TULSIRAMJI GAIKWAD – PATIL College of Engineering & Technology
Department of Information Technology

Operating System

Unit I

Introduction to Operating System (OS):

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

An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

Q1. Describe the Need for an OS, with appropriate definition of OS.

Ans.:- The primary need for the OS arises from the fact that user needs to be provided with services and OS ought to facilitate the provisioning of these services. The central part of a computer system is a processing engine called CPU. A system should make it possible for a user's application to use the processing unit. A user application would need to store information. The OS makes memory available to an application when required. Similarly, user applications need use of input facility to communicate with the application. This is often in the form of a key board, or a mouse or even a joy stick (if the application is a game for instance).

Definition of Operating System:

An Operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being applications programs.

An Operating system is concerned with the allocation of resources and services, such as memory, processors, devices and information. The Operating System correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

Q2. Write notes on User and System View of Operating System.

Ans.:- From the user point of view the primary consideration is always the convenience. It should be easy to use an application. In launching an application, it helps to have an icon which gives a clue which application it is. We have seen some helpful clues for launching a browser, e-mail or even a document preparation application. In other words, the human computer interface which helps to identify an application and its launch is very useful. This hides a lot of details of the more elementary instructions that help in selecting the application.

Similarly, if we examine the programs that help us in using input devices like a key board – all the complex details of character reading program are hidden from the user. The same is true when we write a program. For instance, when we use a programming language like C, a printf command helps to generate the desired form of output. The following figure essentially depicts the basic schema of the use of OS from a user stand point.

However, when it comes to the view point of a system, the OS needs to ensure that all the system users and applications get to use the facilities that they need. Also, OS needs to ensure that system resources are utilized efficiently. For instance, there may be many service requests on a Web server. Each user request need to be serviced. Similarly, there may be many programs residing in the main memory. The system need to determine which programs are active and which need to await some
form of input or output. Those that need to wait can be suspended temporarily from engaging the processor. This strategy alone enhances the processor throughput. In other words, it is important for an operating system to have a control policy and algorithm to allocate the system resources.

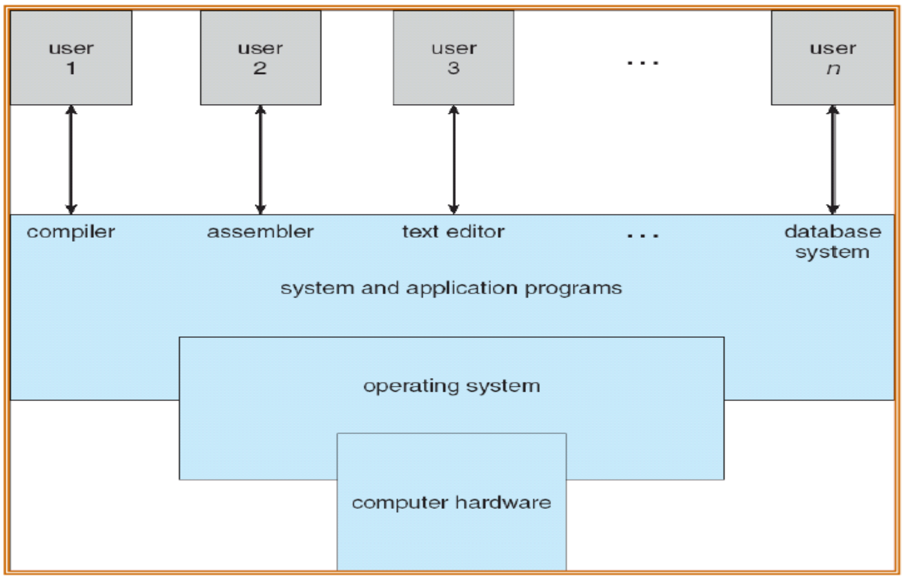


Fig: Conceptual view of Computer System

Q3. Explain various services provided by Operating System.

Ans.:- 1. Program execution

Operating system handles many kinds of activities from user programs to system programs like printer spooler, name servers, file server etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management.

- Loads a program into memory.
- Executes the program.
- Handles program's execution.

- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

2. I/O Operation

I/O subsystem comprised of I/O devices and their corresponding driver software. Drivers hides the peculiarities of specific hardware devices from the user as the device driver knows the peculiarities of the specific device.

Operating System manages the communication between user and device drivers. Following are the major activities of an operating system with respect to I/O Operation.

- I/O operation means read or write operation with any file or any specific I/O device.
- Program may require any I/O device while running.
- Operating system provides the access to the required I/O device when required.

3. File system manipulation

A file represents a collection of related information. Computer can store files on the disk (secondary storage), for long term storage purpose. Few examples of storage media are magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management.

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

4. Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, operating system manages communications between processes. Multiple processes with one another through communication lines in the network.

OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication.

- Two processes often require data to be transferred between them.
- The both processes can be on the one computer or on different computer but are connected through computer network.
- Communication may be implemented by two methods either by Shared Memory or by Message Passing.

5. Error handling

Error can occur anytime and anywhere. Error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling.

- OS constantly remains aware of possible errors.
- OS takes the appropriate action to ensure correct and consistent computing.

6. Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management.

- OS manages all kind of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

7. Protection

Considering a computer systems having multiple users the concurrent execution of multiple processes, then the various processes must be protected from each another's activities.

Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer systems. Following are the major activities of an operating system with respect to protection.

- OS ensures that all access to system resources is controlled.
- OS ensures that external I/O devices are protected from invalid access attempts.
- OS provides authentication feature for each user by means of a password.

Q4. Can you explain various Types of Operating System?

Ans.:- Batch System

- Some computer systems only did one thing at a time. They had a list of the computer system may be dedicated to a single program until its completion, or they may be dynamically reassigned among a collection of active programs in different stages of execution.
- Batch operating system is one where programs and data are collected together in a batch before processing starts. A job is predefined sequence of commands, programs and data that are combined in to a single unit called job.
- Figure below shows the memory layout for a simple batch system. Memory management in batch system is very simple. Memory is usually divided into two areas : Operating system and user program area.
- Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e first come first served fashion.
- When job completed execution, its memory is releases and the output for the job gets copied into an output spool for later printing.
- Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
- Batch systems are inconvenient for users because users can not interact with their jobs to fix problems. There may also be long turn-around times.

- Batch systems are inconvenient for users because users can not interact with their jobs to fix problems. There may also be long turn-around times.
- Example: System id generating monthly bank statement.

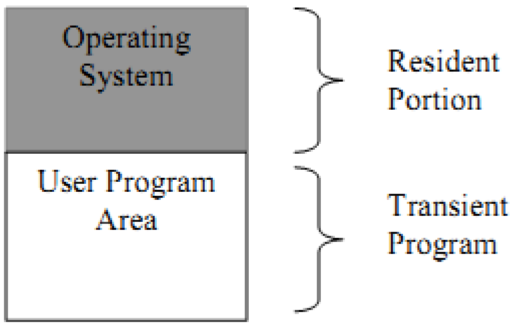


Fig: Memory Layout for a Simple Batch System

Advantages of Batch System

- Move much of the work of the operator to the computer.
- Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of Batch System

- Turn–around time can be large from user standpoint.
- Difficult to debug program.
- A job could enter an infinite loop.
- A job could corrupt the monitor, thus affecting pending jobs.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Time Sharing Systems

- Multi-programmed batched systems provide an environment where the various system resources (for example, CPU, memory, peripheral devices) are utilized effectively.
- Time sharing, or multitasking, is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while it is running.
- An interactive, or hands-on, computer system provides on-line communication between the user and the system. The user gives instructions to the operating system or to a program directly, and receives an immediate response. Usually, a keyboard is used to provide input, and a display screen (such as a cathode-ray tube (CRT) or monitor) is used to provide output.
- If users are to be able to access both data and code conveniently, an on-line file system must be available. A file is a collection of related information defined by its creator. Batch systems are appropriate for executing large jobs that need little interaction.

