consider the following set of processes, with the length of the CPU-burst time given in milliseconds.

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- For non-preemptive SJF, the Gantt Chart is asfollows:



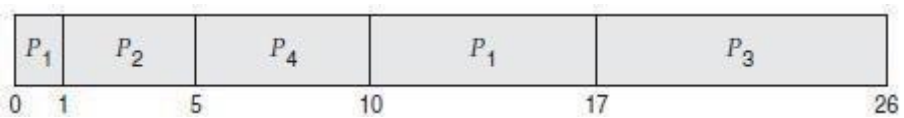
- Waiting time for P1 = 3; P2 = 16; P3 = 9; 4=0 Average waiting time: (3 + 16 + 9 + 0)/4= 7

preemptive SJF/SRTF: Consider the following set of processes, with the length

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

of the CPU- burst time given in milliseconds.

- For preemptive SJF, the Gantt Chart is as follows:



- The average waiting time is ((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 =6.5.

Priority Scheduling

- A priority is associated with each process.
- The CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- Priorities can be defined either internally or externally.
    1. ***Internally-defined*** priorities.
        - Use some measurable quantity to compute the priority of a process.
        - For example: time limits, memory requirements, no. f open files.
    2. ***Externally-defined*** priorities.
        - Set by criteria that are external to the OS For example:
        - importance of the process, political factors
- Priority scheduling can be either preemptive or non-preemptive.
    1. ***Preemptive***

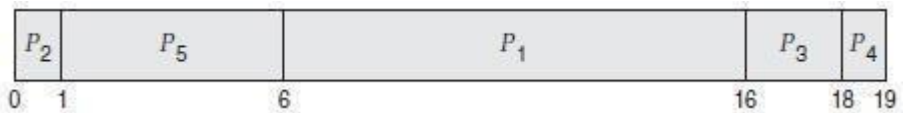
       The CPU is preempted if the priority of the newly arrived process is higher than the priority of the currently running process.
    2. ***Non Preemptive***

       The new process is put at the head of the ready-queue
- Advantage:
    - Higher priority processes can be executed first.
- Disadvantage:
    - Indefinite blocking, where low-priority processes are left waiting

- Example: Consider the following set of processes, assumed to have arrived at time 0, in the order PI, P2, ..., P5, with the length of the CPU-burst time given in milliseconds.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- The Gantt chart for the schedule is as follows:



- The average waiting time is 8.2milliseconds.

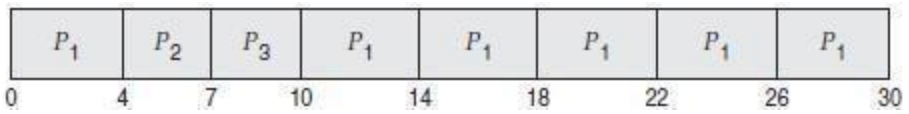
Round Robin Scheduling

- Designed especially for time sharing systems.
- It is similar to FCFS scheduling, but with preemption.
- A small unit of time is called a ***time quantum****(or time slice).*
- Time quantum is ranges from 10 to 100ms.
- The ready-queue is treated as a **circular queue**.
- The CPU scheduler
  - goes around the ready-queue and
  - allocates the CPU to each process for a time interval of up to 1 time quantum.
- To implement:

  The ready-queue is kept as a FIFO queue of processes
- CPU scheduler
  1. Picks the first process from the ready-queue.
  2. Sets a timer to interrupt after 1 time quantum and
  3. Dispatches the process.
- One of two things will then happen.
  1. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily.
  2. If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the OS. The process will be put at the tail of the ready-queue.
- Advantage:
  - Higher average turnaround than SJF.
- Disadvantage:
  - Better response time than SJF.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart for the schedule is as follows:



- The average waiting time is 17/3 = 5.66milliseconds.

- *The RR scheduling algorithm is preemptive.*

  No process is allocated the CPU for more than *1* time quantum in a row. If a process' CPU burst exceeds *1* time quantum, that process is preempted and is put back in the ready- queue.

- The performance of algorithm depends heavily on the size of the time quantum.
  1. If time quantum=very large, RR policy is the same as the FCFS policy.
  2. If time quantum=very small, RR approach appears to the users as though each of n processes has its own processor running at l/n the speed of the real processor.
- In software, we need to consider the effect of context switching on the performance of RR scheduling
  1. Larger the time quantum for a specific process time, less time is spend on context switching.
  2. The smaller the time quantum, more overhead is added for the purpose of context- switching.

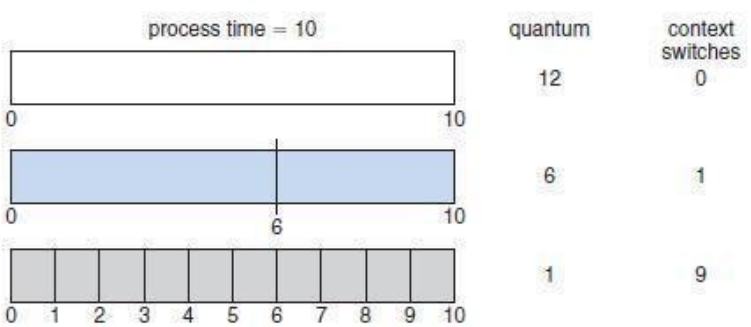


Fig: How a smaller time quantum increases context switches

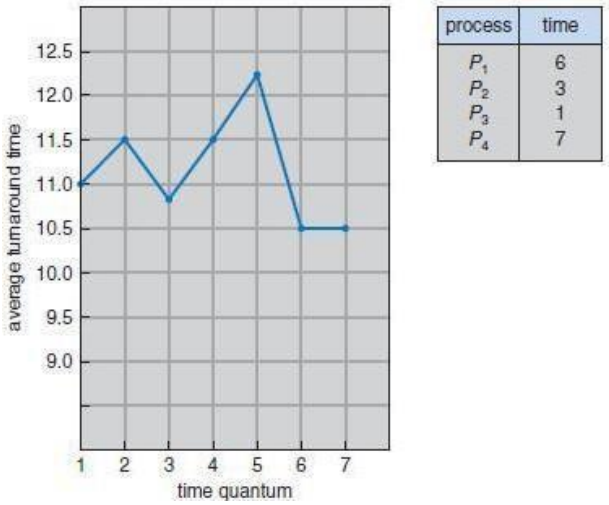
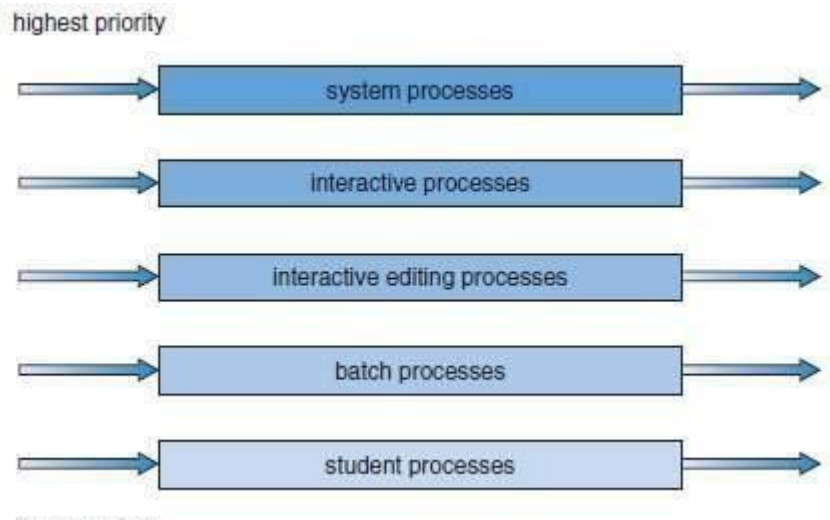


Fig: How turnaround time varies with the time quantum

Multilevel Queue Scheduling

- Useful for situations in which processes are easily classified into different groups.
- For example, a common division is made between
  - foreground (or interactive) processes and
  - background (or batch) processes.
- The ready-queue is partitioned into several separate queues (Figure2.19).
- The processes are permanently assigned to one queue based on some property like
  - memory size
  - process priority or
  - process type.
- Each queue has its own scheduling algorithm.

