BICTE Operating System Notes

Notes By Rajesh parajuli

OPERATING SYSTEMS

OBJECTIVES:

- To learn the fundamentals of Operating Systems.
- To learn the mechanisms of OS to handle processes and threads and their communication
- To learn the mechanisms involved in memory management in contemporary OS
- To gain knowledge on distributed operating system concepts that includes architecture,
- Mutual exclusion algorithms, deadlock detection algorithms and agreement protocols
- To know the components and management aspects of concurrency management

UNIT-I

Introduction: Concept of Operating Systems, Generations of Operating systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System.

Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching

Thread: Definition, Various states, Benefits of threads, Types of threads, Concept of Multithreads.

UNIT-II

Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJF, RR; Multiprocessor scheduling: Real Time scheduling: RM and EDF.

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion,Hardware Solution, Strict Alternation, Peterson's Solution, The Producer/Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

UNIT-III

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and External fragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU).

UNIT-IV

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), Free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance.

I/O Hardware: I/O devices, Device controllers, Direct memory access Principles of I/O Software: Goals of Interrupt handlers, Device drivers, Device independent I/O software.

UNIT-V

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks.

TEXT BOOKS:

1. Operating System Concepts Essentials, 9th Edition by AviSilberschatz, Peter Galvin,Greg Gagne, Wiley Asia Student Edition.
2. Operating Systems: Internals and Design Principles, 5th Edition, William Stallings,Prentice Hall of India.

REFERENCE BOOKS:

1. Operating System: A Design-oriented Approach, 1st Edition by Charles Crowley, Irwin Publishing
2. Operating Systems: A Modern Perspective, 2nd Edition by Gary J. Nutt, Addison-Wesley
3. Design of the Unix Operating Systems, 8th Edition by Maurice Bach, Prentice-Hallof India
4. Understanding the Linux Kernel, 3rd Edition, Daniel P. Bovet, Marco Cesati, O'Reilly and Associates

OUTCOMES:

At the end of the course the students are able to:

- Create processes and threads.
- Develop algorithms for process scheduling for a given specification of CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time.
- For a given specification of memory organization develop the techniques for optimally allocating memory to processes by increasing memory utilization and for improving the access time.
- Design and implement file management system.
- For a given I/O devices and OS (specify) develop the I/O management functions in OS as part of a uniform device abstraction by performing operations for synchronization between CPU and I/O controllers.

INDEX

UNIT-I

Operating System Introduction: Operating Systems Objectives and functions, Computer System Architecture, OS Structure, OS Operations, Evolution of Operating Systems - Simple Batch, Multi programmed, time shared, Personal Computer, Parallel, Distributed Systems, Real-Time Systems, Special - Purpose Systems, Operating System services, user OS Interface, System Calls, Types of System Calls, System Programs, Operating System Design and Implementation, OS Structure, Virtual machines

A computer system is a collection of hardware and software components designed to provide an effective tool for computation.

Hardware generally refers to the electrical, mechanical and electronic parts that make up the computer(*i.e.*, Internal architecture of the computer (or) physical computing equipment).However the hardware is sophisticated, it cannot function properly without a proper driver which can drive it and bring it to the best advantage. For example, a car, even though sophisticated in its features, it cannot function independently without being properly driven by an efficient driver.

Similarly the hardware though technologically innovative, and which presents enhanced features, which needs set of programs to bring it to operation and to the best advantage. So, the driver that drives the hardware is software.

Software refers to the set of programs written to provide services to the system. It gives life and meaning to the hardware and bring it to the operational level, which otherwise is a useless piece of metal.

Software is basically of two types:

1. Application software
2. System software

Application Software: Set of programs written for a specific area of application. For example, word processors, spreadsheets and data base management systems, etc.

System Software: Set of programs written from the point of view of the machine *i.e.*, for the sake of the system. System software provides environment for execution of application software. One cannot aim to develop or write application software, without the presence and aid of system software.

NEED OF AN OPERATING SYSTEM

Operating system is an interface between user and hardware. OS creates user friendly environment.

Suppose when working with DOS-OS, if the user want to delete the program ,he has to type the command C:\DEL FILENAME and press the enter, then the program will be deleted. So ,the user delete the program very easily with the help of OS.

Suppose user want to delete the program without using OS, then he has to write a separate program for DEL command and perform the operation. Every time for doing any operation he has to write a separate program. So ,it is very difficult for the programmer, for that OS provides user friendly environment ,it is the main function of the OS. For example, MS-DOS provides different commands for performing different operations.

When the user sends a command, the OS must make sure that the command is executed or if it is not executed, must arrange for the user to get a message about explaining the error.

Another important function is resource management. The OS acts like a government, the government collects money from various resources and distribute to the different development activities. Similarly the OS collects all resources in the network environment and allocates the resources to requesting processes in an efficient manner. So, it is called as "*Resource Manager*".

The OS controls and co-ordinates the execution of the programs. So, it is sometimes called as Control program (It provides interface to various hardware components such as printer, monitor, keyboard, etc. So, it can able to control the execution of a program).

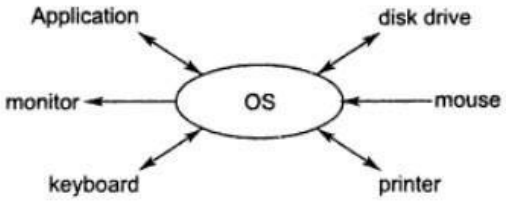
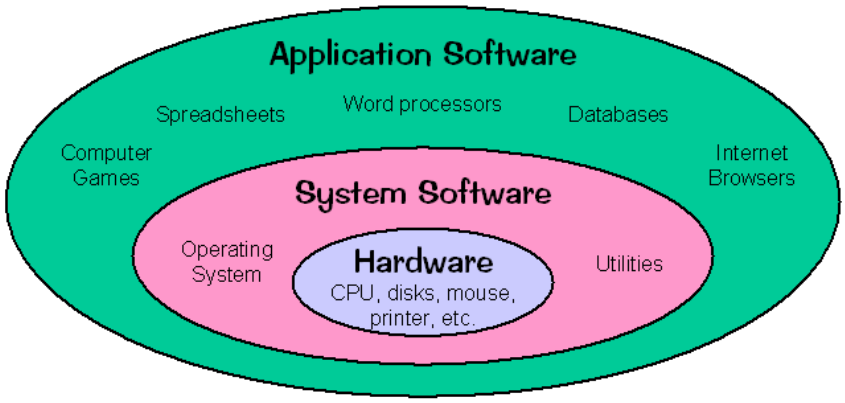


Fig. *OS Acts as Control Program*



OBJECTIVES OF O.S (GOALS)

The OS has 3 main objectives.

- **Convenience.** An OS makes a computer more convenient to the user for using. (Easy-to-use commands, graphical user interface(GUI))
- **Efficiency.** An OS allows the computer system resources to be used in an efficient manner, to ensure good resource utilization efficiency, and provide appropriate corrective actions when it becomes low.
- **Ability to evolve.** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without interfering with service.

Operating system performs the following functions:

1. Booting

Booting is a process of starting the computer operating system starts the computer to work. It checks the computer and makes it ready to work.

2. Memory Management

It is also an important function of operating system. The memory cannot be managed without operating system. Different programs and data execute in memory at one time. if there is no operating system, the programs may mix with each other. The system will not work properly.

3. Loading and Execution

A program is loaded in the memory before it can be executed. Operating system provides the facility to load programs in memory easily and then execute it.

4. Data security

Data is an important part of computer system. The operating system protects the data stored on the computer from illegal use, modification or deletion.

5. Disk Management

Operating system manages the disk space. It manages the stored files and folders in a proper way.

6. Process Management

CPU can perform one task at one time. if there are many tasks, operating system decides which task should get the CPU.

7. Device Controlling

operating system also controls all devices attached to computer. The hardware devices are controlled with the help of small software called device drivers..

8. Providing interface

It is used in order that user interface acts with a computer mutually. User interface controls how you input data and instruction and how information is displayed on screen. The operating system offers two types of the interface to the user:

1. Graphical-line interface: It interacts with of visual environment to communicate with the computer. It uses windows, icons, menus and other graphical objects to issues commands.
2. Command-line interface:it provides an interface to communicate with the computer by typing commands.

Computer System Architecture

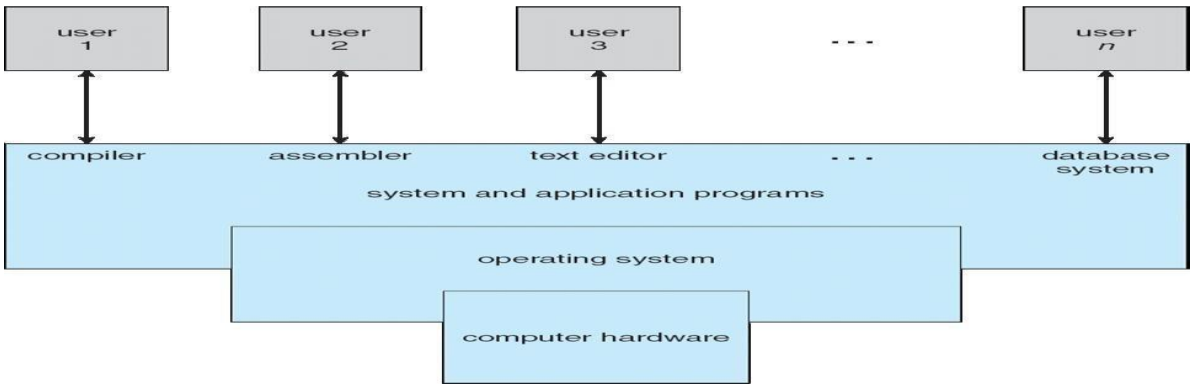
Computer system can be divided into four components Hardware – provides basic computing resources

- 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 Four

Components of a Computer System



Computer architecture means construction/design of a computer. A computer system may be organized in different ways. Some computer systems have single processor and others have multiprocessors. So based on the processors used in computer systems, they are categorized into the following systems.

1. Single-processor system
2. Multiprocessor system
3. Clustered Systems:

1. Single-Processor Systems:

Some computers use only one processor such as microcomputers (or personal computers PCs). On a single-processor system, there is only one CPU that performs all the activities in the computer system. However, most of these systems have other special purpose processors, such as I/O processors that move data quickly among different components of the computers. These processors execute only a limited system programs and do not run the user program. Sometimes

they are managed by the operating system. Similarly, PCs contain a special purpose microprocessor in the keyboard, which converts the keystrokes into computer codes to be sent to the CPU. The use of special purpose microprocessors is common in microcomputer. But it does not mean that this system is multiprocessor. A system that has only one general-purpose CPU, is considered as single- processor system.

2. Multiprocessor Systems:

In multiprocessor system, two or more processors work together. In this system, multiple programs (more than one program) are executed on different processors at the same time. This type of processing is known as multiprocessing. Some operating systems have features of multiprocessing. UNIX is an example of multiprocessing operating system. Some versions of Microsoft Windows also support multiprocessing.

Multiprocessor system is also known as parallel system. Mostly the processors of multiprocessor system share the common system bus, clock, memory and peripheral devices. This system is very fast in data processing.

Types of Multiprocessor Systems:

The multiprocessor systems are further divided into two types; (i). Asymmetric multiprocessing system
(ii). Symmetric multiprocessing system

(i) Asymmetric Multiprocessing System(AMS):

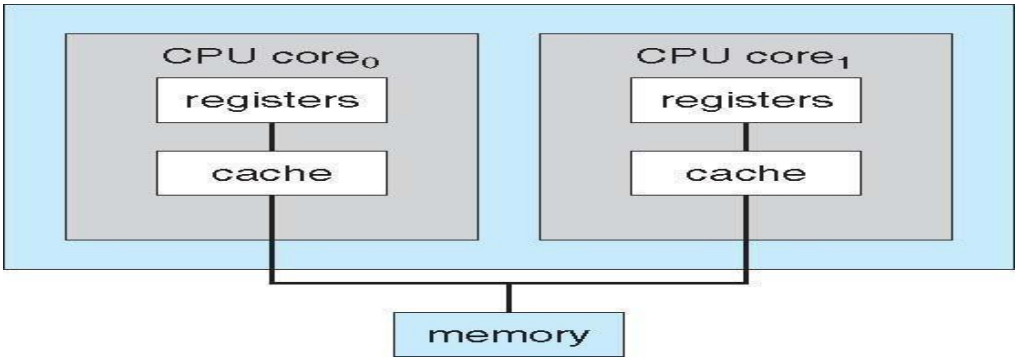
The multiprocessing system, in which each processor is assigned a specific task, is known as Asymmetric Multiprocessing System. For example, one processor is dedicated for handling user's requests, one processor is dedicated for running application program, and one processor is dedicated for running image processing and so on. In this system, one processor works as master processor, while other processors work as slave processors. The master processor controls the operations of system. It also schedules and distributes tasks among the slave processors. The slave processors perform the predefined tasks.

(ii) Symmetric Multiprocessing System(SMP):

The multiprocessing system, in which multiple processors work together on the same task, is known as Symmetric Multiprocessing System. In this system, each processor can perform all types of tasks. All processors are treated equally and no master-slave relationship exists between the processors.

For example, different processors in the system can communicate with each other. Similarly, an I/O can be processed on any processor. However, I/O must be controlled to ensure that the data reaches the appropriate processor. Because all the processors share the same memory, so the input data given to the processors and their results must be separately controlled. Today all modern operating systems including Windows and Linux provide support for SMP.

It must be noted that in the same computer system, the asymmetric multiprocessing and symmetric multiprocessing technique can be used through different operating systems.



A Dual-Core Design

3. Clustered Systems:

Clustered system is another form of multiprocessor system. This system also contains multiple processors but it differs from multiprocessor system. The clustered system consists of two or more individual systems that are coupled together. In clustered system, individual systems (or clustered computers) share the same storage and are linked together ,via Local Area Network (LAN).

A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the other nodes over the LAN. If the monitored machine fails due to some technical fault (or due to other reason), the monitoring machine can take ownership of its storage. The monitoring machine can also restart the applications that were running on the failed machine. The users of the applications see only an interruption of service.

Types of Clustered Systems:

Like multiprocessor systems, clustered system can also be of two types (i). Asymmetric Clustered System
(ii). Symmetric Clustered System

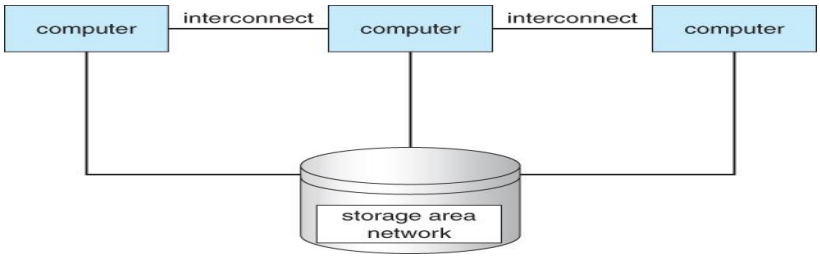
(i). Asymmetric Clustered System:

In asymmetric clustered system, one machine is in hot-standby mode while the other

machine is running the application. The hot-standby host machine does nothing. It only monitors the active server. If the server fails, the hot-standby machine becomes the active server.

(ii). Symmetric Clustered System:

In symmetric clustered system, multiple hosts (machines) run the applications. They also monitor each other. This mode is more efficient than asymmetric system, because it uses all the available hardware. This mode is used only if more than one application be available to run.



Operating System – Structure

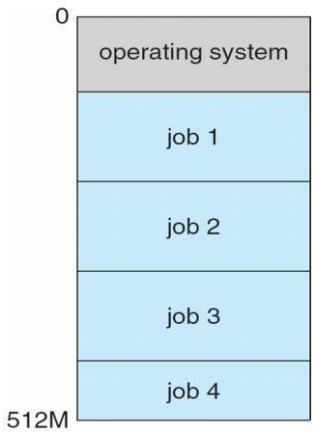
Operating System Structure

- **Multiprogramming** needed for efficiency
- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to
- Execute A subset of total jobs in system is kept in memory

Multiprogramming

When two or more programs are residing in memory at the same time, then sharing the processor is referred to the multiprogramming. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

Following figure shows the memory layout for a multiprogramming system.



Operating system does the following activities related to multiprogramming.

- 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 job in the memory.
- Multiprogramming operating system monitors the state of all active programs and system resources using memory management programs to ensures that the CPU is never idle unless there are no jobs

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

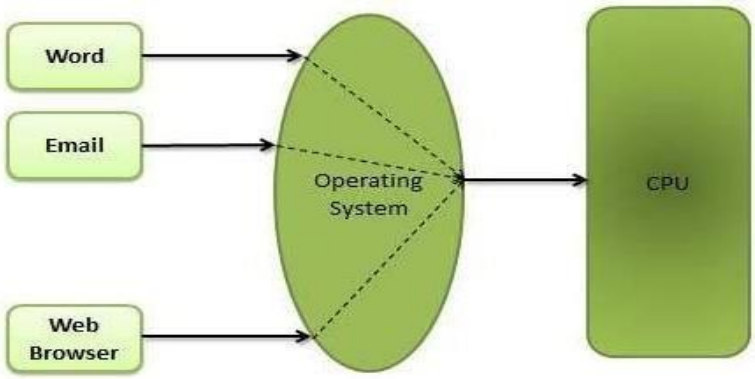
- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

2) Multitasking

Multitasking

Multitasking refers to term where multiple jobs are executed by the CPU simultaneously by switching between them.Switches occur so frequently that the users may interact with each program while it is running. Operating system does the following activities related to multitasking.

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- Operating System handles multitasking in the way that it can handle multiple operations / executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- 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 very short time before it either finishes or needs to perform I/O.

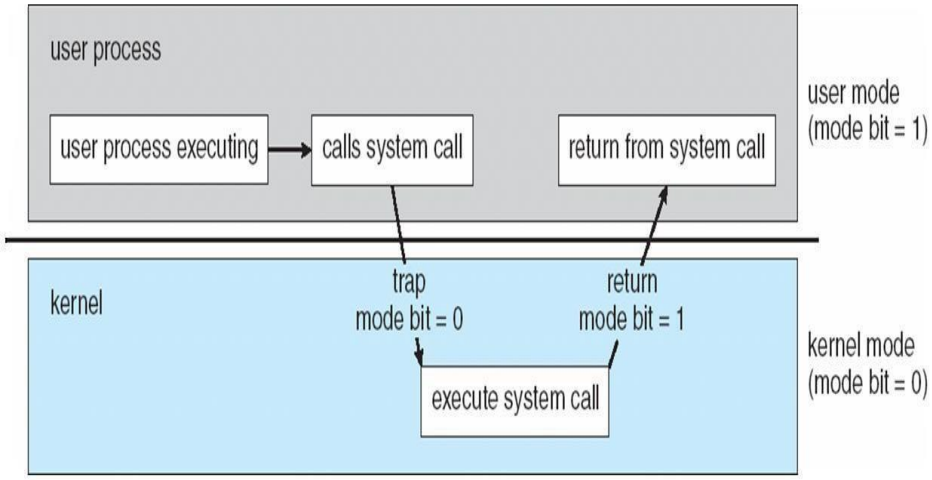
- Since interactive I/O typically runs at people speeds, it may take a long time to completed. During this time a CPU can be utilized by another process.
- Operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Operating-system Operations

1) Dual-Mode Operation·

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least we need two separate modes of operation.user mode and kernel mode.
A bit, called the mode bit is added to the hardware of the computer 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 on behalf of the operating system and one that is executed on behalf of the user, When



the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a.. system call), it must transition from user to kernel mode to fulfill the request.

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 state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users-and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. the hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/0 control timer management and interrupt management.

Timer

We must ensure that the operating system maintains control over the CPU. We must prevent a user program from getting stuck in an infinite loop or not calling system services and never returning control to the operating system. To accomplish this goal, we can use a **timer**. A timer can be set to interrupt the computer after a specified period.

Before turning over control to the user, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time. Clearly, instructions that modify the content of the timer are privileged.

Thus, we can use the timer to prevent a user program from running too long.

EVOLUTION OF OPERATING SYSTEMS

Operating system and computer architecture have had a great deal of influence on each other. Operating systems were developed mainly to facilitate the use of the hardware and to bring it to the best advantage. Here we will briefly make a sketch of the evolutionary path of OS development.

Serial Processing

Before 1950's the programmers directly interact with computer hardware, there was no OS at that time. If the programmer want to execute the program on those days, he has to follow some serial steps:

- Type the program on punched card.
- Convert the punched card to card reader.
- Submit to the computing machine, if any error in the program, the error condition was indicated by lights.
- The programmer examine the registers and main memory to identify the cause of error.
- Take the output on the printers.
- Then the programmer is ready for the next program.

This type of processing is difficult for users, it takes much time and next program should wait for the completion of previous one. The programs are submitted to the machine one after the other. So, this method is called as "Serial processing".

Batch Processing

In olden days(before 1960's), it is difficult to execute a program using computer. Because the computer is located in different rooms, one room for card reader and one for executing the program and another room for printing the output. The user or machine operator, running between these three rooms to complete a job. This problem was solved by batch processing system.

In batch processing technique similar type of jobs batch together and execute at a time. The operator carries the group of jobs at a time from one room to another. Therefore the programmer need not run between these three rooms several times.

The batch processing had an advantage .In that for one batch, the compiler, assembler, the loader etc had to be loaded only once, thus reducing the setup time to some extent. For example, FORTRAN programs were grouped together as one batch say batch 1, the PASCAL programs into another batch say Batch 2, the COBOL programs into another batch say Batch 3, and so on. Now the operator can arrange for the execution of these source programs which has been batched together one by one. After the execution of batch1 was over, the operator would load the compiler, assembler and loader, etc for the batch 2 and so on.

Setup time for batch 1	Runtime for batch 1	Setup time for batch 2	Runtime for batch 2	...	...	...

Fig. *Batch Processing*

The main advantage of batch processing is setup time will be reduced to a large extent, but the disadvantage is that the CPU is idle for the time in between two batches.

If the programs were not batched up together, the set up time would be much more higher.

Setup time for program 1	Runtime for program 1	Setup time for program 2	Runtime for program 2	...	...	...

C

Multiprogramming

Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs. Instead the processor executes part of one program, then part of another, and so on. But to the user it appears that all programs are executing at the same time.

In multiprogramming, number of processes are reside in main memory at a time. The OS picks and begins to execute one of the jobs in the main memory. For example, consider the main memory consisting of 5 jobs at a time, the CPU executes one by one.

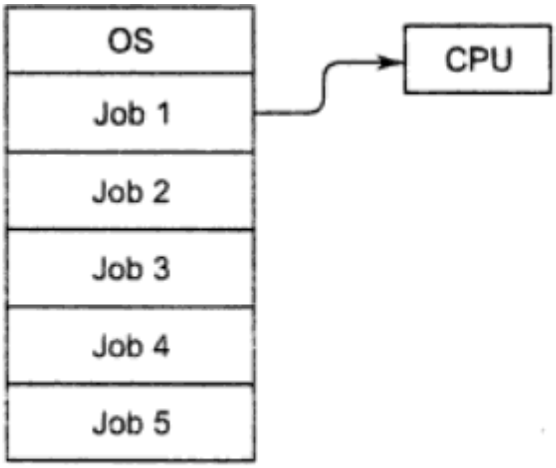


Fig. *Multiprogramming*

In non-multiprogramming system, the CPU can execute only one program at a time, if the running program waiting for any I/O device, the CPU becomes idle, so it will effect on the performance of the CPU.

But in multiprogramming environment, any I/O wait happened in a process, then the CPU switches from that job to another job in the job pool. If enough jobs could be held in main memory at once, the CPU is not idle at any time.

For Example: The idea is common in other life situations. The doctor does not have only one patient at a time, number of patients reside in the hospital under treatment. If the doctor has enough patients a doctor never needs to be idle.

Distributed Systems

A recent trend in computer system is to distribute computation among several processors. The processors in distribute system may vary in size and function, and referred by a number of different names such as sites, nodes, computers and so on depending on the context.

A distributed system is basically a collection of autonomous (independent by function) computer systems which co-operate with one another through their hardware and software interconnections.

In distributed systems, the processors cannot share memory or time, each processor has its own local memory. The processors communicate with one another through various communication lines such as high speed buses .These systems are also called as *"Loosely Coupled systems"*.

Distributed system = Network + Transparency(Invisible)

Advantages

1. **Resource sharing:** If a number of sites connected by a high speed communication lines, it is possible to share the resources from one site to another site.

   For example, S_1 and S_2 are two sites, these are connected by some communication lines, the site S_1 having the printer, but S_2 does not having the printer. Then the system can use the printer at S_1 without moving from S_2 to S_1.Therefore resource sharing is possible in distributed systems.
2. **Computation speedup:** A big computation is partitioned into number of partitions, these sub-partitions run concurrently in distributed systems.

   For example, site S_1 need to execute a big computation, this computation is divided into sub computations and these are executed by some other machines in different sites.
3. **Reliability:** If a resource or a system failed in one site due to technical problems. We can use other systems or other resources in some other sites.
4. **Communication:** Distributed systems provides communication which is not at all possible, that much in a centralized system. For Example, E-mail

Time Sharing Systems

Multiprogramming features were superimposed on batch processing to ensure good utilization of CPU but from the point of view of a user the service was poor as the response time, *i.e*, the time elapsed between submitting a job and getting the results was unacceptably high. Development of interactive terminals changed the scenario. Computation became an on-line activity. A user could provide inputs to a computation from a terminal and could also examine the output of the computation on the same terminal. Hence the response time needed to be drastically reduced. This was achieved by storing programs of several users in memory and providing each user a slice of time on CPU to process his/her program.

Time sharing or multitasking is a logical extension of multiprogramming. In time sharing environment, a number of jobs are loaded on to the memory and a number of users are communicating with the computer through different terminals. The OS allocates a fixed time interval (TIME SLICE) to each program in memory. Thus each program in memory is executed for a fixed interval of time.

As soon as the time allotted for a particular program is completed, the CPU starts executing the next program. This process is continued till all the programs in the memory are executed. A program may need number of time slices for its complete execution. Although the computer system is executing one job at a time, due to the speed of the CPU, every user on a terminal has the feeling that his program that is being executed continuously, because, after every time slice, the user gets a response from the computer. The user on the terminal is communicating with his running program, and is able to debug and experiment with his program.

Thus, the OS for a time sharing computer system has all the capabilities of a multiprogramming OS, but along with an additional capacity of allocating a fixed time slice of CPU to each program.

- Main advantage of time sharing system is efficient CPU utilization.
- The user can interact with the job while it is executing, but it is not possible in batch systems.

Personal-Computer Systems(PCs)

A personal computer (PC) is a small, relatively inexpensive computer designed for an individual user. In price, personal computers range anywhere from a few hundred dollars to thousands of dollars. All are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip.

At home, the most popular use for personal computers is for playing games. Businesses use personal computers for word processing, accounting, desktop publishing, and for running spreadsheet and database management applications.

Parallel Systems

Almost all the systems are uni-processor systems *i.e.*, they have only one CPU. Systems in which there are more than one CPU is called as Multi-processor systems. These systems have been developed to enhance the computing power of a computing system, and the features of this system is that, they share the memory, bus and the peripheral devices. These systems are referred as *"Tightly coupled systems"*. A system consisting of more than one processor and it is a tightly coupled, then the system is called "Parallel system".

In parallel systems number of processors executing their jobs in parallel (simultaneous process). Multi-processor systems are divided into following categories:

- Symmetric
- Asymmetric

In symmetric multi-processing, each processor runs a shared copy of operating system . The processors can communicate with each other and execute these copies concurrently. Thus, in a symmetric system, all the processors share an equal amount of load .Encore's version of UNIX for the Multimax computer is an example of symmetric multiprocessing. In this system various processors execute copies of UNIX operating system, thereby executing M processes if there are M processors.

Asymmetric multi-processing is based on the principle of master-slave relationship. In this system, one of the processors runs the operating system and that processor is called the master processor. Other processors run user processes and are known as slave processors. In other words, the master processor controls, schedules and allocates the task to the slave processors. Asymmetric multi-processing is more common in extremely large systems, where one of the time consuming tasks is processing I/O requests. In the asymmetric systems the processors do not share the equal load.

Advantages:

1. It results in saving money compared to the stand alone systems, since CPU'S can share memory, bus and peripherals.
2. Throughput can be increased
3. They increase the reliability.

Since there are more than one CPU, the failure of one or more of the CPU does not halt the entire system, but only slows down the work. For example, if there are five processors, all the five working together gives full efficiency. If two CPU's fail, then the system still works but only at 60% efficiency. This indicates increased aspect of reliability compared to stand alone systems.

Special purpose systems

a) Real-Time EmbeddedSystems

These devices are found everywhere, from car engines and manufacturing robots to DVDs and microwave ovens. They tend to have very specific tasks.

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

b) Multimedia Systems

Most operating systems are designed to handle conventional data such as text files, programs, word-processing documents, and spreadsheets. However, a recent trend in technology is the incorporation of multimedia data into computer 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 in popular use today. These include audio files such as MP3, DVD movies, video conferencing, and short video clips of movie previews or news stories downloaded over the Internet. Multimedia applications may also include live webcasts (broadcasting over the World Wide Web)

c) Hand held Systems

Handheld Systems include personal digital assistants (PDAs, cellular telephones. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. For example, a PDA is typically about 5 inches in height and 3 inches in width, and it weighs less than one-half pound. Because of their size, most handheld devices have small amounts of memory, slow processors, and small display screens.