- Time-sharing systems were developed to provide interactive use of a computer system at a reasonable cost. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program that is loaded into memory and is executing is commonly referred to as a process. When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output is to a display for the user and input is from a user keyboard. Since interactive I/O typically runs at people speeds, it may take a long time to complete.
- A time-shared operating system allows the many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that she has her own computer, whereas actually one computer is being shared among many users.
- Time-sharing operating systems are even more complex than are multi-programmed operating systems. As in multiprogramming, several jobs must be kept simultaneously in memory, which requires some form of memory management and protection.

Multiprogramming

- When two or more programs are in memory at the same time, sharing the processor is referred to the multiprogramming operating system. Multiprogramming assumes a single processor that is being shared. It increases CPU utilization by organizing jobs so that the CPU always has one to execute.
- Figure below shows the memory layout for a multiprogramming system. The operating system keeps several jobs in memory at a time. This set of jobs is a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the jobs in the memory.

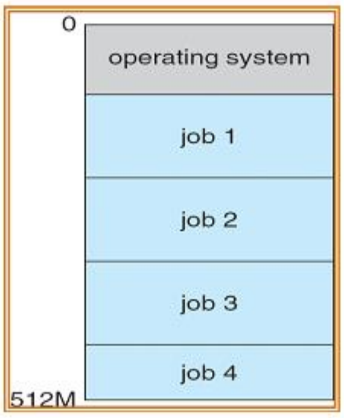


Fig: Memory Layout for Multiprogramming System

- Multiprogrammed system provides an environment in which the various system resources are utilized effectively, but they do not provide for user interaction with the computer system.
- Jobs entering into the system are kept into the memory. Operating system picks the job and begins to execute one of the jobs in the memory. Having several programs in memory at the same time requires some form of memory management.
- Multiprogramming operating system monitors the state of all active programs and system resources. This ensures that the CPU is never idle unless there are no jobs.

Advantages

1. High CPU utilization.
2. It appears that many programs are allotted CPU almost simultaneously.

Disadvantages

1. CPU scheduling is requires.
2. To accommodate many jobs in memory, memory management is required.

Multiprocessor System

Multiprocessor systems (also known as parallel systems or tightly coupled systems) have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

The multiple-processor systems in use today are of two types. Some systems use **asymmetric multiprocessing**, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors. The most common systems use **symmetric multiprocessing (SMP)**, in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors. Figure below illustrates a typical SMP architecture.

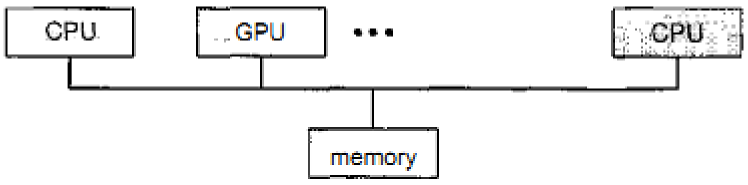


Figure: Symmetric multiprocessing architecture

Since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. These inefficiencies can be avoided if the processors share certain data structures. A multiprocessor system of this form will allow processes and resources—such as memory—to be shared dynamically among the various processors and can lower the variance among the processors.

Advantages

1. Increased throughput. By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N, however; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. Similarly, N programmers working closely together do not produce N times the amount of work a single programmer would produce.

2. Economy of scale. Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.

3. Increased reliability. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

Real Time Operating System

A **real-time operating system** (**RTOS**) is an operating system (OS) intended to serve real time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter.

A real-time operating system (RTOS) is an operating system that guarantees a certain capability within a specified time constraint. For example, an operating system might be designed to ensure that a certain object was available for a robot on an assembly line. In what is usually called a "hard" real-time operating system, if the calculation could not be performed for making the object available at the designated time, the operating system would terminate with a failure. In a "soft" real-time operating system, the assembly line would continue to function but the production output might be lower as objects failed to appear at their designated time, causing the robot to be temporarily unproductive. Some real-time operating systems are created for a special application and others are more general purpose. Some existing general purpose operating systems claim to be a real-time operating systems. To some extent, almost any general purpose operating system such as Microsoft's Windows 2000 or IBM's OS/390 can be evaluated for its real-time operating system qualities. That is, even if an operating system doesn't qualify, it may have characteristics that enable it to be considered as a solution to a particular real-time application problem.

Q5. Give Operating System Structure in brief.

Ans.:- 1. Simple Structure

Many commercial systems do not have well-defined structures. Frequently, such operating systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system. Figure below shows its structure.

In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail. MS-DOS was also limited by the hardware of its era. Because the Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

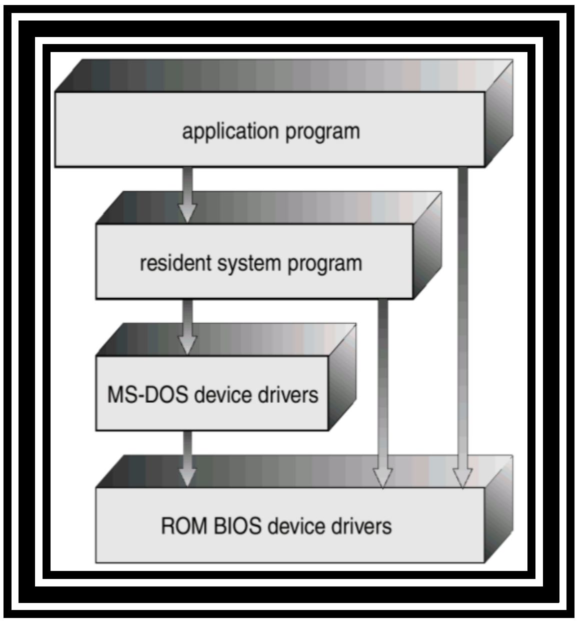


Fig: MS- Dos Layer Structure

Another example of limited structuring is the original UNIX operating system. UNIX is another system that initially was limited by hardware functionality. It consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. We can view the traditional UNIX operating system as being layered, as shown in Figure below. Everything below the system call interface and above the physical hardware is the kernel. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls. Taken in sum, that is an enormous amount of functionality to be combined into one level. This monolithic structure was difficult to implement and maintain.

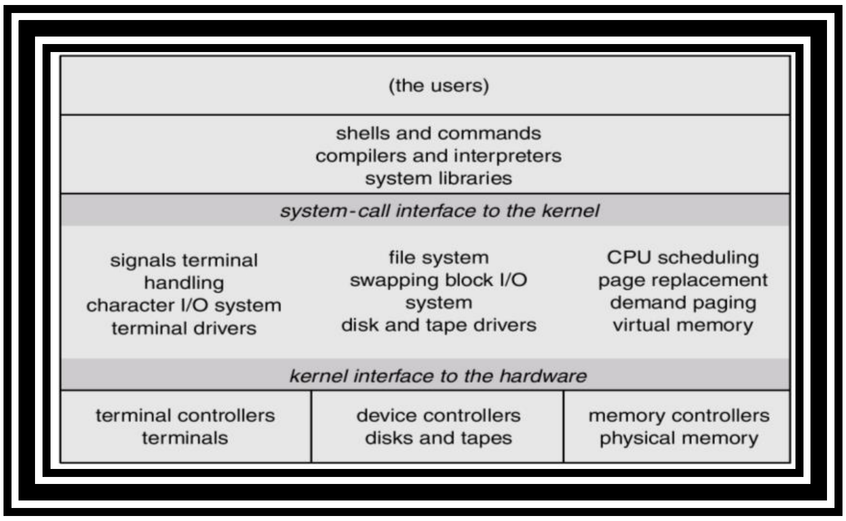


Fig: Unix System Structure

2. Layered Approach

With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS or UNIX systems. The operating system can then retain much greater control over the computer and over the applications that make use of that computer.

