Topics Covered

- What is Operating System
- Abstract view of Operating System
- Goals and functions of Operating System
- Evolution of Operating System
- Multiprogramming Operating System
- Multitasking/Time Sharing Operating System
- Multiprocessing Processing Operating System
- Real Time Operating System
- Distributed Operating System

What is Operating System

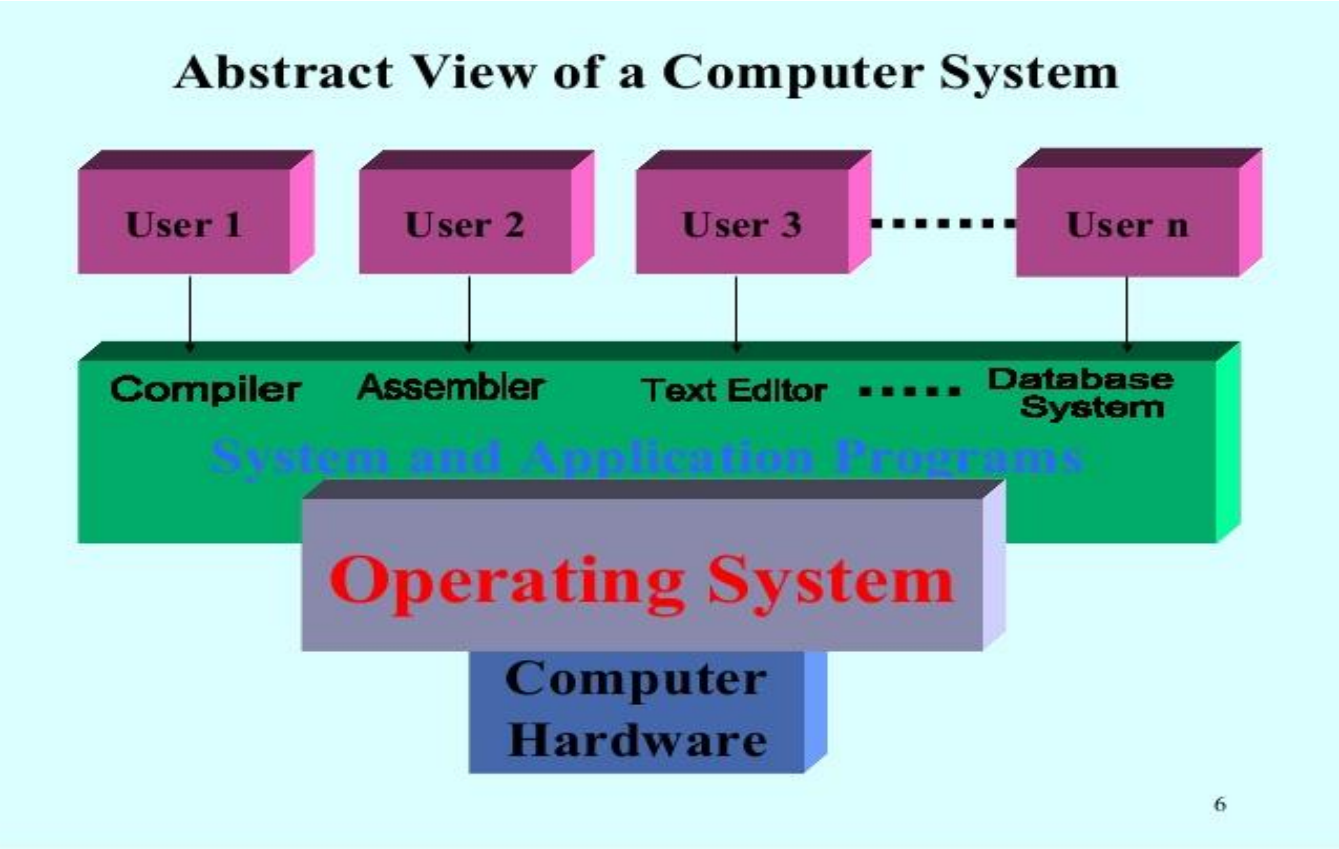
- Whatever used as an interface between the user and the core machine is OS.
    - E.g. steering of car, switch of the fan etc., Buttons over electronic devices.
- Question comes why we need an operating system?
    - To enable everybody to use h/w In a convenient and efficient manner
- Definition of Operating System
    - **There is no exact or precise definition for OS but we can say, "A program or System software"**
        - Which Acts as an intermediary between user & h/w
        - Resource Manager/Allocator - Manage system resources in an unbiased fashion both h/w (mainly CPU time, memory, system buses) & s/w (access, authorization, semaphores) and provide functionality to application programs. OS controls and coordinates the use of resources among various application programs.
        - OS provides platform on which other application programs can be installed, provides the environment within which programs are executed.
- For personal computers - Microsoft windows (82.7%), Mac (13.23), Linux (1.57)
- For mobile phones - Android (87.5), IOS (12.1) etc.