REAL-TIME OS

In a time shared computer system, generally the computer response time is of the order of 0.5 to 2 seconds, which means a user will get computers attention after this much of time. Longer response times may be irritating but not hazardous.

However a real-time OS is needed for the computer systems controlling a process or a real time situation, such as a machine or a satellite. In this case two important points to be noticed are:

- The OS should provide for interactive processing.
- The response time should be very small.

The sensors bring in the data from a device, the OS instructs the computer to analyze the data and send appropriate signals back to the device. Any delay on the part of the computer system or the OS can be catastrophic. Thus, the real-time OS have to work strict time limits and have to be quick. Apart from this, these systems must be highly reliable to avoid failure of the system being controlled.

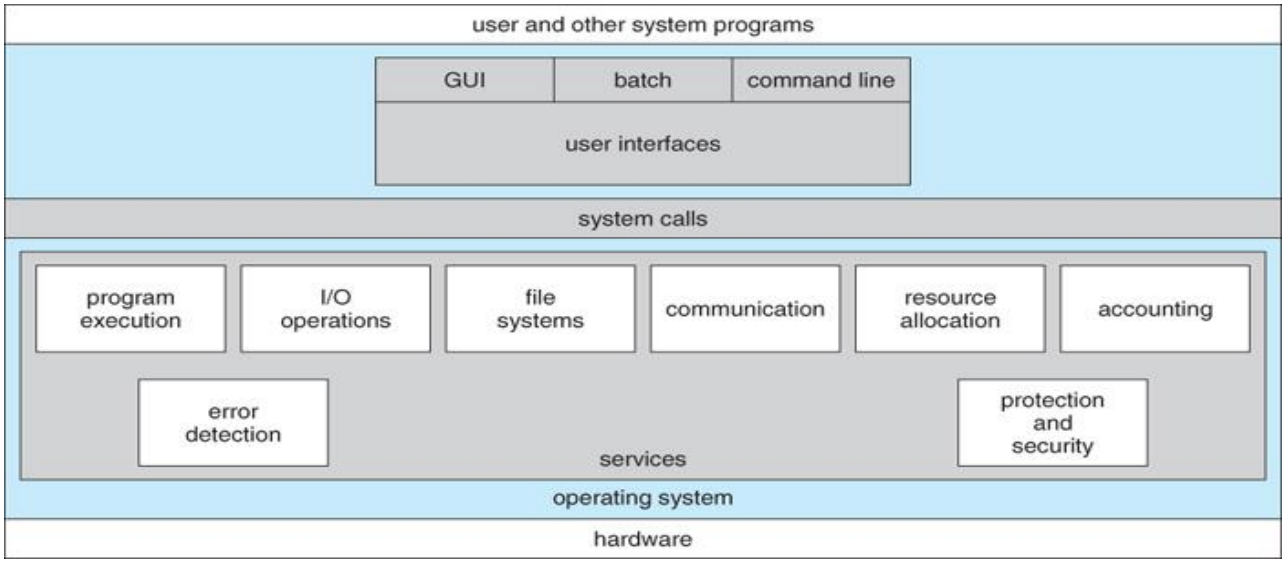
Here the main job of OS is instant handling of the signals or interrupts sent by the device which is being controlled by the computer system.

Real-time systems are systems that have in-built characteristics as supplying immediate response. A primary objective of the real-time system is to provide quick response time. User convenience and resource utilization are of secondary concern to real-time systems.

Real time System is of two types:

- **Hard real-time**
  - Guarantees that critical tasks complete within time.
  - All the delays in the system are bounded.
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- **Soft real-time**
  - Critical time tasks gets priority over other tasks, and retails that priority until it completes.
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Operating System Services



- One set of operating-system services provides functions that are helpful to the user

Communications – Processes may exchange information, on the same computer or between computers over a network Communications may be via shared memory or through message passing (packets moved by the OS)

- Error detection – OS needs to be constantly aware of possible errors May occur in the CPU and memory hardware, in I/O devices, in user program For each type of error, OS should take the appropriate action to ensure correct and consistent computing Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource Sharing

- **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code

Accounting - To keep track of which users use how much and what kinds of computer resources

- **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

Protection involves ensuring that all access to system resources is controlled

- **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
- If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter allows direct command entry

Sometimes implemented in kernel, sometimes by systems program
sometimes multiple flavors implemented – shells
Primarily fetches a command from user and executes it

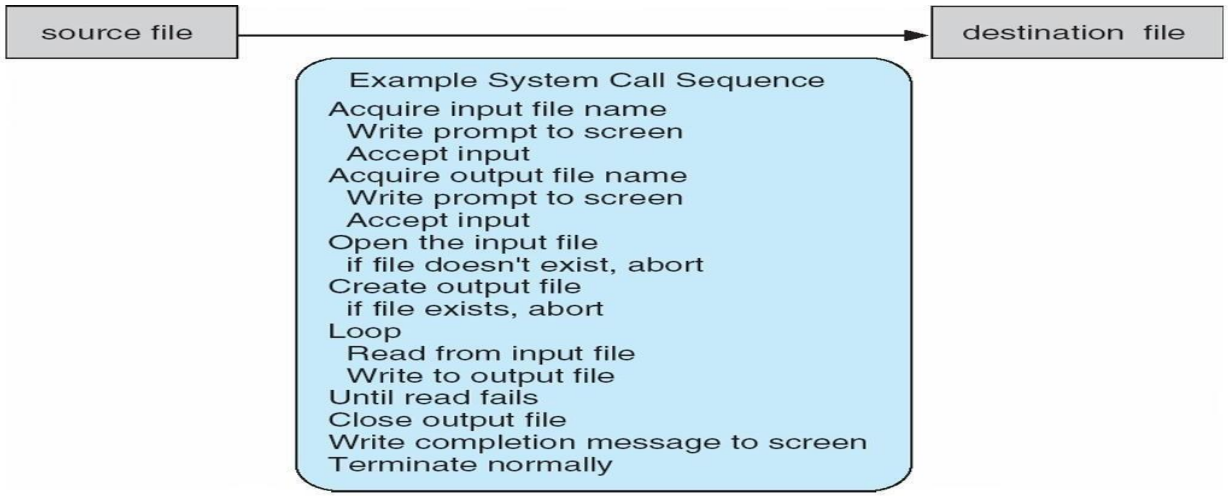
User Operating System Interface - GUI

- User-friendly desktop metaphor interface
- Usually mouse, keyboard, and monitor Icons
- represent files, programs, actions, etc
- Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
- Invented at Xerox PARC
- Many systems now include both CLI and GUI
- interfaces Microsoft Windows is GUI with CLI
- "command" shell
- Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usenThree most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

Example of System Calls



Example of Standard API

Consider the ReadFile() function in the Win32 API—a function for reading from a file

A description of the parameters passed to ReadFile() HANDLE file—the file to be read

LPVOID buffer—a buffer where the data will be read into and written from DWORD bytesToRead—the number of bytes to be read into the buffer LPDWORD bytesRead—the number of bytes read during the last read LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

Typically, a number associated with each system call

System-call interface maintains a table indexed according to these Numbers

The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values

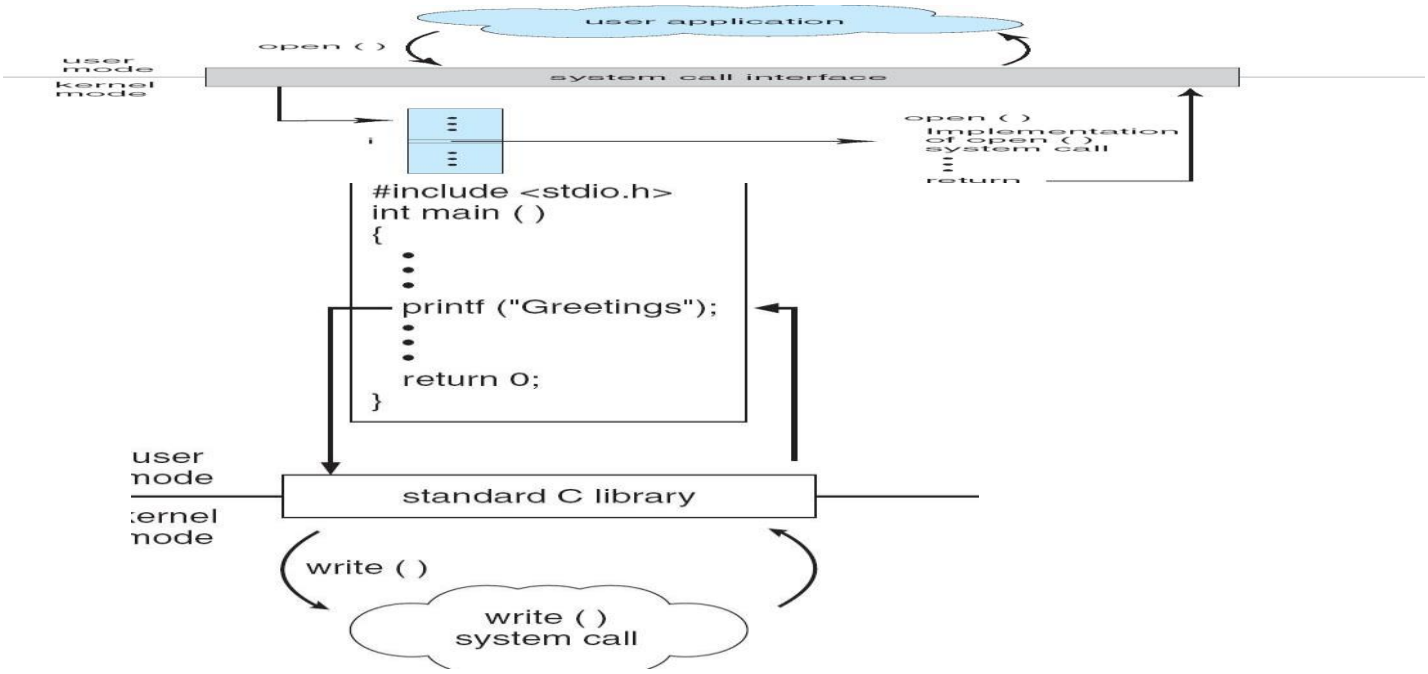
The caller need know nothing about how the system call is implemented Just needs to obey API and understand what OS will

do as a result call Most details of OS interface hidden from programmer by API

Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship

Standard C Library Example



System Call Parameter Passing

- Often, more information is required than simply identity of desired system
- call Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the
- OS Simplest: pass the parameters in *registers*

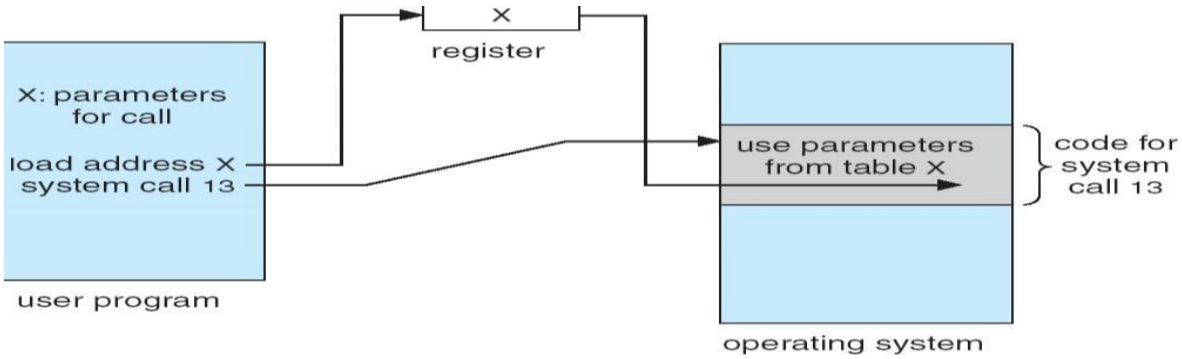
In some cases, may be more parameters than registers

- Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register

 This approach taken by Linux and Solaris

- Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls

1. Process control
2. File management
3. Device management
4. Information maintenance
5. Communications

Process control

A running needs to halt its execution either normally or abnormally.

If a system call is made to terminate the running program, a dump of memory is sometimes taken and an error message generated which can be diagnosed by a debugger

- 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

OS provides an API to make these system calls for managing files

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

Device management

Process requires several resources to execute, if these resources are available, they will be granted and control retuned to user process. Some are physical such as video card and other such as file. User program request the device and release when finished

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

Information maintenance

System calls exist purely for transferring information between the user program and OS. It can 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.

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

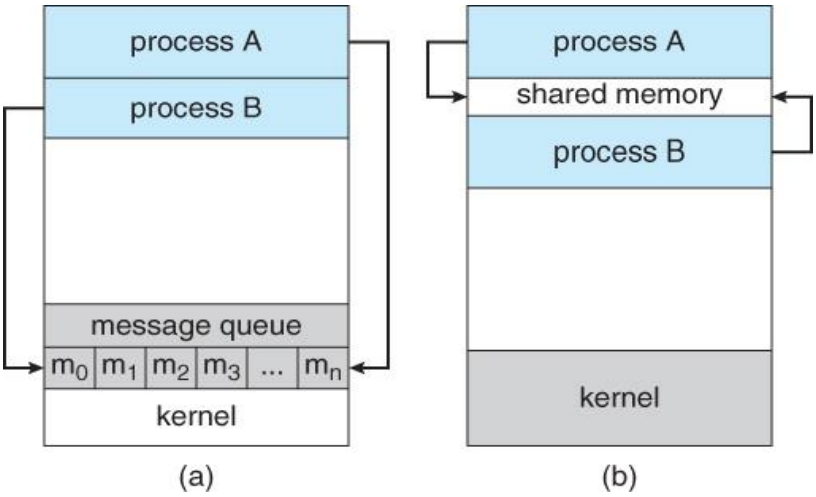
Communications

Two common models of communication

Message-passing model, information is exchanged through an inter process-communication facility provided by the OS.

Shared-memory model, processes use map memory system calls to gain access to regions of memory owned by other processes.

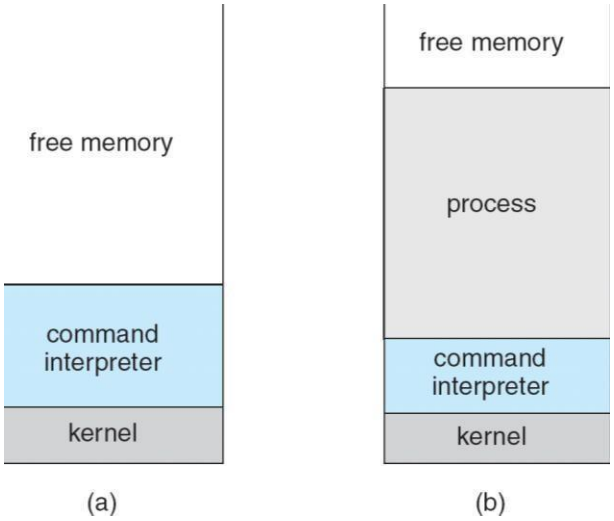
o create, delete communication connection
o send, receive messages
o transfer status information
o attach and detach remote devices



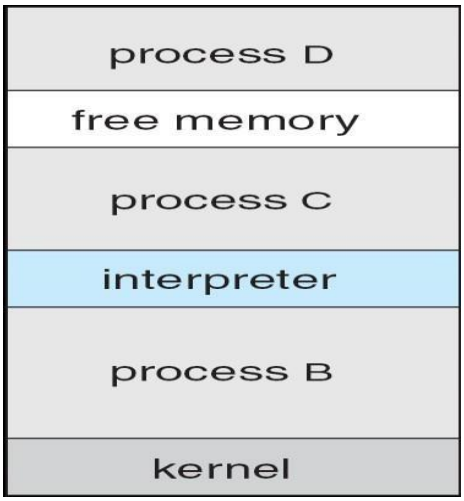
Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

MS-DOS execution



(a) At system startup (b) running a program FreeBSD Running Multiple Programs



System Programs

System programs provide a convenient environment for program development and execution. The can be divided into:

- File manipulation
- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Application programs

Most users' view of the operation system is defined by system programs, not the actual system calls provide a convenient environment for program development and execution

Some of them are simply user interfaces to system calls; others are considerably more complex

File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

- Status information

Some ask the system for info - date, time, amount of available memory, disk space, number of users

Others provide detailed performance, logging, and debugging information

Typically, these programs format and print the output to the terminal or other output devices

Some systems implement a registry - used to store and retrieve configuration information

- File modification

Text editors to create and modify files

Special commands to search contents of files or perform transformations of the text

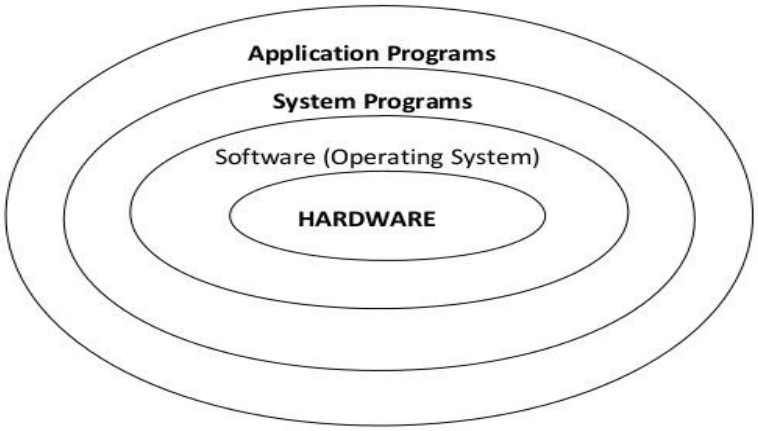
Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided

- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems

- Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

STRUCTURE OF OPERATING SYSTEM:



17

Operating System Design and Implementation

Design and Implementation of OS not "solvable", but some approaches have proven successful

Internal structure of different Operating Systems can vary widely

Start by defining goals and specifications Affected by

choice of hardware, type of system *User* goals and

System goals

User goals – operating system should be convenient 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

Important principle to separate

Policy: What will be done?

Mechanism: How to do it?

Mechanisms determine how to do something, policies decide what will be done

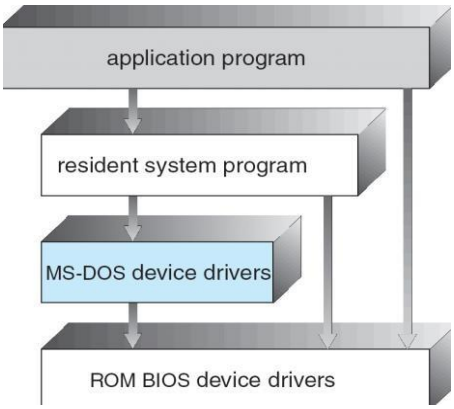
The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Simple Structure

- MS-DOS – written to provide the most functionality in the least space Not divided into
- modules

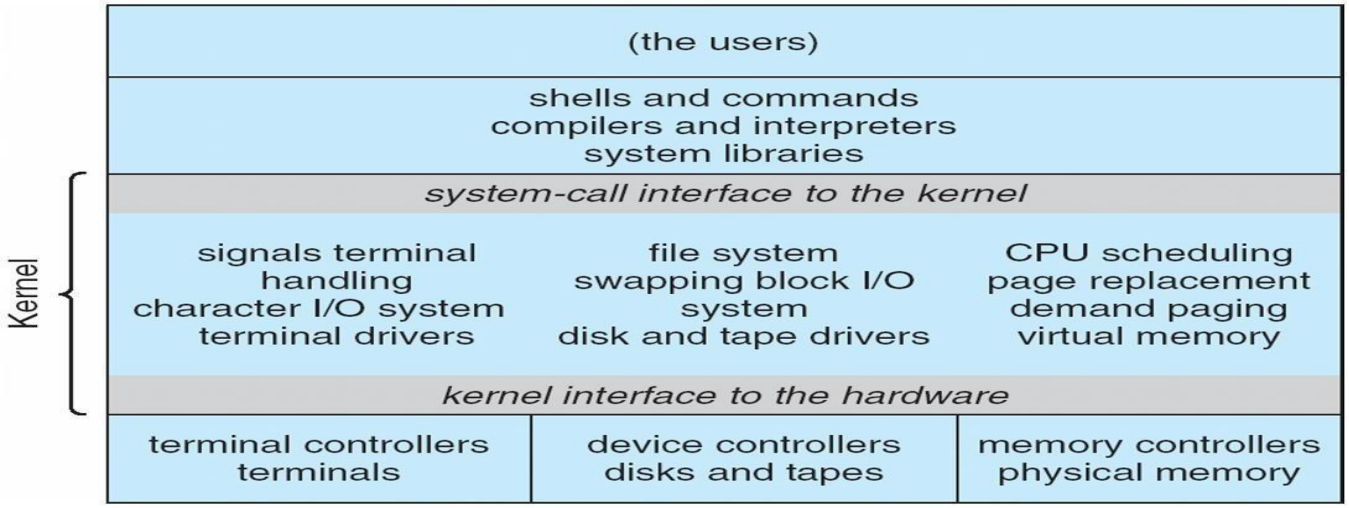
- Although MS-DOS has some structure, its interfaces and levels of Functionality are not well separated

MS-DOS Layer Structure



- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

Traditional UNIX System Structure



UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.

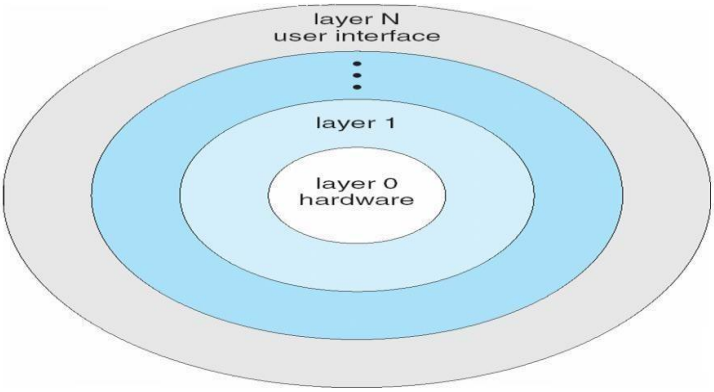
The UNIX OS consists of two separable parts

- Systems programs
- The kernel

Consists of everything below the system-call interface and above the physical hardware
Provides the file system, CPU scheduling, memory management, and other operating-system

functions; a large number of functions for one level

Layered Operating System



Micro kernel System Structure

Moves as much from the kernel into "*user*" space

Communication takes place between user modules using message passing

Benefits:

Easier to extend a microkernel

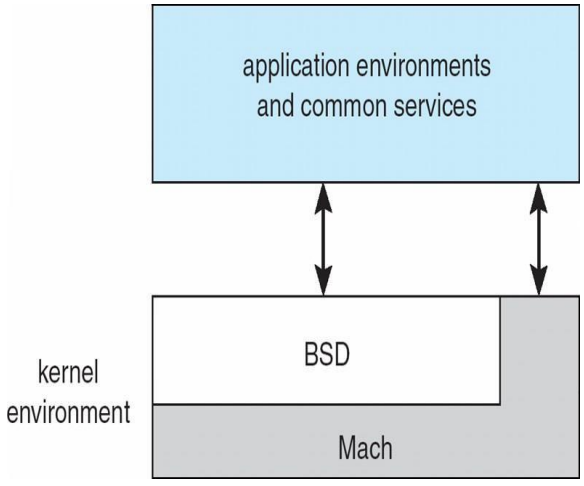
Easier to port the operating system to new architectures More reliable (less code is running in kernel mode)

More secure

Detriments:

Performance overhead of user space to kernel space communication

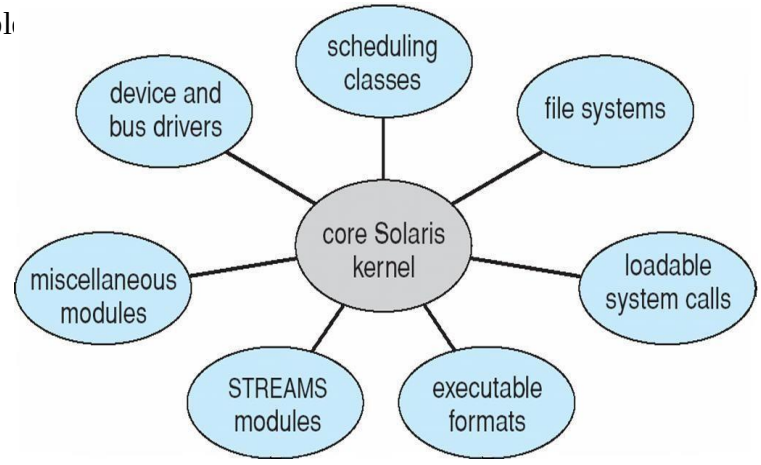
MacOS X Structure



Modules

Most modern operating systems implement kernel modules
Uses object-oriented approach
Each core component is separate
Each talks to the others over known interfaces
Each is loadable as needed within the kernel
Overall, similar to layers but with more flexibl

Solaris Modular Approach

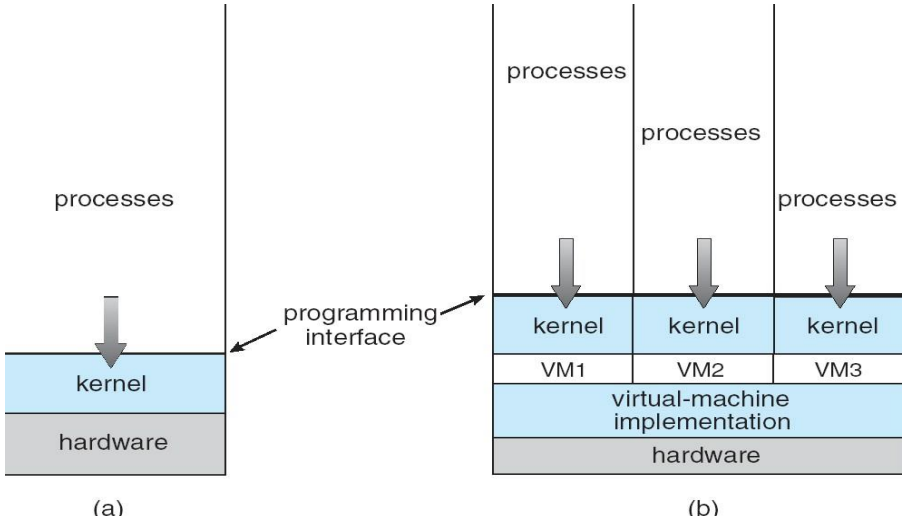


Virtual Machines

A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
A virtual machine provides an interface *identical* to the underlying bare hardware
The operating system host creates the illusion that a process has its own processor and (virtual memory)
Each guest provided with a (virtual) copy of underlying computer

Virtual Machines History and Benefits

First appeared commercially in IBM mainframes in 1972
Fundamentally, multiple execution environments (different operating systems) can share the same hardware
Protect from each other
Some sharing of file can be permitted, controlled
Commutate with each other, other physical systems via networking
Useful for development, testing
Consolidation of many low-resource use systems onto fewer busier systems
"Open Virtual Machine Format", standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms



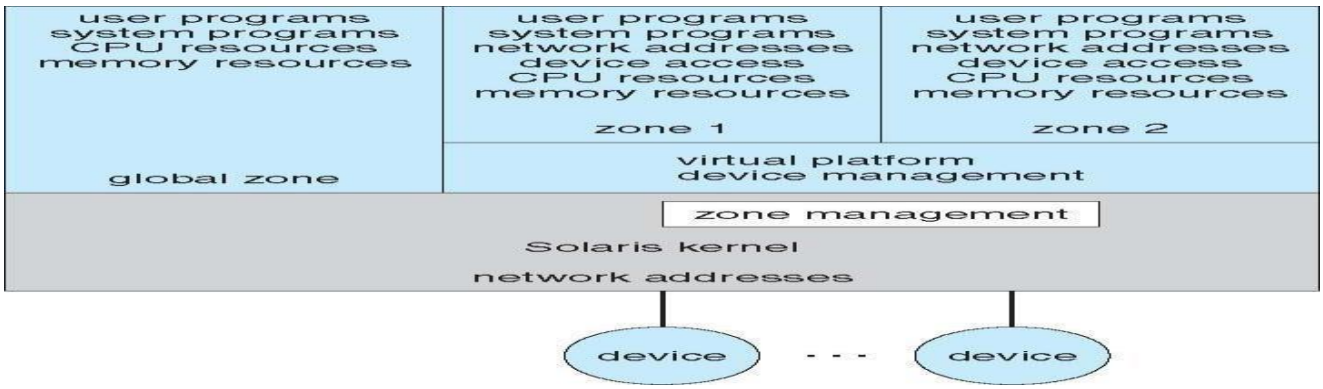
Para-virtualization

Presents guest with system similar but not identical to hardware

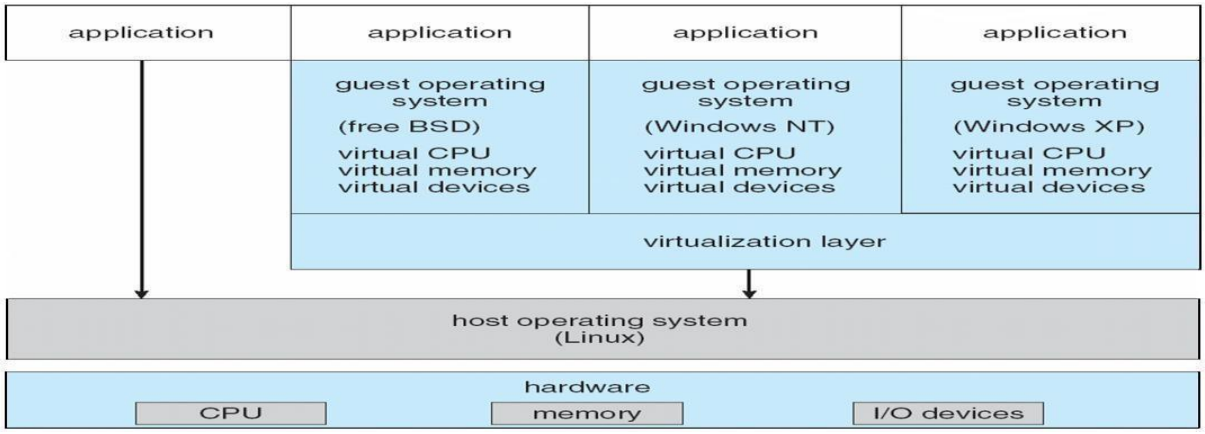
Guest must be modified to run on par virtualized hardware

Guest can be an OS, or in the case of Solaris 10 applications running in containers

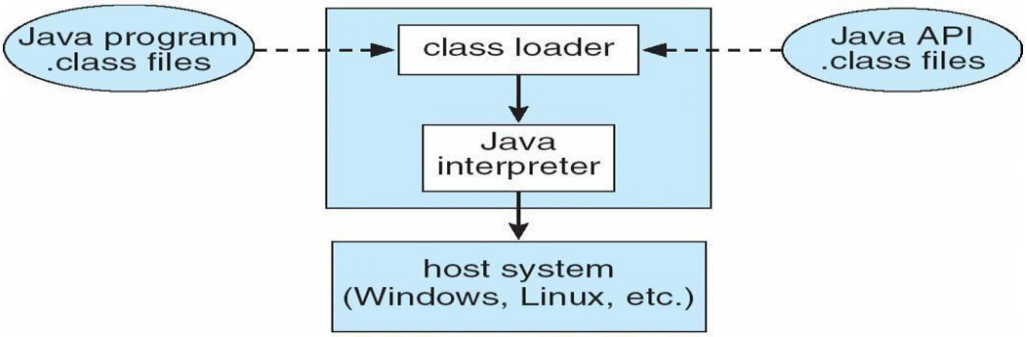
Solaris 10 with Two Containers



VMware Architecture



The Java Virtual Machine



Operating-System Debugging

Debugging is finding and fixing errors, or bugs
generate log files containing error information
Failure of an application can generate core dump file capturing memory of the process
Operating system failure can generate crash dump file containing kernel memory Beyond crashes, performance tuning can optimize system performance
Kernighan's Law: "Debugging is twice as hard as writing the code in the rst place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
Probes fire when code is executed, capturing state data and sending it to consumers of those probes

Process

A process is a program at the time of execution.