A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken up into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface. This layering structure is depicted in Figure below.

An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data. A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn, can
invoke operations on lower-level layers.

The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system is simplified. Each layer is implemented with only those operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

The major difficulty with the layered approach involves appropriately defining the various layers. A final problem with layered implementations is that they tend to be less efficient than other types. These limitations have caused a small backlash against layering in recent years. Fewer layers with more functionality are being designed, providing most of the advantages of modularized code while avoiding the difficult problems of laver definition and interaction.

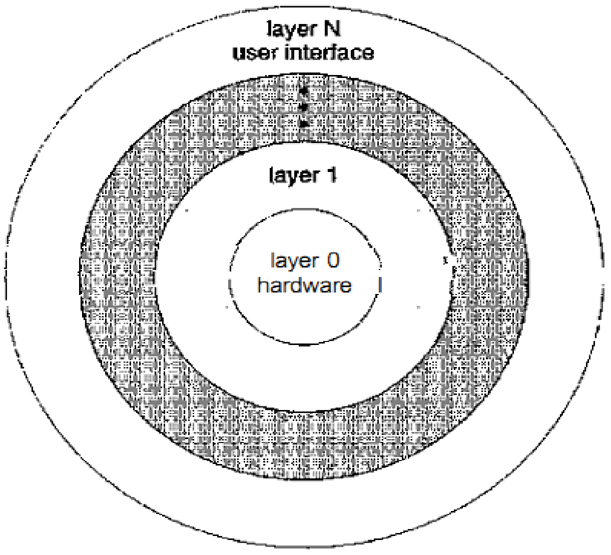


Figure: A layered operating system

System Design Goals:

- **User goals** − operating system should be con-venient to use, easy to learn, reliable, safe, and fast.
- **System goals** − operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Q6. Outline the importance of System Calls in detailed with its method of passing parameters and types.

Ans.:- System calls provide the interface between a runnning program and the operating system. These calls are generally available as assembly-language instructions. Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, Bliss, PL/360).

Three general methods are used to pass parameters between a running program and the operating system:

- Pass parameters in registers.
- Store the parameters in a table in memory, and the table address is passed as a parameter in a register.

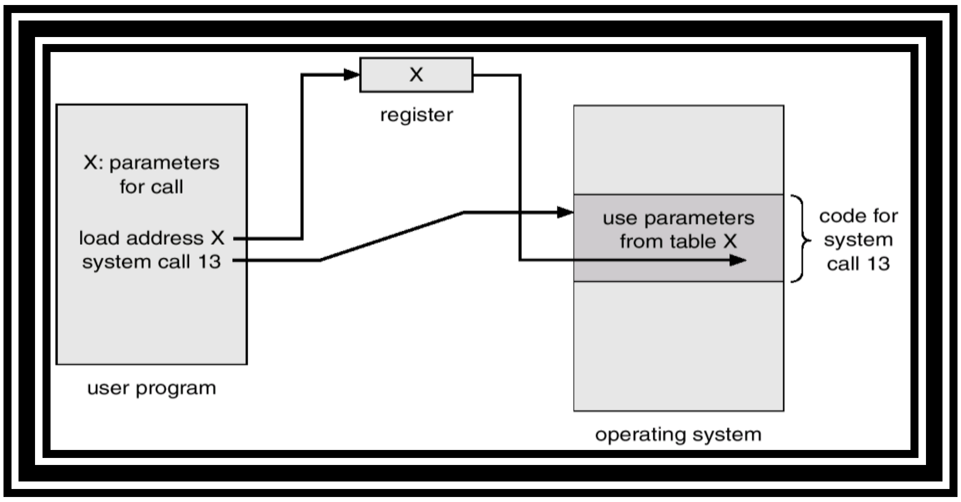


Fig: Passing of Parameters as a Table

- Push (store) the parameters onto the stack by the program, and pop off the stack by the operating system.

Types of System Calls:

1. Process Control

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a debugger—a system program designed to aid the programmer in finding and correcting bugs—to determine the cause of the problem. Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command. In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error. In a GUI system, a pop-up window might alert the user to the error and ask for guidance. In a batch system, the command interpreter usually terminates the entire job and continues with the next job. Some systems allow control cards to indicate special recovery actions in case an error occurs.

2. File Management

We need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it.

We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, a file type, protection codes, accounting information, and so on. At least two system calls, get file attribute and set file attribute, are required for this function. Some operating systems provide many more calls, such as calls for file move and copy.

3. Device Management

A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files). If there are multiple users of the system, the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and close system calls for files. Other operating systems allow unmanaged access to devices. Once the device has been requested (and allocated to us), we can read, write, and (possibly) reposition the device.

4. Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date. Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

In addition, the operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes).

5. Communication

There are two common models of interprocess communication: the message-passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network. Each computer in a network has a host name by which it is commonly known. A host also has a network identifier, such as an IP address. Similarly, each process has a process name, and this name is translated into an identifier by which the operating system can refer to the process. The get host id and get processed system calls do this translation. The identifiers are then passed to the general-purpose open and close calls provided by the file system or to specific open connection and close connection system calls, depending on the system's model of communication. The recipient process usually must give its permission for communication to take place with an accept connection call. Most processes that will be receiving connections are special-purpose daemons, which are systems programs provided for that purpose. They execute a wait for connection call and are awakened when a connection is made. The source of the communication, known as the client, and the receiving daemon, known as a server, then exchange messages by using read message and write message system calls. The close connection call terminates the communication.

In the shared-memory model, processes use shared memory create and shared memory attach system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by the processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for inter-computer communication. Shared memory allows maximum speed and convenience of communication, since it can be done at memory speeds when it takes place within a computer. Problems exist, however, in the areas of protection and synchronization between the processes sharing memory.

Q7. Explain in brief System Programs.

Ans.:- System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex. They can be divided into these categories:

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

• **Status information:** Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information.

• **File modification:** Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.

• **Programming-language support:** Compilers, assemblers, debuggers and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.

• **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.

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

TULSIRAMJI GAIKWAD – PATIL College of Engineering & Technology
Department of Information Technology
Operating System
<u>Unit II</u>

Q1. Explain File Concept in detailed.

Ans.:-

- A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referred by name. Files have unique file names and may be created and deleted. Restrictions on access control usually apply at the file level.
- A file is a container for a collection of information. The file manager provides a protection mechanism to allow users administrator how processes executing on behalf of different users can access the information in a file. File protection is a fundamental property of files because it allows different people to store their information on a shared computer.
- File represents programs and data. Data files may be numeric, alphabetic, binary or alpha numeric. Files may be free form, such as text files. In general, file is sequence of bits, bytes, lines or records.
- A file has a certain defined structure according to its type.
  1 Text File
  2 Source File
  3 Executable File
  4 Object File

File Structure

Four terms are used for files
• Field
• Record
• Database

A field is the basic element of data. An individual field contains a single value. A record is a collection of related fields that can be treated as a unit by some application program.

A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referenced by name. Files have file names and maybe created and deleted. Access control restrictions usually apply at the file level.

A database is a collection of related data. Database is designed for use by a number of different applications.

A database may contain all of the information related to an organization or project, such as a business or a scientific study. The database itself consists of one or more types of files. Usually, there is a separate database management system that is independent of the operating system.

Q2. Write various File Attributes with operations performed on file.