Q An operating system is: **(NET-DEC-2006)**

(A) Collection of hardware components
(B) Collection of input-output devices
(C) Collection of software routines
(D) All the above

Answer: C

Abstract view of operating system



- Computer hardware – CPU, memory units, i/o devices, system bus, registers etc. provides the basic computing resources.
- OS - Control and coordinates the use of the hardware among the various applications programs.
- System and Applications programs - Defines the way in which these resources are used to solve the computing problems of the user.
- User

Goals and Functions of operating system

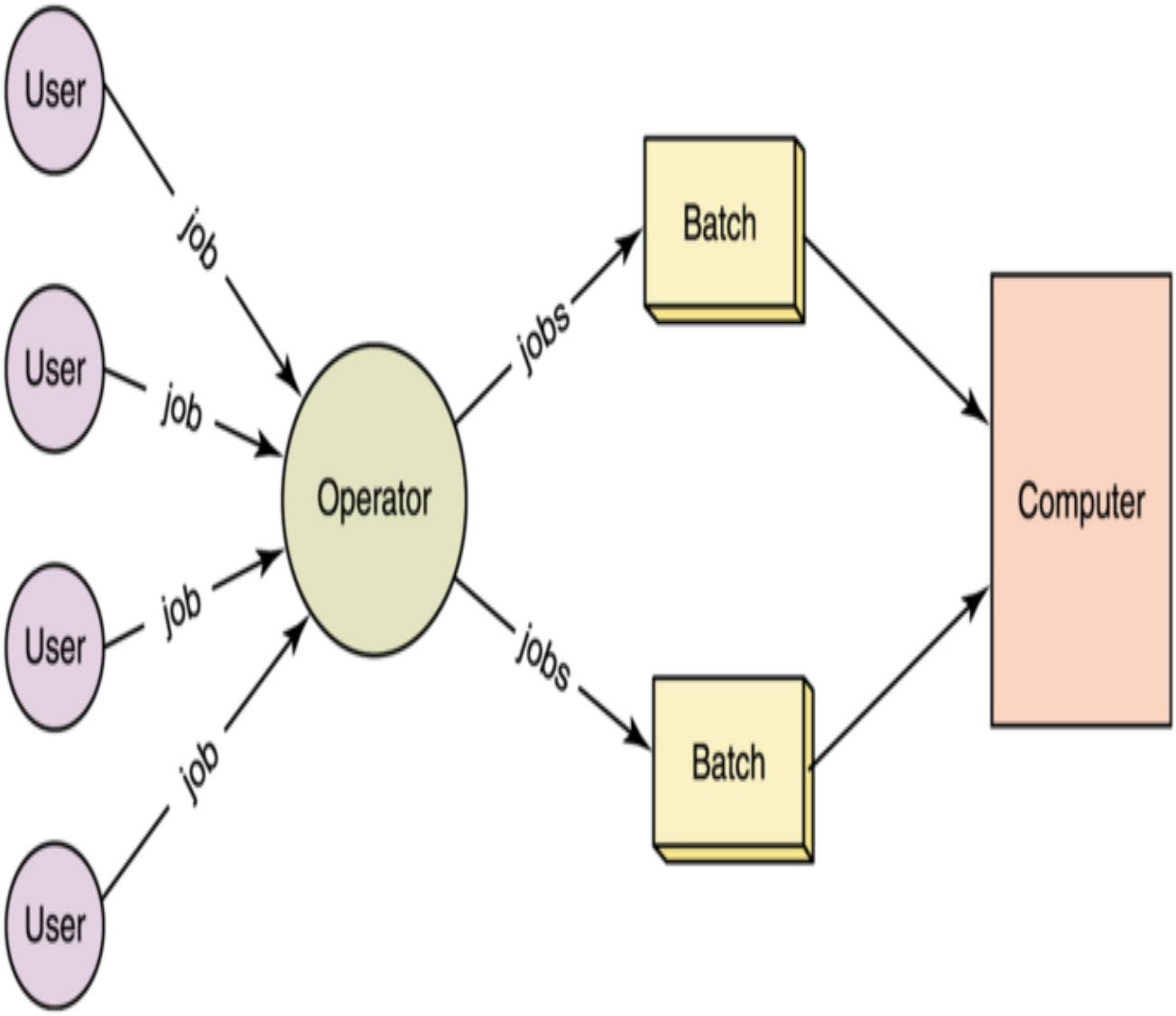
- Goals are the ultimate destination, but we follow functions to implement goals
- Goals
  - Primary goals (Convenience / user friendly)
  - Secondary goals (Efficiency (Using resources in efficient manner) / Reliability / maintainability)
- **Functions of operating system**
  - Process management
  - Memory management
  - I\O device management
  - File management
  - Network management
  - Security & Protection