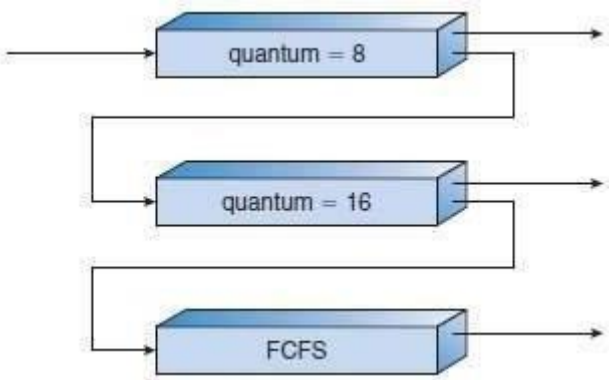
  For example, separate queues might be used for foreground and background processes.



- There must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.
- For example, the foreground queue may have absolute priority over the background queue.
- **Time slice**: each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR 20% to background in FCFS

Multilevel Feedback Queue Scheduling

- A process may move between queues
- The basic idea: Separate processes according to the features of their CPU bursts. For example
  1. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
  2. If a process waits too long in a lower-priority queue, it may be moved to a higher-priority queue This form of aging prevents starvation.



.

Figure 2.20 Multilevel feedback queues

In general, a multilevel feedback queue scheduler is defined by the following parameters:

1. The number of queues.
2. The scheduling algorithm for each queue.
3. The method used to determine when to upgrade a process to a higher priority queue.
4. The method used to determine when to demote a process to a lower priority queue.
5. The method used to determine which queue a process will enter when that process needs service

MULTIPLE PROCESSOR SCHEDULING

- If multiple CPUs are available, the scheduling problem becomes more complex.
- Two approaches:

Asymmetric Multiprocessing The basic idea is:

- A master server is a single processor responsible for all scheduling decisions, I/O processing and other system activities.
- The other processors execute only user code.
- Advantage: This is simple because only one processor accesses the system data structures, reducing the need for data sharing.

Symmetric Multiprocessing

The basic idea is:

- Each processor is self-scheduling.
- To do scheduling, the scheduler for each processor
- Examines the ready-queue and
- Selects a process to execute.

Restriction: We must ensure that two processors do not choose the same process and that processes are not lost from the queue.

Processor Affinity

- In SMP systems,
  1. Migration of processes from one processor to another are avoided and
  2. Instead processes are kept running on same processor. This is known as processor affinity.
- Two forms:
  1. ***Sof tAffinity***
     - When an OS try to keep a process on one processor because of policy, but cannot guarantee it will happen.
     - It is possible for a process to migrate between processors.
  2. ***Hard Affinity***
     - When an OS have the ability to allow a process to specify that it is not to migrate to other processors. Eg: Solaris OS

Load Balancing

- This attempts to keep the workload evenly distributed across all processors in an SMP system.
- Two approaches:
  1. ***PushMigration***

     A specific task periodically checks the load on each processor and if it finds an imbalance, it evenly distributes the load to idle processors.

Symmetric Multithreading

- The basic idea:
    1. Create multiple logical processors on the same physical processor.
    2. Present a view of several logical processors to the OS.
- Each logical processor has its own architecture state, which includes general-purpose and machine-state registers.
- Each logical processor is responsible for its own interrupt handling.
- SMT is a feature provided in hardware, not software.

THREAD SCHEDULING

- On OSs, it is kernel-level threads but not processes that are being scheduled by the OS.
- User-level threads are managed by a thread library, and the kernel is unaware of them.
- To run on a CPU, user-level threads must be mapped to an associated kernel-level thread.

Contention Scope

- Two approaches:

1. Process-Contention scope

- On systems implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on an available LWP.
- Competition for the CPU takes place among threads belonging to the same process.

2. System-Contentionscope

- The process of deciding which kernel thread to schedule on the CPU.
- Competition for the CPU takes place among all threads in the system.
- Systems using the one-to-one model schedule threads using only SCS.

Pthread Scheduling

- P thread API that allows specifying either PCS or SCS during thread creation.
- P threads identifies the following contention scope values:
    1. PTHREAD_SCOPEJPROCESS schedules threads using PCS scheduling.
    2. PTHREAD-SCOPE_SYSTEM schedules threads using SCS scheduling.
- P thread IPC provides following two functions for getting and setting the contention scope policy:
    1. P thread_ attr_ setscope(pthread_attr_t *attr, intscope)
    2. pthread_attr_getscope(pthread_attr_t *attr, int*scop

QUESTION BANK

1. What is a thread? What is TCB?
2. Write a note on multithreading models.
3. What is thread cancellation?
4. What is signal handling?
5. Explain The various Threading issues
6. What do you mean by
    a. Thread pool
    b. Thread specific data
    c. Scheduler activation
7. What is pre-emptive scheduling and non-pre-emptive scheduling?
8. Define the following:
    a. CPU utilization
    b. Throughput
    c. Turnaround time
    d. Waiting time
    e. Response time
9. Explain scheduling algorithms with examples.
10. Explain multilevel and multilevel feedback queue.
11. For the following set of process find the avg. waiting time and avg. turn around using Gantt chart for a) FCFS b) SJF (primitive and non-primitive) c) RR (quantum= 4)

Process	**Arrival Time**	**Burst Time**
P1	0	4
P2	1	2
P3	2	5
P4	3	4

FILE CONCEPT

FILE:

- *A* file is a named collection of related information that is recorded on secondary storage.
- The information in a file is defined by its creator. Many different types of information may be stored in a file source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.

A file has a certain defined which depends on its type.

- *A text* file is a sequence of characters organized into lines.
- *A source* file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An *object* file is a sequence of bytes organized into blocks understandable by the system's linker.
- An *executable* file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters, such as *example*.c
- When a file is named, it becomes independent of the process, the user, and even the system that created it.

A file's attributes vary from one operating system to another but typically consist of these:

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type:** This information is needed for systems that support different types of files.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words, or blocks) and possibly themaximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.

File Operations

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files.

1. **Creating a file:** Two steps are necessary to create a file,
    a) Space in the file system must be found for the file.
    b) An entry for the new file must be made in the directory.
2. **Writing a file:** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
3. **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. Again, the directory issearched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place,the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file-position pointer.
4. **Repositioning within a file:** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/0. This file operation is also known as files seek.
5. **Deleting a file:** To delete a file, search the directory for the named file. Having found the associated directory entry, then release all file space, so that it can be reused by other files, and erase the directory entry.
6. **Truncating a file:** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged but lets the file be reset to length zero and its file space released.

- Other common operations include appending new information to the end of an existing file and renaming an existing file.
- Most of the file operations mentioned involve searching the directory for the entry associated with the named file.
- To avoid this constant searching, many systems require that an open () system call be made before a file is first used actively.
- The operating system keeps a small table, called the **open file table** containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required.

- The implementation of the open() and close() operations is more complicated in an environment where several processes may open the file simultaneously
- The operating system uses two levels of internal tables:
    1. A per-process table
    2. A system-wide table

The per-process table:

- Tracks all files that a process has open. Stored in this table is information regarding the use of the file by the process.
- Each entry in the per-process table in turn points to a system-wide open-file table.

The system-wide table

- contains process-independent information, such as the location of the file on disk, access dates, and file size. Once a file has been opened by one process, the system-wide table includes an entry for the file.

Several pieces of information are associated with an open file.

1. **File pointer:** On systems that do not include a file offset as part of the read() and write() system calls, the system must track the last read write location as a current- file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.
2. **File-open count:** As files are closed, the operating system must reuse its open- file table entries, or it could run out of space in the table. Because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open counter tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.
3. **Disk location of the file:** Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory sothat the system does not have to read it from disk for each operation.
4. **Access rights:** Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/0 requests.

File Types

- The operating system should recognize and support file types. If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.
- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts-**a name and an extension**, usuallyseparated by a period character
- The system uses the extension to indicate the type of the file and the type of operations that can be done on that file.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- File types also can be used to indicate the internal structure of the file. For instance source and object files have structures that match the expectations of the programs that read them. Certain files must conform to a required structure that is understood by the operating system. For example: the operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is.
- The operating system support multiple file structures: the resulting size of the operating system also increases. If the operating system defines five different file structures, it needs to contain the code to support these file structures.
- It is necessary to define every file as one of the file types supported by the operating system. When new applications require information structured in ways not supported by

the operating system, severe problems may result.