Ans.:-

- File attributes vary from one operating system to another. The common attributes are,
- Name – only information kept in human-readable form.
- Identifier – unique tag (number) identifies file within file system
- Type – needed for systems that support different types
- Location – pointer to file location on device
- Size – current file size
- Protection – controls who can do reading, writing, executing
- Time, date, and user identification – data for protection, security, and usage monitoring. This Information about files are kept in the directory structure, which is maintained on the disk

File Operations

- **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **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.
- **Reading a file.** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched 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. Both the read and write operations use this same pointer, saving space and reducing system complexity.
- 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/O. This file operation is also known as a file seek.
- Deleting a file. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- 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—except for file length—but lets the tile be reset to length zero and its file space released.

Q3. How file can be access by using various Access Methods?

Ans.:-

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.

Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is most common; for example, editors and compilers usually access files in this fashion.

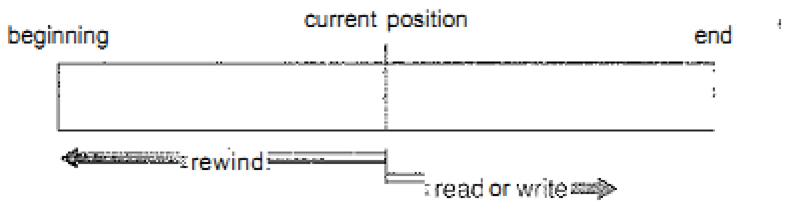


Figure: Sequential-access file.

Reads and writes make up the bulk of the operations on a file. A read operation- read next-reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation- write next- appends to the end of the file and advances to the
end of the newly written material (the new end of file). Such 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.

Direct Access

Another method is direct access (or relative 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. Thus, 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. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

As a simple example, on an airline-reservation system, we might store all the information about a particular flight (for example, flight 713) in the block identified by the flight number. Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people's names or search a small in-memory index to determine a block to read and search.

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 effect a read n, we would, position to n and then read next.

The block number provided by the user to the operating system is normally a relative block number. A relative block number is an index relative to the beginning of the file. This, the first relative block of the file is 0, the next is 1, and so on, even though the actual absolute disk address of the block may be 14703 for the first block and 3192 for the second. The use of relative block numbers allows the operating system to decide where the file should be placed and helps to prevent the user from accessing portions of the file system that may not be part of her file. Some systems start their relative block numbers at 0; others start at 1.

Q4. Discuss Directory structure in brief.

Ans.:-

1. 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, however, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names. If two users call their data file test, then the unique-name rule is violated. Although file names are generally selected to reflect the content of the file, they are often limited in length, complicating the
task of making file names unique. The MS-DOS operating system allows only 11-character file names; UNIX, in contrast, allows 255 characters. 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.

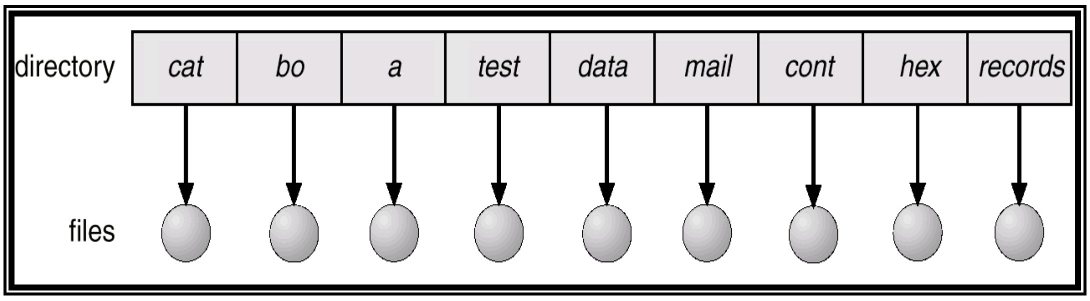


Fig: Single-Level Directory

2. Two-Level Directory

In the two-level directory structure, each user has his own user file directory (LTD). The UFDs have similar structures, but each lists only the files of a single user. 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.

When a user refers to a particular file, only his own UFD is searched. Thus, 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's file that has the same name.

The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new UFD and adds an entry for it to the MFD. The execution of this program might be restricted to system administrators.

Although the two-level directory structure solves the name-collision problem, it still has disadvantages. This structure effectively isolates one user from another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files. Some systems simply do not allow local user files to be accessed by other users.

If access is to be permitted, one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a two-level directory, we must give both the user name and the file name. A two-level directory can be thought of as a tree, or an inverted tree, of height 2. The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves. The files are the leaves of the tree. Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file). Thus, a user name and a file name define a path name. Every file in the system has a path name. To name a file uniquely, a user must know the path name of the file desired.

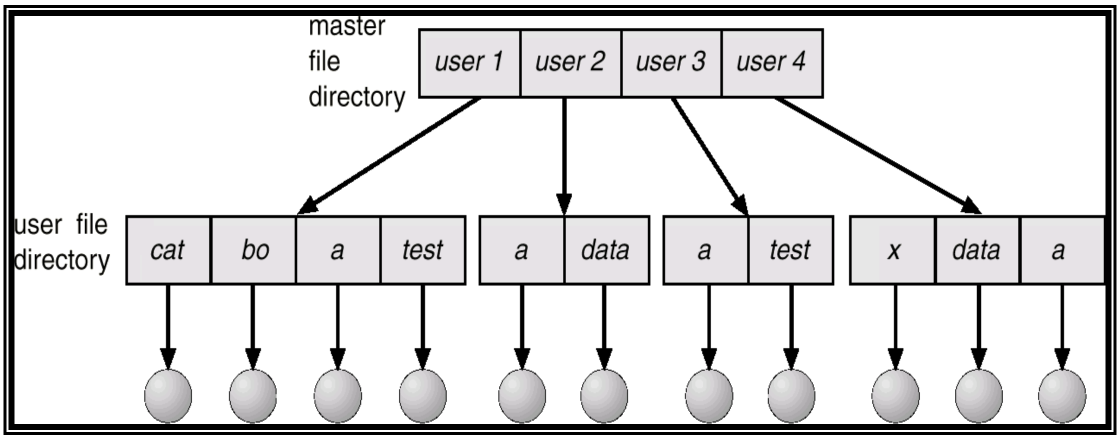


Fig: Tree Structured Directory

3. Tree Structured Directory

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 (or subdirectory) 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.

In normal use, each process has a current directory. The current directory should contain most of the files that are of current interest to the process. When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file. To change directories, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory. Thus, the user can change his current directory whenever he desires. From one change director y system call to the next, all open system calls search the current directory for the specified file. Note that the search path may or may not contain a special entry that stands for "the current directory."

The initial current directory of the login shell of a user is designated when the user job starts or the user logs in. The operating system searches the accounting file (or some other predefined location) to find an entry for this user (for accounting purposes). In the accounting file is a pointer to (or the
name of) the user's initial directory. This pointer is copied to a local variable for this user that specifies the user's initial current directory. From that shell, other processes can be spawned. The current directory of any subprocess is usually the current directory of the parent when it was spawned.

Path names can be of two types: absolute and relative. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory.