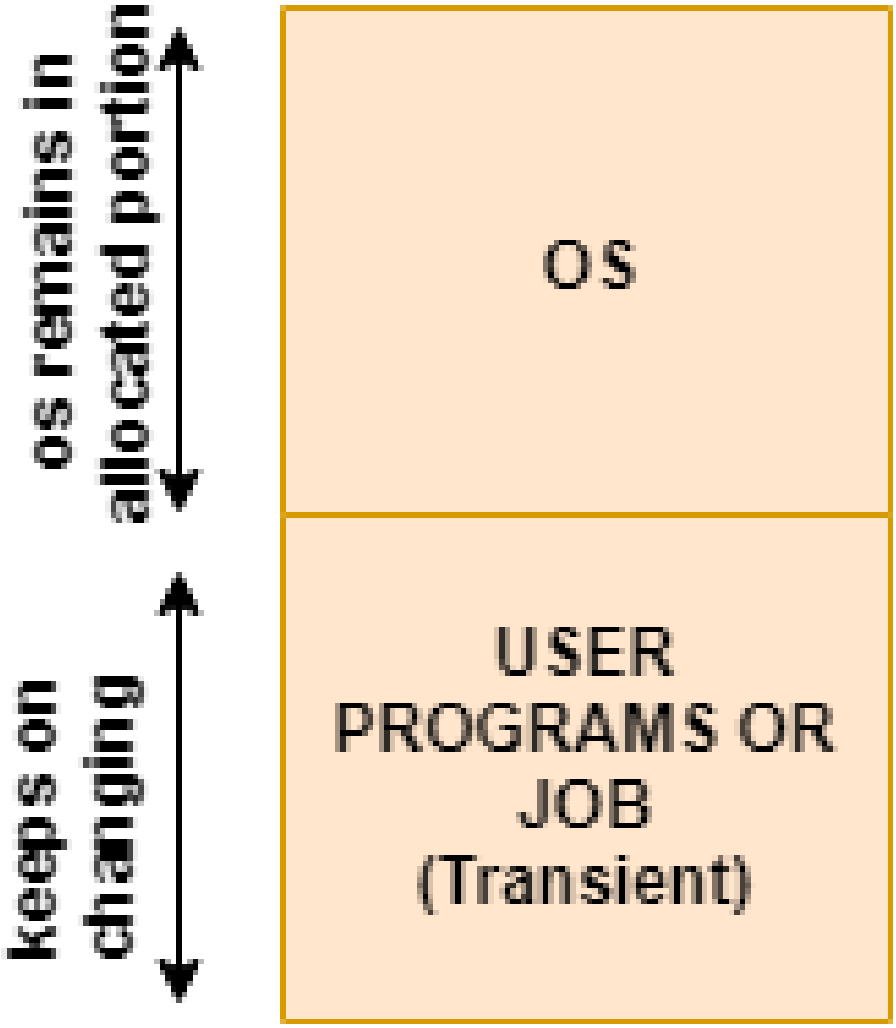
Evolution of Operating System

- Early computers were not interactive device, there user use to prepare a job which consist three parts
    - Program
    - Control information
    - Input data
- Only one job is given input at a time as there was no memory (secondary memory), computer will take the input then process it and then generate output.
- Common input/output device were punch card or tape drives.
- So, these devices were very slow, and processor remain ideal most of the time.

Batch Operating System

- To speed up the processing job with similar types (programming language) were batched together and were run through the processor as a group (batch).
- In some system grouping is done by the operator while in some systems it is performed by the 'Batch Monitor' resided in the low end of main memory)
- Then jobs (as a deck of punched cards) are bundled into batches with similar requirement.
- Then the submitted jobs were 'grouped as FORTRAN jobs, COBOL jobs etc.





Advantage

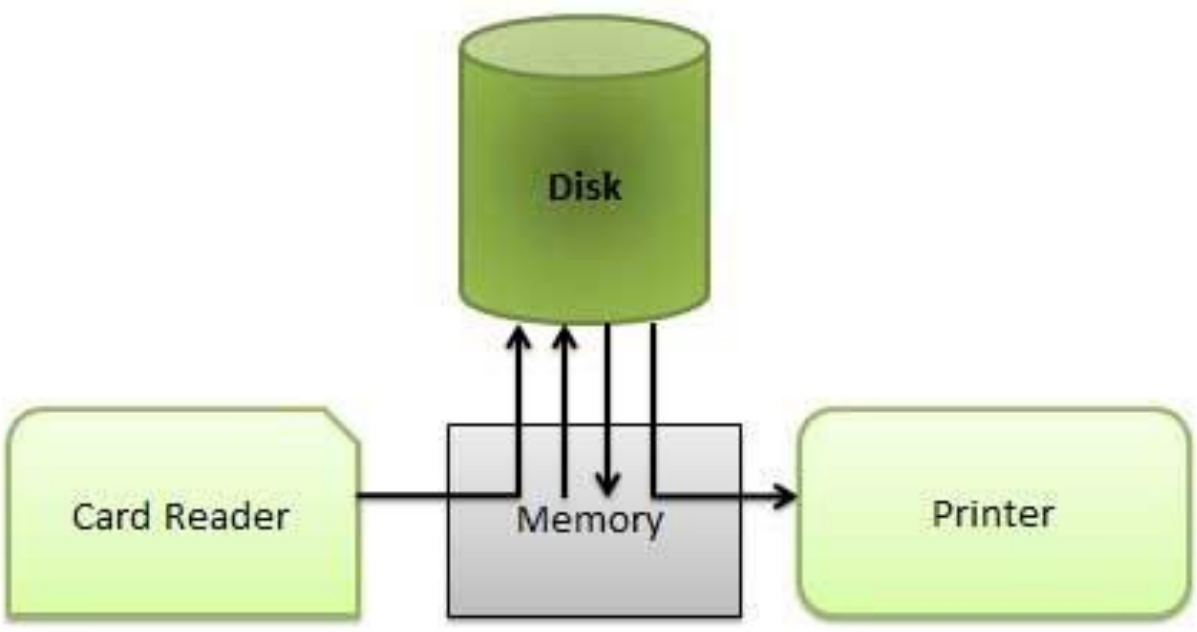
- The batched jobs were executed automatically one after another saving its time by performing the activities (like loading of compiler) only for once. It resulted in improved system utilization due to reduced turnaround time.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention

Disadvantage

- Memory limitation – memory was very limited, because of which interactive process or multiprogramming was not possible

Spooling

- **(**"Spool" is technically an acronym for simultaneous peripheral operations online.**)**
- In a computer system peripheral equipment, such as printers and punch card readers etc, are very slow relative to the performance of the rest of the system. Spooling is useful because devices access data at different rates.
- Spooling is a process in which data is temporarily held to be used and executed by a device, program or the system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.
- Generally, the spool is maintained on the computer's physical memory, buffers or the I/O device-specific interrupts.