Differences between Process and Program

Process	Program
Process is a dynamic object	Program is a static object
Process is sequence of instruction execution	Program is a sequence of instructions
Process loaded in to main memory	Program loaded into secondary storage devices
Time span of process is limited	Time span of program is unlimited
Process is a active entity	Program is a passive entity

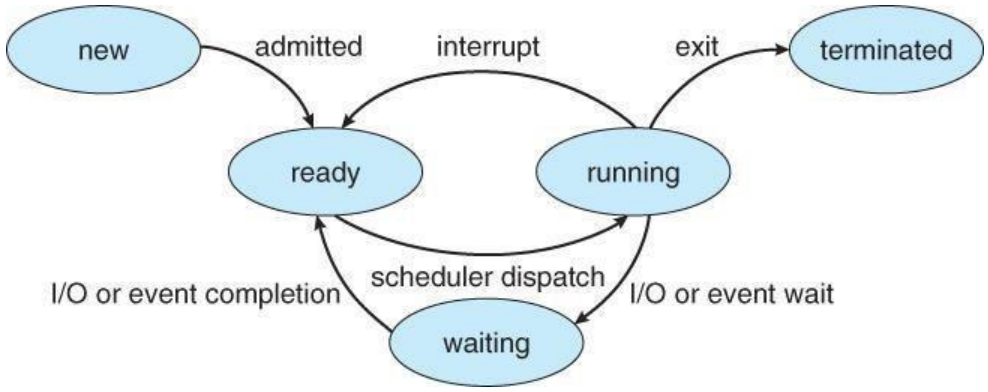
Process States

When a process executed, it changes the state, generally the state of process is determined by the current activity of the process. Each process may be in one of the following states:

1. New : The process is beingcreated.
2. Running : The process is beingexecuted.
3. Waiting : The process is waiting for some event tooccur.
4. Ready : The process is waiting to be assigned to a processor.
5. Terminated : The Process has finishedexecution.

Only one process can be running in any processor at any time, But many process may be in ready and waiting states. The ready processes are loaded into a “ready queue”.

Diagram of process state



a) **New ->Ready** : OS creates process and prepares the process to be executed,thenOSmoved the process into readyqueue.

b) **Ready->Running** : OS selects one of the Jobs from ready Queue and move themfrom ready to Running.

c) **Running->Terminated** : When the Execution of a process has Completed, OSterminatesthatprocess from running state. Sometimes OS terminates the process for someother reasons including Time exceeded, memory unavailable, access violation, protection Error, I/O failure and soon.

d) **Running->Ready** : When the time slot of the processor expired (or) If the processorreceivedanyinterrupt signal, the OS shifted Running -> ReadyState.

e) **Running -> Waiting** : A process is put into the waiting state, if the process need an event occur (or) an I/O Devicerequire.

f) **Waiting->Ready** : A process in the waiting state is moved to ready state whenthe eventforwhichit has beenCompleted.

Process Control Block:

Each process is represented in the operating System by a Process Control Block.

It is also called Task Control Block. It contains many pieces of information associated with a specific Process.

Process State
Program Counter
CPU Registers
CPU Scheduling Information
Memory – Management Information
Accounting Information
I/O Status Information

Process Control Block

1. **ProcessState** : The State may be new, ready, running, and waiting,Terminated…
2. **ProgramCounter** : indicates the Address of the next Instruction to be executed.
3. **CPUregisters** : registers include accumulators, stack pointers, General purpose Registers….

4. **CPU-SchedulingInfo** : includes a process pointer, pointers to schedulingQueues,other scheduling parametersetc.
5. **Memory management Info**: includes page tables, segmentation tables, value of base and limit registers.
6. **AccountingInformation:** includes amount of CPU used, time limits, Jobs(or)Process numbers.
7. **I/O StatusInformation**: Includes the list of I/O Devices Allocated to theprocesses, list of open files.

Threads:

A process is divide into number of light weight process, each light weight process is said to be a Thread. The Thread has a program counter (Keeps track of which instruction to execute next), registers (holds its current working variables), stack (execution History).

Thread States:

1. bornState : A thread is justcreated.
2. readystate : The thread is waiting forCPU.
3. running : System assigns the processor to thethread.
4. sleep : A sleeping thread becomes ready after the designated sleep timeexpires.
5. dead : The Execution of the threadfinished.

Eg: Word processor.

Typing, Formatting, Spell check, saving are threads.

Differences between Process and Thread

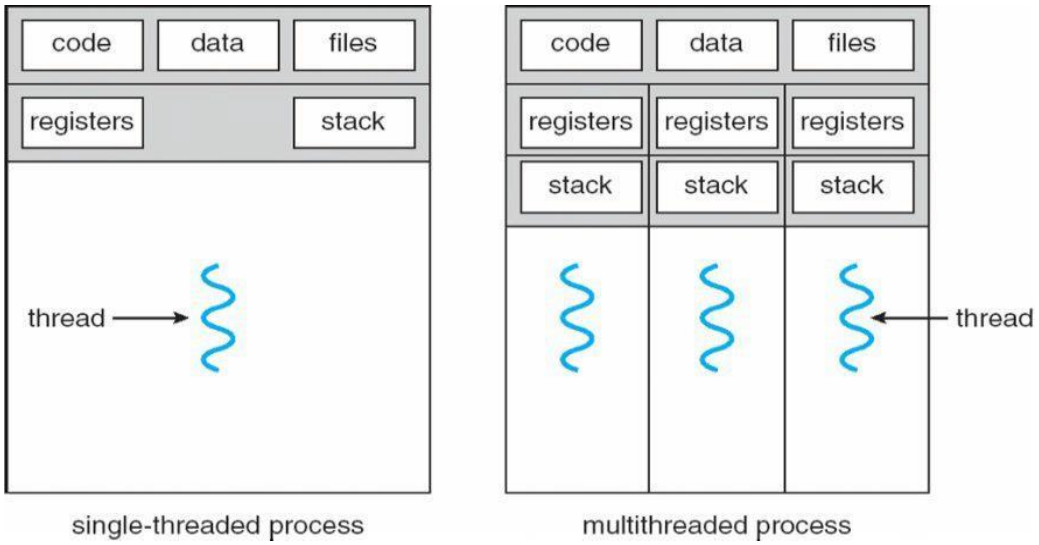
Process	Thread
Process takes more time to create.	Thread takes less time to create.
it takes more time to complete execution & terminate.	Less time to terminate.
Execution is very slow.	Execution is very fast.
It takes more time to switch b/w two processes.	It takes less time to switch b/w two threads.
Communication b/w two processes is difficult .	Communication b/w two threads is easy.
Process can't share the same memory area.	Threads can share same memory area.
System calls are requested to communicate each other.	System calls are not required.
Process is loosely coupled.	Threads are tightly coupled.
It requires more resources to execute.	Requires few resources to execute.

Multithreading

A process is divided into number of smaller tasks each task is called a Thread. Number of Threads with in a Process execute at a time is called Multithreading.
If a program, is multithreaded, even when some portion of it is blocked, the whole program is not blocked.The rest of the program continues working If multiple CPU's are available.
Multithreading gives best performance.If we have only a single thread, number of CPU's available, No performance benefits achieved.

- Process creation is heavy-weight while thread creation is light-weight

 Can simplify code, increase efficiency



 Kernels are generally multithreaded

CODE- Contains instruction
DATA- holds global variable **FILES-** opening and closing files
REGISTER- contain information about CPU state
STACK-parameters, local variables, functions
Types Of Threads:

1) User Threads : Thread creation, scheduling, management happen in user space by Thread Library. user threads are faster to create and manage. If a user thread performs a system call, which blocks it, all the other threads in that process one also automatically blocked, whole process is blocked.

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls areblocking.
- Multithreaded application cannot take advantage ofmultiprocessing.

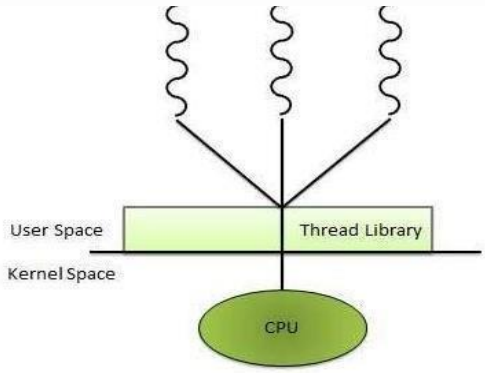
2) Kernel Threads: kernel creates, schedules, manages these threads .these threads are slower, manage. If one thread in a process blocked, over all process need not be blocked.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the userthreads.
- Transfer of control from one thread to another within same process requires a mode switch to the Kernel.



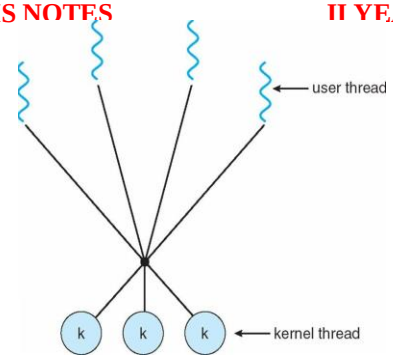
Multithreading Models

Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

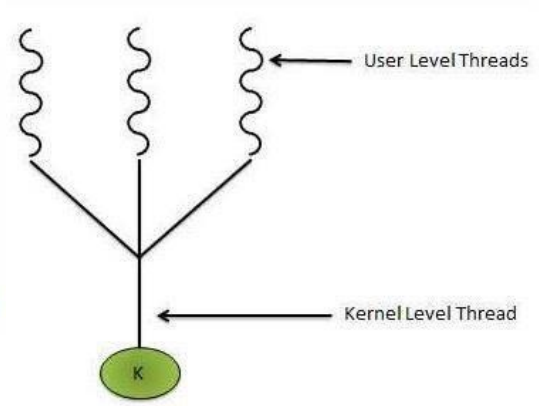
In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine. Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



Many to One Model

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

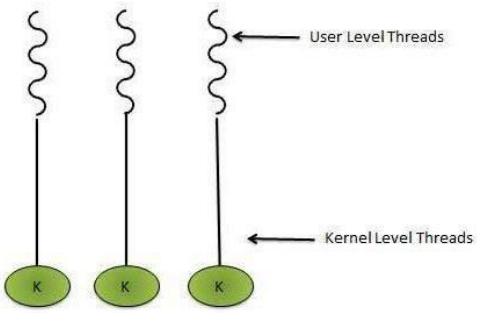
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model

There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User Level & Kernel Level Thread

S.N.	User Level Threads	Kernel Level Thread
1	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User level thread is generic and can run on any operating system.	Kernel level thread is specific to the operating system.
4	Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

UNIT-II

Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJF, RR; Multiprocessor scheduling: Real Time scheduling: RM and EDF.

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer/Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

PROCESS SCHEDULING:

CPU is always busy in **Multiprogramming**. Because CPU switches from one job to another job. But in **simple computers** CPU sit idle until the I/O request granted.

scheduling is a important OS function. All resources are scheduled before use.(cpu, memory, devices…..)

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing

. **Scheduling Objectives**

Maximize throughput.
Maximize number of users receiving acceptable response times.
Be predictable.
Balance resource use.
Avoid indefinite postponement.
Enforce Priorities.
Give preference to processes holding key resources

SCHEDULING QUEUES: people live in rooms. Process are present in rooms knows as queues. There are 3types

1. **job queue**: when processes enter the system, they are put into a **job queue**, which consists all processes in the system. Processes in the job queue reside on mass storage and await the allocation of main memory.
2. **ready queue**: if a process is present in main memory and is ready to be allocated to cpu for execution, is kept in **readyqueue.**
3. **device queue**: if a process is present in waiting state (or) waiting for an i/o event to complete is said to bein device queue.(or)

The processes waiting for a particular I/O device is called device queue.

Schedulers : There are 3 schedulers

1. Long term scheduler.
2. Medium term scheduler
3. Short term scheduler.

Scheduler duties:

- Maintains the queue.
- Select the process from queues assign to CPU.

Types of schedulers

1. Long term scheduler:
select the jobs from the job pool and loaded these jobs into main memory (ready queue).
Long term scheduler is also called job scheduler.

2. Short term scheduler:
select the process from ready queue, and allocates it to the cpu.
If a process requires an I/O device, which is not present available then process enters device queue.
short term scheduler maintains ready queue, device queue. Also called as cpu scheduler.

3. Medium term scheduler: if process request an I/O device in the middle of the execution, then the process removed from the main memory and loaded into the waiting queue. When the I/O operation completed, then the job moved from waiting queue to ready queue. These two operations performed by medium term scheduler.

Comparison between Scheduler

S.N.	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch: Assume, main memory contains more than one process. If cpu is executing a process, if time expires or if a high priority process enters into main memory, then the scheduler saves information about current process in the PCB and switches to execute the another process. The concept of moving CPU by scheduler from one process to other process is known as context switch.

Non-Preemptive Scheduling: **CPU** is assigned to one process, CPU do not release until the competition of that process. The CPU will assigned to some other process only after the previous process has finished.

Preemptive scheduling: here CPU can release the processes even in the middle of the execution. CPU received a signal from process p2. OS compares the priorities of p1 ,p2. If p1>p2, CPU continues the execution of p1. If p1<p2 CPU preempt p1 and assigned to p2.

Dispatcher: The main job of dispatcher is switching the cpu from one process to another process. Dispatcher connects the cpu to the process selected by the short term scheduler.

Dispatcher latency: The time it takes by the dispatcher to stop one process and start another process is known as dispatcher latency. If the dispatcher latency is increasing, then the degree of multiprogramming decreases.

SCHEDULING CRITERIA:

1. **Throughput**: how many jobs are completed by the cpu with in a timeperiod.
2. **Turn around time** : The time interval between the submission of the process and time of the completion is turn around time.

TAT = Waiting time in ready queue + executing time + waiting time in waiting queue for I/O.

3. **Waiting time**: The time spent by the process to wait for cpu to beallocated.
4. **Response time**: Time duration between the submission and firstresponse.
5. **Cpu Utilization**: CPU is costly device, it must be kept as busy aspossible.

Eg: CPU efficiency is 90% means it is busy for 90 units, 10 units idle.

CPU SCHEDULINGALGORITHMS:

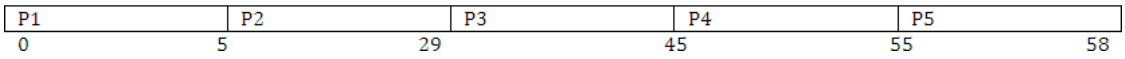
1. **First come First served scheduling: (FCFS):** The process that request the CPU first is holds the cpu first. If a process request the cpu then it is loaded into the ready queue, connect CPU to that process.

Consider the following set of processes that arrive at time 0, the length of the cpu burst time given in milli seconds.

burst time is the time, required the cpu to execute that job, it is in milli seconds.

Process	Burst time(milliseconds)
P1	5
P2	24
P3	16
P4	10
P5	3

Chart:



Average turn around time:

Turn around time= waiting time + burst time

Turn around time for p1= 0+5=5.
Turn around time for p2=5+24=29 Turn around time for p3=29+16=45 Turn around time for p4=45+10=55 Turn around time for p5= 55+3=58
Average turn around time= (5+29++45+55+58/5) = 187/5 =37.5 millisecounds

Average waiting time:

waiting time= starting time- arrival time

Waiting time for p1=0
Waiting time for p2=5-0=5
Waiting time for p3=29-0=29
Waiting time for p4=45-0=45
Waiting time for p5=55-0=55
Average waiting time= 0+5+29+45+55/5 = 125/5 = 25 ms.

Average Response Time :

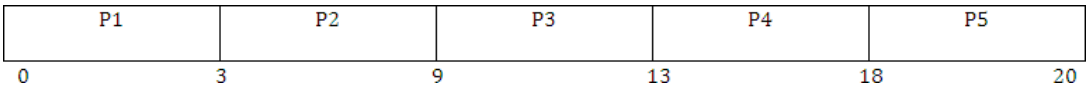
Formula : First Response - Arrival Time Response Time for P1 =0
Response Time for P2 => 5-0 = 5
Response Time for P3 => 29-0 = 29
Response Time for P4 => 45-0 = 45
Response Time for P5 => 55-0 = 55
Average Response Time => (0+5+29+45+55)/5 =>25ms

1) First Come FirstServe:

It is Non Primitive Scheduling Algorithm.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8

Process arrived in the order P1, P2, P3, P4, P5.
P1 arrived at 0 ms.
P2 arrived at 2 ms.
P3 arrived at 4 ms.
P4 arrived at 6 ms.
P5 arrived at 8 ms.



Average Turn Around Time

Formula : Turn around Time =**:** waiting time + burst time
Turn Around Time for P1 => 0+3= 3
Turn Around Time for P2 => 1+6 = 7
Turn Around Time for P3 => 5+4 = 9
Turn Around Time for P4 => 7+ 5= 12
Turn Around Time for P5 => 2+ 10=12
Average Turn Around Time => (3+7+9+12+12)/5 =>43/5 = 8.50 ms.

Average Response Time :

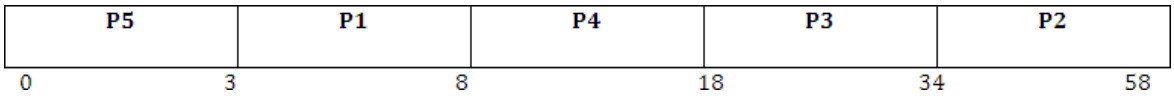
Formula : Response Time = First Response - Arrival Time
Response Time of P1 = 0
Response Time of P2 => 3-2 = 1
Response Time of P3 => 9-4 = 5
Response Time of P4 => 13-6 = 7
Response Time of P5 => 18-8 =10
Average Response Time => (0+1+5+7+10)/5 => 23/5 = 4.6 ms
Advantages: Easy to Implement, Simple.

Disadvantage: Average waiting time is very high.

2) Shortest Job First Scheduling (SJF):

Which process having the smallest CPU burst time, CPU is assigned to that process . If two process having the same CPU burst time, FCFS is used.

PROCESS	CPU BURST TIME
P1	5
P2	24
P3	16
P4	10
P5	3



P5 having the least CPU burst time (3ms). CPU assigned to that (P5). After completion of P5 short term scheduler search for nest (P1).....

Average Waiting Time :

Formula = Staring Time - Arrival Time
waiting Time for P1 => 3-0 = 3
waiting Time for P2 => 34-0 = 34
waiting Time for P3 => 18-0 = 18
waiting Time for P4 =>8-0=8
waiting time for P5=0
Average waiting time => (3+34+18+8+0)/5 => 63/5 =12.6 ms

Average Turn Around Time :

Formula = waiting Time + burst Time

Turn Around Time for P1 => 3+5 =8
Turn Around for P2 => 34+24 =58
Turn Around for P3 => 18+16 = 34

Turn Around Time for P4 => 8+10 =18
Turn Around Time for P5 => 0+3 = 3
Average Turn around time => (8+58+34+18+3)/5 => 121/5 = 24.2 ms

Average Response Time :

Formula : First Response - Arrival Time

First Response time for P1 =>3-0 = 3
First Response time for P2 => 34-0 = 34
First Response time for P3 => 18-0 = 18
First Response time for P4 => 8-0 = 8
First Response time for P5 = 0
Average Response Time => (3+34+18+8+0)/5 => 63/5 = 12.6 ms
SJF is Non primitive scheduling algorithm
Advantages : **Least average waiting time**
Least average turn around time Least average response time
Average waiting time (FCFS) = 25 ms
Average waiting time (SJF) = 12.6 ms 50% time saved in SJF.
Disadvantages:

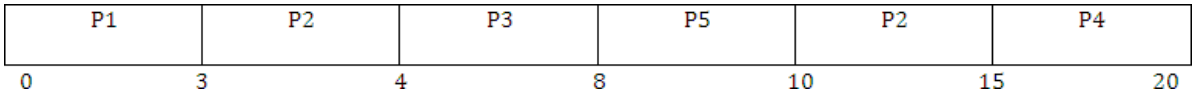
- Knowing the length of the next CPU burst time is difficult.
- Aging (Big Jobs are waiting for long time for CPU)

3) Shortest Remaining Time First (SRTF):

This is primitive scheduling algorithm.

Short term scheduler always chooses the process that has term shortest remaining time. When a new process joins the ready queue , short term scheduler compare the remaining time of executing process and new process. If the new process has the least CPU burst time, The scheduler selects that job and connect to CPU. Otherwise continue the old process.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8



P1 arrives at time 0, P1 executing First , P2 arrives at time 2. Compare P1 remaining time and P2 (3-2 = 1) and 6. So, continue P1 after P1, executing P2, at time 4, P3 arrives, compare P2 remaining time (6-1=5) and 4 (4<5) .So, executing P3 at time 6, P4 arrives. Compare P3 remaining time and P4 (4-2=2) and 5 (2<5). So, continue P3 , after P3, ready queue consisting P5 is the least out of three. So execute P5, next P2, P4.

FORMULA : Finish time - Arrival

Time Finish Time for P1 => 3-0 = 3

Finish Time for P2 => 15-2 = 13

Finish Time for P3 => 8-4 =4

Finish Time for P4 => 20-6 = 14

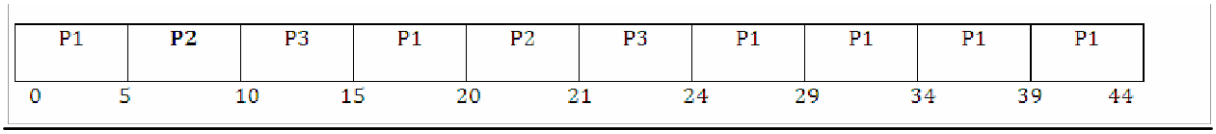
Finish Time for P5 => 10-8 = 2

Average Turn around time => 36/5 = 7.2 ms.

4)ROUND ROBIN SCHEDULING ALGORITHM :

It is designed especially for time sharing systems. Here CPU switches between the processes. When the time quantum expired, the CPU switched to another job. A small unit of time, called a time quantum or time slice. A time quantum is generally from 10 to 100 ms. The time quantum is generally depending on OS. Here ready queue is a circular queue. CPU scheduler picks the first process from ready queue, sets timer to interrupt after one time quantum and dispatches the process.

PROCESS	BURST TIME
P1	30
P2	6
P3	8



AVERAGE WAITING TIME :

Waiting time for P1 => 0+(15-5)+(24-20) => 0+10+4 = 14
Waiting time for P2 => 5+(20-10) => 5+10 = 15
Waiting time for P3 => 10+(21-15) => 10+6 = 16
Average waiting time => (14+15+16)/3 = 15 ms.

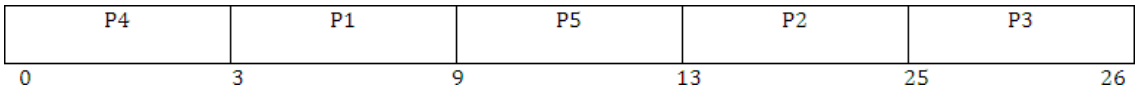
AVERAGE TURN AROUND TIME :

FORMULA : Turn around time = waiting time + burst Time
Turn around time for P1 => 14+30 =44
Turn around time for P2 => 15+6 = 21
Turn around time for P3 => 16+8 = 24
Average turn around time => (44+21+24)/3 = 29.66 ms

5) PRIORITY SCHEDULING :

PROCESS	BURST TIME	PRIORITY
P1	6	2
P2	12	4
P3	1	5
P4	3	1
P5	4	3

P4 has the highest priority. Allocate the CPU to process P4 first next P1, P5, P2, P3.



AVERAGE WAITING TIME :

Waiting time for P1 => 3-0 =3
Waiting time for P2 => 13-0 = 13
Waiting time for P3 => 25-0 = 25
Waiting time for P4 => 0
Waiting time for P5 => 9-0 =9

Average waiting time => (3+13+25+0+9)/5 = 10 ms

AVERAGE TURN AROUND TIME :

Turn around time for P1 =>3+6 = 9
Turn around time for P2 => 13+12= 25
Turn around time for P3 => 25+1 = 26
Turn around time for P4 => 0+3= 3
Turn around time for P5 => 9+4 = 13

Average Turn around time => (9+25+26+3+13)/5 = 15.2 ms

Disadvantage: Starvation

Starvation means only high priority process are executing, but low priority process are waiting for the CPU for the longest period of the time.

Multiple – processor scheduling:

When multiple processes are available, then the scheduling gets more complicated, because there is more than one CPU which must be kept busy and in effective use at all times.

Load sharing resolves around balancing the load between multiple processors. Multi processor systems may be heterogeneous (It contains different kinds of CPU's) (or) Homogeneous(all the same kind of CPU).

1) Approaches to multiple-processor scheduling

a)Asymmetric multiprocessing

One processor is the master, controlling all activities and running all kernel code, while the other runs only user code.

b)Symmetric multiprocessing:

Each processor schedules its own job. Each processor may have its own private queue of ready processes**.**

2) Processor Affinity

Successive memory accesses by the process are often satisfied in cache memory. what happens if the process migrates to another processor. the contents of cache memory must be invalidated for the first processor, cache for the second processor must be repopulated. Most Symmetric multi processor systems try to avoid migration of processes from one processor to another processor, keep a process running on the same processor. This is called processor affinity.

a) Soft affinity:

Soft affinity occurs when the system attempts to keep processes on the same processor but makes no guarantees.

b) Hard affinity:

Process specifies that it is not to be moved between processors.

3) Load balancing:

One processor wont be sitting idle while another is overloaded. Balancing can be achived through push migration or pull migration.

Push migration:

Push migration involves a separate process that runs periodically(e.g every 200 ms) and moves processes from heavily loaded processors onto less loaded processors.

Pull migration:

Pull migration involves idle processors taking processes from the ready queues of the other processors.

Real time scheduling:

Real time scheduling is generally used in the case of multimedia operating systems. Here multiple processes compete for the CPU. How to schedule processes A,B,C so that each one meets its deadlines. The general tendency is to make them pre-emptable, so that a process in danger of missing its deadline can preempt another process. When this process sends its frame, the preempted process can continue from where it had left off. Here throughput is not so significant. Important is that tasks start and end as per their deadlines.

RATE MONOTONIC (RM) SCHEDULING ALGORITHM

Rate monotonic scheduling Algorithm works on the principle of preemption. Preemption occurs on a given processor when higher priority task blocked lower priority task from execution. This blocking occurs due to priority level of different tasks in a given task set. rate monotonic is a preemptive algorithm which means if a task with shorter period comes during execution it will gain a higher priority and can block or preemptive currently running tasks. In RM priorities are assigned according to time period. Priority of a task is inversely proportional to its timer period. Task with lowest time period has highest priority and the task with highest period will have lowest priority.