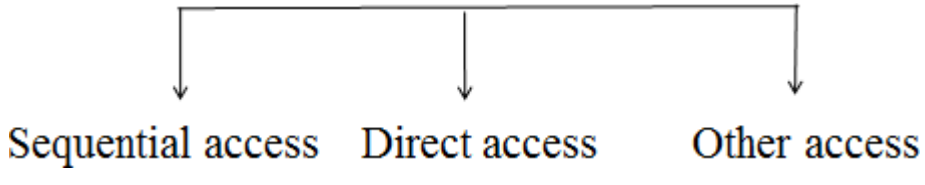
- Example: The Macintosh operating system supports a minimal number of file structures. It expects files to contain two parts: a resource fork and data fork.
  - **The resource fork** contains information of interest to the user.
  - **The data fork** contains program code or data

Internal File Structure

- Internally, locating an offset within a file can be complicated for the operating system. Disk systems typically have a well-defined block size determined by the size of a sector.
- All disk I/0 is performed in units of one block (physical record), and all blocks are the same size. It is unlikely that the physical record size will exactly match the length of the desired logical record.
- Logical records may even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.

ACCESS METHODS

- Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.
- Some of the common methods are:



1. Sequential methods

- The simplest access method is sequential methods. Information in the file is processed in order, one record after the other.
- Reads and writes make up the bulk of the operations on a file.
- A read operation (next-reads) reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location
- The write operation (write next) appends to the end of the file and advances to the end of the newly written material.
- A file can be reset to the beginning and on some systems, a program may be able to skip forward or backward n records for some integer n-perhaps only for n =1.

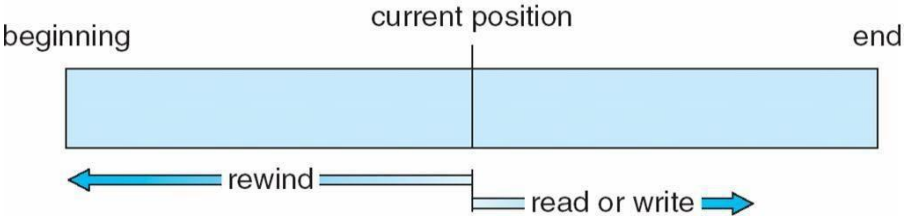


Figure: Sequential-access file.

2. Direct Access

- A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order.
- The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records.
- Example: if we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.
- Direct-access files are of great use for immediate access to large amounts of information such as Databases, where searching becomes easy and fast.
- For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have read n, where n is the block number, rather than read next, and ·write n rather than write next.
- An alternative approach is to retain read next and write next, as with sequential access, and to add an operation position file to n, where n is the block number. Then, to affect a read n, we would position to n and then read next.

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Figure: Simulation of sequential access on a direct-access file.

3. Other Access Methods:

- Other access methods can be built on top of a direct-access method. These methods generally involve the construction of an index for the file.
- The **Index**, is like an index in the back of a book contains pointers to the various blocks. To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

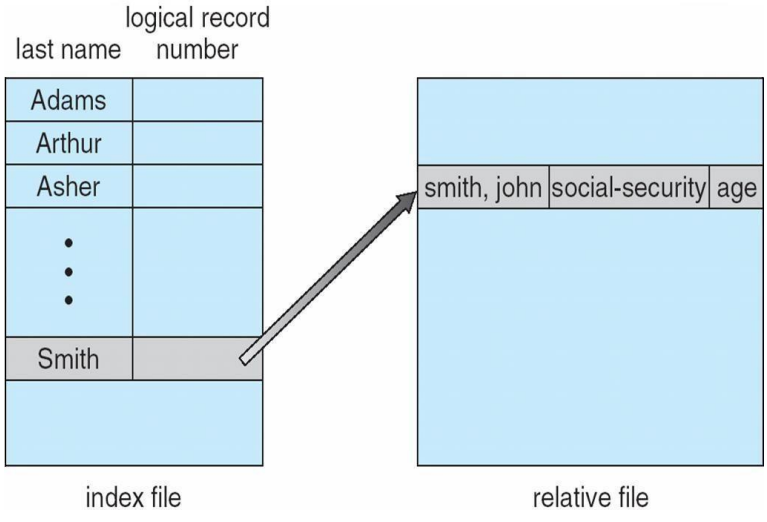


Figure: Example of index and relative files

DIRECTORY AND DISK STRUCTURE

- Files are stored on random-access storage devices, including hard disks, optical disks, and solid state (memory-based) disks.
- A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control
- Disk can be subdivided into partitions. Each disks or partitions can be RAID protected against failure.
- Partitions also known as minidisks or slices. Entity containing file system known as a volume. Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory**or **volume table of contents.**

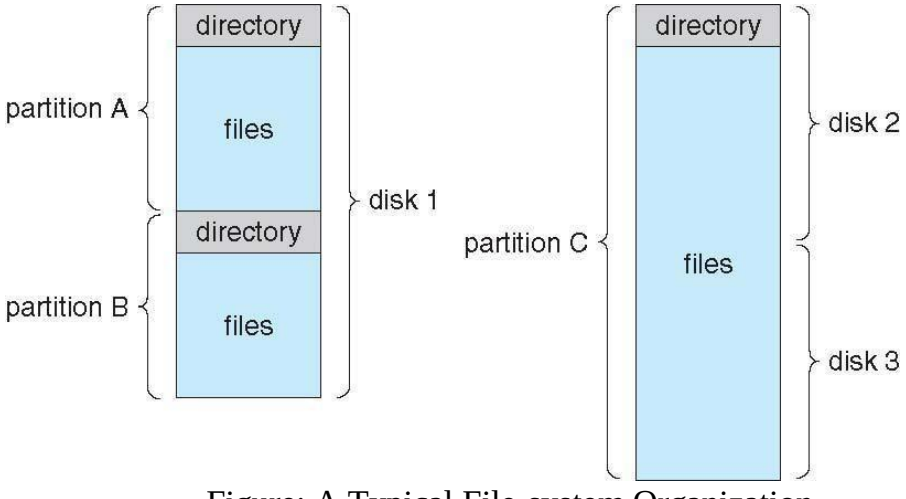


Figure: A Typical File-system Organization

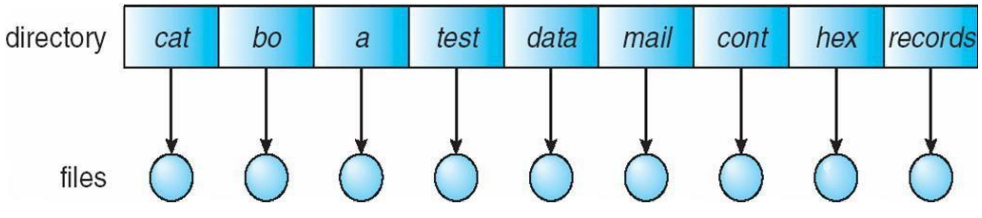
Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries. A directory contains information about the files including attributes location and ownership. To consider a particular directory structure, certain operations on the directory have to be considered:
  - **Search for a file:** Directory structure is searched for a particular file in directory. Files have symbolic names and similar names may indicate a relationship between files. Using this similarity it will be easy to find all whose name matches a particular pattern.
  - **Create a file:** New files needed to be created and added to the directory.
  - **Delete a file:** When a file is no longer needed, then it is able to remove it from thedirectory.
  - **List a directory:** It is able to list the files in a directory and the contents of the directory entry for each file in the list.
  - **Rename a file:** Because the name of a file represents its contents to its users, It is possible to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
  - **Traverse the file system:** User may wish to access every directory and every file within a directory structure. To provide reliability the contents and structure of the entire file system is saved at regular intervals.

- The most common schemes for defining the logical structure of a directory are described below
  1. Single-level Directory
  2. Two-Level Directory
  3. Tree-Structured Directories
  4. Acyclic-Graph Directories
  5. General Graph Directory

1. Single-level Directory

- The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand

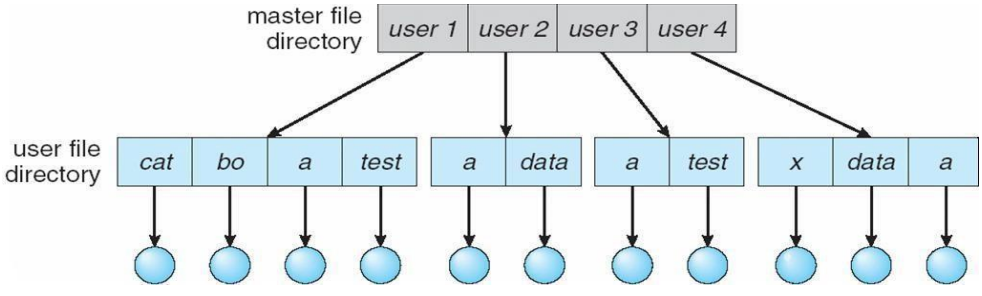


- A single-level directory has significant limitations, when the number of files increases or when the system has more than one user.

- As directory structure is single, uniqueness of file name has to be maintained, which is difficult when there are multiple users.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.
- It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system. Keeping track of so many files is a daunting task.