- The most common implementation of spooling can be found in typical input/output devices such as the keyboard, mouse and printer. For example, in printer spooling, the documents/files that are sent to the printer are first stored in the memory. Once the printer is ready, it fetches the data and prints it.
- A spooler works by intercepting the information going to the printer, parking it temporarily on disk or in memory. The computer can send the document information to the spooler at full speed, then immediately return control of the screen to you.
- The spooler, meanwhile, hangs onto the information and feeds it to the printer at the slow speed the printer needs to get it. So if your computer can **spool,** you can work while a document is being printed.
- Even experienced a situation when suddenly for some seconds your mouse or keyboard stops working? Meanwhile, we usually click again and again here and there on the screen to check if its working or not. When it actually starts working, what and wherever we

pressed during its hang state gets executed very fast because all the instructions got stored in the respective device's spool.

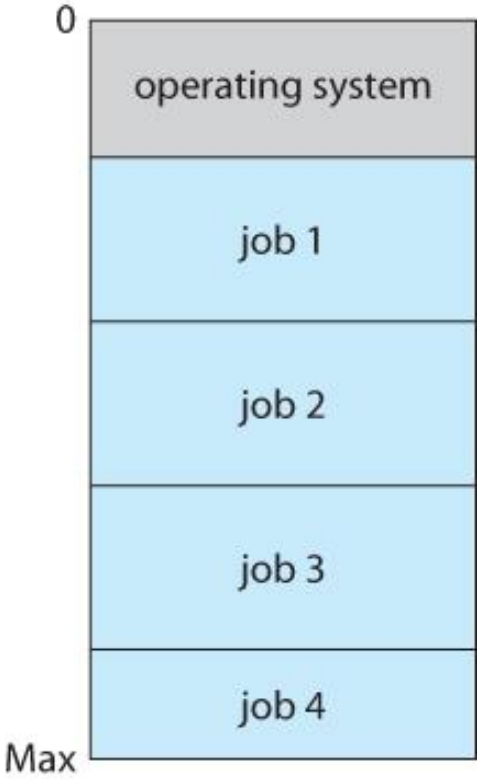
- Spooling is capable of overlapping I/O operation for one job with processor operations for another job. i.e. multiple processes can write documents to a print queue without waiting and resume with their work.

Q Which of the following is an example of a spooled device? **(GATE-1996) (1 Marks)**
(A) a line printer used to print the output of a number of jobs
(B) a terminal used to enter input data to a running program
(C) a secondary storage device in a virtual memory system
(D) a graphic display device
Answer: (A)

Q which of the following is an example of a spooled device? **(Gate-1998) (1 Marks)**
(a) The terminal used to enter the input data for the C program being executed
(b) An output device used to print the output of a number of jobs so
(c) The secondary memory device in a virtual storage system
(d) The swapping area on a disk used by the swapper
Answer (b)

Multiprogramming Operating System

- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. The basic idea of multiprogramming operating system is it keeps several jobs in main memory simultaneously.
- The jobs are kept initially on the disk in the **job pool**. This pool consists of all processes residing on disk awaiting allocation of main memory.
- The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete.
- In a non-multi-programmed system, the CPU would sit idle and wait but in a multi-programmed system, the operating system simply switches to, and executes, another job. When ***that*** job needs to wait, the CPU switches to ***another*** job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. So, conclusion is as long as at least one job needs to execute, the CPU is never idle.



- **Advantage**
  - High and efficient CPU utilization.
  - Less response time or waiting time or turnaround time
  - In most of the applications multiple tasks are running and multiprogramming systems better handle these type of applications
  - Several processes share CPU time
- **Disadvantage**
  - It is difficult to program a system because of complicated schedule handling.
  - To accommodate many jobs in main memory, complex memory management is required.

Q Which of the following features will characterize an OS as multi-programmed OS? **(NET-DEC-2012) (Gate-2002- 1 Marks)**

(a) More than one program may be loaded into main memory at the same time.

(b) If a program waits for certain event another program is immediately scheduled.

(c) If the execution of a program terminates, another program is immediately scheduled.

(A) (a) only

(B) (a) and (b) only

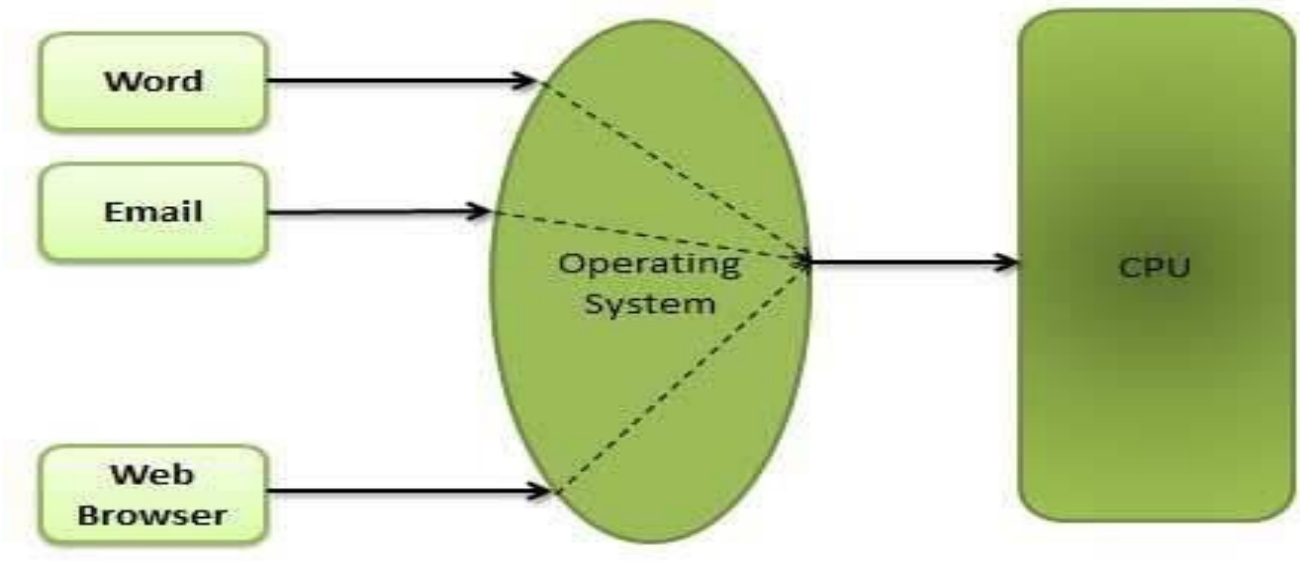
(C) (a) and (c) only

(D) (a), (b) and (c) only

Ans: d

Multitasking Operating system / Time sharing / Multiprogramming with round robin / fair share