For example, we have a task set that consists of three tasks as follows

Tasks	Execution time(Ci)	Time period(Ti)
T1	0.5	3
T2	1	4
T3	2	6

Table 1. Task set

$$\mathbf{U} = 0.5/3 + 1/4 + 2/6 = 0.167 + 0.25 + 0.333 = 0.75$$

As processor utilization is less than 1 or 100% so task set is schedulable and it also satisfies the above equation of rate monotonic scheduling algorithm.

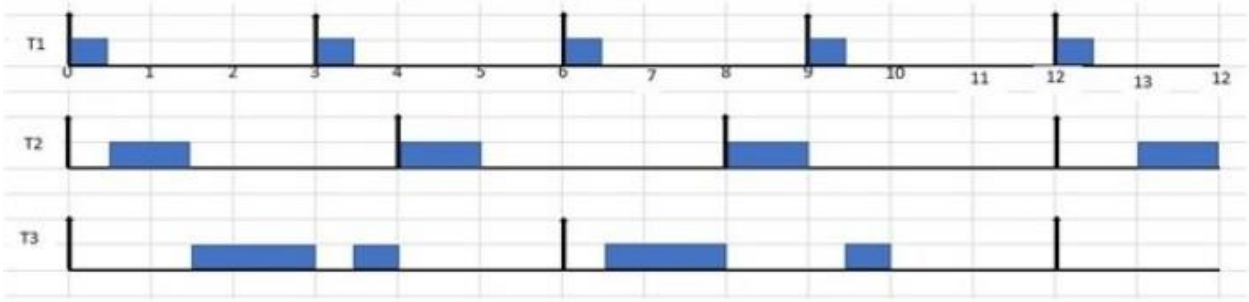


Figure 1. RM scheduling of Task set in table 1.

A task set given in table 1 it RM scheduling is given in figure 1. The explanation of above is as follows

1. According to RM scheduling algorithm task with shorter period has higher priority so T1 has high priority, T2 has intermediate priority and T3 has lowest priority. At t=0 all the tasks are released. Now T1 has highest priority so it executes first till t=0.5.
2. At t=0.5 task T2 has higher priority than T3 so it executes first for one-time units till t=1.5. After its completion only one task is remained in the system that is T3, so it starts its execution and executes till t=3.
3. At t=3 T1 releases, as it has higher priority than T3 so it preempts or blocks T3 and starts it execution till t=3.5. After that the remaining part of T3 executes.
4. At t=4 T2 releases and completes it execution as there is no task running in the system at this time.
5. At t=6 both T1 and T3 are released at the same time but T1 has higher priority due to shorter period so it preempts T3 and executes till t=6.5, after that T3 starts running and executes till t=8.
6. At t=8 T2 with higher priority than T3 releases so it preempts T3 and starts its execution.
7. At t=9 T1 is released again and it preempts T3 and executes first and at t=9.5 T3 executes its remaining part. Similarly, the execution goes on.

Earliest Deadline First (EDF) Scheduler Algorithm

The EDF is a dynamic algorithm, Job priorities are re-evaluated at every decision point, this re-evaluation is based on relative deadline of a job or task, the closer to the deadline, the higher the priority. The EDF has the following advantages:

1. Very flexible (arrival times and deadlines do not need to be known before implementation).
2. Moderate complexity.
3. Able to handle aperiodic jobs.

The EDF has the following disadvantages:

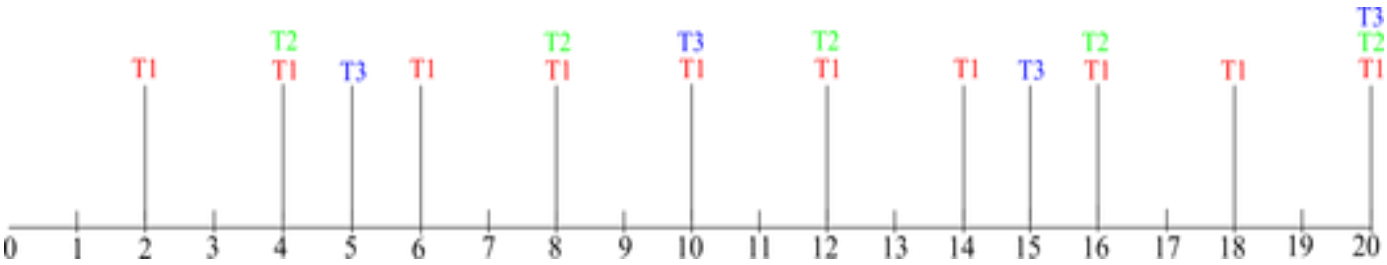
1. Optimally requires pre-emptive jobs.
2. Not optimal on several processors.
3. Difficult to verify.

Example

Consider the following task set in Table 1. P represents the Period, e the Execution time and D stands for the Deadline. Assume that the job priorities are re-evaluated at the release and deadline of a job.

	P	**e**	**D**
T1	2	0.5	2
T2	4	1	4
T3	5	1.5	5

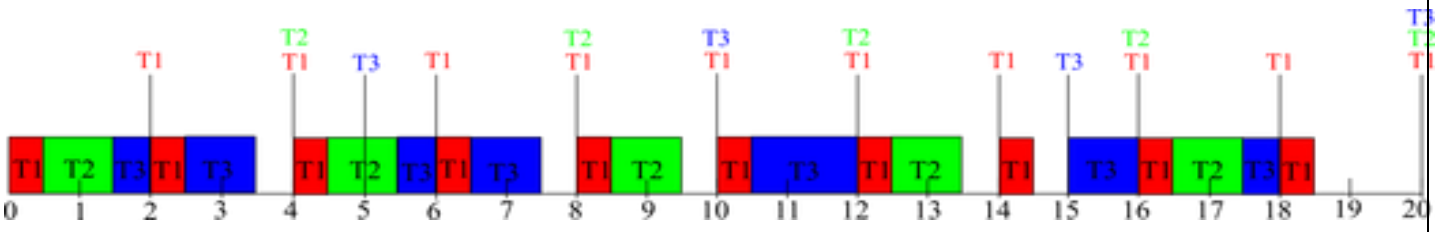
Solution



Mark all deadlines related to all the tasks

- First mark all deadlines related to the tasks as shown in Fig. 1. T1, T2 and T3 are represented with Red, Green and Blue colour respectively. The schedule is from 0 – 20ms as shown.
- At T = 0, T1 has the closest deadline, so schedule T1.
- At T = 0.5, T1 is completed, its next release time is at 2ms. T2 is closer to its deadline so T2 is scheduled next and executes for 1s.
- At T = 1.5, T2 job is completed. T3 is next because it is closer to its deadline while T2 has not been released.
- At T = 2, a new instance of T1 is released, therefore, T3 is interrupted and has 1ms left to complete execution. T1 executes
- At T = 2.5, The only ready job is T3 which is scheduled until completion.
- At T = 4, a new instance of T1 is released which executes for 0.5ms.
- At T = 4.5, T1 is now completed, so T2 is now the task closest to its deadline and is scheduled.
- At T = 5.5, T3 is scheduled but is pre-empted at T = 6 so runs for 0.5ms
- At T = 6, a new instance of T1 is released and therefore scheduled.
- At T = 6.5, T3 is closest to its deadline because T1 and T3 have not been released. So T3 is allowed to complete its execution which is 1ms.
- At T = 8, a new instance of T1 is released and is scheduled.
- At T = 8.5, T2 is the task having the closest deadline and so is scheduled to run for its execution time.
- At T = 10, the next release of T1 is scheduled.

- At T = 10.5, the next job with the closest deadline is T3 because the next T2 job will be released at T = 12. So T3 is scheduled until completion.
- At T = 12, the next release of T1 is scheduled.
- At T = 12.5, T2 is scheduled as it is the job with the closest deadline.
- At T = 14, the next release of T1 is scheduled.
- At T = 15, the next release of T3 is scheduled because it is now the job with the closest deadline because the next release of T1 and T2 is at 16ms. T3 runs for 1ms.
- At T = 16, T3 is pre=empted because a new release of T1 which has the closest deadline is now available.
- T = 16.5, T2 is the job with the closest deadline, so it is scheduled for the duration of its execution time.
- At T = 17.5, since T1 and T2 have completed, T3 resumes execution to complete its task which ran for only 1ms the last time. T3 completes execution at T = 18.
- At T = 18, a new instance of T1 is released and scheduled to run for its entire execution time.
- At T = 18.5, no job is released yet because a new release of T1, T2 and T3 are at 20ms.
- Fig. 2 shows the EDF schedule from T = 0 to T = 20.
- .



Inter Process communication:

Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action. Coordination of simultaneous processes to complete a task is known as process synchronization.

The critical section problem

Consider a system , assume that it consisting of n processes. Each process having a segment of code. This segment of code is said to be critical section.

E.G: Railway Reservation System.

Two persons from different stations want to reserve their tickets, the train number, destination is common, the two persons try to get the reservation at the same time. Unfortunately, the available berths are only one; both are trying for that berth.

It is also called the critical section problem. Solution is when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the **entry section**. The critical section may be followed by an **exit section**. The remaining code is the **remainder section.**

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (1);
```

Figure General structure of a typical process P_i.

A solution to the critical section problem must satisfy the following 3 requirements: 1.mutual exclusion:

Only one process can execute their critical section at any time.

2. **Progress:**

When no process is executing a critical section for a data, one of the processes wishing to enter a critical section for data will be granted entry.

3. **Bounded wait:**

No process should wait for a resource for infinite amount of time.

Critical section:

The portion in any program that accesses a shared resource is called as critical section (or) critical region.

Peterson's solution:

Peterson solution is one of the solutions to critical section problem involving two processes. This solution states that when one process is executing its critical section then the other process executes the rest of the code and vice versa.

Peterson solution requires two shared data items:

1) **turn**: indicates whose turn it is to enter into the critical section. If turn == i ,then process i is allowed into their critical section.

2) **flag:** indicates when a process wants to enter into critical section. when

process i wants to enter their critical section, it sets flag[i] to true.

```
do {flag[i] = TRUE; turn = j;
while (flag[j] && turn == j);
critical section
flag[i] = FALSE;
remainder section
} while (TRUE);
```

Synchronization hardware

In a uniprocessor multiprogrammed system, mutual exclusion can be obtained by disabling the interrupts before the process enters its critical section and enabling them after it has exited the critical section**.**

Disable interrupts
Critical section
Enable interrupts

Once a process is in critical section it cannot be interrupted. This solution cannot be used in multiprocessor environment. since processes run independently on different processors.

In multiprocessor systems, **Testandset** instruction is provided, it completes execution without interruption. Each process when entering their critical section must set **lock**, to prevent other processes from entering their critical sections simultaneously and must release the lock when exiting their critical sections.

```
do {
acquire
lock
critical
section
release
lock
remainder
section
} while (TRUE);
```

A process wants to enter critical section and value of lock is false then **testandset** returns false and the value of lock becomes true. thus for other processes wanting to enter their critical sections **testandset** returns true and the processes do busy waiting until the process exits critical section and sets the value of lock to false.

- **Definition:**

```
boolean TestAndSet(boolean&lock){
boolean temp=lock;
Lock=true;
return temp;
}
```

Algorithm for TestAndSet

```
do{
        while testandset(&lock)
        //do nothing
        //critical section
        lock=false
remainder section
}while(TRUE);
```

Swap instruction can also be used for mutual exclusion

Definition

```
Void swap(boolean &a, boolean &b)
{
boolean temp=a;
a=b;
b=temp;
}
```

Algorithm

```
do
{
key=true;
while(key=true)
swap(lock,key);
critical section
lock=false;
remainder section
}while(1);
```

lock is global variable initialized to false.each process has a local variable key. A process wants to enter critical section,since the value of lock is false and key is true.

lock=false
key=true
after swap instruction,
lock=true
key=false

now key=false becomes true,process exits repeat-until,and enter into critical section. When process is in critical section (lock=true),so other processes wanting to enter critical section will have
lock=true
key=true
Hence they will do busy waiting in repeat-until loop until the process exits critical section and sets the value of lock to false.

Semaphores

A semaphore is an integer variable.semaphore accesses only through two operations.

1) **wait:** wait operation decrements the count by1.

If the result value is negative,the process executing the wait operation is blocked.

2) **signaloperation:**

Signal operation increments by 1,if the value is not positive then one of the process blocked in wait operation unblocked.

```
wait (S) {
while S <= 0 ; //
no-op
 S--;
}

signal (S)
{

S++;
}
```

In binary semaphore count can be 0 or 1. The value of semaphore is initialized to 1.

```
do {
wait (mutex);
// Critical Section
signal (mutex);
// remainder section
} while (TRUE);
```

First process that executes wait operation will be immediately granted sem.count to 0. If some other process wants critical section and executes wait() then it is blocked,since value becomes -1. If the process exits critical section it executes signal().sem.count is incremented by 1.blocked process is removed from queue and added to ready queue.

Problems:

1) **Deadlock**

Deadlock occurs when multiple processes are blocked.each waiting for a resource that can only be freed by one of the other blocked processes.

2) **Starvation**

one or more processes gets blocked forever and never get a chance to take their turn in the critical section.

3) **Priority inversion**

If low priority process is running ,medium priority processes are waiting for low priority process,high priority processes are waiting for medium priority processes.this is called Priority inversion.

The two most common kinds of semaphores are **counting semaphores** and **binary semaphores**. Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked).

Classic problems of synchronization

1) **Bounded-buffer problem**

Two processes share a common ,fixed –size buffer.

Producer puts information into the buffer, consumer takes it out.

The problem arise when the producer wants to put a new item in the buffer,but it is already full. The solution is for the producer has to wait until the consumer has consumed atleast one buffer. similarly if the consumer wants to remove an item from the buffer and sees that the buffer is empty,it goes to sleep until the producer puts something in the buffer and wakes it up.

synchronisation problems:

i) we must guard against attempting to write data to the buffer when the buffer is full; ie the producer must wait for an 'empty space'.

ii) we must prevent the consumer from attempting to read data when the buffer is empty; ie, the consumer must wait for 'data available'.

To provide for each of these conditions, we require to employ three semaphores which are defined in the following table:

Semaphore	*Purpose*	*Initial Value*
free	mutual exclusion for buffer access	1
space	space available in buffer	N
data	data available in buffer	0

The structure of the producer process

```
do {
// produce an item in
nextp wait (empty);
wait (mutex);
// add the item to the
buffer signal (mutex);
signal (full);
} while (TRUE);
```

The structure of the consumer process

```
do {
wait
(full);
wait
(mutex);
// remove an item from buffer to
nextc signal (mutex);
signal (empty);
// consume the item in nextc
} while (TRUE);
```

2) The readers-writers problem

A database is to be shared among several concurrent processes.some processes may want only to read the database,some may want to update the database.If two readers access the shared data simultaneously no problem.if a write,some other process access the database simultaneously problem arised.Writes have excusive access to

the shared database while writing to the database.This problem is known as readers- writes problem.

First readers-writers problem

No reader be kept waiting unless a writer has already obtained permission to use the shared resource.

Second readers-writes problem:

Once writer is ready,that writer performs its write as soon as possible.

A process wishing to modify the shared data must request the lock in write mode. multiple processes are permitted to concurrently acquire a reader-writer lock in read mode. A reader writer lock in read mode. but only one process may acquire the lock for writing as exclusive access is required for writers.

Semaphore mutex initialized to 1

- Semaphore wrt initialized to 1
- Integer read count initialized to 0

The structure of a writer process

```
do {
wait (wrt) ;
// writing is
performed
signal (wrt) ;
} while (TRUE);
```

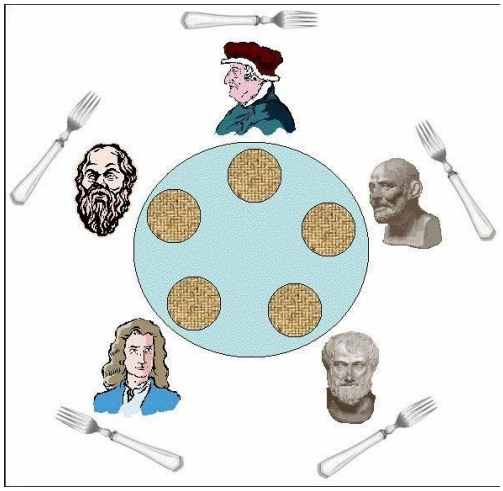
The structure of a reader process

```
do {
wait (mutex) ;
readcount ++ ;
if (readcount == 1)
wait (wrt) ;
signal (mutex)
// reading is performed wait (mutex) ;
readcount
- - ;
if (readcount == 0)
 signal (wrt) ;
signal (mutex) ;
} while (TRUE);
```

3) Dining Philosophers problem

Five philosophers are seated on 5 chairs across a table. Each philosopher has a plate full of noodles. Each philosopher needs a pair of forks to eat it. There are only 5 forks available all together. There is only one fork between any two plates of noodles.

In order to eat, a philosopher lifts two forks, one to his left and the other to his right. if he is successful in obtaining two forks, he starts eating after some time, he stops eating and keeps both the forks down.



What if all the 5 philosophers decide to eat at the same time ?

All the 5 philosophers would attempt to pick up two forks at the same time. So,none of them succeed.

One simple solution is to represent each fork with a semaphore.a philosopher tries to grab a fork by executing wait() operation on that semaphore.he releases his forks by executing the signal() operation.This solution guarantees that no two neighbours are eating simultaneously.

Suppose all 5 philosophers become hungry simultaneously and each grabs his left fork,he will be delayed forever.

The structure of Philosopher *i*:

```
do{
wait ( chopstick[i] );
wait ( chopStick[ (i + 1) % 5] );
// eat
signal ( chopstick[i] );
signal (chopstick[ (i + 1) % 5] );
// think
} while (TRUE);
```

Several remedies:

1) Allow at most 4 philosophers to be sitting simultaneously at the table.
2) Allow a philosopher to pickup his fork only if both forks are available.
3) An odd philosopher picks up first his left fork and then right fork. an even philosopher picks up his right fork and then his left fork.

MONITORS

The disadvantage of semaphore is that it is unstructured construct. Wait and signal operations can be scattered in a program and hence debugging becomes difficult.

A monitor is an object that contains both the data and procedures needed to perform allocation of a shared resource. To accomplish resource allocation using monitors, a process must call a **monitor entry routine**. Many processes may want to enter the monitor at the same time. but only one process at a time is allowed to enter. Data inside a monitor may be either global to all routines within the monitor (or) local to a specific routine. Monitor data is accessible only within the monitor. There is no way for processes outside the monitor to access monitor data. This is a form of information hiding.

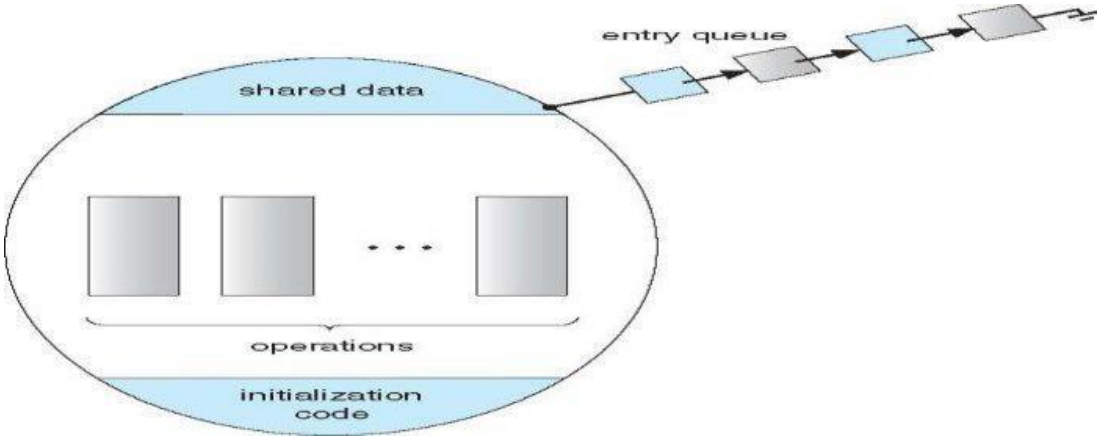
If a process calls a monitor entry routine while no other processes are executing inside the monitor, the process acquires a lock on the monitor and enters it. while a process is in the monitor, other processes may not enter the monitor to acquire the resource. If a process calls a monitor entry routine while the other monitor is locked the monitor makes the calling process wait outside the monitor until the lock on the monitor is released. The process that has the resource will call a monitor entry routine to release the resource. This routine could free the resource and wait for another requesting process to arrive monitor entry routine calls signal to allow one of the waiting processes to enter the monitor and acquire the resource. Monitor gives high priority to waiting processes than to newly arriving ones.

Structure:

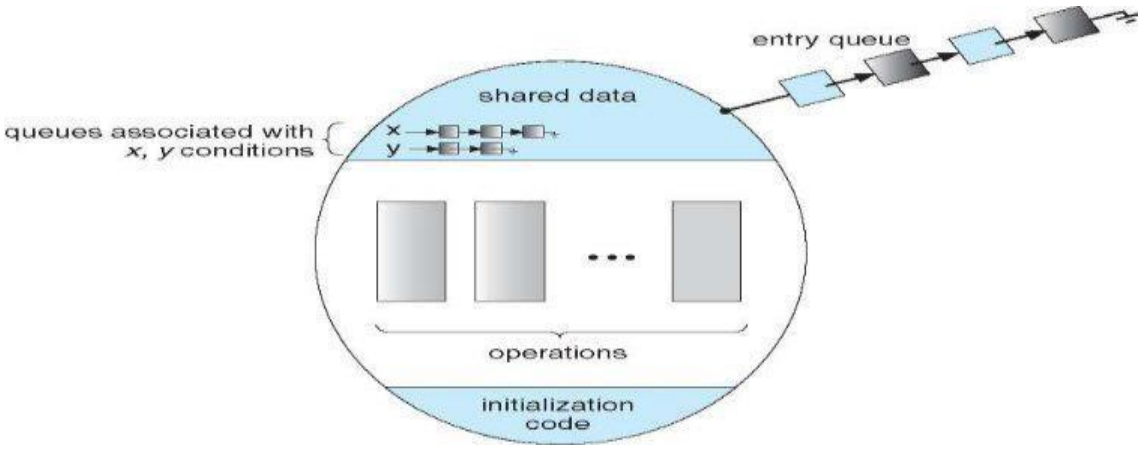
```
monitor monitor-name
{
// shared variable declarations
procedure P1 (…) { …. }
procedurePn (…) {……}
Initialization code (…) { … }
}
}
```

Processes can call procedures p1,p2,p3……They cannot access the local variables of the monitor

Schematic view of a Monitor



Monitor with Condition Variables



Monitor provides condition variables along with two operations on them i.e. wait and signal.

wait(condition variable)

signal(condition variable)

Every condition variable has an associated queue.A process calling wait on a particular condition variable is placed into the queue associated with that condition variable.A process calling signal on a particular condition variable causes a process waiting on that condition variable to be removed from the queue associated with it.

Solution to Producer consumer problem using monitors:

```
monitor
producerconsumer
condition
full,empty;
int count;
procedure insert(item)
{
if(count==MAX)
wait(full) ;
insert_item(item);
count=count+1;
if(count==1)
signal(empty);
}
procedure remove()
{
if(count==0)
wait(empty);
remove_item(item);
count=count-1;
 if(count==MAX-1)
signal(full);
}
procedure producer()
{
producerconsumer.insert(item);
}
procedure consumer()
{
producerconsumer.remove();
}
```

Solution to dining philosophers problem using monitors

```
        void test(int i) {
           if ((state[(i + 4) % 5] != eating) &&
              (state[i] == hungry) &&
              (state[(i + 1) % 5] != eating)) {
                 state[i] = eating;
                 self[i].signal();
           }
        }

        void init() {
           for (int i = 0; i < 5; i++)
              state[i] = thinking;
        }
     }
```

Figure A monitor solution to the dining-philosopher problem.

```
     test((i + 1) % 5);
  }
```

A philosopher may pickup his forks only if both of them are available.A philosopher can eat only if his two neighbours are not eating.some other philosopher can delay himself when he is hungry.

Diningphilosophers.Take_forks( ) : acquires forks ,which may block the process.

Eat noodles ( )

Diningphilosophers.put_forks( ): releases the forks.

Resuming processes within a monitor

If several processes are suspended on condion x and x.signal( ) is executed by some process. then

how do we determine which of the suspended processes should be resumed next ?

solution is FCFS(process that has been waiting the longest is resumed first).In many circumstances, such simple technique is not adequate. alternate solution is to assign priorities and wake up the process with the highest priority.

Resource allocation using monitor

boolean inuse=false;

conditionavailable;

//conditionvariable