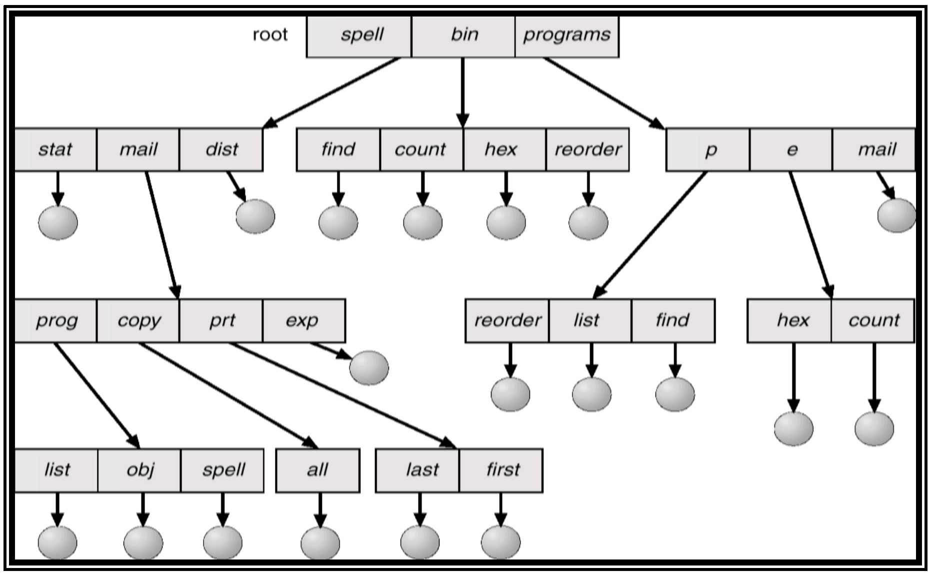


Figure: Tree-structured directory structure

4. Acyclic Graph Directory

Consider two programmers who are working on a joint project. The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own 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- that is, a graph with no cycles- 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.

An acyclic-graph directory structure is more flexible than is a simple tree structure, but it is also more complex. Several problems must be considered carefully. A file may now have multiple absolute path names. Consequently, distinct file names may refer to the same file. This situation is similar to the aliasing problem for programming languages. If we are trying to traverse the entire file system—to find a file, to accumulate statistics on all files, or to copy all files to backup storage—this problem becomes significant, since we do not want to traverse shared structures more than once.

Another problem involves deletion. For reusing the space allocated to a shared file, one possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointers to the now-nonexistent file. Worse, if the remaining file pointers contain actual disk addresses, and the space is subsequently reused for other files, these dangling pointers may point into the middle of other files. Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted. We could keep a list of all references to a file (directory entries or symbolic links). When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link or directory entry is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty.

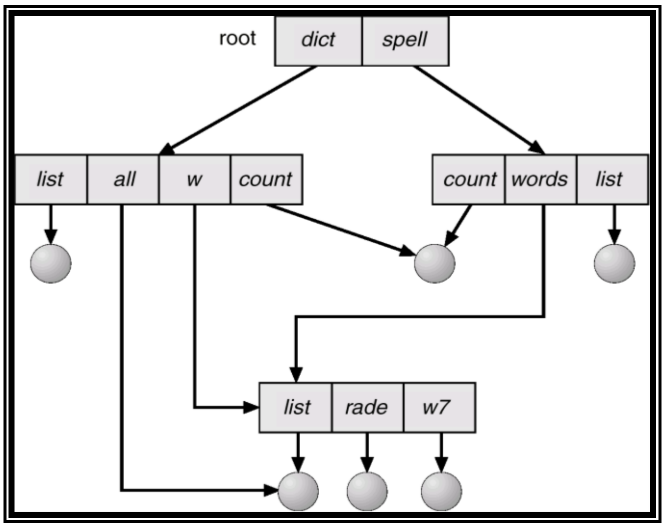


Fig: Acyclic Graph Directory

Q5. Illustrate the techniques used for Directory Implementation.

Ans.:-

Linear List

The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. This method is simple to program but time-consuming to execute. To create a new file, we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it. To reuse the directory entry, we can do one of several things. We can mark the entry as unused (by assigning it a special name, such as an all-blank name, or with a used-unused, bit in each entry), or we can attach it to a list of free directory entries. A third alternative is to copy the last entry in the directory into the freed location and to decrease the length of the directory. A linked list can also be used to decrease the time required to delete a file.

The real disadvantage of a linear list of directory entries is that finding a file requires a linear search. Directory information is used frequently, and users will notice if access to it is slow. In fact, many operating systems implement a software cache to store the most recently used directory information. A cache hit avoids the need to constantly reread the information from disk. A sorted list allows a binary search and decreases the average search time. However, the requirement that the list be kept sorted may complicate creating and deleting files, since we may have to move substantial amounts of directory information to maintain a sorted directory. A more sophisticated tree data structure, such as a B-tree, might help here. An advantage of the sorted list is that a sorted directory listing can be produced without a separate sort step.

Hash Table

Another data structure used for a file directory is a hash table. With this method, a linear list stores the directory entries, but a hash data structure is also used. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for collisions- situations in which two file names hash to the same location. The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size. For example, assume that we make a linear-probing hash table that holds 64 entries. The hash function converts file names into integers from 0 to 63, for instance, by using the remainder of a division by 64. If we later try to create a 65th file, we must enlarge the directory hash table- say, to 128 entries. As a result, we need a new hash function that must map file names to the range 0 to 127, and we must reorganize the existing directory entries to reflect their new hash-function values.

Alternatively, a chained-overflow hash table can be used. 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. Lookups may be somewhat slowed, because searching for a name might require stepping through a linked list of colliding table entries. Still, this method is likely to be much faster than a linear search through the entire directory.

Q6. Explain how Free-Space Management is carried out?

Ans.:-

Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files, if possible.

Bit Vector

Free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.

For example consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free-space bit map would be

001111001111110001100000011100000 …..

The main advantage of this approach is that it is relatively simple and efficient to find the first free block or n consecutive free blocks on the disk.

The calculation of the block number is

(number of bits per word) x (number of 0-value words) + offset of first 1 bit

Linked List

Another approach is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.

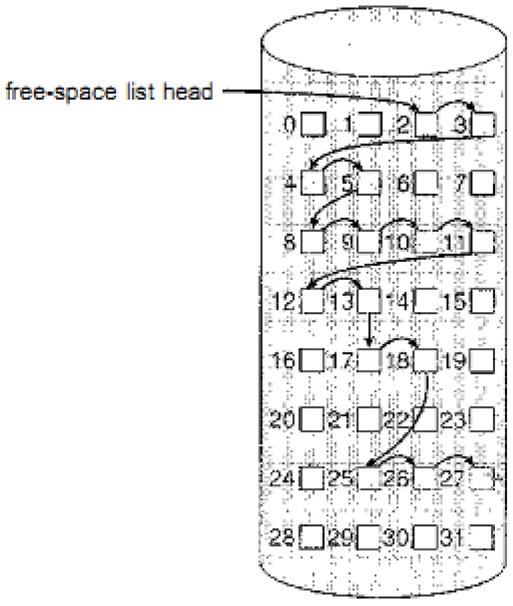


Fig: Linked Free Space on a Disk

Grouping

A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these blocks are actually free. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly, unlike in the standard linked-list approach.

Counting

Several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous allocation algorithm or through clustering. A list of n free disk addresses, we can keep the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as count is generally greater than 1.

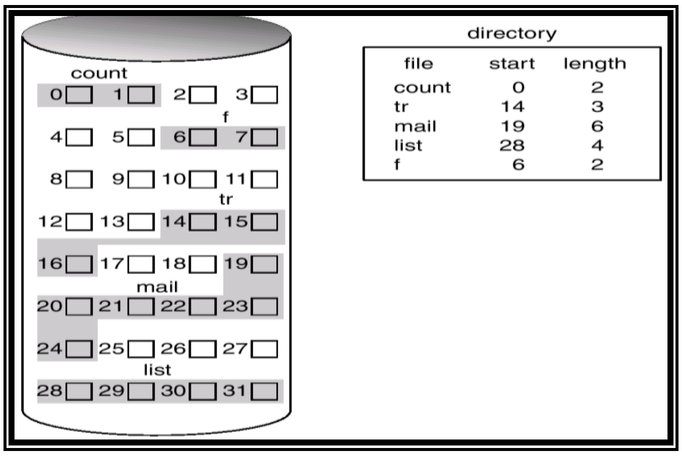
Q7. Explain various Allocation Methods.