- In the modern operating systems, we are able to play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all running at the same time. (by context switching, the illusion of parallelism is achieved)
- A task in a multitasking operating system is not a whole application program but it can also refer to a "thread of execution" when one process is divided into sub-tasks.
- For multitasking to take place, firstly there should be multiprogramming i.e. presence of multiple programs ready for execution. And secondly the concept of time sharing.
- Time sharing (or multitasking) is a logical extension of multiprogramming, it allows many users to share the computer simultaneously. the CPU executes multiple jobs (May belong to different user) by switching among them, but the switches occur so frequently that, each user is given the impression that the entire computer system is dedicated to his/her use, even though it is being shared among many users.
- In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time. minimal lag in overall performance and without affecting the operations of each task.



Q If you want to execute more than one program at a time, the systems software that are used must be capable of: **(NET-DEC-2005)**

(A) word processing
(B) virtual memory
(C) compiling
(D) multitasking

Answer: D

Multiprocessing Operating System/ Tightly coupled system

- Multiprocessor Operating System refers to the use of two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral devices. These systems are referred as tightly coupled systems
- So, in multiprocessing, multiple concurrent processes each can run on a separate CPU, here we achieve a true parallel execution of processes. Parallel processing is the ability of the CPU to simultaneously process incoming jobs.
- Multiprocessing becomes most important in computer system, where the complexity of the job is more, and CPU divides and conquers the jobs. Generally, the parallel processing is used in the fields like artificial intelligence and expert system, image processing, weather forecasting etc.
- **Multiprocessing can be of two types**
    - Symmetric multiprocessing - In a symmetric multi-processing, a single OS instance controls two or more identical processors connected to a single shared main memory. Most of the multi-processing PC motherboards utilize symmetric multiprocessing. Here each processor runs an identical copy of operating system and these copies communicate with each other. SMP means that all processors are peers; no boss–worker relationship exists between processors. Windows, Mac OSX and Linux.
    - Asymmetric - This scheme defines a master-slave relationship, where one processor behaves as a master and control other processor which behaves as slaves. For e.g. It may require that only one particular CPU respond to all hardware interrupts, whereas all other work in the system may be distributed equally among CPUs; or execution of kernel-mode code may be restricted to only one particular CPU, whereas user-mode code may be executed in any combination of processors. Multiprocessing systems are often easier to design if such restrictions are imposed, but they tend to be less efficient than systems in which all CPUs are utilized. A boss processor controls the system, other processors either look to the boss for instruction or have predefined tasks. This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.
- **Advantage of multiprocessing**
    - Increased Throughput - By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N,

however; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. These types of systems are used when very high speed is required to process a large volume of data.

- Economy of Scale - Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.
- Increased Reliability (fault tolerance) - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fail, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether
- Increased reliability of a computer system is crucial in many applications. The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation. Some systems go beyond graceful degradation and are called fault tolerant, because they can suffer a failure of any single component and still continue operation.

- **Disadvantages**
  - It's more complex than simple operating systems.
  - It requires context switching which may impacts performance.
  - If one processor fails then it will affect in the speed
  - large main memory required

Q In which of the following, ready to execute processes must be present in RAM? **(NET-DEC-2008)**

(A) multiprocessing
(B) multiprogramming
(C) multitasking
(D) all of the above

Answer: D

Real time Operating system

- A real time system is a time bound system which has well defined fixed time constraints. Processing must be done within the defined constraints or the system will fail. Very fast and quick respondent systems. Valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.
- RTOS intended to serve real-time applications that process data as it comes in, typically without buffer delays. Are used in an environment where a large number of events (generally external) must be accepted and processed in a short time.
- Little swapping of programs between primary and secondary memory. Most of the time, processes remain in primary memory in order to provide quick response, therefore, memory management in real time system is less demanding compared to other systems.
- For example, a measurement from a petroleum refinery indicating that temperature is getting too high and might demand for immediate attention to avoid an explosion. Airlines reservation system, Air traffic control system, Systems that provide immediate updating, Systems that provide up to the minute information on stock prices, Defense application systems like as RADAR.
- There are three different types of RTOS which are following
    - **Soft real-time operating system** - The soft real-time operating system has certain deadlines, may be missed and they will take the action at a time t=0+. The soft real-time operating system is a type of OS and it does not contain constrained to extreme rules. The critical time of this operating system is delayed to some extent. The examples of this operating system are the digital camera, mobile phones and online data etc.
    - **Hard real-time operating system** - This is also a type of OS and it is predicted by a deadline. The predicted deadlines will react at a time t = 0. Some examples of this operating system are air bag control in cars, anti-lock brake, and engine control system etc.