```
monitorentry void get resource()
{
if(inuse)                    //is resource inuse
{
wait(available);                         wait until available issignaled
}
inuse=true;                  //indicate resource is now inuse
}
monitor entry void return resource()
{
inuse=false;                 //indicate resource
is not in use signal(available); //signal a
waiting process to proceed
}
```

UNIT-III

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and External fragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU).

Logical And Physical Addresses

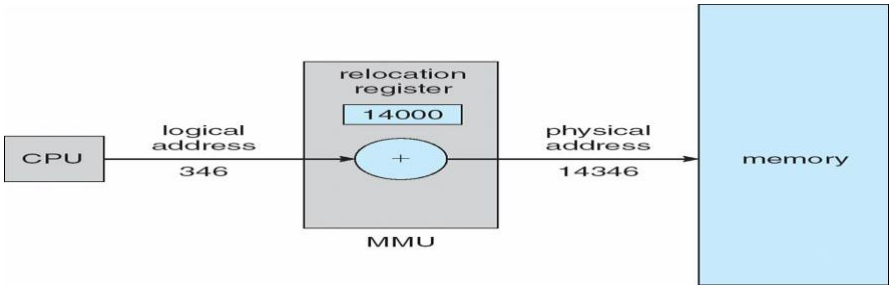
An address generated by the CPU is commonly refereed as **Logical Address**, whereas the address seen by the memory unit that is one loaded into the memory address register of the memory is commonly refereed as the **Physical Address**. The compile time and load time address binding generates the identical **logical and physical addresses**. However, the execution time addresses binding scheme results in differing **logical and physical addresses**.

The set of all **logical addresses** generated by a program is known as **Logical Address Space**, where as the set of all **physical addresses** corresponding to these logical addresses is **Physical Address Space**. Now, the run time mapping from virtual address to **physical address** is done by a hardware device known as **Memory Management Unit**. Here in the case of mapping the base register is known as **relocation register**. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory .Let's understand this situation with the help of example: If the base register contains the value 1000,then an attempt by the user to address location 0 is dynamically relocated to location 1000,an access to location 346 is mapped to location 1346.

Memory-Management Unit (MMU)

Hardware device that maps virtual to physical address

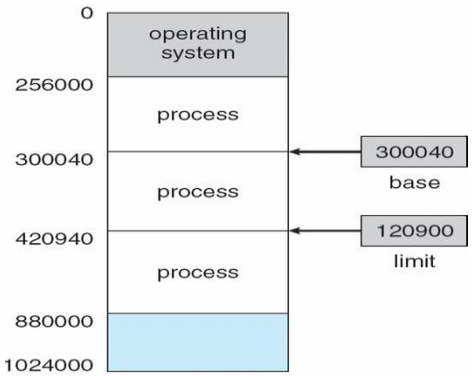
-
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses



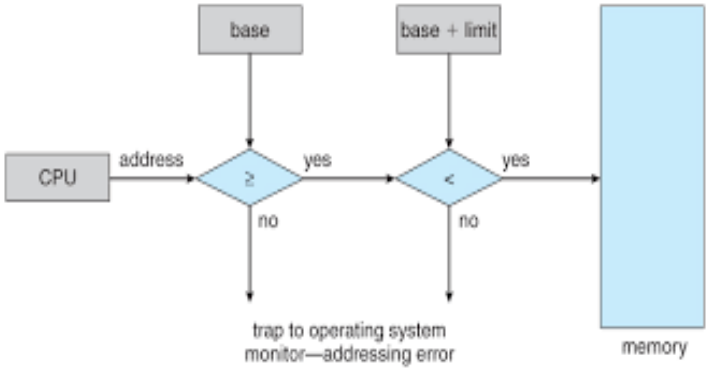
The user program never sees the **real physical address** space, it always deals with the **Logical addresses**. As we have two different type of addresses **Logical address** in the range (0 to max) and **Physical addresses** in the range(R to R+max) where R is the value of relocation register. The user generates only **logical addresses** and thinks that the process runs in location to 0 to max. As it is clear from the above text that user program supplies only logical addresses, these **logical addresses** must be mapped to **physical address** before they are used.

Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space



HARDWARE PROTECTION WITH BASE AND LIMIT



Binding of Instructions and Data to Memory

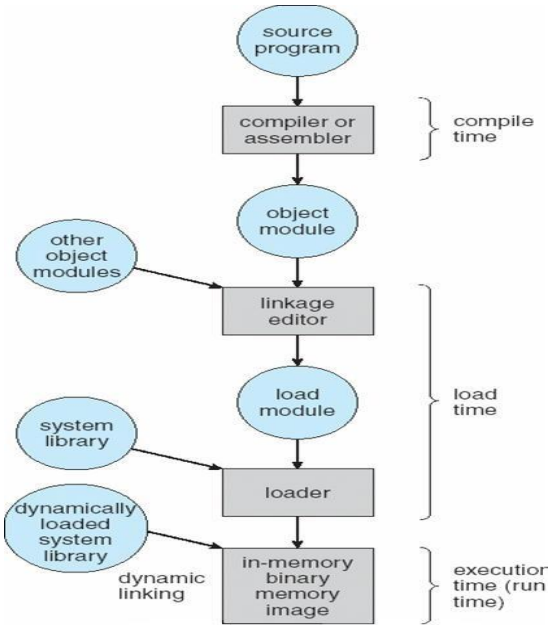
Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes

Load time: Must generate **relocatable code** if memory location is not known at compile time

- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required implemented through program design

Dynamic Linking

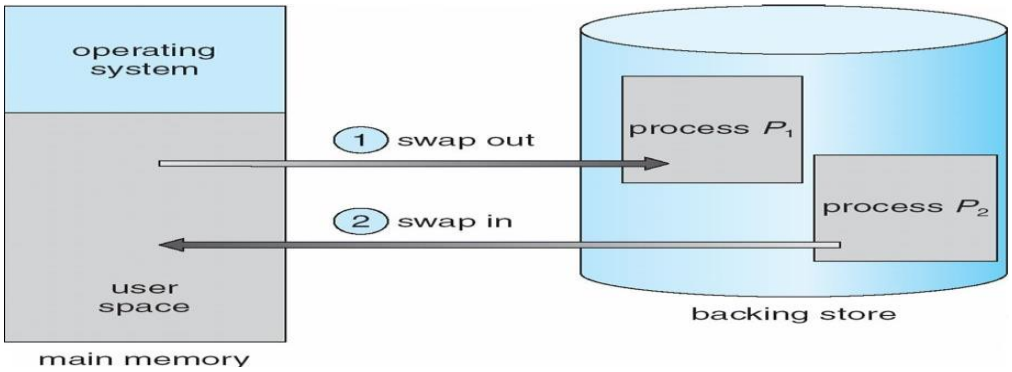
- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library
- routine Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address Dynamic
- linking is particularly useful for libraries
- System also known as **shared libraries**

Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped and Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



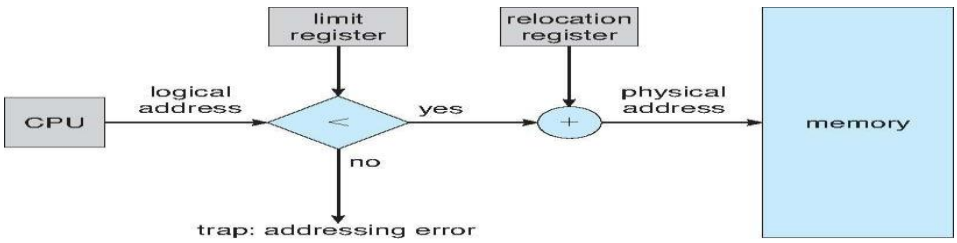
Contiguous Allocation

- Main memory usually into two partitions:
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memorynRelocation registers used to protect user processes from each other, and from changing operating-system code and data

Base register contains value of smallest physical address

- Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers



- Multiple-partition allocation
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it

Contiguous memory allocation is one of the efficient ways of allocating main memory to the processes. The memory is divided into two partitions. One for the Operating System and another for the user processes. Operating System is placed in low or high memory depending on the interrupt vector placed. In contiguous memory allocation each process is contained in a single contiguous section of memory.

Memory protection

Memory protection is required to protect Operating System from the user processes and user processes from one another. A relocation register contains the value of the smallest physical address for example say 100040. The limit register contains the range of logical address for example say 74600. Each logical address must be less than limit register. If a logical address is greater than the limit register, then there is an addressing error and it is trapped. The limit register hence offers memory protection.

The MMU, that is, Memory Management Unit maps the logical address dynamically, that is at run time, by adding the logical address to the value in relocation register. This added value is the physical memory address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values. The advantage of relocation register is that it provides an efficient way to allow the Operating System size to change dynamically.

Memory allocation

There are two methods namely, multiple partition method and a general fixed partition method. In multiple partition method, the memory is divided into several fixed size partitions. One process occupies each partition. This scheme is rarely used nowadays. Degree of multiprogramming depends on the number of partitions. Degree of multiprogramming is the number of programs that are in the main memory. The CPU is never left idle in multiprogramming. This was used by IBM OS/360 called MFT. MFT stands for Multiprogramming with a Fixed number of Tasks.

Generalization of fixed partition scheme is used in MVT. MVT stands for Multiprogramming with a Variable number of Tasks. The Operating System keeps track of which parts of memory are available and which is occupied. This is done with the help of a table that is maintained by the Operating System. Initially the whole of the available memory is treated as

one large block of memory called a **hole**. The programs that enter a system are maintained in an input queue. From the hole, blocks of main memory are allocated to the programs in the input queue. If the hole is large, then it is split into two, and one half is allocated to the arriving process and the other half is returned. As and when memory is allocated, a set of holes in scattered. If holes are adjacent, they can be merged.

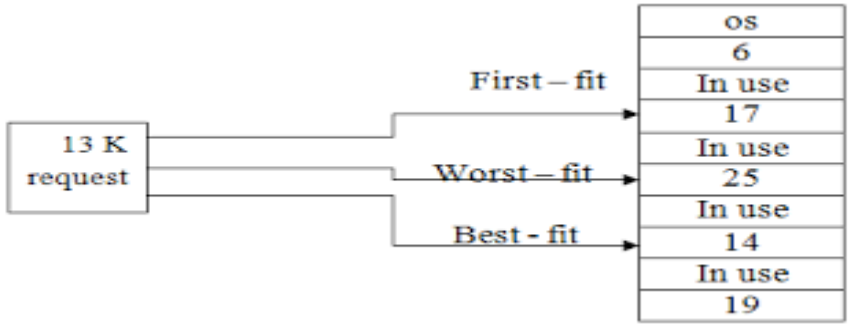
Now there comes a general dynamic storage allocation problem. The following are the solutions to the dynamic storage allocation problem.

- First fit: The first hole that is large enough is allocated. Searching for the holes starts from the beginning of the set of holes or from where the previous first fit search ended.

- Best fit: The smallest hole that is big enough to accommodate the incoming process is allocated. If the available holes are ordered, then the searching can be reduced.

- Worst fit: The largest of the available holes is allocated.

Example:



First and best fits decrease time and storage utilization. First fit is generally faster.

Fragmentation

The disadvantage of contiguous memory allocation is **fragmentation**. There are two types of fragmentation, namely, internal fragmentation and External fragmentation.

Internal fragmentation

When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 18464 bytes is available. Let the size of the process be 18462. If the hole is allocated to this process, then two bytes are left which is not used. These two bytes which cannot be used forms the internal fragmentation. The worst part of it is that the overhead to maintain these two bytes is more than two bytes.

External fragmentation

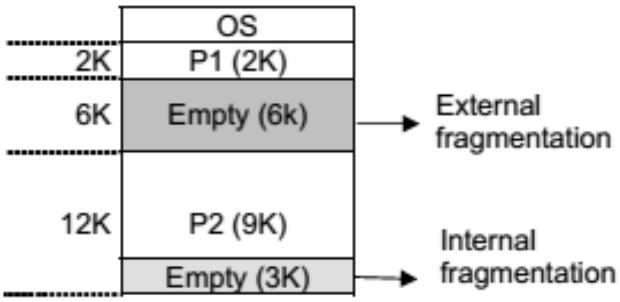
All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of

fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block. It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

One more solution to external fragmentation is to have the logical address space and physical address space to be non contiguous. Paging and Segmentation are popular non contiguous allocation methods.

Example for internal and external fragmentation



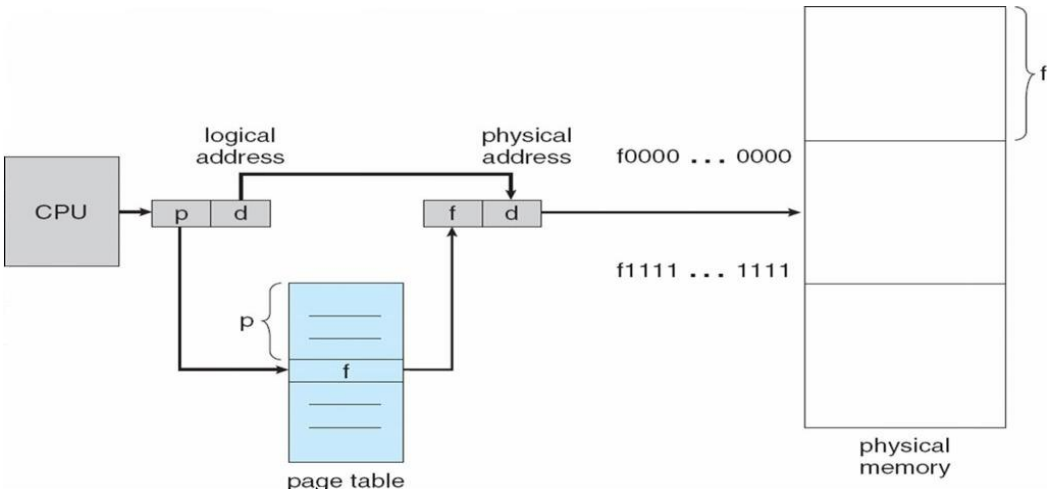
Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Paging Hardware



Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

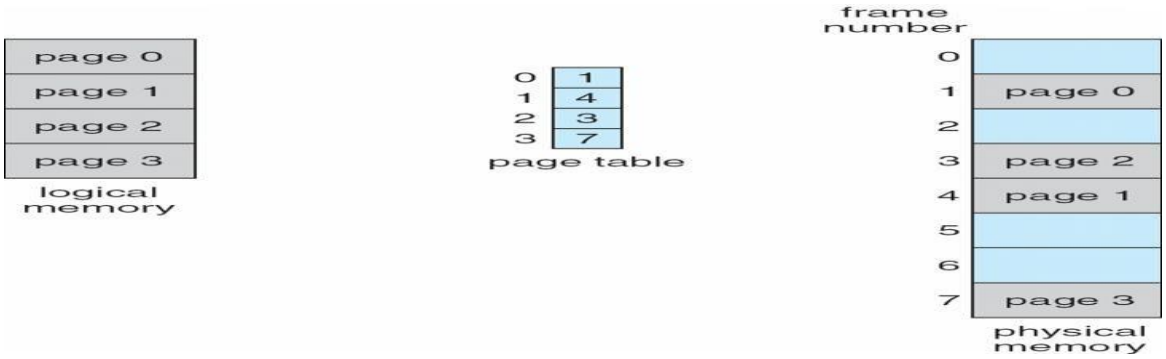
Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

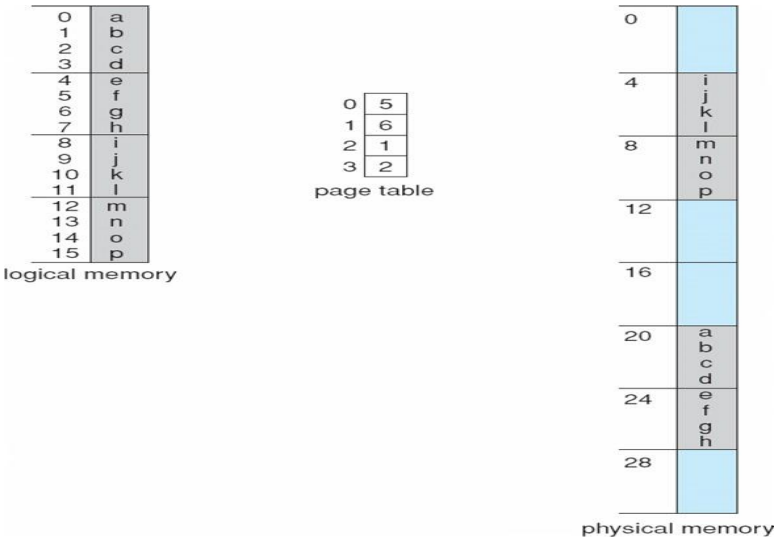
Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

Paging Model of Logical and Physical Memory

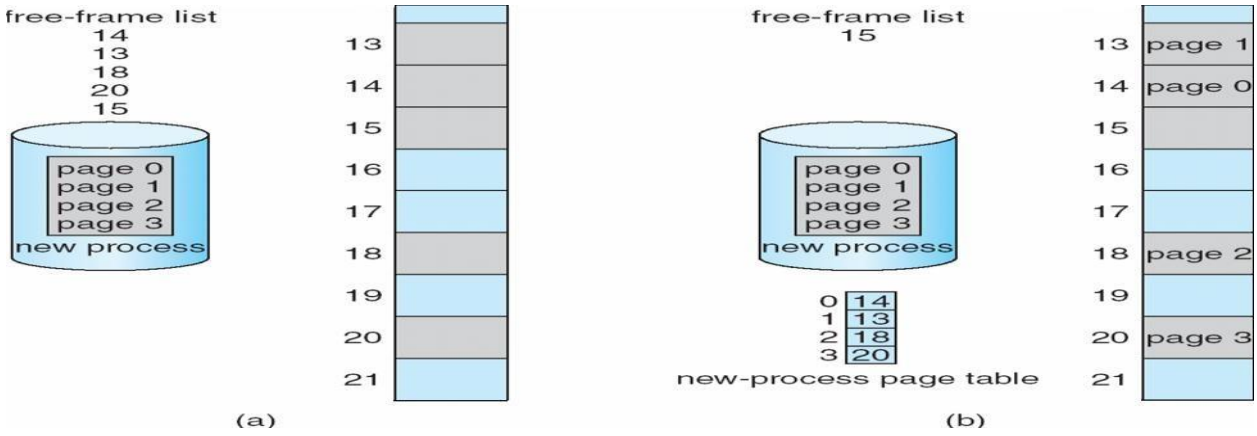


Paging Example



32-byte memory and 4-byte pages

Free Frames



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a

computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

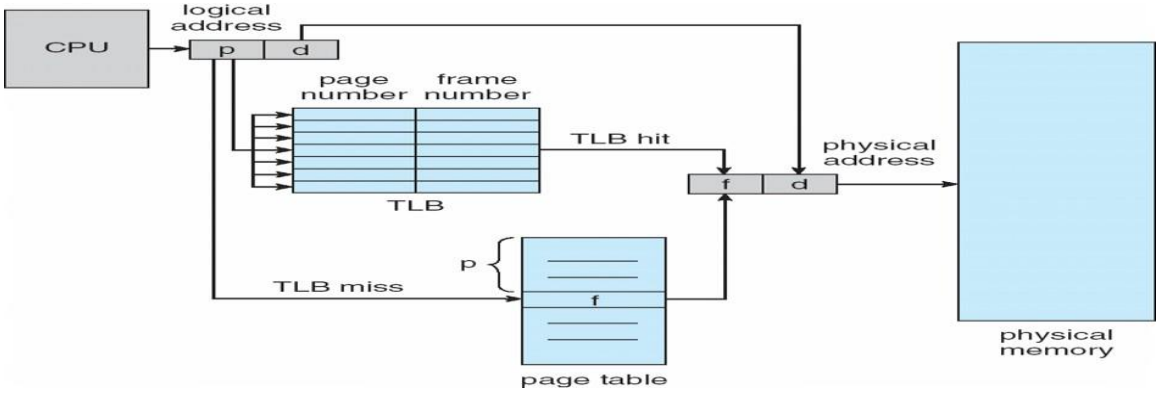
This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

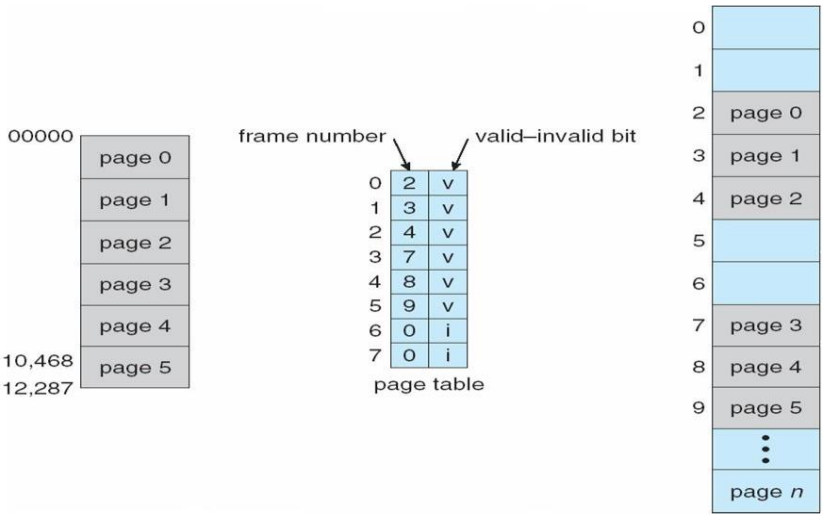
The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Paging Hardware With TLB



Memory Protection

- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal
- page "invalid" indicates that the page is not in the process' logical address space
- Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

Shared code

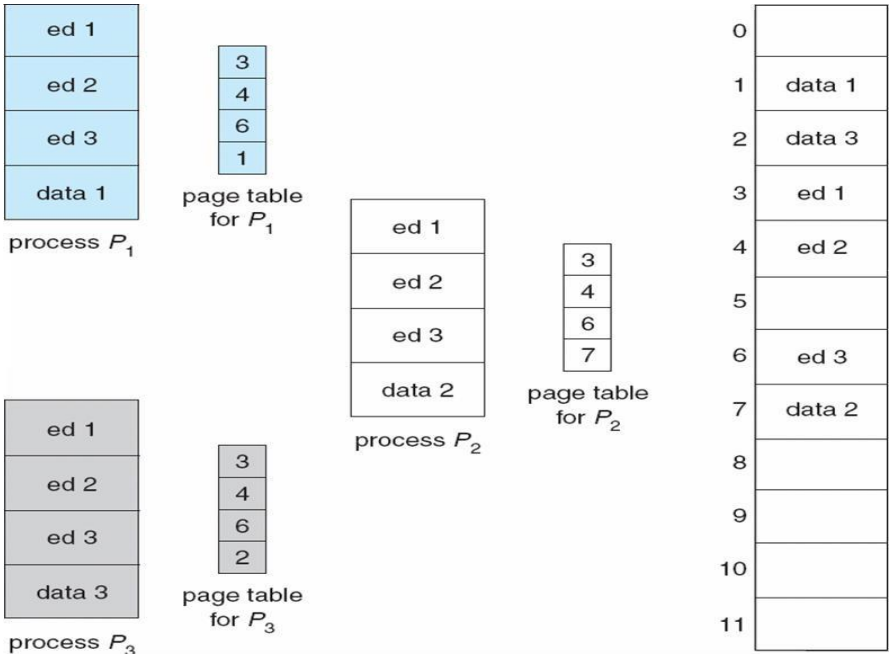
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

Each process keeps a separate copy of the code and data

-
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



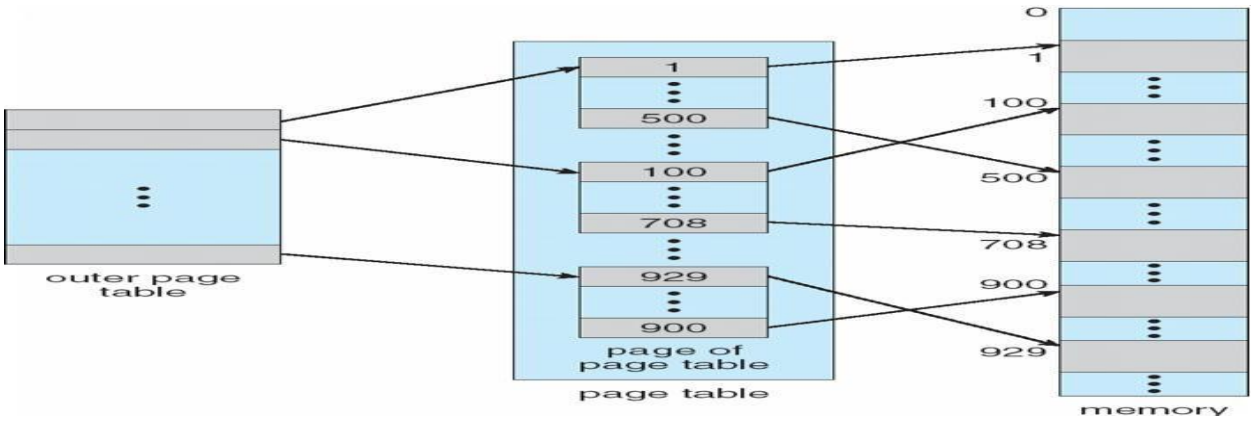
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

Break up the logical address space into multiple page tables A simple technique is a two-level page table

Two-Level Page-Table Scheme



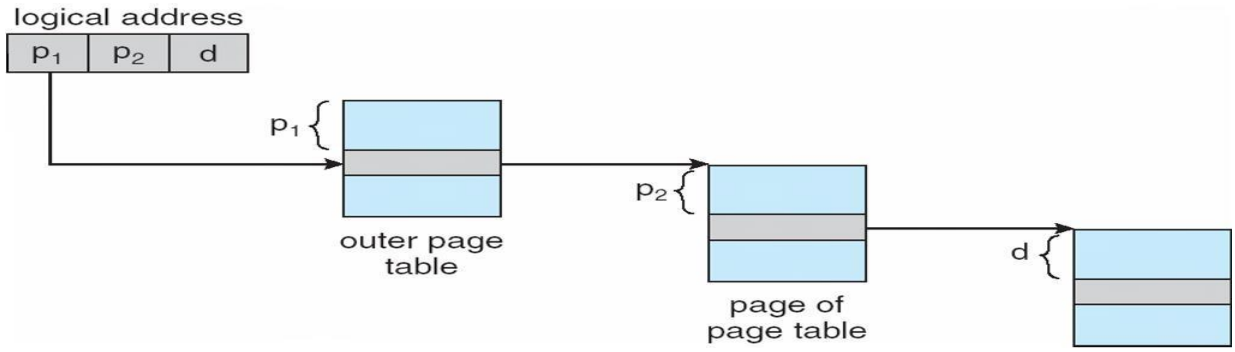
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided
- into: a page number consisting of 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
- a 12-bit page number a 10-bit page offset
- Thus, a logical address is as follows:
-

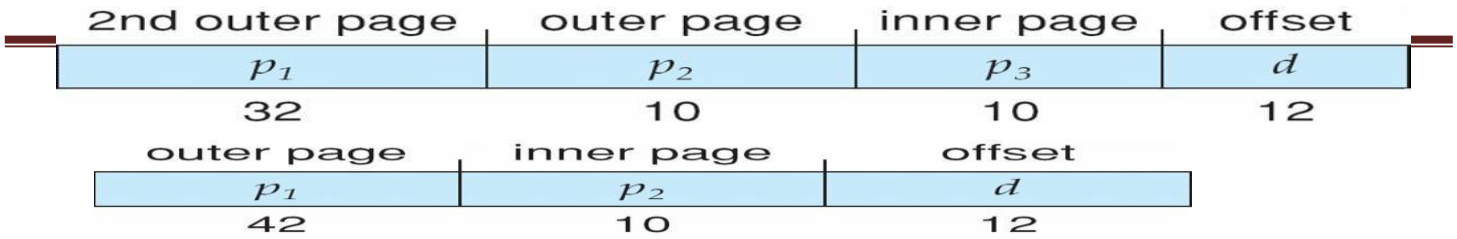
where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Page number		page offset
p_{12}	p_{10}	d_{10}

Address-Translation Scheme



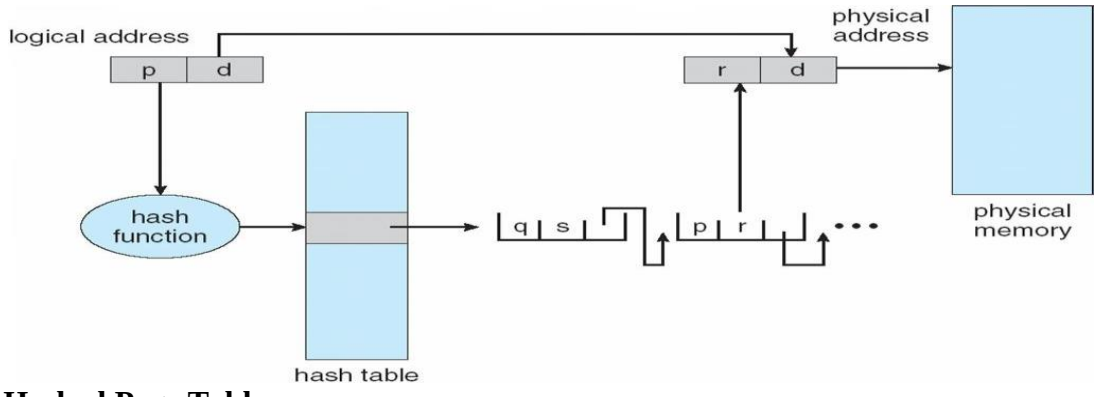
Three-level Paging Scheme



Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
- This page table contains a chain of elements hashing to the same
- location Virtual page numbers are compared in this chain searching for
- a match

If a match is found, the corresponding physical frame is extracted



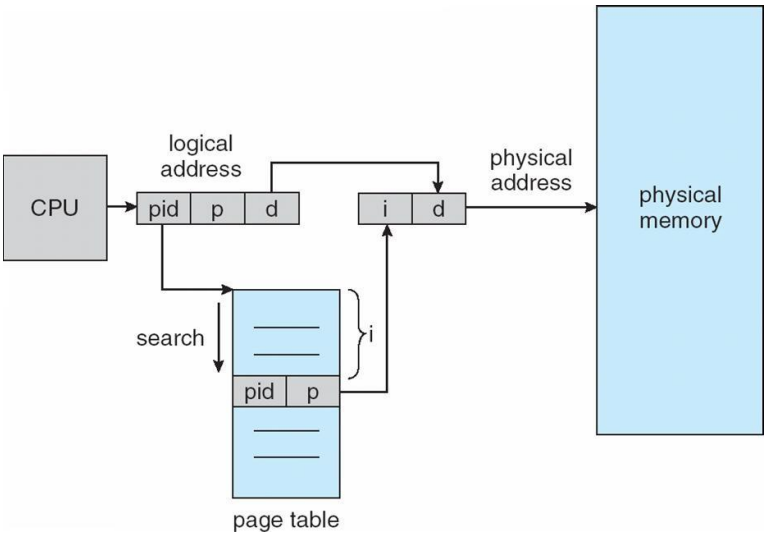
Hashed Page Table

Inverted Page Table

One entry for each real page of memory

- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

Inverted Page Table Architecture



Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

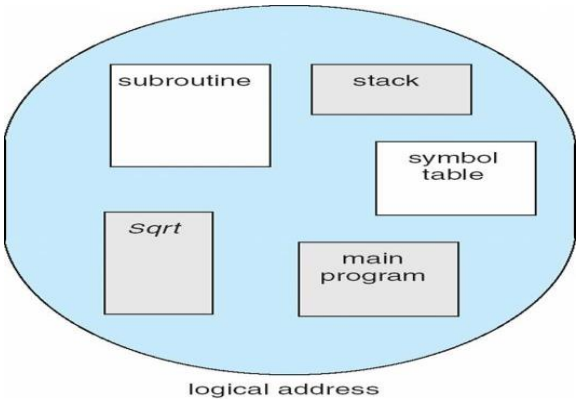
- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Segmentation

- Memory-management scheme that supports user view of memory A program is a collection of segments

- A segment is a logical unit such as:
- main program
- Procedure
- function method
- object

- local variables, global variables
- common block
- stack
- symbol table
- arrays



User's View of a Program

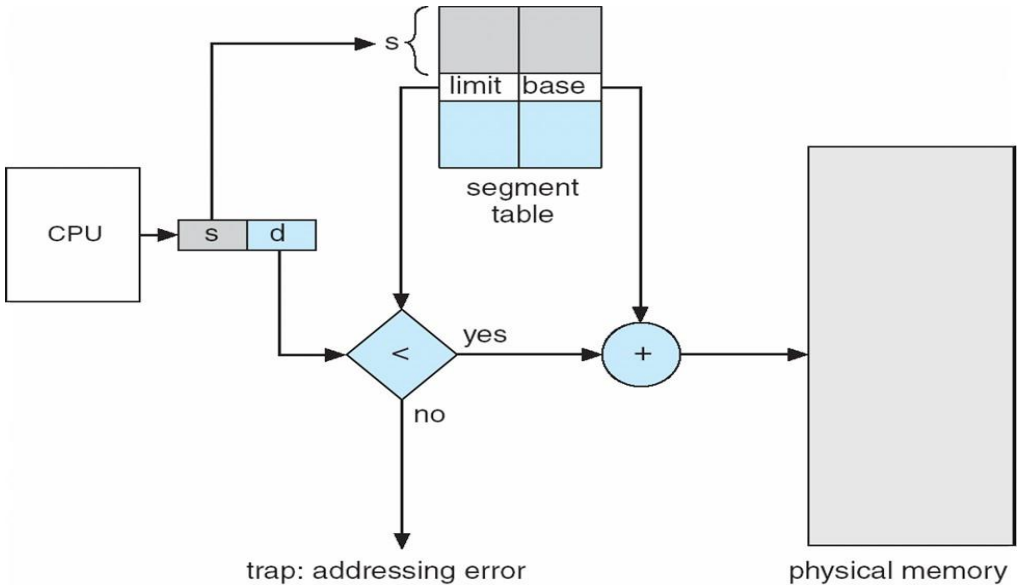
Segmentation Architecture

- Logical address consists of a two tuple:

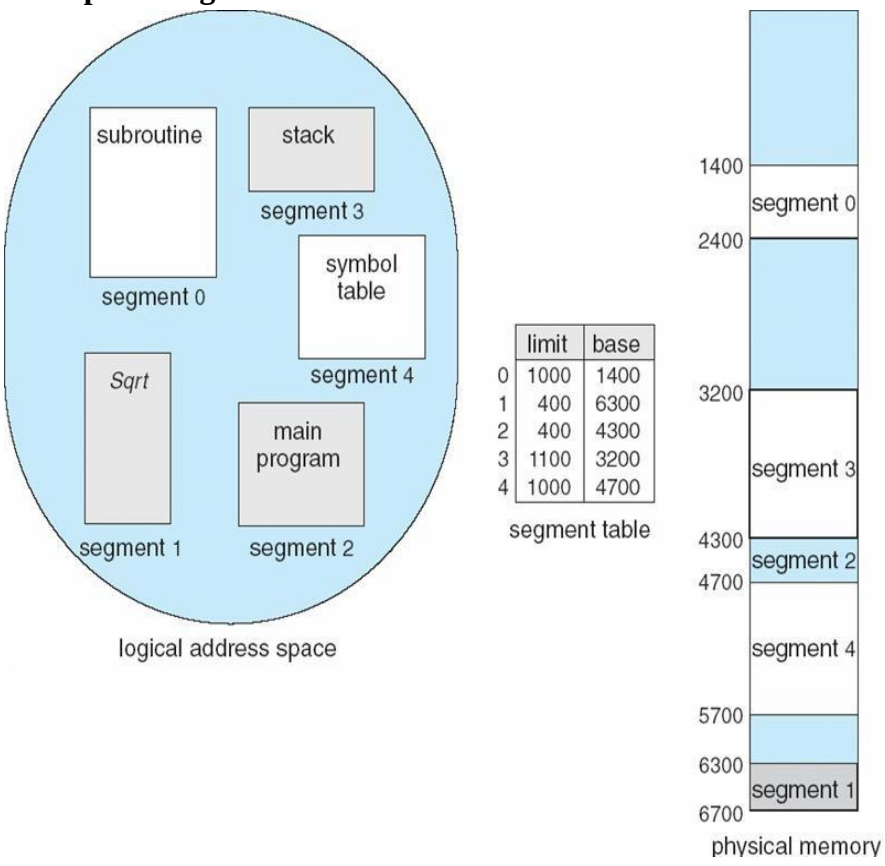
o <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:space
- **base** – contains the starting physical address where the segments reside in memory
- **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*** < **STLR**
- Protection
- With each entry in segment table associate:
- validation bit = 0 Þ illegal segment
- read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation
- problem A segmentation example is shown in the following diagram

Segmentation Hardware



Example of Segmentation



Segmentation with paging

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.

An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

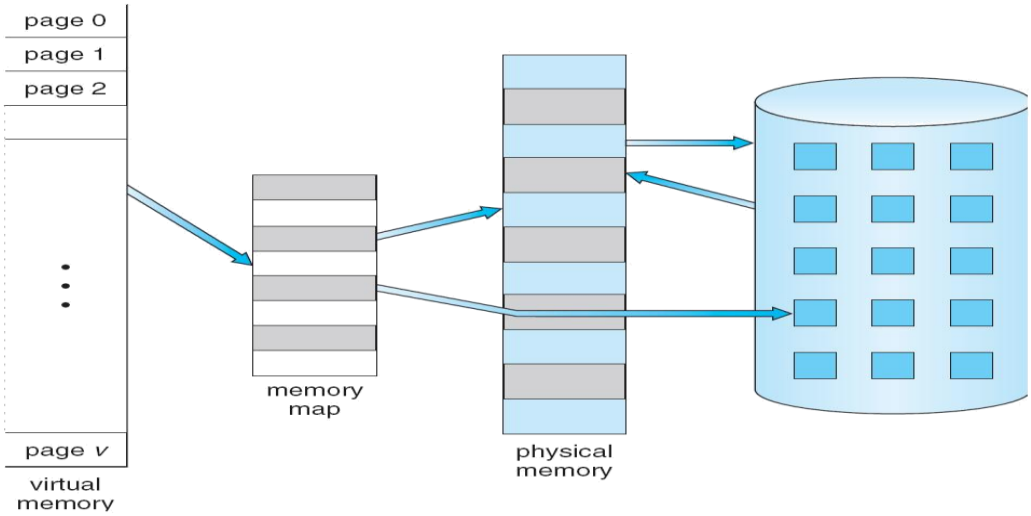


Fig. Diagram showing virtual memory that is larger than physical memory.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.

2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

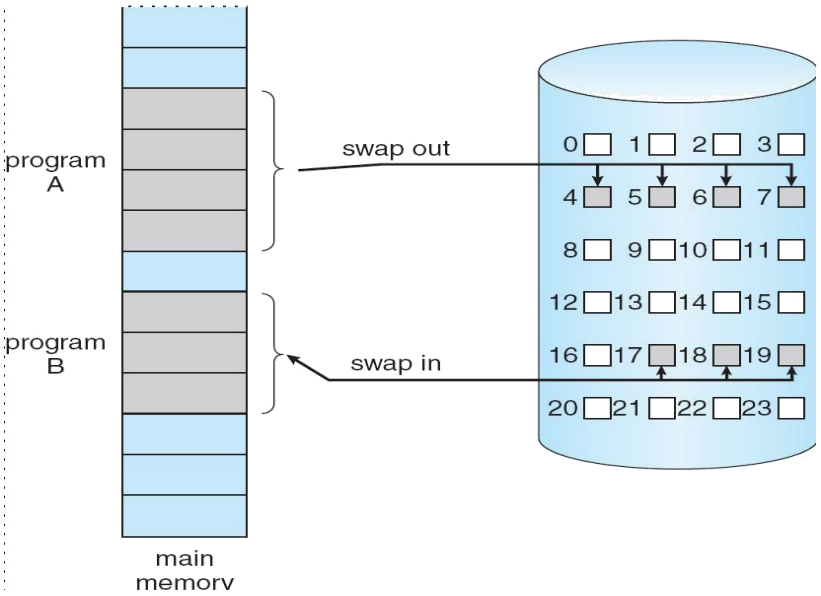
Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Fig. Transfer of a paged memory to continuous disk space



Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

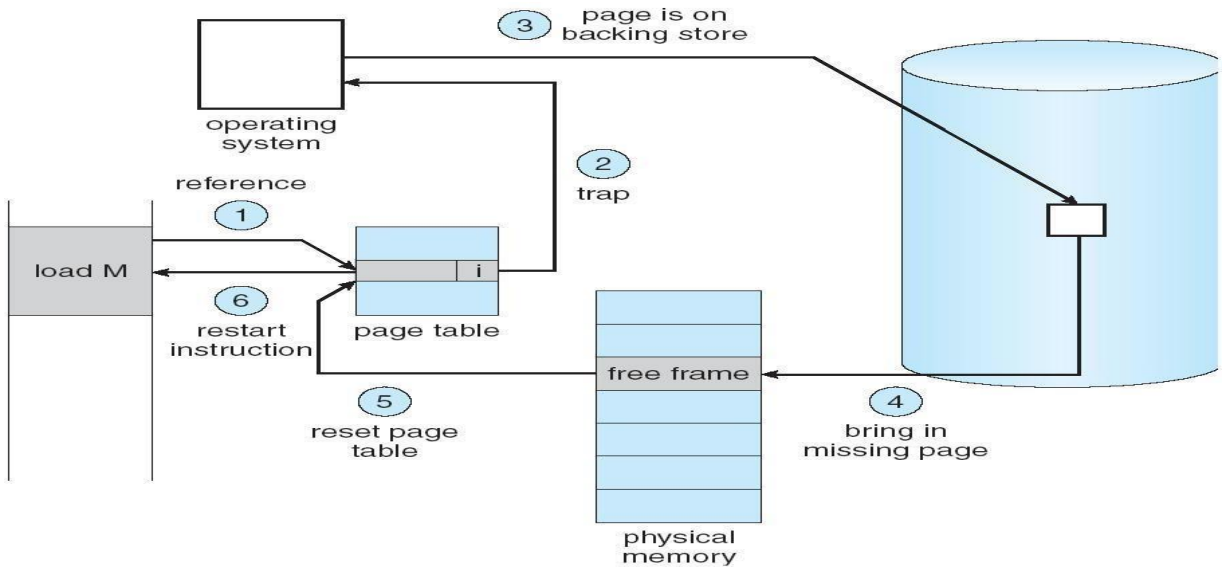
Initially only those pages are loaded which will be required the process immediately.
The pages that are not moved into the memory are marked as invalid in the page table. For

an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.

Fig. Steps in handling a page fault



6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It is not a big issue for small programs, but for larger programs it affects performance drastically.

What is dirty bit?

When a bit is modified by the CPU and not written back to the storage, it is called as a dirty bit. This bit is present in the memory cache or the virtual storage space.

Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

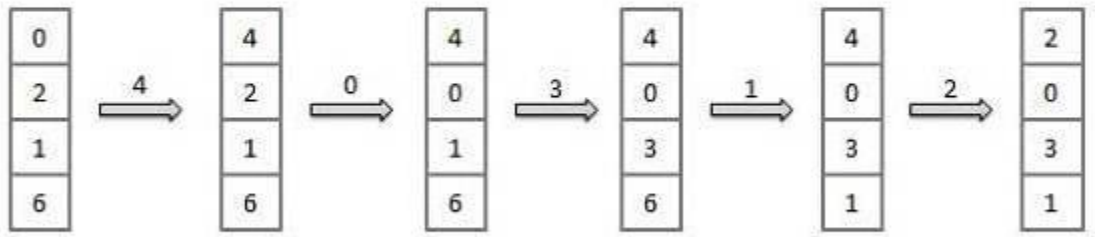
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0 First In First Out (FIFO) algorithm
- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

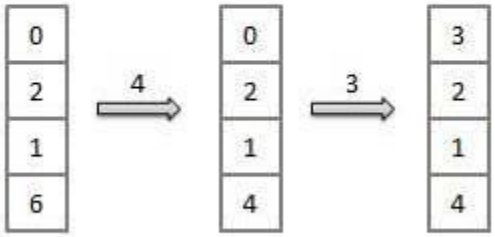
Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



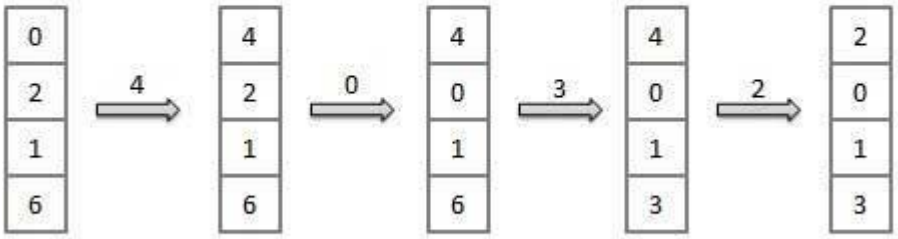
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



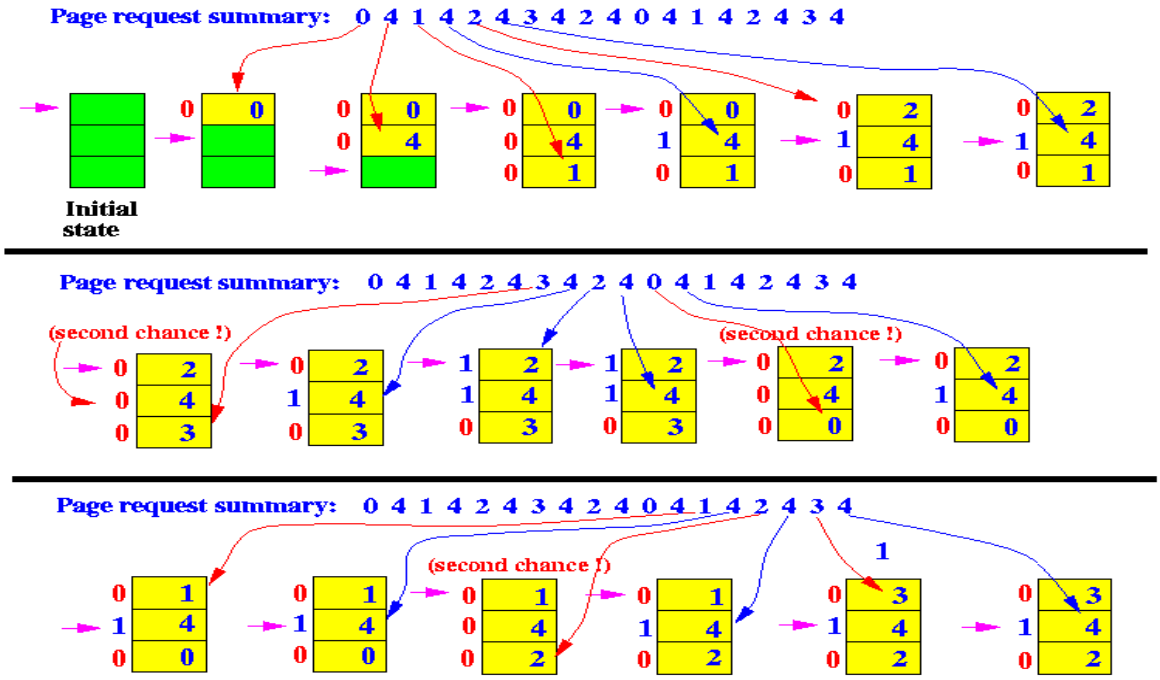
Fault Rate = 8 / 12 = 0.67

Second chance page replacement algorithm

- **Second Chance** replacement policy is called the **Clock** replacement policy...
- In the Second Chance page replacement policy, the candidate pages for removal are **consider** in a round robin matter, and a page that has been **accessed between consecutive considerations** will not be replaced.

The page replaced is the one that - considered in a round robin matter - has not been accessed since its last consideration.

- Implementation:
  - Add a "second chance" bit to each memory frame.
  - Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
  - A new page read into a memory frame has the second chance bit set to ZERO (0)
  - When you need to find a page for removal, look in a round robin manner in the memory frames:
    - If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
    - If the second chance bit is ZERO, replace the page in that memory frame.
- The following figure shows the behavior of the program in paging using the Second Chance page replacement policy:



  - We can see notably that the **bad** replacement decision made by FIFO is **not present** in Second chance!!!
  - There are a total of **9 page read operations** to satisfy the total of 18 page requests - just as good as the more computationally expensive LRU method !!!

NRU (Not Recently Used) Page Replacement Algorithm - This algorithm requires that each page have two additional status bits 'R' and 'M' called reference bit and change bit respectively. The reference bit(R) is automatically set to 1 whenever the page is referenced. The change bit (M) is set to 1 whenever the page is modified. These bits are stored in the PMT and are updated on every memory reference. When a page fault occurs, the memory manager inspects all the pages and divides them into 4 classes based on R and M bits.

- **Class 1: (0,0)** – neither recently used nor modified - the best page to replace.
- **Class 2: (0,1)** – not recently used but modified - the page will need to be written out before replacement.
- **Class 3: (1,0)** – recently used but clean - probably will be used again soon.
- **Class 4: (1,1)** – recently used and modified - probably will be used again, and write out will be needed before replacing it.

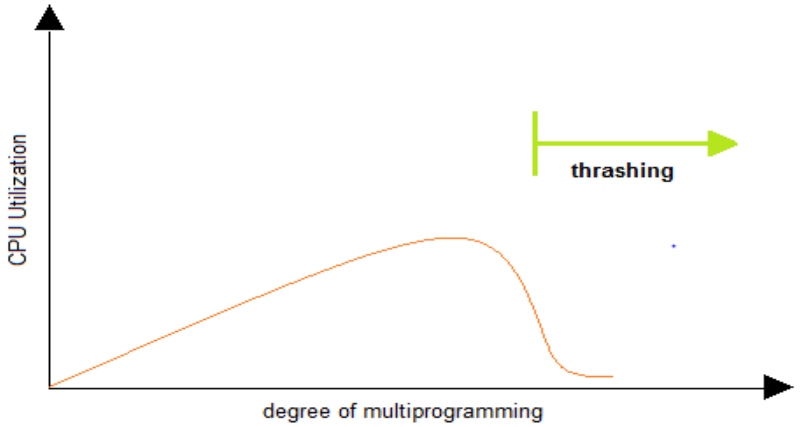