2. Two-Level Directory

- In the two-level directory structure, each user has its own **user file directory** (UFD). The UFDs have similar structures, but each lists only the files of a single user.
- When a user refers to a particular file, only his own UFD is searched. Different users may have files with the same name, as long as all the file names within each UFD are unique.
- To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD thus; it cannot accidentally delete another user'sfile that has the same name.
- When a user job starts or a user logs in, the system's Master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.



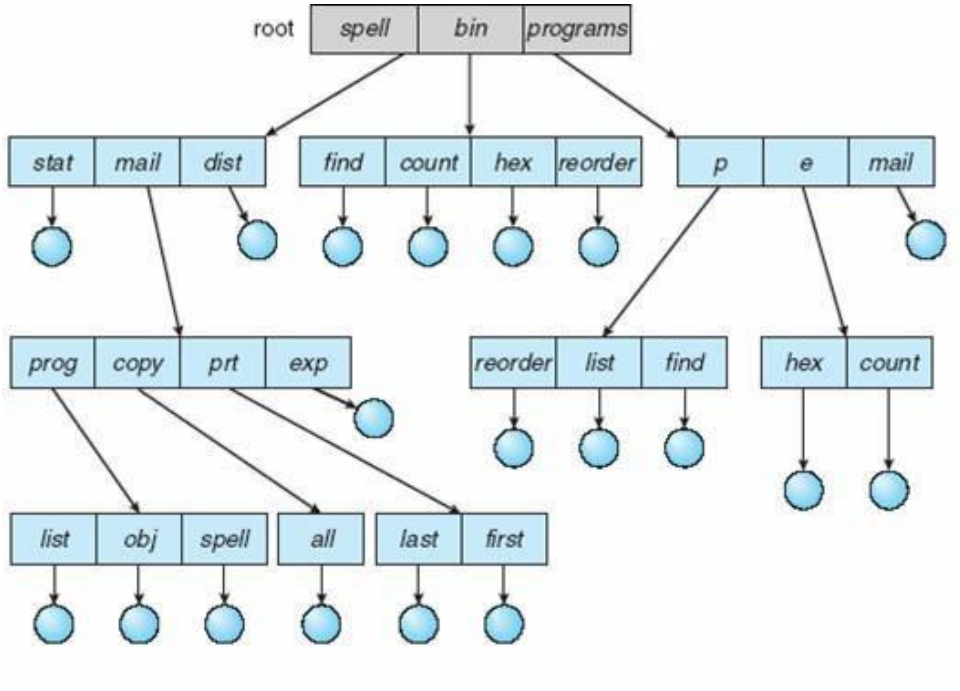
- **Advantage:**
  - No file name-collision among different users.
  - Efficient searching.
- **Disadvantage**
  - Users are isolated from one another and can't cooperate on the same task.

3. Tree Structured Directories

- A tree is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1). Special system calls are used to create and

delete directories.

- **Two types of path-names:**
  1. Absolute path-name: begins at the root.
  2. Relative path-name: defines a path from the current directory.



How to delete directory?

1. To delete an empty directory: Just delete the directory.
2. To delete a non-empty directory:
   - First, delete all files in the directory.
   - If any subdirectories exist, this procedure must be applied recursively to them.

Advantage:

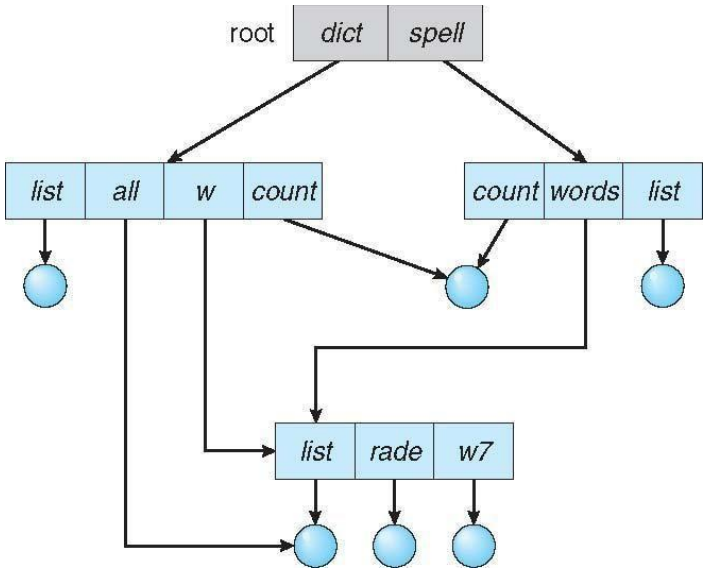
- Users can be allowed to access the files of other users.

Disadvantages:

- A path to a file can be longer than a path in a two-level directory.
- Prohibits the sharing of files (or directories).

4. Acyclic Graph Directories

- The common subdirectory should be shared. A shared directory or file will exist in the file system in two or more places at once. A tree structure prohibits the sharing of files or directories.
- An acyclic graph is a graph with no cycles. It allows directories to share subdirectories and files.



- The same file or subdirectory may be in two different directories. The acyclic graph is a natural generalization of the tree-structured directory scheme.

Two methods to implement shared-files (or subdirectories):

1. Create a new directory-entry called a link. A link is a pointer to another file (or subdirectory).
2. Duplicate all information about shared-files in both sharing directories.

Two problems:

1. A file may have multiple absolute path-names.
2. Deletion may leave dangling-pointers to the non-existent file.

Solution to deletion problem:

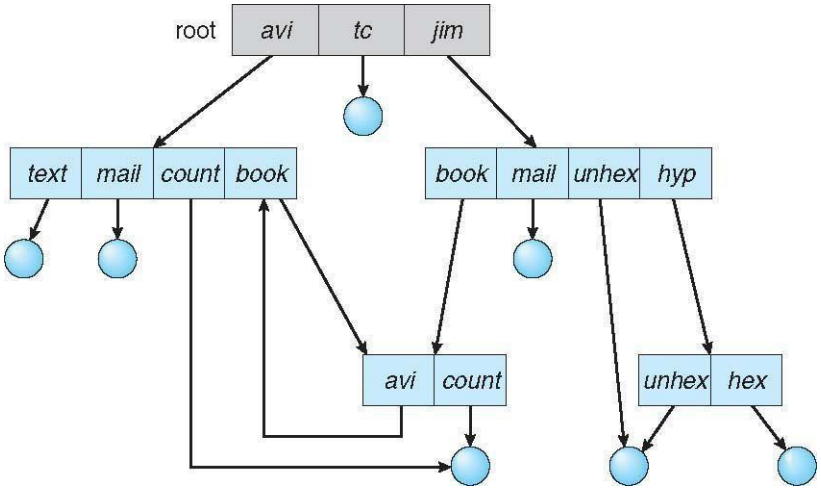
1. Use back-pointers: Preserve the file until all references to it are deleted.
2. With symbolic links, remove only the link, not the file. If the file itself is deleted, the link can be removed.

5. General Graph Directory

- **Problem:** If there are cycles, we want to avoid searching components twice.
- **Solution:** Limit the no. of directories accessed in a search.
- **Problem:** With cycles, the reference-count may be non-zero even when it is no longer possible to refer to a directory (or file). (A value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted).
- **Solution:** Garbage-collection scheme can be used to determine when the last reference has been deleted.

Garbage collection involves

- First pass traverses the entire file-system and marks everything that can be accessed.
- A second pass collects everything that is not marked onto a list of free-space



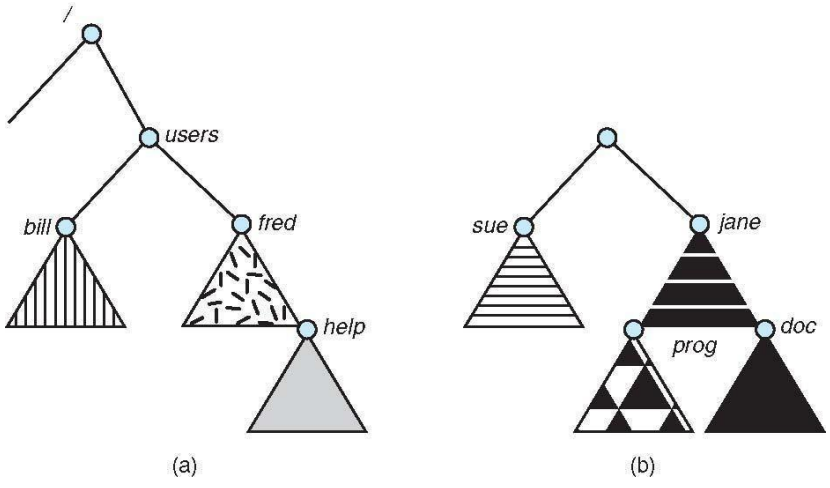
FILE SYSTEM MOUNTING

- A file must be *opened* before it is used, a file system must be *mounted* before it can be available to processes on the system
- **Mount Point:** The location within the file structure where the file system is to be attached.