Distributed OS

- A distributed system is a collection of processors or loosely coupled nodes that do not share memory or a clock. Instead, each node has its own local memory. The nodes communicate with one another through various networks, such as high-speed buses and the Internet.
- The nodes in a distributed system may vary in size and function. They may include small microprocessors, personal computers, and large general-purpose computer systems.
- There are four major reasons for building distributed systems: resource sharing, computation speedup, reliability, and communication.
- Resource Sharing
    - If a number of different sites (with different capabilities) are connected to one another, then a user at one site may be able to use the resources available at another. In general, resource sharing in a distributed system provides mechanisms for sharing files at remote sites, processing information in a distributed database, printing files at remote sites, using remote specialized hardware devices (such as a supercomputer), and performing other operations.
- Computation Speedup
    - If a particular computation can be partitioned into sub computations that can run concurrently, then a distributed system allows us to distribute the sub computations among the various sites. The sub computations can be run concurrently and thus provide computation speedup. In addition, if a particular site is currently overloaded with jobs, some of them can be moved to other, lightly loaded sites. This movement of jobs is called load sharing or job migration. Automated load sharing, in which the distributed operating system automatically moves jobs, is not yet common in commercial systems.
- Reliability
    - If one site fails in a distributed system, the remaining sites can continue operating, giving the system better reliability. If the system is composed of multiple large autonomous installations (that is, general-purpose computers), the failure of one of them should not affect the rest. If, however, the system is composed of small machines, each of which is responsible for some crucial system function (such as the web server or the file system), then a single failure may halt the operation of the whole system. In general, with enough redundancy (in both hardware and data), the system can continue operation, even if some of its sites have failed. The failure of a site must be detected by the system, and appropriate action may be needed to

recover from the failure. The system must no longer use the services of that site. In addition, if the function of the failed site can be taken over by another site, the system must ensure that the transfer of function occurs correctly. Finally, when the failed site recovers or is repaired, mechanisms must be available to integrate it back into the system smoothly.

- Communication
  - When several sites are connected to one another by a communication network, users at the various sites have the opportunity to exchange information. Such functions include file transfer, login, mail, and remote procedure calls (RPCs).
- The advantage of a distributed system is that these functions can be carried out over great distances. Two people at geographically distant sites can collaborate on a project, for example. By transferring the files of the project, logging in to each other's remote systems to run programs, and exchanging mail to coordinate the work, users minimize the limitations inherent in long- distance work.
- The advantages of distributed systems have resulted in an industry-wide trend toward downsizing. Many companies are replacing their mainframes with networks of workstations or personal computers. Companies get a bigger bang for the buck (that is, better functionality for the cost), more flexibility in locating resources and expanding facilities, better user interfaces, and easier maintenance.

Q Match the following: **(NET-June-2015)**

List - I	**List - II**
(a) Spooling	**(i)** Allows several jobs in memory to improve CPU utilization
(b) Multiprogramming	**(ii)** Access to shared resources among geographically dispersed computers in a transparent way
(c) Time sharing	**(iii)** Overlapping I/O and computations
(d) Distributed computing	**(iv)** Allows many users to share a computer simultaneously by switching processor frequently

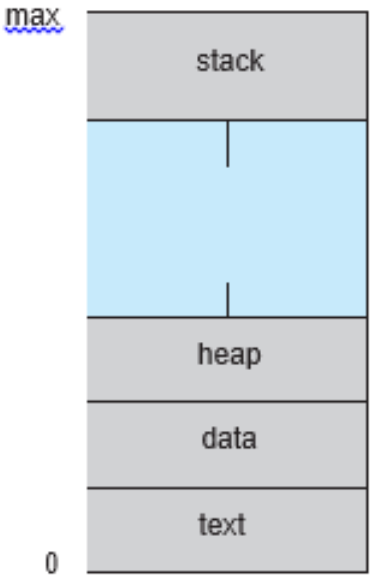
codes:

	(a)	**(b)**	**(c)**	**(d)**
1)	iii	i	ii	iv
2)	iii	i	iv	ii
3)	iv	iii	ii	i
4)	ii	iii	iv	i

Ans: 2

Process

- In general, a process is a program in execution.
- A Program is not a Process by default. A program is a passive entity, i.e. a file containing a list of instructions stored on disk (secondary memory) (often called an executable file).
- A program becomes a Process when an executable file is loaded into main memory and when it's PCB is created
- A process on the other hand is an Active Entity, which require resources like main memory, CPU time, registers, system buses etc.
- Even if two processes may be associated with same program, they will be considered as two separate execution sequences and are totally different process. For instance, if a user has invoked many copies of web browser program, each copy will be treated as separate process. even though the text section is same but the data, heap and stack sections can vary.



- A Process consists of following sections:
  - **Text section:** also known as Program Code.
  - **Stack:** which contains the temporary data (Function Parameters, return addresses and local variables).
  - **Data Section:** containing global variables.
  - **Heap:** which is memory dynamically allocated during process runtime.

Q Process is a: **(ISRO 2009)**

a) A program in high level language kept on disk

b) Contents of main memory

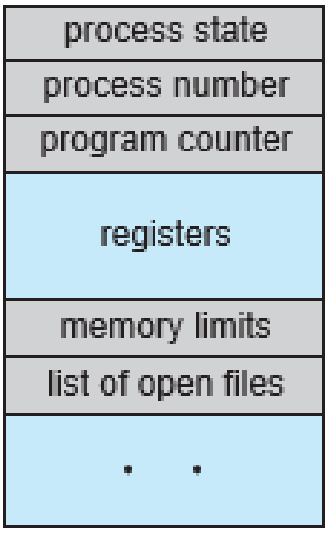
c) A program in execution

d) A job in secondary memory

Ans. C

Process Control Block (PCB)