This algorithm removes a page at random from the lowest numbered non-empty class.

Thrashing

If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process' execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.

In fact, look at any process that does not have "enough" frames. Although it is technically possible to reduce the number of allocated frames to the minimum, there is some (larger) number of pages in active use. If the process does not have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again. The process continues to fault, replacing pages for which it then faults and brings back in right away.

This high paging activity is called **thrashing**. A process is thrashing if it is spending more time paging than executing.



UNIT-IV

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), Free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance.

I/O Hardware: I/O devices, Device controllers, Direct memory access Principles of I/O Software: Goals of Interrupt handlers, Device drivers, Device independent I/O software.

File System

File Concept:

Computers can store information on various storage media such as, magnetic disks, magnetic tapes, optical disks. The physical storage is converted into a logical storage unit by operating system. The logical storage unit is called FILE. A file is a collection of similar records. A record is a collection of related fields that can be treated as a unit by some application program. A field is some basic element of data. Any individual field contains a single value. A data base is collection of related data.

Student	Marks	Marks	Fail/Pas
KUMA	85	86	P
LAKSH	93	92	P

DATA FILE

Student name, Marks in sub1, sub2, Fail/Pass is fields. The collection of fields is called a RECORD. **RECORD:**

LAKSH	93	92	P

Collection of these records is called a data file.

FILE ATTRIBUTES :

1. Name : A file is named for the convenience of the user and is referred by its name. A name is usually a string of characters.
2. Identifier : This unique tag, usually a number ,identifies the file within the file system.
3. Type : Files are of so many types. The type depends on the extension of the file.

Example:
.exe Executable file
.obj Object file
.src Source file

4. Location : This information is a pointer to a device and to the location of the file on that device.

5. Size : The current size of the file (in bytes, words,blocks).
6. Protection : Access control information determines who can do reading, writing, executing and so on.
7. Time, Date, User identification : This information may be kept for creation, last modification,last use.

FILE OPERATIONS

1. Creating a file : Two steps are needed to create a file. They are:
   - *Check whether the space is available ornot.*
   - *If the space is available then made an entry for the new file in the directory. The entry includes name of the file, path of the file,etc…*
2. Writing a file : To write a file, we have to know 2 things. One is name of the file and second is the information or data to be written on the file, the system searches the entired given location for the file. If the file is found, the system must keep a write pointer to the location in the file where the next write is to take place.
3. Reading a file : To read a file, first of all we search the directories for the file, if the file is found, 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.
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. This operation is also called file seek.
5. Deleting a file : To delete a file, first of all search the directory for named file, then released the file space and erase the directoryentry.
6. Truncating a file : To truncate a file, remove the file contents only but, the attributes are as itis.

FILE TYPES: The name of the file split into 2 parts. One is name and second is Extension. The file type is depending on extension of the file.

File Type	Extension	Purpose
Executable	.exe .com .bin	Ready to run (or) ready to run machine
Source code	.c .cpp .asm	Source code in various languages.
Object	.obj .o	Compiled, machine
Batch	.bat .sh	Commands to the command

Text	.txt .doc	Textual data, documents
Word processor	.doc .wp .rtf	Various word processor formats
Library	.lib .dll	Libraries of routines for
Print or View	.pdf .jpg	Binary file in a format for
Archive	.arc .zip	Related files grouped into a
Multimedia	.mpeg .mp3 .avi	Binary file containing audio or audio/video

FILE STRUCTURE

File types also can be used to indicate the internal structure of the file. 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. If OS supports multiple file structures, the resulting size of OS is large. If the OS defines 5 different file structures, it needs to contain the code to support these file structures. All OS must support at least one structure that of an executable file so that the system is able to load and run programs.

INTERNAL FILE STRUCTURE

In UNIX OS, defines all files to be simply stream of bytes. Each byte is individually addressable by its offset from the beginning or end of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks, say 512 bytes per block.

The logical record size, physical block size, packing determines how many logical records are in each physical block. The packing can be done by the user's application program or OS. A file may be considered a sequence of blocks. If each block were 512 bytes, a file of 1949 bytes would be allocated 4 blocks (2048 bytes). The last 99 bytes

would be wasted. It is called internal fragmentation all file systems suffer from internal fragmentation, the larger the block size, the greater the internal fragmentation.

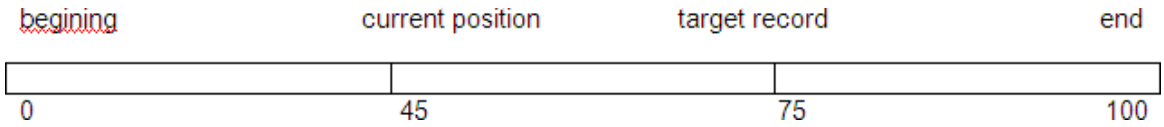
FILE ACCESS METHODS

Files stores information, this information must be accessed and read into computer memory. There are so many ways that the information in the file can be accessed.

1. Sequential file access:

Information in the file is processed in order i.e. one record after the other.
Magnetic tapes are supporting this type of file accessing.

Eg : A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record then, it access sequentially from 45, 46, 47
…….. 74, 75. So the read/write head traverse all the records between 45 to 75.



2. Direct access:

Direct access is also called relative access. Here records can read/write randomly without any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block.

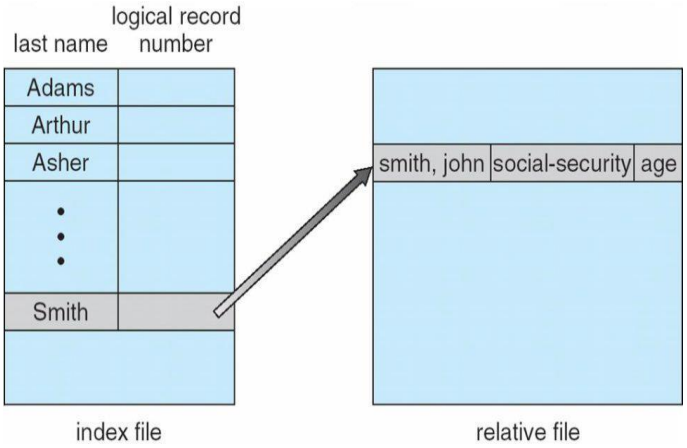
Eg : A disk containing of 256 blocks, the position of read/write head is at 95th block. The block is to be read or write is 250th block. Then we can access the 250th block directly without any restrictions.

Eg : CD consists of 10 songs, at present we are listening song 3, If we want to listen song 10, we can shift to 10.

3. Indexed Sequential File access

The main disadvantage in the sequential file is, it takes more time to access a Record .Records are organized in sequence based on a key field.
Eg :
A file consisting of 60000 records,the master index divide the total records into 6 blocks, each block consisiting of a pointer to secondary index.The secondary index divide the 10,000 records into 10 indexes.Each index consisting of a pointer to its orginal

location.Each record in the index file consisting of 2 field, A key field and a pointer field.



DIRECTORY STRUCTURE

Sometimes the file system consisting of millions of files,at that situation it is very hard to manage the files. To manage these files grouped these files and load one group into one partition.

Each partition is called a directory .a directory structure provides a mechanism for organizing many files in the file system.

OPERATION ON THE DIRECTORIES :

1. Search for a file : Search a directory structure for requiredfile.

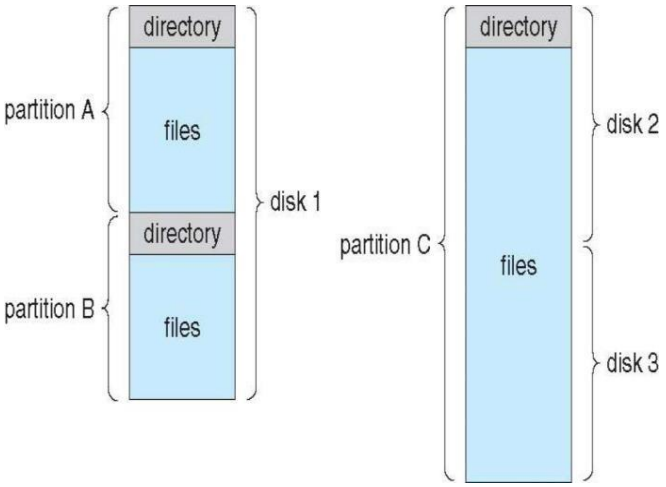
2. createafile : New files need to be created, added to thedirectory.

3. Deleteafile : When a file is no longer needed,we want to remove it fromthe directory.

4. List adirectory : We can know the list of files in thedirectory.

5. Renameafile : When ever we need to change the name of the file,wecanchange thename.

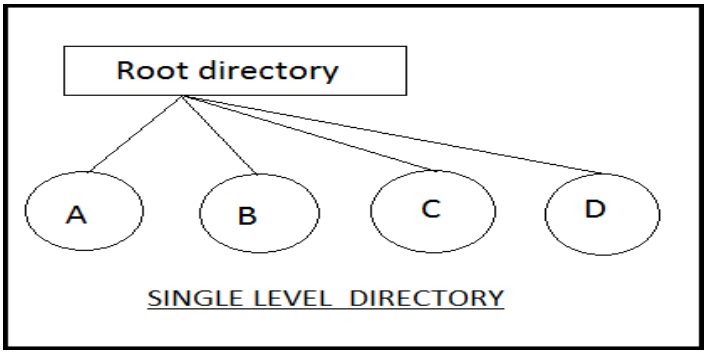
6. Traverse the file system : We need to access every directory and every file with in a directory structure we can traverse the file system



The various directory structures

1. Single level directory:

The directory system having only one directory,it consisting of all files some times it is said to be root directory.



E.g :- Here directory containing 4 files (A,B.C,D).the advantage of the scheme is its simplicity and the ability to locate files quickly.The problem is different users may accidentally use the same names for their files.

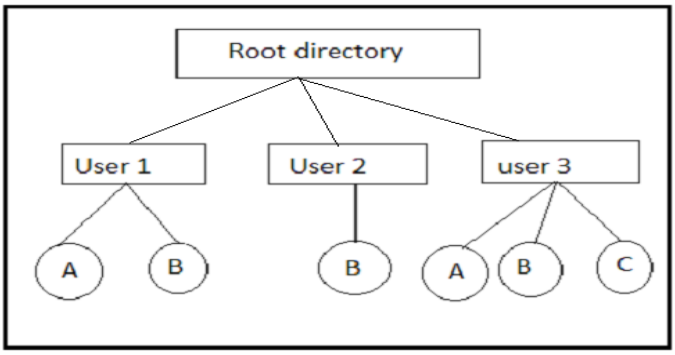
E.g :- If user 1 creates a files caled sample and then later user 2 to creates a file called sample,then user2's file will overwrite user 1 file.Thats why it is not used in the multi user system.

2. Two level directory:

The problem in single level directory is different user may be accidentally use

the same name for their files. To avoid this problem each user need a private directory,

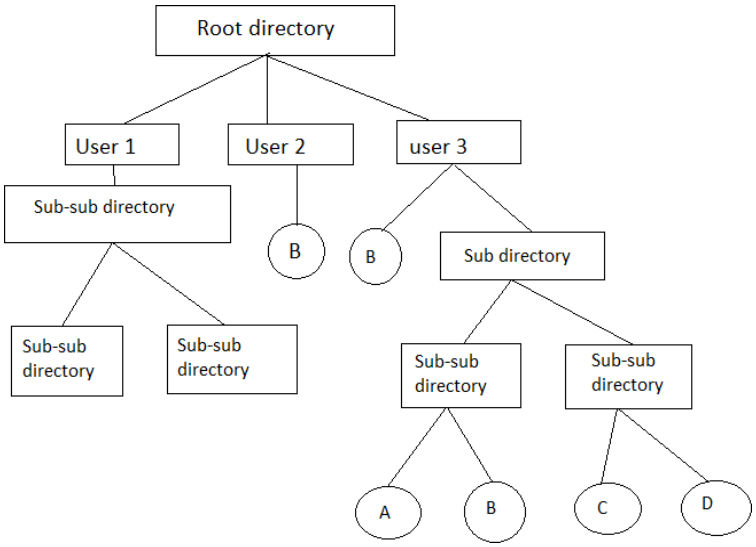
Names chosen by one user don't interfere with names chosen by a different user.



Root directory is the first level directory.user 1,user2,user3 are user level of directory A,B,C are files.

3. Tree structured directory:

Two level directory eliminates name conflicts among users but it is not satisfactory for users with a large number of files.To avoid this create the sub-directory and load the same type of files into the sub-directory.so, here each can have as many directories are needed.



There are 2 types of path

1. Absoulte path
2. Relative path

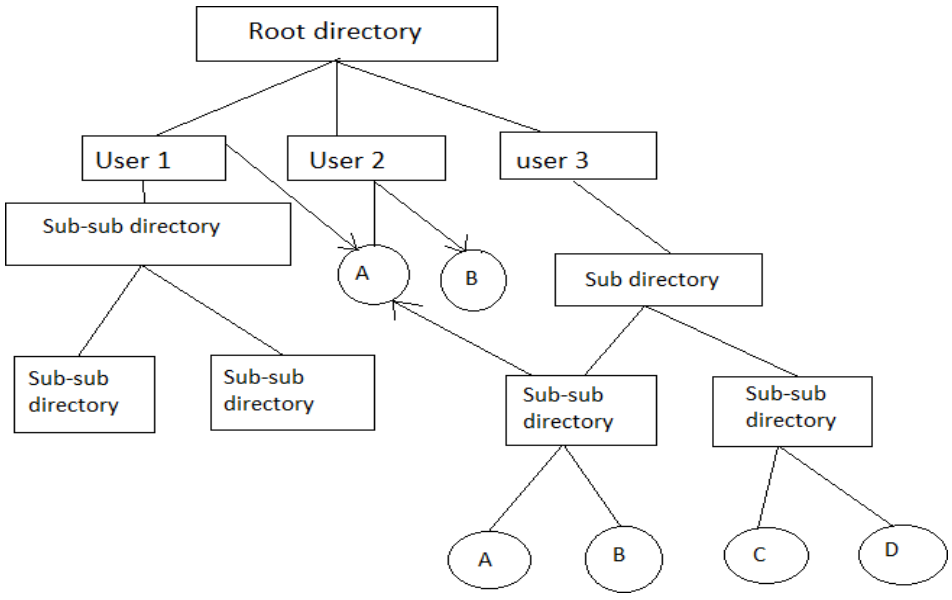
Absoulte path : Begging with root and follows a path down to specified files giving directory, directory name on the path.
Relative path : A path from current directory.

4. Acyclic graphdirectory

Multiple users are working on a project, the project files can be stored in a comman sub-directory of the multiple users. This type of directory is called acyclic graph directory .The common directory will be declared a shared directory. The graph contain no cycles with shared files, changes made by one user are made visible to other users.A file may now have multiple absolute paths. when shared directory/file is deleted, all pointers to the directory/ files also to be removed.

5. General graph directory:

When we add links to an existing tree structured directory, the tree structure is destroyed, resulting is a simple graph structure.

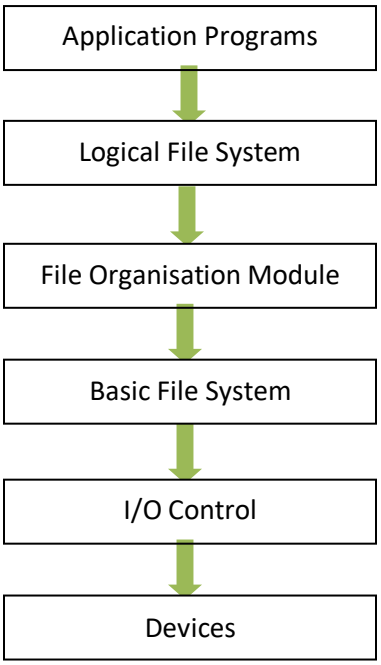


Advantages :- Traversing is easy. Easy sharing is possible.

File system structure:

Disk provides the bulk of secondary storage on which a file system is maintained. They have 2 characteristics that make them a convenient medium for storing multiple files.

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 same place.
2. A disk can access directly any block of information it contains.



I/O Control: consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system. The device driver writes specific bit patterns to special locations in the I/O 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 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 (Eg. Drive 1, cylinder 73, track2, sector 10).

The File Organization Module knows about files and their logical blocks and physical blocks. By knowing the type of file allocation used and the location of the file, file organization module can translate logical block address to physical addresses for the basic file system to transfer. Each file's logical blocks are numbered from 0 to n. so, physical blocks containing the data usually do not match the logical numbers. A translation is needed to locate each block.

The Logical File System manages all file system structure except the actual data (contents of file). It maintains file structure via file control blocks. A file control block (inode in Unix file systems) contains information about the file, ownership, permissions, location of the file contents.

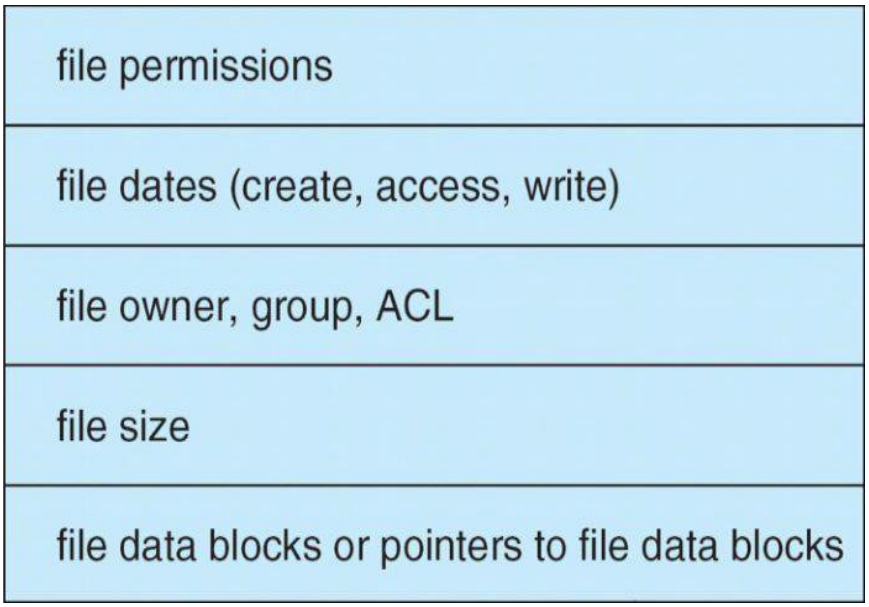
File System Implementation:

Overview:

A Boot Control Block (per volume) can contain information needed by the system to boot an OS from that volume. If the disk does not contain an OS, this block can be empty.

A Volume Control Block (per volume) contains volume (or partition) details, such as number of blocks in the partition, size of the blocks, a free block, count and free block pointers, free FCB count, FCB pointers.

A Typical File Control Block



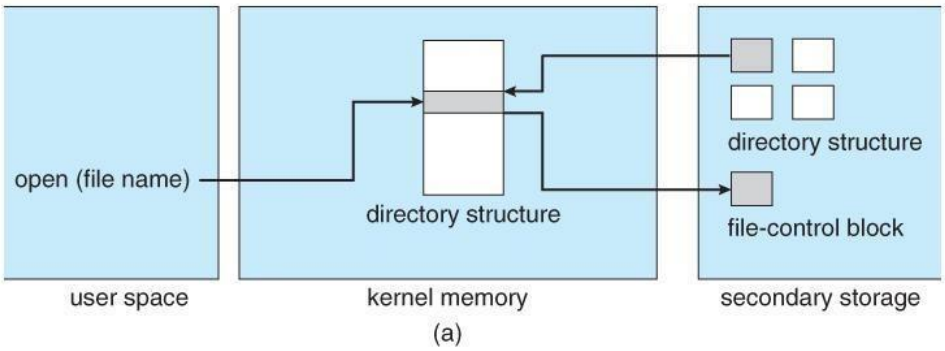
A Directory Structure (per file system) is used to organize the files. A PER-FILE FCB contains many details about the file.

A file has been created; it can be used for I/O. First, it must be opened. The open( ) call passes a file name to the logical file system. The open( ) system call 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*. If the file is not already open, the directory structure is searched for the given file name. Once the file is found, FCB is copied into a system

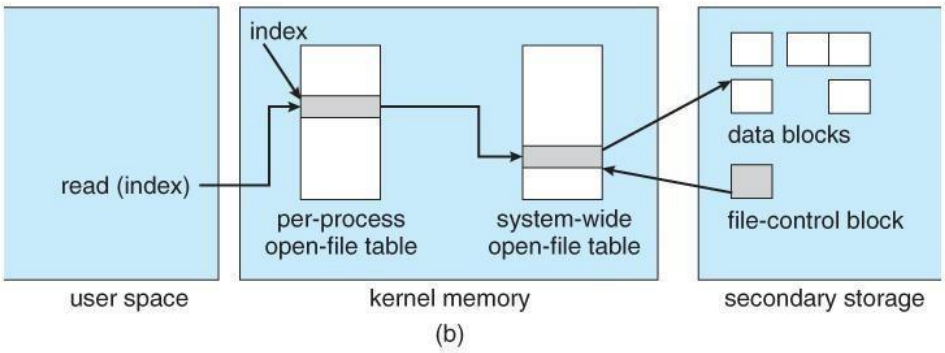
wide open file table in memory. This table not only stores the FCB but also tracks the number of processes that have the file open.

Next, an entry is made in the per – process open file table, with the pointer to the entry in the system wide open file table and some other fields. These are the fields include a pointer to the current location in the file (for the next read/write operation) and the access mode in which the file is open. The open () call returns a pointer to the appropriate entry in the per-process file system table. All file operations are preformed via this pointer. When a process closes the file the per- process table entry is removed. And the system wide entry open count is decremented. When all users that have opened the file close it, any updated metadata is copied back to the disk base directory structure. System wide open file table entry is removed.

System wide open file table contains a copy of the FCB of each open file, other information. Per process open file table, contains a pointer to the appropriate entry in the system wide open file table, other information.



(a)



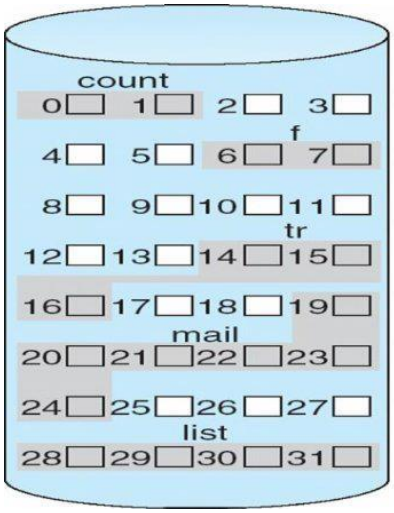
(b)

Allocation Methods – Contiguous

An allocation method refers to how disk blocks are allocated for files:

Contiguous allocation – each file occupies set of contiguous blocks o Best performance in most cases

- Simple – only starting location (block #) and length (number ofblocks) are required
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**



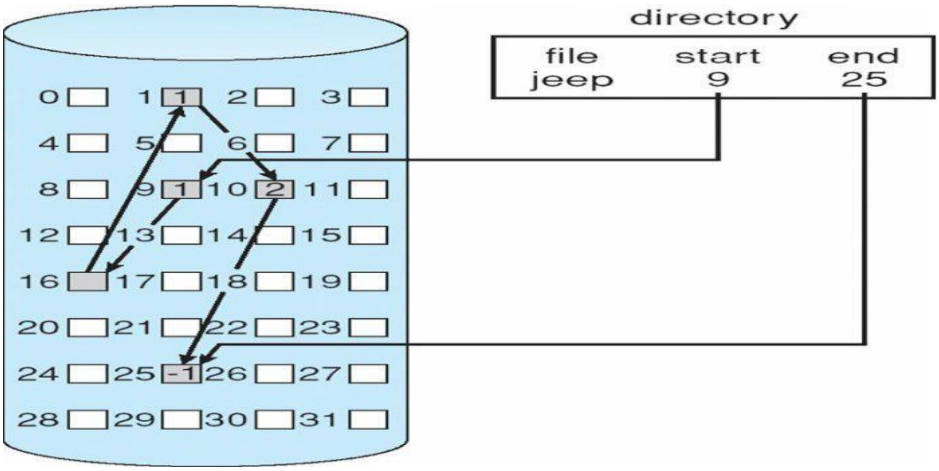
directory

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

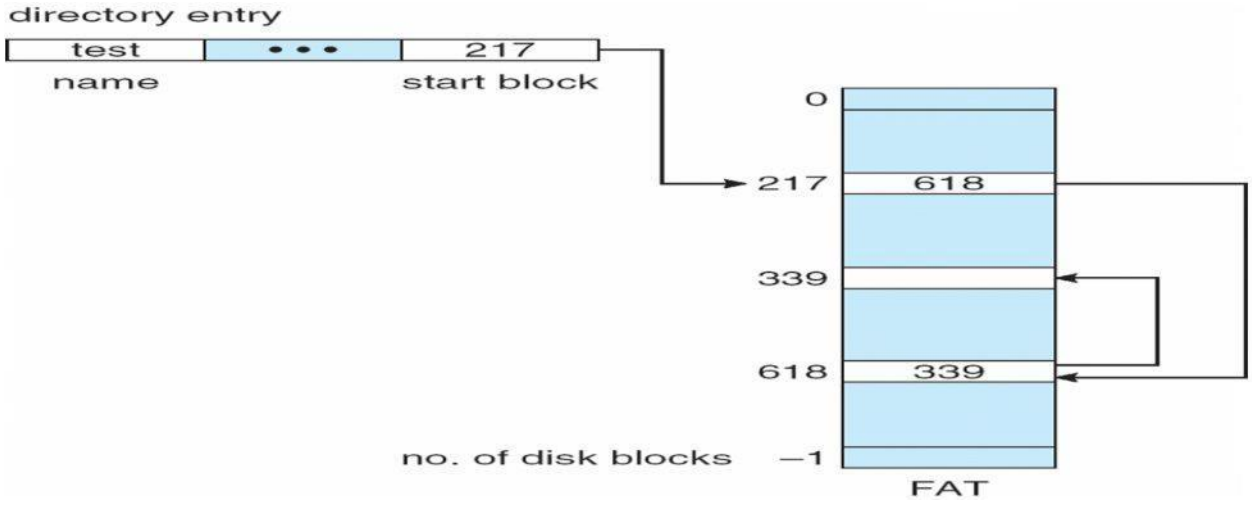
Linked

Linked allocation – each file a linked list of blocks o File ends at nil pointer

- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks FAT (File Allocation Table) variation
- Beginning of volume has table, indexed by block number
- Much like a linked list, but faster on disk and cacheable

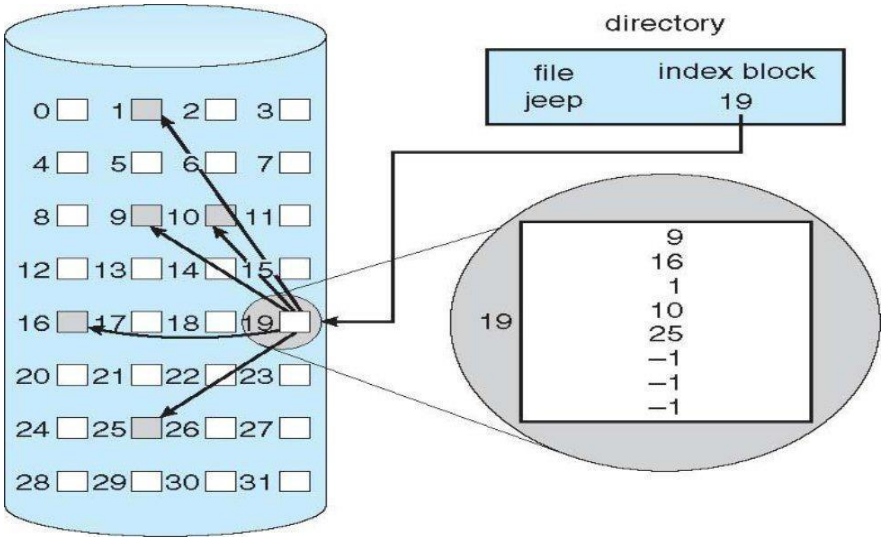


File-Allocation Table



Indexed allocation

- Each file has its own **index block**(s) of pointers to its data blocks



Free-Space Management

File system maintains **free-space list** to track available blocks/clusters Linked list (free list)

- o Cannot get contiguous space easily
- o No waste of space
- o No need to traverse the entire list

1. **Bitmap or Bit vector –**

A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block. The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

Advantages –

- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

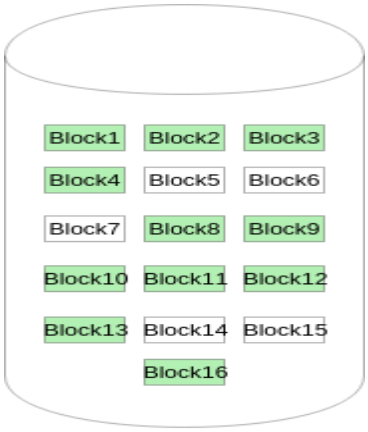
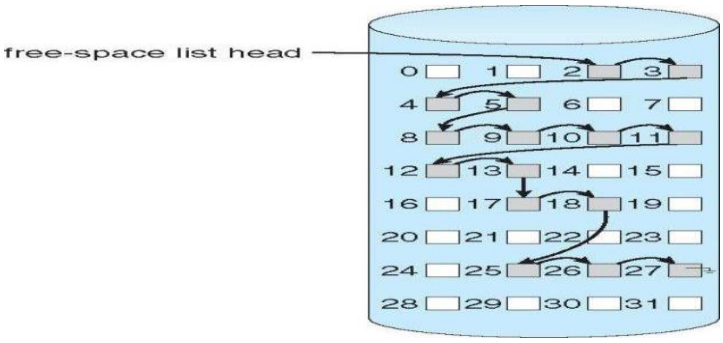


Figure - 1

Linked Free Space List on Disk



In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

Grouping

Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one).

An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily

Counting