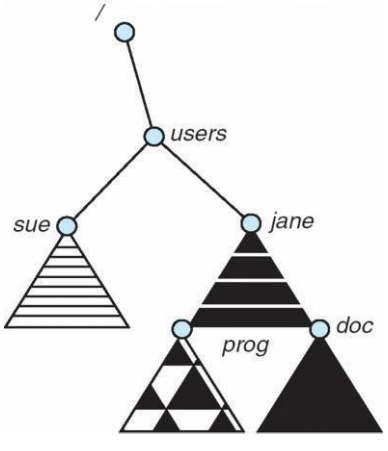
The mounting procedure:

- The operating system is given the name of the device and the mount point.
- The operating system verifies that the device contains a valid file system. It does so by asking the device driver to read the device directory and verifying that the directory has the expected format
- The operating system notes in its directory structure that a file system is mounted at the specified mount point.

To illustrate file mounting, consider the file system shown in figure. The triangles represent sub-trees of directories that are of interest



- Figure (a) shows an existing file system,
- while Figure 1(b) shows an un-mounted volume residing on */device/dsk.* At this point, only the files on the existing file system can be accessed.



- Above figure shows the effects of mounting the volume residing on */device/dsk* over */users.*
- If the volume is un-mounted, the file system is restored to the situation depicted in first Figure.

FILE SHARING

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a protection scheme.
- On distributed systems, files may be shared across a network.
- Network File-system (NFS) is a common distributed file-sharing method.

Multiple Users

File-sharing can be done in 2 ways:

1. The system can allow a user to access the files of other users by default or
2. The system may require that a user specifically grant access.

To implement file-sharing, the system must maintain more file- & directory-attributes than on a single-user system.
Most systems use concepts of file owner and group.

1. Owner

- The user who may change attributes & grant access and has the most control over the file (or directory).
- Most systems implement owner attributes by managing a list of user-names and user IDs

2. Group

- The group attribute defines a subset of users who can share access to the file.
- Group functionality can be implemented as a system-wide list of group-names and group IDs.
- Exactly which operations can be executed by group-members and other users is definable by the file's owner.
- The owner and group IDs of files are stored with the other file-attributes and can be used to allow/deny requested operations.

Remote File Systems

It allows a computer to mount one or more file-systems from one or more remote- machines.
There are three methods:

1. Manually transferring files between machines via programs like ftp.
2. Automatically DFS (Distributed file-system): remote directories are visible from a local machine.
3. Semi-automatically via www (World Wide Web): A browser is needed to gain access to the remote files, and separate operations (a wrapper for ftp) are used to transfer files.

ftp is used for both anonymous and authenticated access. **Anonymous access** allows a user to transfer files without having an account on the remote system.

Client Server Model

- Allows clients to mount remote file-systems from servers.
- The machine containing the files is called the **server**. The machine seeking access to the files is called the **client**.
- A server can serve multiple clients, and a client can use multiple servers.
- The server specifies which resources (files) are available to which clients.
- A client can be specified by a network-name such as an IP address.

Disadvantage:

- Client identification is more difficult.
- In UNIX and its NFS (network file-system), authentication takes place via the client networking information by default.
- Once the remote file-system is mounted, file-operation requests are sent to the server via the DFS protocol.

Distributed Information Systems

- Provides unified access to the information needed for remote computing.
- The DNS (domain name system) provides hostname-to-network address translations.
- Other distributed info. systems provide username/password space for a distributed facility

Failure Modes

- Local file-systems can fail for a variety of reasons such as failure of disk (containing the file-system), corruption of directory-structure & cable failure.
- Remote file-systems have more failure modes because of the complexity of network-systems.
- The network can be interrupted between 2 hosts. Such interruptions can result from hardware failure, poor hardware configuration or networking implementation issues.
- DFS protocols allow delaying of file-system operations to remote-hosts, with the hope that the remote-host will become available again.
- To implement failure-recovery, some kind of state information may be maintained on both the client and the server.

Consistency Semantics

- These represent an important criterion of evaluating file-systems that supports file- sharing. These specify how multiple users of a system are to access a shared-file simultaneously.
- In particular, they specify when modifications of data by one user will be observed by other users.
- These semantics are typically implemented as code with the file-system.

- These are directly related to the process-synchronization algorithms.
- A successful implementation of complex sharing semantics can be found in the Andrew file-system (AFS).

UNIX Semantics

- UNIX file-system (UFS) uses the following consistency semantics:
    1. Writes to an open-file by a user are visible immediately to other users who have this file opened.
    2. One mode of sharing allows users to share the pointer of current location into a file. Thus, the advancing of the pointer by one user affects all sharing users.
- A file is associated with a single physical image that is accessed as an exclusive resource.
- Contention for the single image causes delays in user processes.

Session Semantics

The AFS uses the following consistency semantics:

1. Writes to an open file by a user are not visible immediately to other users that have the same file open.
2. Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect these changes.

- A file may be associated temporarily with several (possibly different) images at the same time.
- consequently, multiple users are allowed to perform both read and write accesses concurrently on their images of the file, without delay.
- Almost no constraints are enforced on scheduling accesses.

Immutable Shared Files Semantics

- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has 2 key properties:
    1. File-name may not be reused
    2. File-contents may not be altered.
- Thus, the name of an immutable file signifies that the contents of the file are fixed.
- The implementation of these semantics in a distributed system is simple, because the sharingis disciplined

PROTECTION

- When information is stored in a computer system, we want to keep it safe from physical damage (reliability) and improper access (protection).
- Reliability is generally provided by duplicate copies of files.
- For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer.
- File owner/creator should be able to control what can be done and by whom.

Types of Access

- Systems that do not permit access to the files of other users do not need protection. This is too extreme, so controlled-access is needed.
- Following operations may be controlled:
    1. ***Read:*** Read from the file.
    2. ***Write:*** Write or rewrite the file.
    3. ***Execute:*** Load the file into memory and execute it.
    4. ***Append:*** Write new information at the end of the file.
    5. ***Delete:*** Delete the file and tree its space for possible reuse.
    6. ***List:*** List the name and attributes of the file.

Access Control

- Common approach to protection problem is to make access dependent on identity of user.
- Files can be associated with an ACL (access-control list) which specifies username and types of access for each user.

Problems:

1. Constructing a list can be tedious.
2. Directory-entry now needs to be of variable-size, resulting in more complicated space management.

Solution:

- These problems can be resolved by combining ACLs with an ‘owner, group, universe’ access control scheme
- To reduce the length of the ACL, many systems recognize 3 classifications of users:
    1. ***Owner:*** The user who created the file is the owner.
    2. ***Group:*** A set of users who are sharing the file and need similar access is a group.
    3. ***Universe:*** All other users in the system constitute the universe.

Other Protection Approaches

- A password can be associated with each file.
- **Disadvantages:**
    1. The no. of passwords you need to remember may become large.
    2. If only one password is used for all the files, then all files are accessible if it is discovered.
    3. Commonly, only one password is associated with all of the user's files, so protection is all-or nothing.

- In a multilevel directory-structure, we need to provide a mechanism for directory protection.
- The directory operations that must be protected are different from the File-operations:
    1. Control creation & deletion of files in a directory.
    2. Control whether a user can determine the existence of a file in a directory.

IMPLEMENTATION OF FILE SYSTEM

FILE SYSTEM STRUCTURE

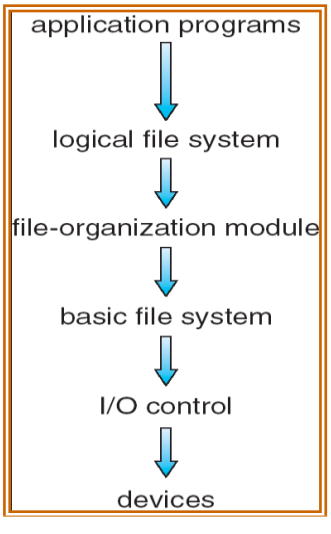
- Disks provide the bulk of secondary-storage on which a file-system is maintained.

The disk is a suitable medium for storing multiple files.