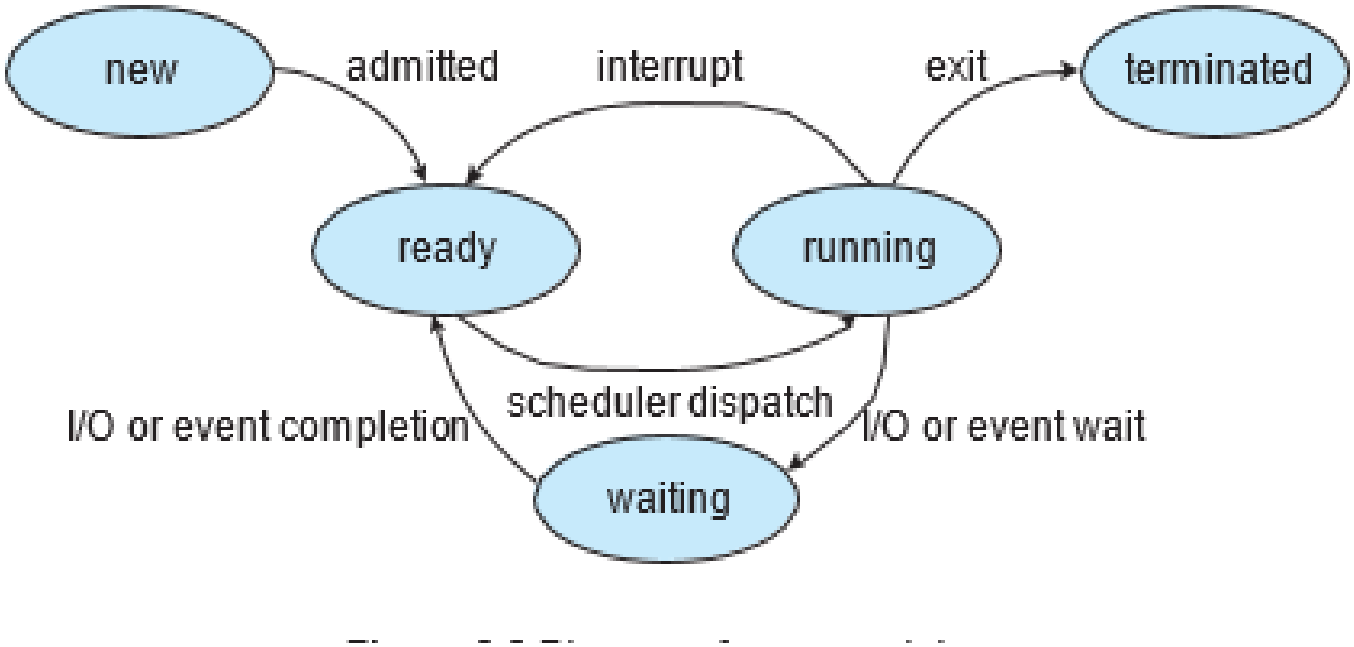
- Each process is represented in the operating system by a process control block (PCB) — also called a task control block. PCB simply serves as the repository for any information that may vary from process to process. It contains many pieces of information associated with a specific process, including these:



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

Process States

- A Process changes states as it executes. The state of a process is defined in parts by the current activity of that process. A process may be in one of the following states:
- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting (Blocked):** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.



Q How many states can a process be in? (NET-DEC-2007)
a) 2 **b)** 3 **c)** 4 **d)** 5
Answer: D

Q On a system with N CPUs, what is the minimum and maximum number of processes that can be in the ready, run, and blocked states, if we have n process?

	Min	**Max**
Ready	0	All P-n process
Run	0	N
Block	0	All process

Q The state of a process after it encounters an I/O instruction is **(ISRO 2013)**

a) ready **b)** blocked **c)** idle **d)** running

Ans. B

Q There are three processes in the ready queue. When the currently running process requests for I/O how many process switches take place? **(ISRO 2011)**

a) 1 **b)** 2 **c)** 3 **d)** 4

Ans. A

Q Which combination of the following features will suffice to characterize an OS as a multi-programmed OS? **(GATE-2002) (2 Marks)**

(a) More than one program may be loaded into main memory at the same time for execution.

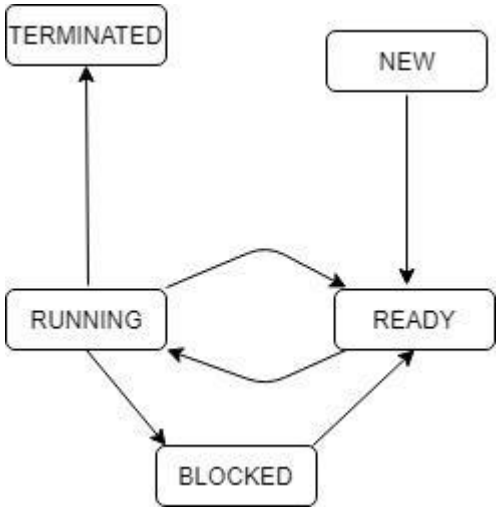
(b) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

(c) If the execution of program terminates, another program is immediately scheduled for execution.