Because space is frequently contiguously used and freed, with contiguous- allocation allocation, extents, or clustering.

Keep address of first free block and count of following free blocks. Free space list then has entries containing addresses and counts.

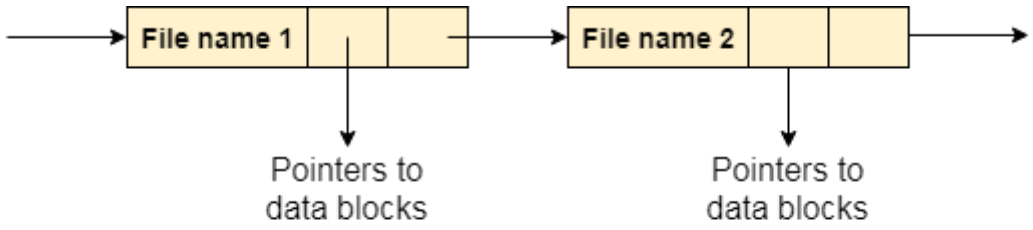
Directory Implementation

1. Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

Characteristics

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



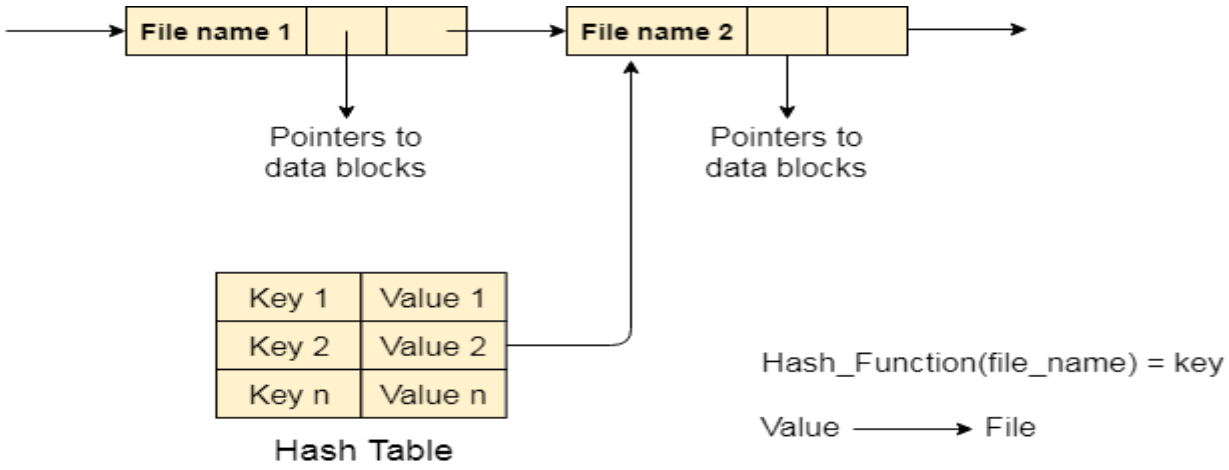
Linear List

2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



Efficiency and Performance

Efficiency dependent on:

- Disk allocation and directory algorithms
- Types of data kept in file's directory entry

Performance

- Disk cache – separate section of main memory for frequently used blocks
- free-behind and read-ahead – techniques to optimize sequential access
- improve PC performance by dedicating section of memory as virtual disk, or RAM disk

I/O Hardware: I/O devices

Input/output devices are the devices that are responsible for the input/output operations in a computer system.

Basically there are following two types of input/output devices:

- Block devices
- Character devices

Block Devices

A block device stores information in block with fixed-size and own-address.

It is possible to read/write each and every block independently in case of block device.

In case of disk, it is always possible to seek another cylinder and then wait for required block to rotate under head without mattering where the arm currently is. Therefore, disk is a block addressable device.

Character Devices

A character device accepts/delivers a stream of characters without regarding to any block structure.

Character device isn't addressable.

Character device doesn't have any seek operation.

There are too many character devices present in a computer system such as printer, mice, rats, network interfaces etc. These four are the common character devices.

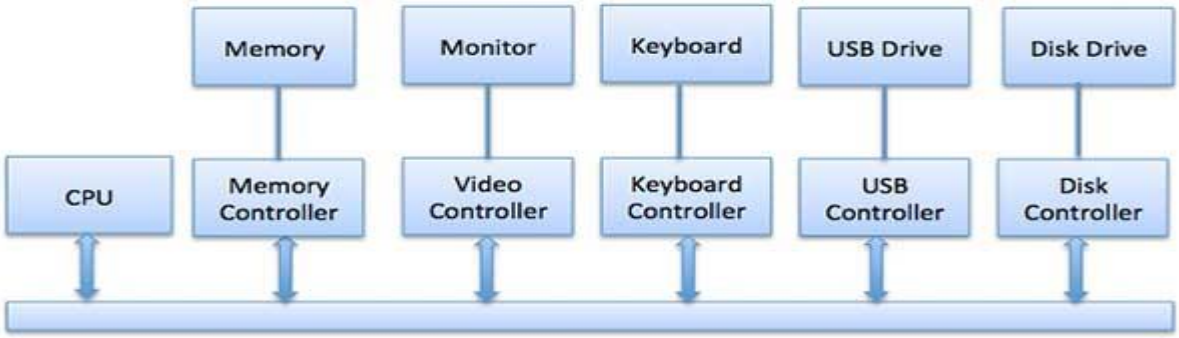
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

- **Synchronous I/O** − In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** − I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

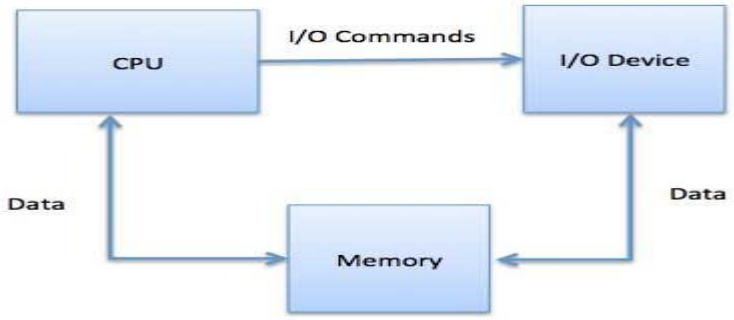
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

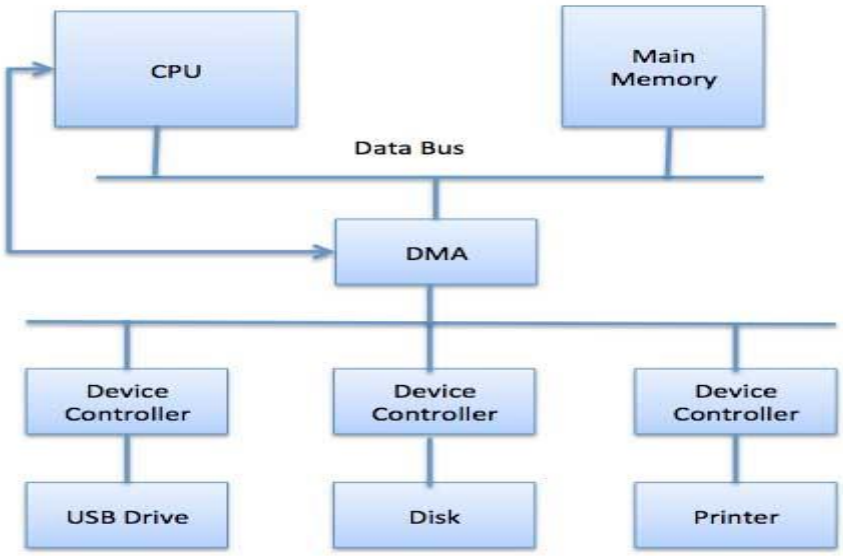
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



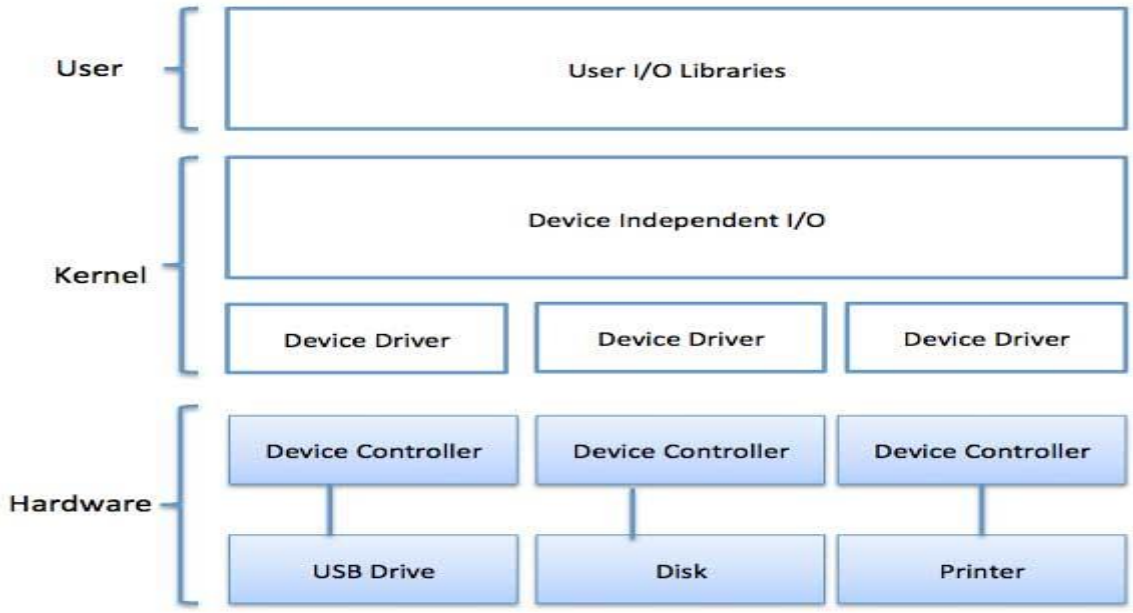
The operating system uses the DMA hardware as follows −

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

I/O software is often organized in the following layers −

- **User Level Libraries** − This provides simple interface to the user program to perform input and output. For example, **stdio** is a library provided by C and C++ programming languages.
- **Kernel Level Modules** − This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
- **Hardware** − This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.



Device Drivers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs −

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback functions in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architecture, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to

write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software −

- Uniform interfacing for device drivers
- Device naming - Mnemonic names mapped to Major and Minor device numbers
- Device protection
- Providing a device-independent block size
- Buffering because data coming off a device cannot be stored in final destination.
- Storage allocation on block devices
- Allocation and releasing dedicated devices
- Error Reporting

User-Space I/O Software

These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

Kernel I/O Subsystem

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

- **Scheduling** − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering** − Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
- **Caching** − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **Spooling and Device Reservation** − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.
- **Error Handling** − An operating system that uses protected memory can guard against many kinds of hardware and application errors.

UNIT-V

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks.

DEADLOCKS

System model:

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, I/O devices are examples of resource types. If a system has 2 CPUs, then the resource type CPU has 2 instances.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its task. The number of resources as it requires to carry out its task. The number of resources requested may not exceed the total number of resources available in the system. A process cannot request 3 printers if the system has only two.

A process may utilize a resource in the following sequence:

(I) REQUEST: The process requests the resource. If the request cannot be granted immediately (if the resource is being used by another process), then therequesting process must wait until it can acquire theresource.

(II) USE: The process can operate on the resource .if the resource is a printer, the process can print on theprinter.

(III) RELEASE: The process release theresource.

For each use of a kernel managed by a process the operating system checks that the process has requested and has been allocated the resource. A system table records whether each resource is free (or) allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

To illustrate a deadlocked state, consider a system with 3 CDRW drives. Each of 3 processes holds one of these CDRW drives. If each process now requests another drive, the 3 processes will be in a deadlocked state. Each is waiting for the event "CDRW is released" which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

Deadlocks may also involve different resource types. Consider a system with one printer and one DVD drive. The process P_i is holding the DVD and process P_j is holding the printer. If P_i requests the printer and P_j requests the DVD drive, a deadlock occurs.

DEADLOCK CHARACTERIZATION:

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

NECESSARY CONDITIONS:

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. MUTUAL EXCLUSION: Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until theresource has beenreleased.
2. HOLD AND WAIT: A process must be holding at least one resource and waitingto acquire additional resources that are currently being held by otherprocesses.
3. NO PREEMPTION: Resources cannot be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed itstask.
4. CIRCULAR WAIT: A set {P_0,P_1,…..P_n} of waiting processes must exist such that P_0 is waiting for resource held by P_1, P_1 is waiting for a resource held by P_2,……,P_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held byP_0.

RESOURCE ALLOCATION GRAPH

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. the set of vertices V is partitioned into 2 different types of nodes:

P = {P_1, P_2….P_n}, the set consisting of all the active processes in the system. R= {R_1, R_2….R_m}, the set consisting of all resource types in the system.

A directed edge from process P_i to resource type R_j is denoted by P_i ->R_j. It signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource.

A directed edge from resource type R_j to process P_i is denoted by R_j ->P_i, it signifies that an instance of resource type R_j has been allocated to process P_i.

A directed edge P_i ->R_j is called a requested edge. A directed edge R_j->$P_i$is called an assignmentedge.

We represent each process P_i as a circle, each resource type R_j as a rectangle. Since resource type R_j may have more than one instance. We represent each such instance as a dot within the rectangle. A request edge points to only the rectangle Rj. An assignment edge must also designate one of the dots in therectangle.

When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource, as a result, the assignment edge is deleted.

The sets P, R, E:

P= {P_1, P_2, P_3}

R= {R_1, R_2, R_3, R_4}

E= {P_1 ->R_1, P_2 ->R_3, R_1 ->P_2, R_2 ->P_2, R_2 ->P_1, R_3 ->P_3}

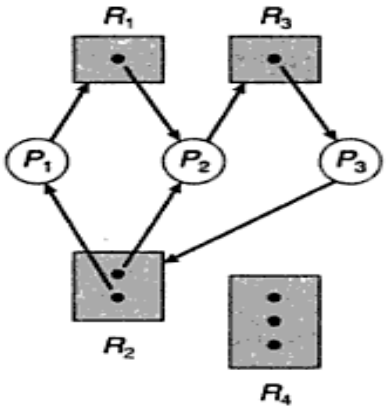


Figure Resource-allocation graph with a deadlock.

One instance of resource type R1
Two instances of resource type R2
One instance of resource type R3
Three instances of resource type R4

PROCESS STATES:

Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.

Process P2 is holding an instance of R1 and an instance of R2 and is waiting for instance of R3.

Process P3 is holding an instance of R3.

If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 ->R2 is added to the graph.

2 cycles:

P1 ->R1 ->P2 ->R3 ->P3 ->R2 ->P1

P2 ->R3 ->P3 ->R2 ->P2

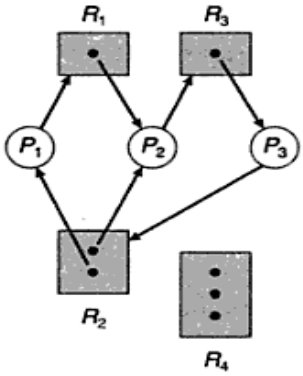


Figure Resource-allocation graph with a deadlock.

Processes P1, P2, P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3.process P3 is waiting for either process P1 (or) P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.

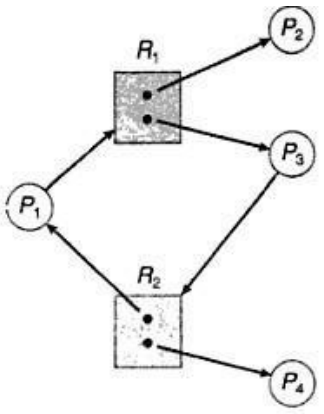


Figure Resource-allocation graph with a cycle but no deadlock.

We also have a cycle: P1 ->R1 ->P3 ->R2 ->P1

However there is no deadlock. Process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

DEADLOCK PREVENTION

For a deadlock to occur, each of the 4 necessary conditions must held. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

Mutual Exclusion – not required for sharable resources; must hold for non sharable resources

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources

- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
- Low resource utilization; starvation possible

No Preemption –

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- Preempted resources are added to the list of resources for which the process is waiting
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

Requires that the system has some additional *a priori* information available

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes .

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in **safe state** if there exists a sequence <*P1, P2, …, Pn*> of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the *Pj*, with *j* <*I*

That is:

- If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished
- When *Pj* is finished, *Pi* can obtain needed resources, execute, return allocated resources, and terminate
- When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on If a system is in safe state no deadlocks

If a system is in unsafe state possibility of deadlock

Avoidance ensure that a system will never enter an unsafe state

Avoidance algorithms

Single instance of a resource type

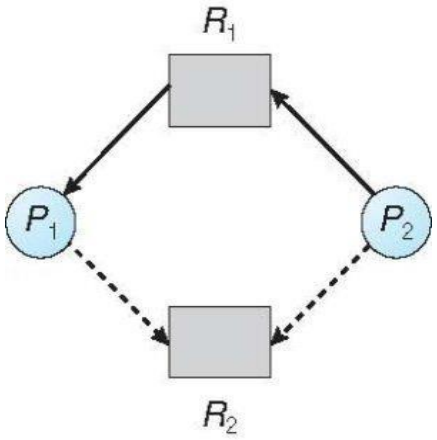
- Use a resource-allocation graph Multiple instances of a resource type
- Use the banker's algorithm

Resource-Allocation Graph Scheme

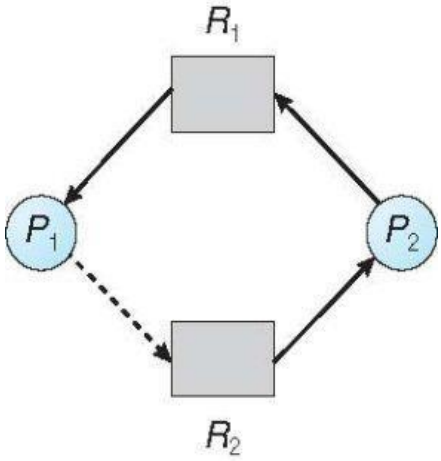
Claim edge*PiÆRj* indicated that process *Pj* may request resource *Rj*; represented by a dashed line

Claim edge converts to request edge when a process requests a resource

Request edge converted to an assignment edge when the resource is allocated to the process When a resource is released by a process, assignment edge reconverts to a claim edge Resources must be claimed *a priori* in the system



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

Multiple instances

Each process must a priori claim maximum use

When a process requests a resource it may have to wait

When a process gets all its resources it must return them in a finite amount of time Let *n* = number of processes, and *m* = number of resources types.

Available*:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type *Rj*available

Max*: n x m* matrix. If *Max* [*i,j*] = *k*, then process *Pi*may request at most *k* instances of resource type *Rj*

Allocation*: n* x *m* matrix. If Allocation[*i,j*] = *k* then *Pi* is currently allocated *k* instances of *Rj*

Need*: n* x *m* matrix. If *Need*[*i,j*] = *k*, then *Pi* may need *k* more instances of *Rj*to complete its task

Need [*i,j]* = *Max*[*i,j*] – *Allocation* [*i,j*]

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize: Work = Available

Finish [i] = false fori = 0, 1, …,n- 1
2. Find an isuch that both:
(a) Finish [i] = false
(b) Needi=Work
If no such iexists, go to step 4
3. Work = Work + Allocation$_i$
Finish[i] = true
go to step 2
4. IfFinish [i] == true for all i, then the system is in a safe state

Resource-Request Algorithm for Process *Pi*

Request = request vector for process *Pi*. If *Requesti*[*j*] = *k* then process *Pi* wants *k* instances of resource type *Rj*

1. If *Requesti*£*Needi*go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If *Requesti*£*Available*, go to step 3. Otherwise *Pi* must wait, since resources are not available
3. Pretend to allocate requested resources to *Pi* by modifying the state as follows:

Available = Available – Request;
Allocationi= Allocationi + Requesti;
Needi=Needi – Requesti;

- *If safe the resources are allocated to Pi*
- *If unsafe Pi must wait, and the old resource-allocation state is restored*

Example of Banker's Algorithm(REFER CLASS NOTES)

consider 5 processes *P*0 through *P*4; 3 resource types:
A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time *T*0:

Allocation	*Max*	*Available*
A B C	*A B C*	*A B C*
*P*0 0 1 0	7 5 3	3 3 2
*P*1 2 0 0	3 2 2	
*P*2 3 0 2	9 0 2	
*P*3 2 1 1	2 2 2	
*P*4 0 0 2	4 3 3	

Σ The content of the matrix *Need* is defined to be *Max* – *Allocation Need*

A B C

The system is in a safe state since the sequence <*P*1, *P*3, *P*4, *P*2, *P*0>

satisfies safety criteria

***P*1 Request (1,0,2)**

Check that Request £ Available (that is, (1,0,2) £ (3,3,2) true

Allocation	*Need*	*Available*
A B C	*A B C*	*A B C*
*P*0 0 1 0	7 4 3	2 3 0
*P*1 3 0 2	0 2 0	
*P*2 3 0 2	6 0 0	
*P*3 2 1 1	0 1 1	
*P*4 0 0 2	4 3 1	

Executing safety algorithm shows that sequence <*P*1, *P*3, *P*4, *P*0, *P*2> satisfies safety requirement

Deadlock Detection

Allow system to enter deadlock state

Detection algorithm

Recovery scheme

Single Instance of Each Resource Type

Maintain *wait-for* graph

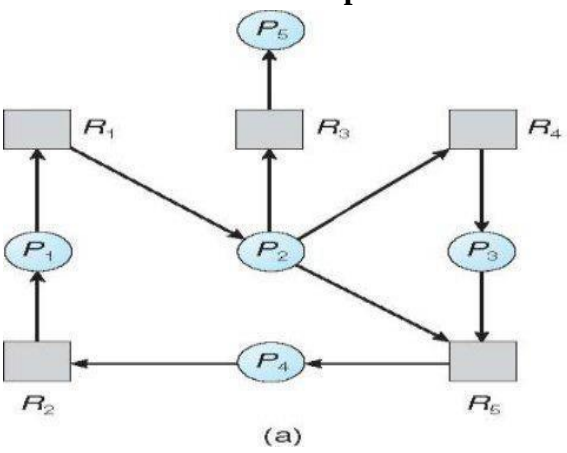
Nodes are processes *PiÆP*

*j*if *Pi*is waiting for*Pj*

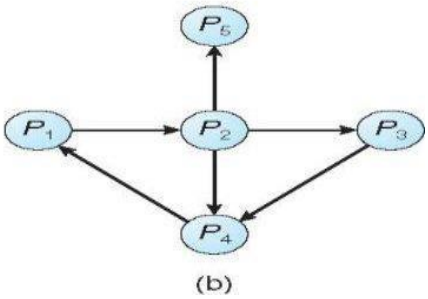
Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

An algorithm to detect a cycle in a graph requires an order of *n*2 operations, where *n* is the number of vertices in the graph

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

Several Instances of a Resource Type

Available*:* A vector of length *m* indicates the number of available resources of each type. **Allocation***:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

Request*:* An *n* x *m* matrix indicates the current request of each process.

If *Request* [*i*][*j*] = *k*, then process *Pi* is requesting *k* more instances of resource type.*Rj*.

Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if Allocationiπ 0, then

Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:

(a) Finish[i] == false

(b) Requesti£Work

If no such i exists, go to step 4

3. Work = Work + Allocation$_i$

Finish[*i*] = *true*

go to step 2

4. If *Finish*[*i*] == false, for some *i*, 1 £*i*£*n*, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then *Pi* is deadlocked

Recovery from Deadlock:

Process Termination

Abort all deadlocked processes

Abort one process at a time until the deadlock cycle is eliminated In which order should we choose to abort?

- Priority of the process
- How long process has computed, and how much longer to completion
- Resources the process has used
- Resources process needs to complete
- How many processes will need to be terminated
- Is process interactive or batch?

Resource Preemption

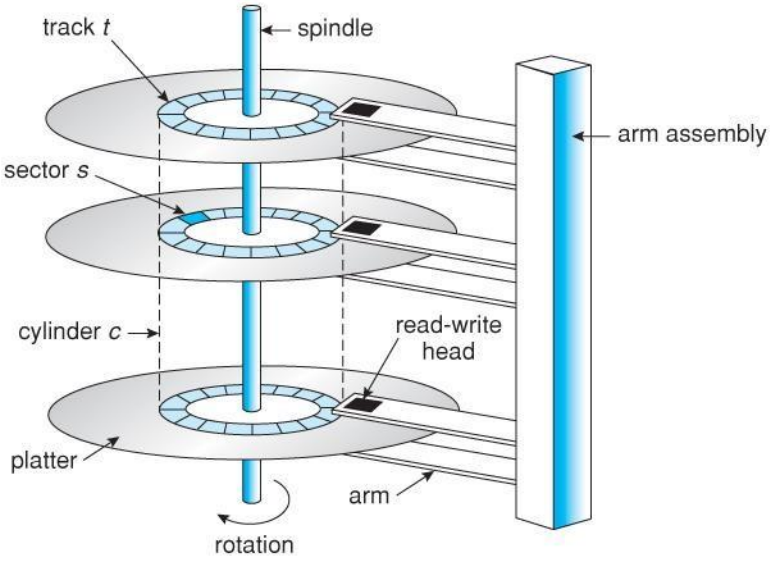
Selecting a victim – minimize cost

Rollback – return to some safe state, restart process for that state

Starvation – same process may always be picked as victim, include number of rollback in cost factor

Secondary storage structure:
Overview of mass storage structure