Ans.:-

Contiguous Allocation

- The contiguous allocation method requires each file to occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. Notice that with this ordering assuming that only one job is accessing the disk, accessing block b + 1 after block b normally requires no head movement.
- When head movement is needed, it is only one track. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal. 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!), 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 allocate 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. The contiguous disk-space-allocation problem can be seen to be a particular application of the general dynamic storage-allocation First Fit and Best Fit are the most common strategies used to select a free hole from the set of available holes. Simulations have shown that both first-fit and best-fit are more efficient than worst-fit in terms of both time and storage utilization. Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first-fit is generally faster.
- These algorithms suffer from the problem of external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks. It becomes a problem when the largest contiguous chunks is insufficient for a request; storage is fragmented into a number of holes, no one 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 either a minor or a major problem.
- To prevent loss of significant amounts of disk space to external fragmentation, the user had to run repacking routine that copied the entire file system onto another floppy disk or onto a tape. The original floppy disk was then freed completely, creating one large contiguous free space. The routine then copied the files back onto the floppy disk by allocating contiguous space from this one large hole. This scheme effectively compacts all free space into one contiguous space, solving the fragmentation problem.

- The time cost is particularly severe for large hard disks that use contiguous allocation, where compacting all the space may take hours and may be necessary on a weekly basis. During this down time, normal system operation generally cannot be permitted, so such compaction is avoided at all costs on production machines.
- A major problem is determining how much space is needed for a file. When the file is created, the total amount of space it will need must be found and allocated.
- The user will normally over estimate the amount of space needed, resulting in considerable wasted space.



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Fig: Contiguous Allocation of Disk Space

Linked Allocation

- Linked allocation solves all problems of contiguous allocation. With link allocation, each file is a linked list disk blocks; the disk blocks may be scattered anywhere on the disk.
- This pointer is 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 a free block to be found via the free-space management system, and this new block is the written to, and is linked to the end of the file
- There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. Notice also that there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks. Consequently, it is never necessary to compact disk space.
- The major problem is that it can be used effectively for only sequential access files. To find the ith block of a file we must start at the beginning of that file, and follow the pointers until we get to the ith block. Each access to a pointer requires a disk read and sometimes a disk seek. Consequently, it is inefficient to support a direct-access capability for linked allocation files.

- Linked allocation is the space required for the pointers If a pointer requires 4 bytes out of a 512 Byte block then 0.78 percent of the disk is being used for pointer, rather than for information.
- The usual solution to this problem is to collect blocks into multiples, called clusters, and to allocate the clusters rather than blocks. For instance, the file system define a cluster as 4 blocks and operate on the disk in only cluster units.
- Pointers then use a much smaller percentage of the file's disk space. This method allows the logical-to-physical block mapping to remain simple, but improves disk throughput (fewer disk head seeks) and decreases the space needed for block allocation and free-list management. The cost of this approach an increase in internal fragmentation.
- Yet another problem is reliability. Since the files are linked together by pointers scattered all over the disk, consider what would happen if a pointer— were lost or damaged. Partial solutions are to use doubly linked lists or to store the file name and relative block number in each block; however, these schemes require even more overhead for each file.
- An important variation, on the linked allocation method is the use of a file allocation table (FAT). This simple but efficient method of disk-space allocation is used by the MS-DOS and OS/2 operating systems. A section of disk at the beginning of each-partition 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 much as is 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 then contains the block number of the next block in the file. This chain continues until the last block, which has a special end-of-file value -as the table entry. Unused blocks are indicated by a 0 table value. Allocating a new block to a file is a simple matter of finding the first 0-valued table entry, and replacing the previous end-of-file value with the address of the new block. The 0 is then replaced with the end-off ile value. An illustrative example is the FAT structure of for a file consisting of disk blocks 217, 618, and 339.

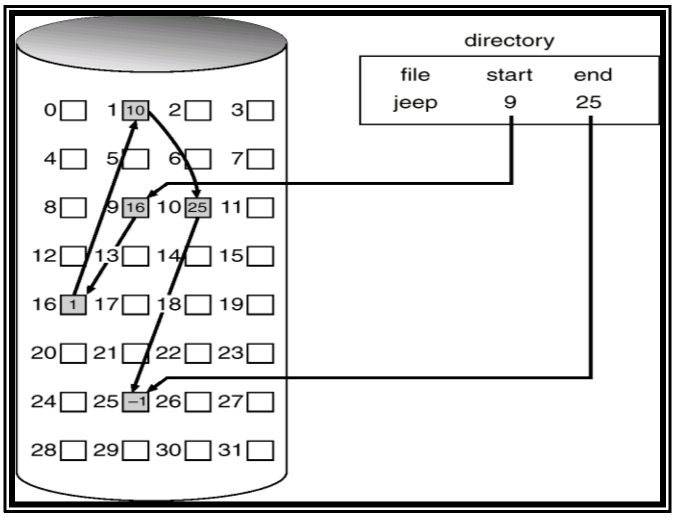


Fig: Linked Allocation

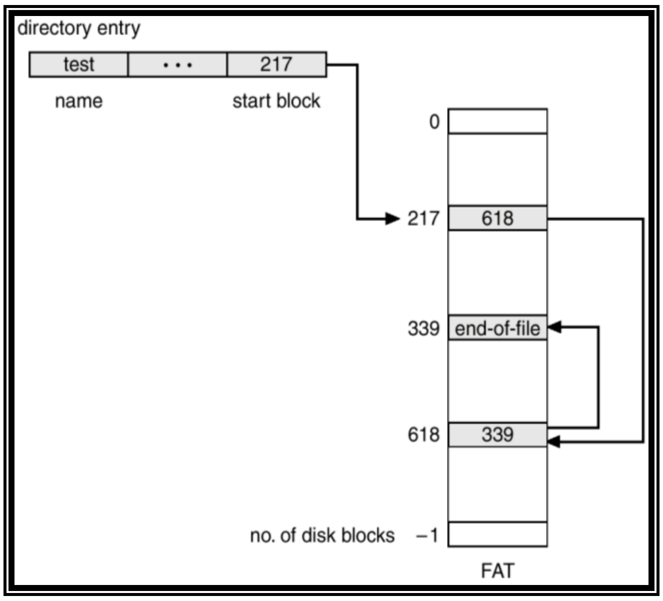


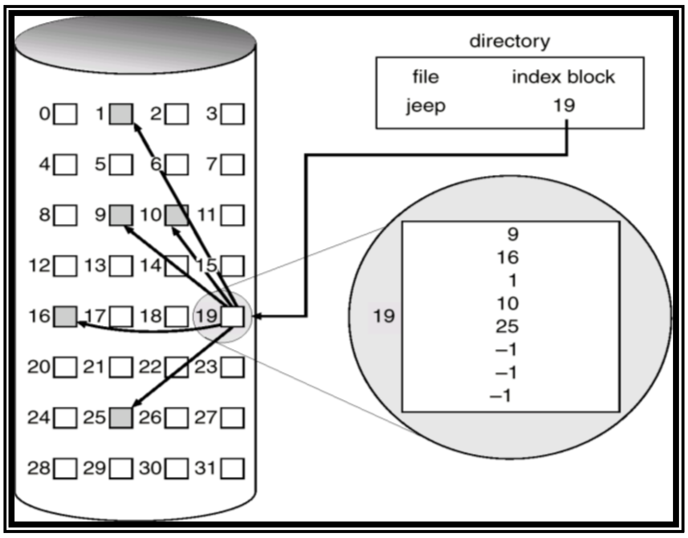
Fig: file Allocation table

Indexed Allocation

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. The absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order Indexed allocation solves this problem by bringing all the pointers together into one location: the 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.
- 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.
- Allocation supports direct access, without suffering from external fragmentation because any free block on he disk may satisfy a request for more space.
- Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

1. Linked scheme. An index block is normally one disk block. Thus, it can be read and
 written directly by itself.
2. Multilevel index. A variant of the linked representation is to use 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 that block to find the desired data block.