- This is because of two characteristics
  1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same place.
  2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires only moving the read-write heads and waiting for the disk to rotate.
- To improve I/O efficiency, I/O transfers between memory and disk are performed in units of blocks. Each block has one or more sectors. Depending on the disk drive, sector-size varies from 32 bytes to 4096 bytes. The usual size is 512 bytes.
- File-systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily
- Design problems of file-systems:
  1. Defining how the file-system should look to the user.
  2. Creating algorithms & data-structures to map the logical file-system onto the physical secondary-storage devices.

Layered File Systems:

- The file-system itself is generally composed of many different levels. Every level in design uses features of lower levels to create new features for use by higher levels.



- File system provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.

A file system poses two quite different design problems.

1. The first problem is defining how the file system should look to the user. This task involves defining a file and its attributes, the operations allowed on a file, and the directory structure for organizing files.
2. The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

The file system itself is generally composed of many different levels. The structure shown in Figure is an example of a layered design. Each level in the design uses the features of lower levels to create new features for use by higher levels.

- The lowest level, **the *I/O control***, consists of **device drivers** and interrupts handlers to transfer information between the main memory and the disk system.

A device driver can be thought of as a translator. Its input consists of high-level commands such as "retrieve block 123."
Its output consists of low level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/0 device to the rest of the system.

- The device driver usually writes specific bit patterns to special locations in the I/0 controller's memory to tell the controller which device location to act on and what actions to take.
- The **basic file system** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive 1, cylinder 73, track 2, sector10).

This layer also manages the memory buffers and caches that hold various file-system, directory, and data blocks.
A block in the buffer is allocated before the transfer of a disk block can occur. When the buffer is full, the buffer manager must find more buffer memory or free up buffer space to allow a requested I/O to complete.

File organization

- Module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file- organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.
- Each file's logical blocks are numbered from 0 (or 1) through *N*. Since the physical blocks containing the data usually do not match the logical numbers, a translation is needed to locate each block.
- The file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

Logical file system

- Manages metadata information. Metadata includes all of the file- system structure except the actual *data* (or contents of the files). The logical file system manages the directory structure to provide the file organization module with the information the latter needs, given a symbolic file name. It maintains file structure via **file-control blocks (FCB)**.
- FCB contains information about the file, including ownership, permissions, and location of the file contents.

File System Implementation

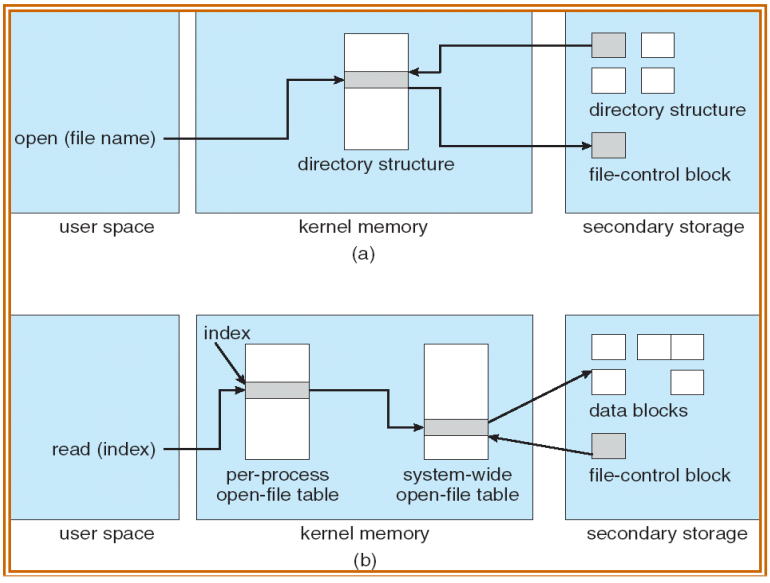
On disk, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

- **Boot Control Block**: On disk, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.
- **Volume Control Block** : (per volume) contains volume (or partition) details, such as the number of blocks in the partition, the size of the blocks, a free-block count and free-block pointers and a free-FCB count and FCB pointers.
- **A directory structure** (per file system) is used to organize the files.
- **A per-file FCB** contains many details about the file. It has a unique identifier number to allow association with a directory entry.

The in-memory information is used for both file-system management and performance improvement via caching. The data are loaded at mount time, updated during file-system operations, and discarded at dismount. Several types of structures may be included.

- An in-memory mount table contains information about each mounted volume.
- An in-memory directory-structure cache holds the directory information of recently accessed directories.

- The system wide open file table contains a copy of the FCB of each open file, as well as other information.
- The per -process open file table contains a pointer to the appropriate entry in the system- wide open-file table, as well as other information.



- Buffers hold file-system blocks when they are being read from disk or written to disk.

Steps for creating a file:

1) An application program calls the logical file system, which knows the format of the directory structures
2) The logical file system allocates a new file control block(FCB)
   - If all FCBs are created at file-system creation time, an FCB is allocated from the free list
3) The logical file system then
   - Reads the appropriate directory into memory
   - Updates the directory with the new file name and FCB
   - Writes the directory back to the disk

UNIX treats a directory exactly the same as a file by means of a type field in the i node Windows NT implements separate system calls for files and directories and treats directories asentities separate from files.

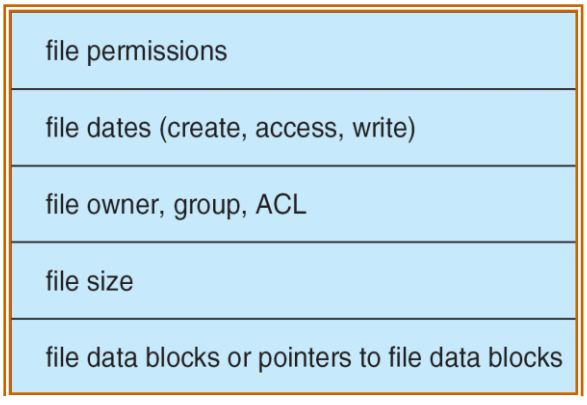
Steps for opening a file:

1) The function first searches the system-wide open-file table to see if the file is already in use by another process
   - If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table
   - This algorithm can have substantial overhead; consequently, parts of the directory structure are usually cached in memory to speed operations

2) Once the file is found, the FCB is copied into a system-wide open-file table in memory
    - This table also tracks the number of processes that have the file open
3) Next, an entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table
4) The function then returns a pointer/index to the appropriate entry in the per-process file-system table
    - All subsequent file operations are then performed via this pointer
    - UNIX refers to this pointer as the file descriptor
    - Windows refers to it as the file handle Steps for closing a file:

1) The per-process table entry is removed
2) The system-wide entry's open count is decremented
3) When all processes that have opened the file eventually close it

Any updated metadata is copied back to the disk-based directory structure. The system-wide open-file table entry is removed



Partitions and Mounting

- Each partition can be either "raw," containing no file system, or "cooked," containing a file system.
- Raw disk is used where no file system is appropriate.
- UNIX swap space can use a raw partition, for example, as it uses its own format on disk and does not use a file system.
- Boot information can be stored in a separate partition. Again, it has its own format, because at boot time the system does not have the file-system code loaded and therefore cannot interpret the file-system format.
- Boot information is a sequential series of blocks, loaded as an image into memory.
- Execution of the image starts at a predefined location, such as the first byte. This boot loader in turn knows about the file-system structure to be able to find and load the kernel and start it executing.
- The root partition which contains the operating-system kernel and sometimes other system files, is mounted at boot time.

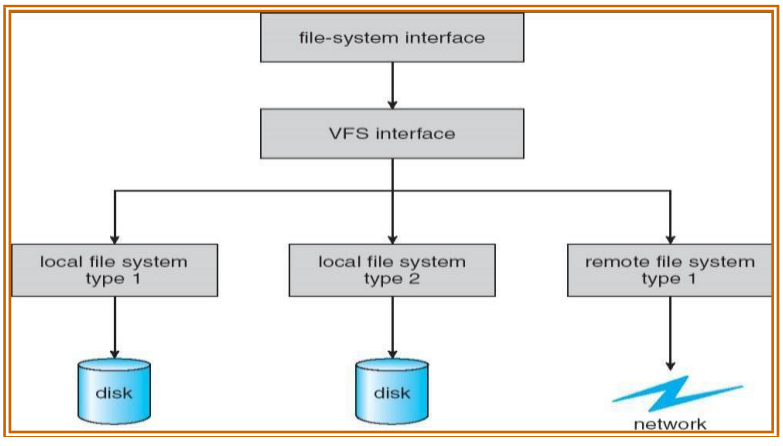
- Other volumes can be automatically mounted at boot or manually mounted later, depending on the operating system.
- After successful mount operation, the operating system verifies that the device contains a valid file system.
- It is done by asking the device driver to read the device directory and verifying that the directory has the expected format.
- If the format is invalid, the partition must have its consistency checked and possibly corrected, either with or without user intervention.
- Finally, the operating system notes in its in-memory mount table that a file system is mounted, along with the type of the file system.
- The root partition is mounted at boot time
    - It contains the operating-system kernel and possibly other system files
- Other volumes can be automatically mounted at boot time or manually mounted later
- As part of a successful mount operation, the operating system verifies that the storage device contains a valid file system
    - It asks the device driver to read the device directory and verify that the directory has the expected format
    - If the format is invalid, the partition must have its consistency checked and possibly corrected
    - Finally, the operating system notes in its in-memory mount table structure that a file system is mounted along with the type of the file system

Virtual file Systems

The file-system implementation consists of three major layers, as depicted schematically inFigure. The first layer is the file-system interface, based on the open(), read(), write(), and close() calls and on file descriptors.