Magnetic disks: Magnetic disks provide the bulk of secondary storage for modern computer system. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material. We store information by it magnetically on the platters.



Moving head disk mechanism

A read /write head files just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are sub divided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.

When the disk in use, a driver motor spins it at high speed. Most drivers rotate 60 to 200 times per second. Disk speed has 2 parts. The transfer rate is the at which data flow between the drive and the computer. To read/write, the head must be positioned at the desired track and at the beginning of the desired sector on the track, the time it takes to position the head at the desired track is called seek time. Once the track is selected the disk controller waits until desired sector reaches the read/write head. The time it takes to reach the desired sector is called ***latency time or rotational dealy-access time***. When the desired sector reached the read/write head, then the real data transferring starts.

A disk can be removable. Removable magnetic disks consist of one platter, held in a plastic case to prevent damage while not in the disk drive. Floppy disks are in expensive removable magnetic disks that have a soft plastic case containing a flexible platter. The storage capacity of a floppy disk is 1.44MB.

A disk drive is attached to a computer by a set of wires called an I/O bus. The data transfer on a bus are carried out by special processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is built into each disk drive . to perform i/o operation, the host controller operates the disk drive hardware to carry out the command. Disk controllers have built in cache, data transfer at the disk drive happens b/w cache and disk surface. Data transfer at the host, occurs b/w cache and host controller.

Magnetic Tapes: magnetic tapes was used as an early secondary storage medium. It is permanent and can hold large amount of data. It access time is slow compared to main memory and magnetic disks. Tapes are mainly used for back up, for storage of infrequently used information. Typically they store 20GB to 200GB.

Disk Structure: most disks drives are addressed as large one dimensional arrays of logical blocks. The one dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. sector 0 is the fist sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinder from outermost to inner most. As we move from outer zones to inner zones, the number of sectors per track decreases. Tracks in outermost zone hold 40% more sectors then innermost zone. The number of sectors per track has been increasing as disks technology improves, and the outer zone of a disk usually has several hundred sectors per track. Similarly, the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders.

Disk attachment

Computer access disk storage is 2 ways.

1. Via I/O ports(host attachedstorage)
2. Via a remote host in a distributed file system(network attachedstorage).

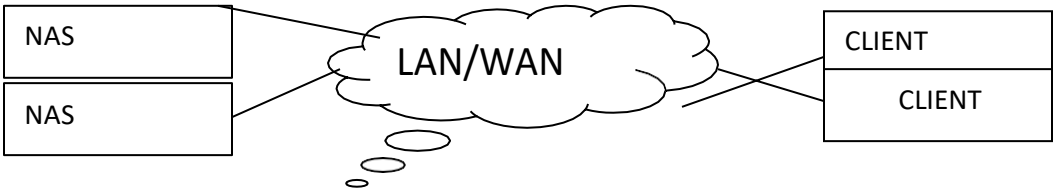
1 .Host attached storage : host attached storage are accessed via local I/O ports. The desktop pc uses an I/O bus architecture called IDE. This architecture supports maximum of 2 drives per I/O bus. High end work station and servers use SCSI and FC.

SCSI is an bus architecture which have large number of conductor's in a ribbon cable (50 or 68) scsi protocol supports maximum of 16 drives an bus. Host consists of a controller card (SCSI Initiator) and upto 15 storage device called SCSI targets.

Fc(fiber channel) is the high speed serial architecture. It operates mostly on optical fiber (or) over 4 conductor copper cable. It has 2 variants. One is a large switched fabric having a 24-bit address space. The other is an (FC-AL) arbitrated loop that can address 126 devices.

A wide variety of storage devices are suitable for use as host attached.(hard disk,cd ,dvd,tape devices)

2. Network-attached storage: A(NAS) is accessed remotely over a data network .clients access network attached storage via remote procedure calls. The rpc are carried via tcp/udp over an ip network-usually the same LAN that carries all data traffic to theclients.



NAS provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host attached storage .but it tends to be less efficient and have lower performance than direct attached storage.

3. Storage area network: The drawback of network attached storage(NAS) is storage I/O operations consume bandwidth on the data network. The communication b/w servers and clients competes for bandwidth with the communication among servers and storagedevices.

A storage area network(SAN) is a private network using storage protocols connecting servers and storage units. The power of a SAN is its flexibility. multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. SANs make it possible for clusters of server to share the same storage

Disk Scheduling Algorithms

Disk scheduling algorithms are used to allocate the services to the I/O requests on the disk . Since seeking disk requests is time consuming, disk scheduling algorithms try to minimize this latency. If desired disk drive or controller is available, request is served immediately. If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the Operating System has to choose which pending request to service next. The OS relies on the type of algorithm it needs when dealing and choosing what particular disk request is to be processed next. The objective of using these algorithms is keeping Head movements to the amount as possible. The less the head to move, the faster the seek time will be. To see how it works, the different disk scheduling algorithms will be discussed and examples are also provided for better understanding on these different algorithms.

1. First Come First Serve(FCFS)

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa . The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement (THM) of the read/write head :

95, 180, 34, 119, 11, 123, 62, 64

Consider that the read/write head is positioned at location 50. Prior to this track location 199 was serviced. Show the total head movement for a 200 track disk (0-199).

Solution:

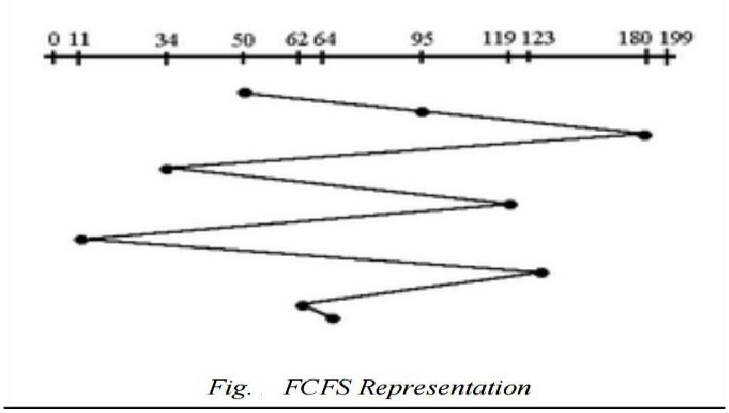


Fig. FCFS Representation

Total Head Movement Computation: (THM) =

(180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62) =

130 + 146 + 85 + 108 + 112 + 61 + 2 (THM) = 644 tracks

Assuming a seek rate of 5 milliseconds is given, we compute for the seek time using the formula: Seek Time = THM * Seek rate
=644 * 5 ms
Seek Time = 3,220 ms.

2. Shortest Seek Time First(SSTF):

This algorithm is based on the idea that that he R/W head should proceed to the track that is closest to its current position . The process would continue until all the track requests are taken care of. Using the same sets of example in FCFS the solution are as follows:

Solution:

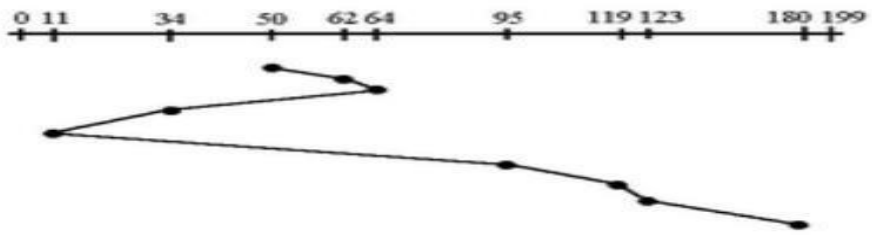


Fig. SSTF Representation

(THM) = (64-50) + (64-11) + (180-11) =

14 + 53 + 169 (THM) = 236 tracks

Seek Time = THM * Seek rate

= 236 * 5ms
Seek Time = 1,180 ms

In this algorithm, request is serviced according to the next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34 . The process would continue up to the last track request. There are a total of 236 tracks and a seek time of 1,180 ms, which seems to be

a better service compared with FCFS which there is a chance that starvation3 would take place. The reason for this is if there were lots of requests closed to each other, the other requests will never be handled since the distance will always be greater.

3. SCAN Scheduling Algorithm

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm. Using the same sets of example in FCFS the solution are as follows:

Solution:

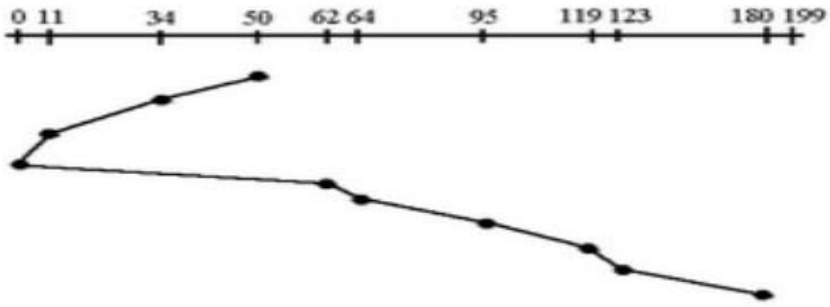


Fig. SCAN Representation

(THM) = (50-0) + (180-0)
= 50 + 180
(THM) = 230

*Seek Time = THM * Seek rate*
*= 230 * 5ms*
Seek Time = 1,150 ms

This algorithm works like an elevator does. In the algorithm example, it scans down towards the nearest end and when it reached the bottom it scans up servicing the requests that it did not get going down. If a request comes in after it has been scanned, it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks and a seek time of 1,150. This is optimal than the previous algorithm.

4 .Circular SCAN (C-SCAN)Algorithm

This algorithm is a modified version of the SCAN algorithm. C-SCAN sweeps the disk from end-to-end, but as soon it reaches one of the end tracks it then moves to the

other end track without servicing any requesting location. As soon as it reaches the other end track it then starts servicing and grants requests headed to its direction. This algorithm improves the unfair situation of the end tracks against the middle tracks. Using the same sets of example in FCFS the solution are as follows:

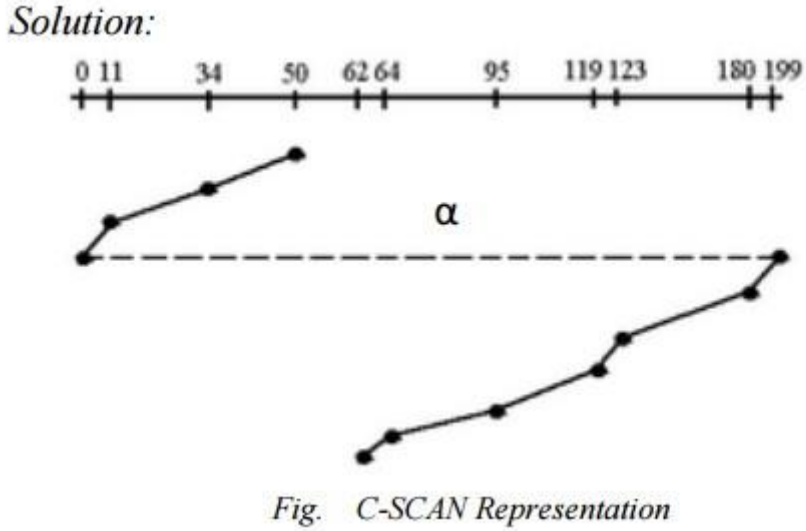


Fig. C-SCAN Representation

Notice that in this example an alpha3 symbol (α) was used to represent the dash line. This return sweeps is sometimes given a numerical value which is included in the computation of the THM . As analogy, this can be compared with the carriage return lever of a typewriter. Once it is pulled to the right most direction, it resets the typing point to the leftmost margin of the paper . A typist is not supposed to type during the movement of the carriage return lever because the line spacing is being adjusted . The frequent use of this lever consumes time, same with the time consumed when the R/W head is reset to its starting position.

Assume that in this example, α has a value of 20ms, the computation would be as follows: (THM) = (50-0) + (199-62) + α
= 50 + 137 + 20 (THM)

= 207 tracks

Seek Time = THM * Seek rate

= 187 * 5ms Seek Time = 935 ms .

The computation of the seek time excluded the alpha value because it is not an actual seek or search of a disk request but a reset of the access arm to the starting position .

Disk management

Disk formatting: A magnetic disk is a blank slate. It is just a platter of **a magnetic recording material.** before a disk can store data , it must be divided into sectors that the disk controller can read and write. This process is called low level formatting (or)physical formatting. low level formatting fills the disk with a special data structure for each sector .the Data structure **for a sector** typically consists of a header, a data **area**, a trailer . the header and trailer contain information used by the disk controller ,such as a sector number and a**n** error correcting code(ECC). When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area . when the sector is read ,the EC**C** is recalculated and compared with the stored value. If the stored and calculated **numbers** are different, this mismatch indicates that the data area of this sector has become corrupted, and that the disk **sector** may be bad. ECC contains enough information, **if** only few bits of data have been corrupted, to enable the controller to identify which bits have changed and calculate what their correct values should be. The controller automatically does the ECC processing what ever a sector is read/written for many hard disks, when the disk controller is instructed to low level format the disk, it can also be told how many bytes of data space to leave between the header and trailer of all sectors.

Before it can use a disk to hold files , OS still needs to record its own data structures on the disk. It does in 2 steps. The first step is to partition the disk in to one/more groups of cylinders. OS can treat each partition as a separate disk. The second step is logical formatting (or)creation of file system. In this step, OS stores the initial File system **d**ata structures on to the disk. These data structures include maps of free and allocate space and initial empty directo**ry**.

***Boot block*:-**

When a computer is powered up -it must have an initial program to run. This initial bootstrap program initializes all aspects of the system, from CPU registers to device controllers, and the contents of main memory, and then starts the OS. To do its job, the bootstrap program finds the OS kernel on disk, loads that kernel into memory and jumps to an initial address to begin the OS execution. For most computers, the bootstrap is stored in ROM. This location is convenient, because ROM needs no initialization and is at a fixed location that the CPU can start executing when powered up, ROM is read only, it cannot be infected by computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot ROM whose job is to bring in a full bootstrap program from disk. The full bootstrap program is stored in the boot blocks at a fixed location on the disk. A disk that has a boot partition is called

a boot disk or system disk. The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

Bad blocks:-

A Block in the disk damaged due to the manufacturing defect or virus or physical damage. This defector block is called Bad block. MS-DOS format command, scans the disk to find bad blocks. If format finds a bad block, it tells the allocation methods not to use that block. Chkdsk program search for the bad blocks and to lock them away. Data that resided on the bad blocks usually are lost. The OS tries to read logical block 87.
The controller calculates ECC and finds that the sector is bad. It reports this finding to the OS. The next time the system is rebooted, a special command is run to tell the SCS controller to replace the bad sector
with a spare.
After that, whenever the system requests logical block 87, the request is translated into the replacement sectors address by the controller.

Sector slipping:-

Logical block 17 becomes defective and the first available spare follows sector 202. Then, sector slipping remaps all the sectors from 17 to 202, sector 202 is copied into the spare, then sector 201 to 202, 200 to 201 and so on. Until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18.

Swap space management:-

System that implements swapping may use swap space to hold an entire process image, including the code and data segments. Paging systems may simply store pages that have been pushed out of main memory. Note that it may be safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort processes. Overestimation wastes disk space that could otherwise be used for files, but it does no other harm. Some systems recommend the amount to be set aside for swap space. Linux has suggested setting swap space to double the amount of physical memory. Some OS allow the use of multiple swap spaces. These swap spaces as put on separate disks so that load placed on the (I/O) system by paging and swapping can be spread over the systems I/O devices.

Swap space location:-

A Swap space can reside in one of two places. It can be carved out of normal file system (or) it can be in a separate disk partition. If the swap space is simply a large file, within the file system, normal file system methods used to create it, name it, allocate its space. It is easy to implement but inefficient. External fragmentation can greatly increase swapping times by forcing multiple seeks during reading/writing of a process image. We can improve performance by caching the block location information in main memory and by using special tools to allocate physically contiguous blocks for the swap file. Alternatively, swap space can be created in a separate raw partition. a separate swap space storage manager is used to allocate
/deal locate the blocks from the raw partition. this manager uses algorithms optimized for speed rather than storage efficiency. Internal fragmentation may increase but it is acceptable because life of data in swap space is shorter than files. since swap space is reinitialized at boot time, any fragmentation is short lived. the raw partition approach creates a fixed amount of swap space during disk partitioning adding more swap space requires either repartitioning the disk (or) adding another swap space elsewhere.