Q8. Can you describe Disk Performance Parameters in brief?

Ans.:-

When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track.

Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position the head at the track is known as seek time.

When once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as rotational delay, or rotational latency. The sum of the seek time, if any, and the rotational delay equals the access time, which is the time it takes to get into position to read or write.

Once the head is in position, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation; the time required for the transfer is the transfer time.

Seek Time Seek time is the time required to move the disk arm to the required track. It turns out that this is a difficult quantity to pin down. The seek time consists of two key components: the initial startup time and the time taken to traverse the tracks that have to be crossed once the access arm is up to speed.

$$T_s = m \times n + s$$

Rotational Delay Disks, other than floppy disks, rotate at speeds ranging from 3600 rpm up to, as of this writing, 15,000 rpm; at this latter speed, there is one revolution per 4 ms. Thus, on the average, the rotational delay will be 2 ms. Floppy disks typically rotate at between 300 and 600 rpm. Thus the average delay will be between 100 and 50 ms.

Transfer Time The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

$$T= b/rN$$

where
T = transfer time
b = number of bytes to be transferred
N = number of bytes on a track
r = rotation speed, in revolutions per second
Thus the total average access time can be expressed as
$T_a = T_s +$
where Ts is the average seek time.

Q9. Write notes on Disk Scheduling:

Ans.:-

The amount of head needed to satisfy a series of I/O request can affect the performance. If desired disk drive and controller are available, the request can be serviced immediately. If a device or controller is busy, any new requests for service will be placed on the queue of pending requests for that drive. When one request is completed, the operating system chooses which
pending request to service next.

Different types of scheduling algorithms are as follows.

1. First Come, First Served scheduling algorithm(FCFS).

2. Shortest Seek Time First (SSTF) algorithm
3. SCAN algorithm
4. Circular SCAN (C-SCAN) algorithm
5. Look Scheduling Algorithm

First Come, First Served scheduling algorithm(FCFS):

The simplest form of scheduling is first-in-first-out (FIFO) scheduling, which processes items from the queue in sequential order. This strategy has the advantage of being fair, because every request is honored and the requests are honored in the order received. With FIFO, if there are only a few processes that require access and if many of the requests are to clustered file sectors, then we can hope for good performance.

Priority: With a system based on priority (PRI), the control of the scheduling is outside the control of disk management software.

Last In First Out: ln transaction processing systems, giving the device to the most recent user should result. In little or no arm movement for moving through a sequential file. Taking advantage of this locality improves throughput and reduces queue length.

Shortest Seek Time First (SSTF) algorithm:

The SSTF policy is to select the disk I/O request the requires the least movement of the disk arm from its current position. Scan With the exception of FIFO, all of the policies described so far can leave some request unfulfilled until the entire queue is emptied. That is, there may always be new requests arriving that will be chosen before an existing request.

The choice should provide better performance than FCFS algorithm.

Under heavy load, SSTF can prevent distant request from ever being serviced. This phenomenon is known as starvation. SSTF scheduling is essentially a from of shortest job first scheduling. SSTF scheduling algorithm are not very popular because of two reasons.

1. Starvation possibly exists.
2. it increases higher overheads.

SCAN scheduling algorithm:

The scan algorithm has the head start at track 0 and move towards the highest numbered track, servicing all requests for a track as it passes the track. The service direction is then reserved and the scan proceeds in the opposite direction, again picking up all requests in order.

SCAN algorithm is guaranteed to service every request in one complete pass through the disk. SCAN algorithm behaves almost identically with the SSTF algorithm. The SCAN algorithm is sometimes called elevator algorithm.

C SCAN Scheduling Algorithm

The C-SCAN policy restricts scanning to one direction only. Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again.

This reduces the maximum delay experienced by new requests.

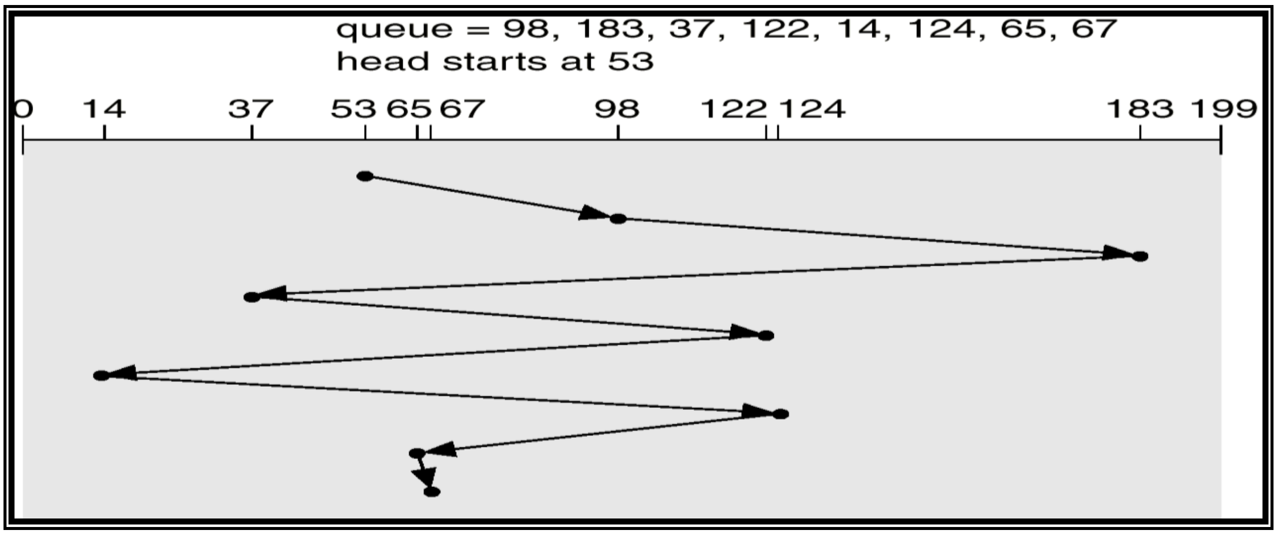
LOOK Scheduling Algorithm

Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling, reverse direction and repeat. This algorithm is similar to innermost and outermost track on each circuit.

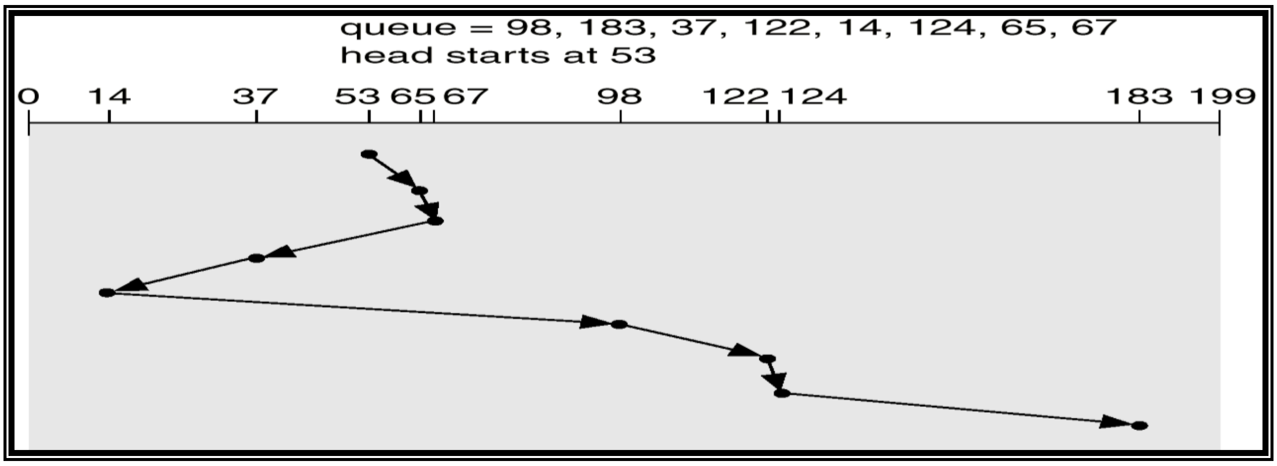
C LOOK Scheduling Algorithm

The C-LOOK policy restricts scanning to one direction only. Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again.