The second layer is called the virtual file system (vfs) layer. The VFS layer serves two important functions:

- It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.
- It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a vnode that contains a numerical designator for a network-wide unique file. This network-wide uniqueness is required for support of network file systems.
- The kernel maintains one vnode structure for each active node.
- Thus, the VFS distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.



Directory Implementation

Selection of directory allocation and directory management algorithms significantly affects the efficiency, performance, and reliability of the file system

One Approach: Direct indexing of a linear list

- Consists of a list of file names with pointers to the data blocks
- Simple to program
- Time-consuming to search because it is a linear search.
- Sorting the list allows for a binary search; however, this may complicate creating and deleting files
- To create a new file, we must first search the directory to be sure that no existing file has the same name.
- Add a new entry at the end of the directory.
- To delete a file, we search the directory for the named file and then release the space allocated to it.
- To reuse the directory entry, we can do one of several things. Mark the entry as unused.
- An alternative is to copy the last entry in the directory into the freed location and to decrease the length of the directory.
- Directory information is used frequently, and users will notice if access to it is slow.

Another Approach: List indexing via a hash function

- Takes a value computed from the file name and returns a pointer to the file name in the linear list
- Greatly reduces the directory search time
- **Can result in collisions** – situations where two file names hash to the same location
- A hash table are its generally fixed size and the dependence of the hash function on that size. (i.e., fixed number of entries).
- Each hash entry can be a linked list instead of an individual value, and we can resolve collisions by adding the new entry to the linked list.

ALLOCATION METHODS

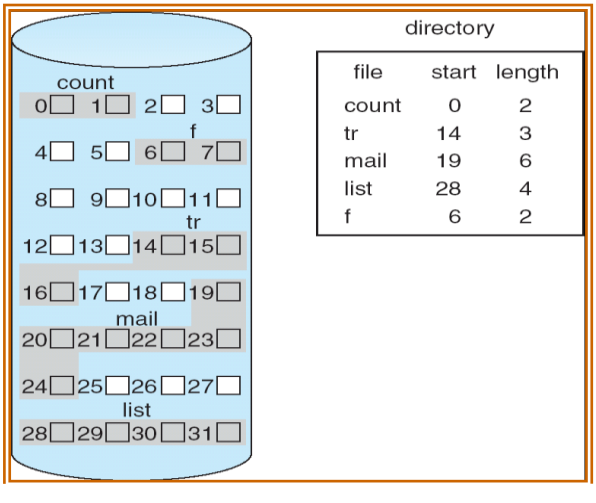
Allocation methods address the problem of allocating space to files so that disk space is utilized effectively and files can be accessed quickly.

Three methods exist for allocating disk space

- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous allocation:

- Requires that each file occupy a set of contiguous blocks on the disk
- Accessing a file is easy – only need the starting location (block #) and length (number of blocks)
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is *n* blocks long and starts at location *b*, then it occupies blocks *b*, *b* + 1, *b* + 2, ... ,*b* + *n* - 1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Accessing a file that has been allocated contiguously is easy. For sequential access, the file system remembers the disk address of the last block referenced and when necessary, reads the next block. For direct access to block *i* of a file that starts at block *b*, we can immediately access block *b* + *i*. Thus, both sequential and direct access can be supported by contiguous allocation.



Disadvantages:

1. Finding space for a new file is difficult. The system chosen to manage free space determines how this task is accomplished. Any management system can be used, but some are slower than others.
2. Satisfying a request of size *n* from a list of free holes is a problem. First fit and best fit are the most common strategies used to select a free hole from the set of available

holes.

3. The above algorithms suffer from the problem of external fragmentation.
    - As files are allocated and deleted, the free disk space is broken into pieces.
    - External fragmentation exists whenever free space is broken into chunks.
    - It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.
    - Depending on the total amount of disk storage and the average file size, external fragmentation may be a minor or a major problem.

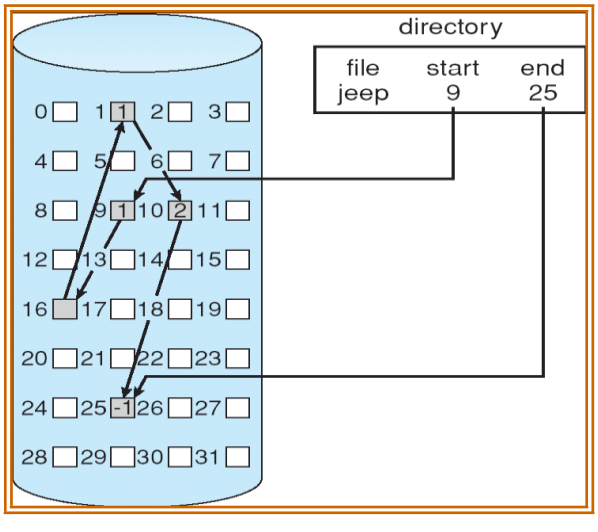
Linked Allocation:

- Solves the problems of contiguous allocation
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- The directory contains a pointer to the first and last blocks of a file
- Creating a new file requires only creation of a new entry in the directory
- Writing to a file causes the free-space management system to find a free block

- This new block is written to and is linked to the end of the file
- Reading from a file requires only reading blocks by following the pointers from block to block.

Advantages

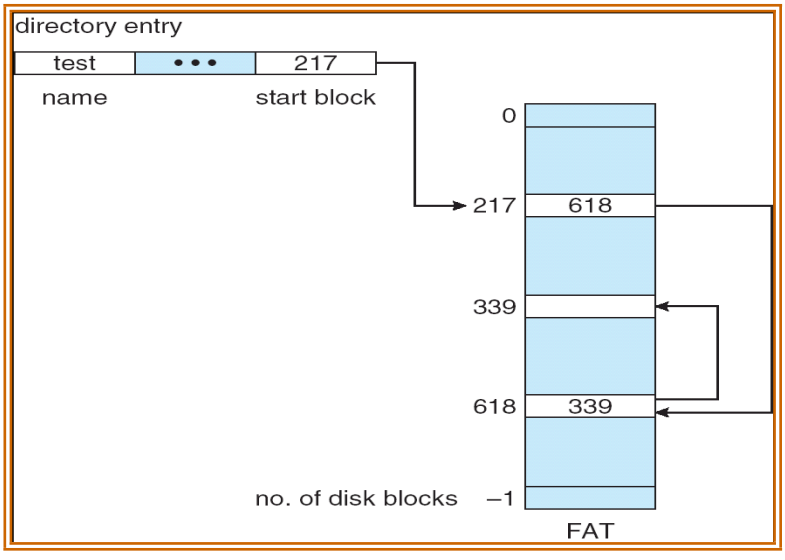
- There is no external fragmentation
- Any free blocks on the free list can be used to satisfy a request for disk space
- The size of a file need not be declared when the file is created
- A file can continue to grow as long as free blocks are available
- It is never necessary to compact disk space for the sake of linked allocation (however, file access efficiency may require it)



- Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.
- For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointers are not made available to the user. A disk address (the pointer) requires 4 bytes in the disk.
- To **create** a new file, we simply create a new entry ile the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointeris initialized to *nil* (the end-of-list pointer value) to signify an empty file. The size field is also set to 0.
- A **write** to the file causes the free-space management system to filed a free block, and this new block is written to and is linked to the end of the file.
- To **read** a file, we simply read blocks by following the pointers from block to block. There is no external fragmentation with linked allocation, and any free block on the free- space list can be used to satisfy a request. The size of a file need not be declared whenthat file is created.
- A file can continue to grow as long as free blocks are available. Consequently, it is never necessary to compact disk space.
- **Disadvantages:**
    1. The major problem is that it can be used effectively only for sequential-access files. To filed the i th block of a file, we must start at the beginning of that file and follow the pointers until we get to the ith block.
    2. Space required for the pointers. Solution is clusters. Collect blocks into multiples and allocate clusters rather than blocks
    3. Reliability - the files are linked together by pointers scattered all over the disk and if a pointer were lost or damaged then all the links are lost.