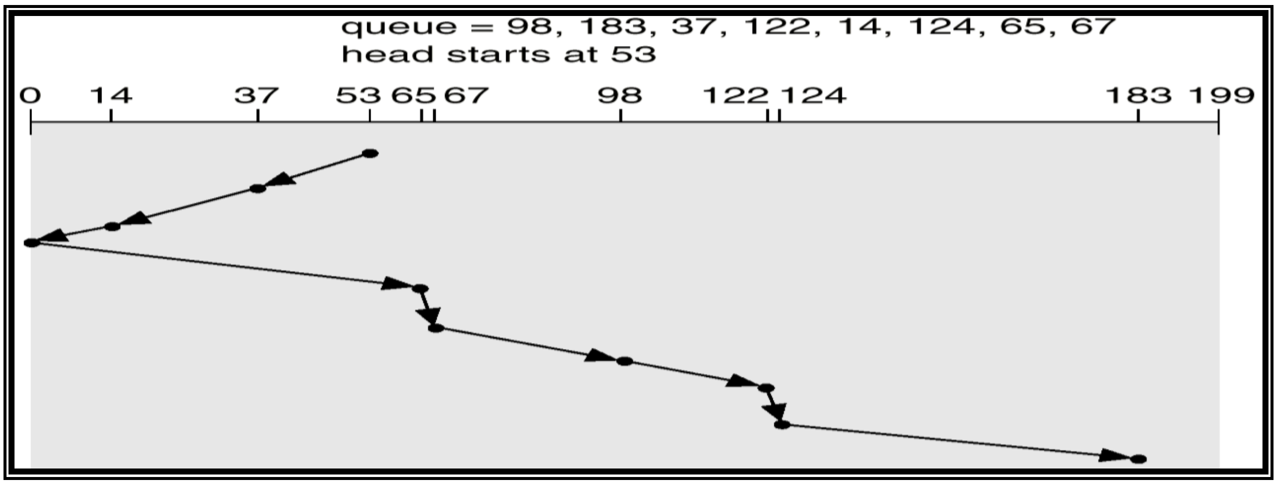
FCFS



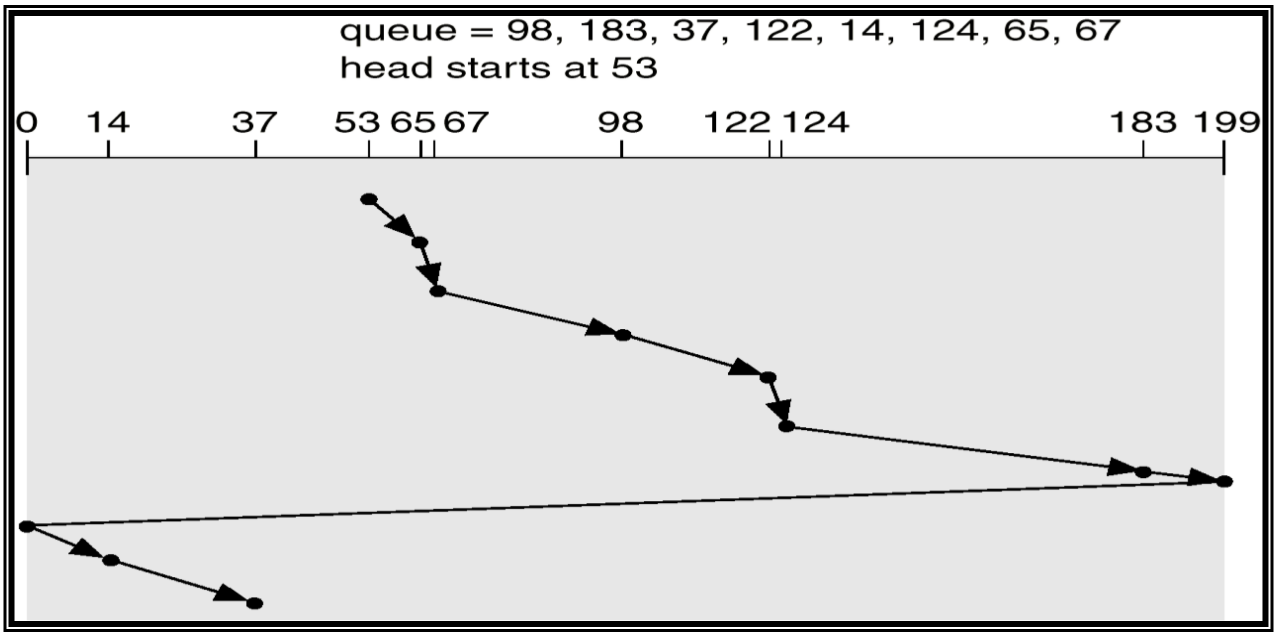
SSTF


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 53 65 67 98 122 124 183 199

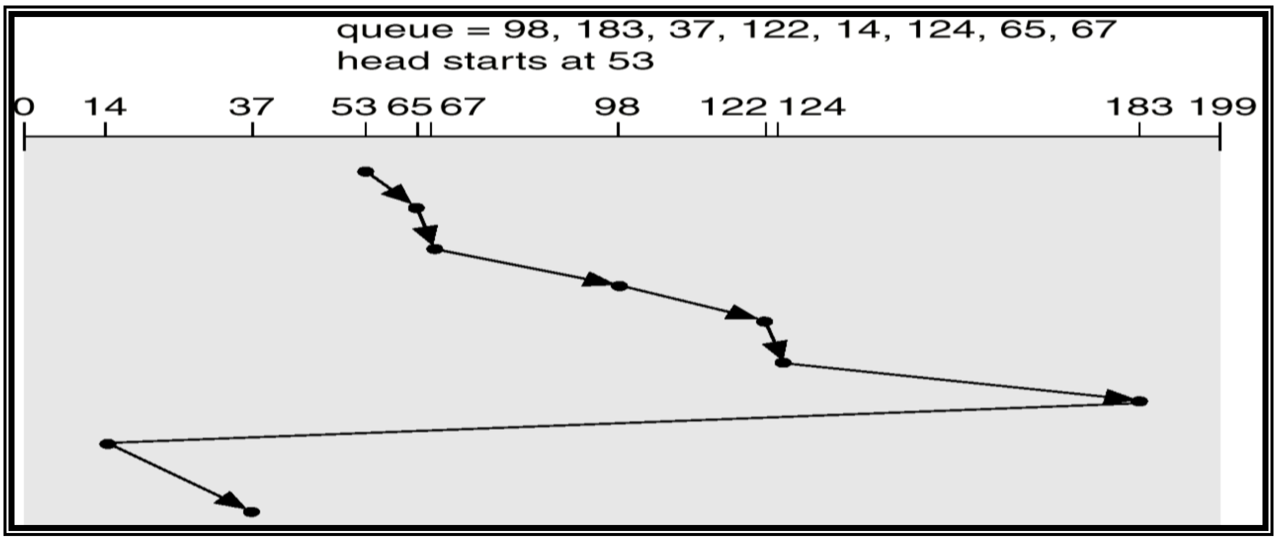
SCAN


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 53 65 67 98 122 124 183 199

C- SCAN


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 53 65 67 98 122 124 183 199

C Look


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199