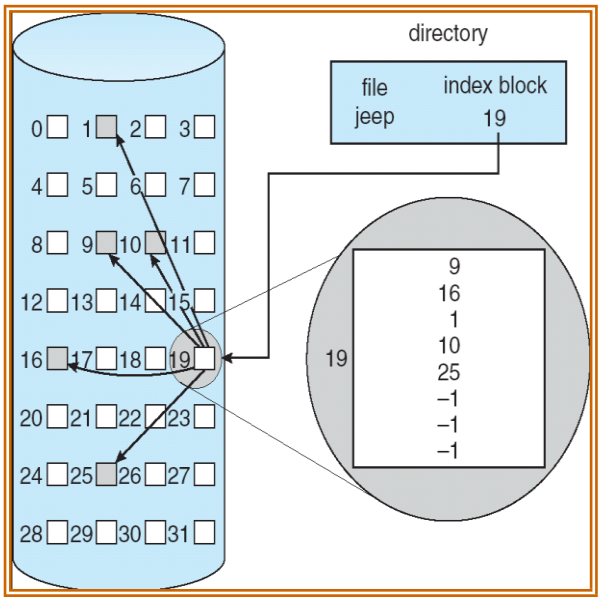
File Allocation Table:

- A section of disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed by block number.
- The FAT is used in much the same way as a linked list. The directory entry contains the block number of the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- The chain continues until it reaches the last block, which has a special end-of-file value as the table entry.
- An unused block is indicated by a table value of 0.
- Consider a FAT with a file consisting of disk blocks 217, 618, and 339.



Indexed allocation:

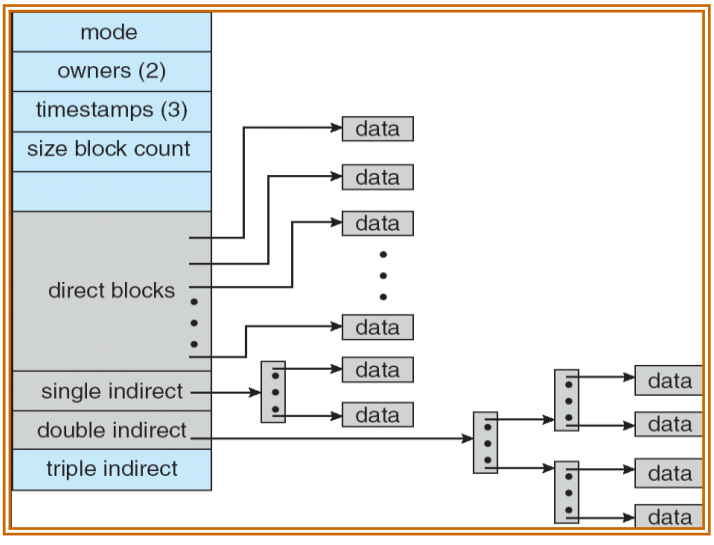
- Brings all the pointers together into one location called index block.
- Each file has its own index block, which is an array of disk-block addresses.
- The *ith* entry in the index block points to the *ith* block of the file. The directory contains the address of the index block. To find and read the *ith* block, we use the pointer in the *ith* index-block entry.
- When the file is created, all pointers in the index block are set to *nil.* When the ith block is first written, a block is obtained from the free-space manager and its address is put in the ith index- block entry.
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.
- Disadvantages :
  - Suffers from some of the same performance problems as linked allocation
  - Index blocks can be cached in memory; however, data blocks may be spread all over the disk volume.
  - Indexed allocation does suffer from wasted space.
  - The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.



If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue. Mechanisms for this purpose include the following:

a) **Linked scheme.** An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we can link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is *nil* (for a small file) or is a pointer to another index block (for a large file).

b) **Multilevel index.** A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size



c) **Combined scheme.** For eg. 15 pointers of the index block is maintained in the file's i node. The first 12 of these pointers point to **direct blocks**; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files (of no more than 12 blocks) do not need a separate index block. If the block size is 4 KB, then up to 48 KB of data can be accessed directly. The next three pointers point to indirect blocks. The first points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The second points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The last pointer contains the address of a **triple indirect block**.

Performance

- Contiguous allocation requires only one access to get a disk block. Since we can easily keep the initial address of the file in memory, we can calculate immediately the disk address of the *ith* block and read it directly.
- For linked allocation, we can also keep the address of the next block in memory and read it directly. This method is fine for sequential access. Linked allocation should not be used for an application requiring direct access.
- Indexed allocation is more complex. If the index block is already in memory, then the access can be made directly. However, keeping the index block in memory requires considerable space. If this memory space is not available, then we may have to read first the index block and then the desired data block.

Free Space Management

The space created after deleting the files can be reused. Another important aspect of disk management is keeping track of free space in memory. The list which keeps track of free space in memory is called the free-space list. To create a file, search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free- space list. When a file is deleted, its disk space is added to the free-space list. The free-space list, isimplemented in different ways as explained below.

a) Bit Vector

- Fast algorithms exist for quickly finding contiguous blocks of a given size
- One simple approach is to use a ***bit vector***, in which each bit represents a disk block, set to 1 if free or 0 if allocated.

For example, consider a disk where blocks 2,3,4,5,8,9, 10,11, 12, 13, 17and 18 are free, and the rest of the blocks are allocated. The free-space bit map would be
0011110011111100011

- Easy to implement and also very efficient in finding the first free block or ‘n’ consecutive free blocks on the disk.

- The down side is that a 40GB disk requires over 5MB just to store the bitmap.

b) Linked List

a. A linked list can also be used to keep track of all free blocks.
b. Traversing the list and/or finding a contiguous block of a given size are not easy, but fortunately are not frequently needed operations. Generally the system just adds and removes single blocks from the beginning of the list.
c. The FAT table keeps track of the free list as just one more linked list on the table.

c) Grouping

a. A variation on linked list free lists. It stores the addresses of n free blocks in the first free block. The first n-1 blocks are actually free. The last block contains the addresses of another n free blocks, and so on.
b. The address of a large number of free blocks can be found quickly.

d) Counting

a. When there are multiple contiguous blocks of free space then the system can keep track of the starting address of the group and the number of contiguous free blocks.
b. Rather than keeping al list of n free disk addresses, we can keep the address of first free block and the number of free contiguous blocks that follow the first block.
c. Thus the overall space is shortened. It is similar to the extent method of allocating blocks.

e) Space Maps

a. Sun's ZFS file system was designed for huge numbers and sizes of files, directories, and even file systems.
b. The resulting data structures could be inefficient if not implemented carefully. For example, freeing up a 1 GB file on a 1 TB file system could involve updating thousands of blocks of free list bit maps if the file was spread across the disk.
c. ZFS uses a combination of techniques, starting with dividing the disk up into (hundreds of) ***Meta slabs*** of a manageable size, each having their own space map.
d. Free blocks are managed using the counting technique, but rather than write the information to a table, it is recorded in a log-structured transaction record. Adjacent free blocks are also coalesced into a larger single free block.
e. An in-memory space map is constructed using a balanced tree data structure, constructed from the log data.
f. The combination of the in-memory tree and the on-disk log provide for very fast and efficient management of these very large files and free blocks.

QUESTION BANK

1. What is a file? Distinguish between contiguous and linked allocation methods with the neat diagram.
2. Explain file allocation methods by taking an example with the neat diagram. Write the advantages and disadvantages.
3. Explain free space management. Explain typical file control block, with a neat sketch.
4. Distinguish between single level directory structure and two level directory structures. What are its advantages and disadvantages?
5. Explain the access matrix model of implementing protection in operating system.
6. For the following page reference string **1,2,3,4,1,2,5,1,2,3,4,5**. Calculate the page faults using FIFO, Optimal and LRU using 3 and 4 frames.
7. Explain Demand paging in detail.
8. For the following page reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1**. Calculate the page faults using FIFO, Optimal and LRU using 3 and 4 frames.
9. Explain copy-on-write process in virtual memory.
10. What is a page fault? with the supporting diagram explain the steps involved in handling page fault.
11. Illustrate how paging affects the system performance.
12. Explain the various types of directory structures.
13. Explain the various file attributes.
14. Explain the various file operations.
15. Explain the various mechanism of implementing file protection.