(A) a **(B)** a and b **(C)** a and c **(D)** a, b and c

Answer: (D)

Q The process state transition diagram in below figure is representative of Untitled Diagram **(GATE-1996) (1 Marks)**



a) a batch operating system

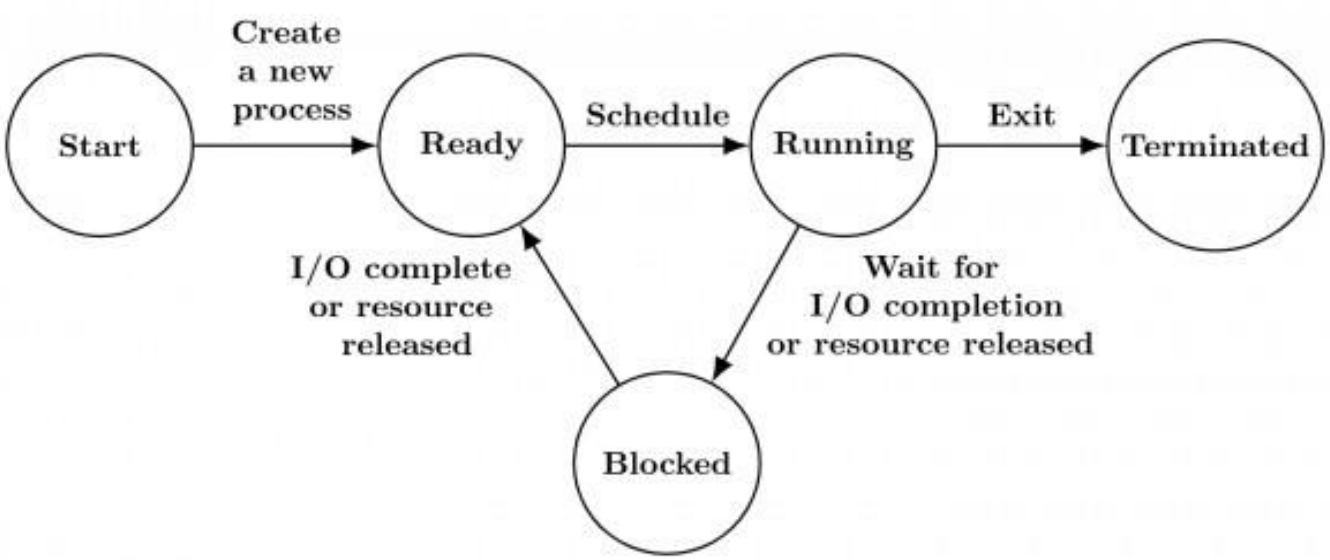
b) an operating system with a preemptive schedular

c) an operating system with a non-preemptive schedular

d) a uni-programmed operating system

Ans. B

Q The process state transition diagram of an operating system is as given below.



Q Which of the following must be FALSE about the above operating system? **(GATE-2006) (1 Marks)**

a) It is a multiprogram operating system

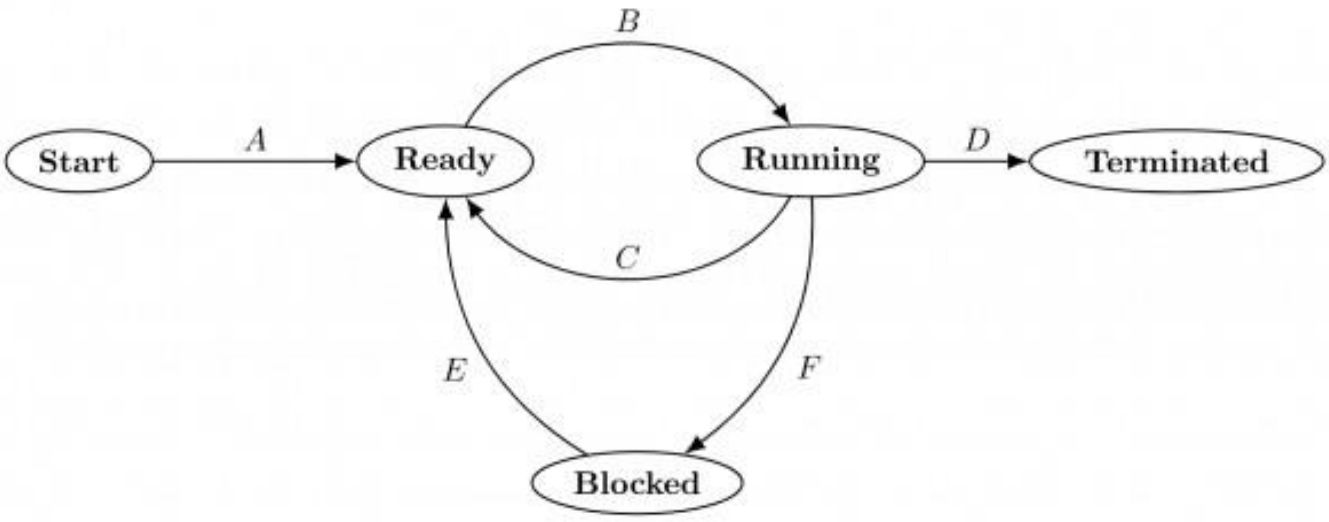
b) It uses preemptive scheduling

c) It uses non-preemptive scheduling

d) It is a multi-user operating system

Ans. B

Q In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state:



Now consider the following statements:

I) If a process makes a transition D, it would result in another process making transition A immediately.

II) A process P2 in blocked state can make transition E while another process P1 is in

running state.

III) The OS uses preemptive scheduling.

IV) The OS uses non-preemptive scheduling.

Which of the above statements are TRUE? **(GATE-2009) (2 Marks)**

a) I and II **b)** I and III **c)** II and III **d)** II and IV

Ans. C

Q A computer handles several interrupt sources of which the following are relevant for this question.

- Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
- Interrupt from Mouse (raises interrupt if the mouse is moved or a button is pressed)
- Interrupt from Keyboard (raises interrupt when a key is pressed or released)
- Interrupt from Hard Disk (raises interrupt when a disk read is completed)

Which one of these will be handled at the HIGHEST priority? **(GATE - 2011) (1 Marks)**

(A) Interrupt from Hard Disk

(B) Interrupt from Mouse

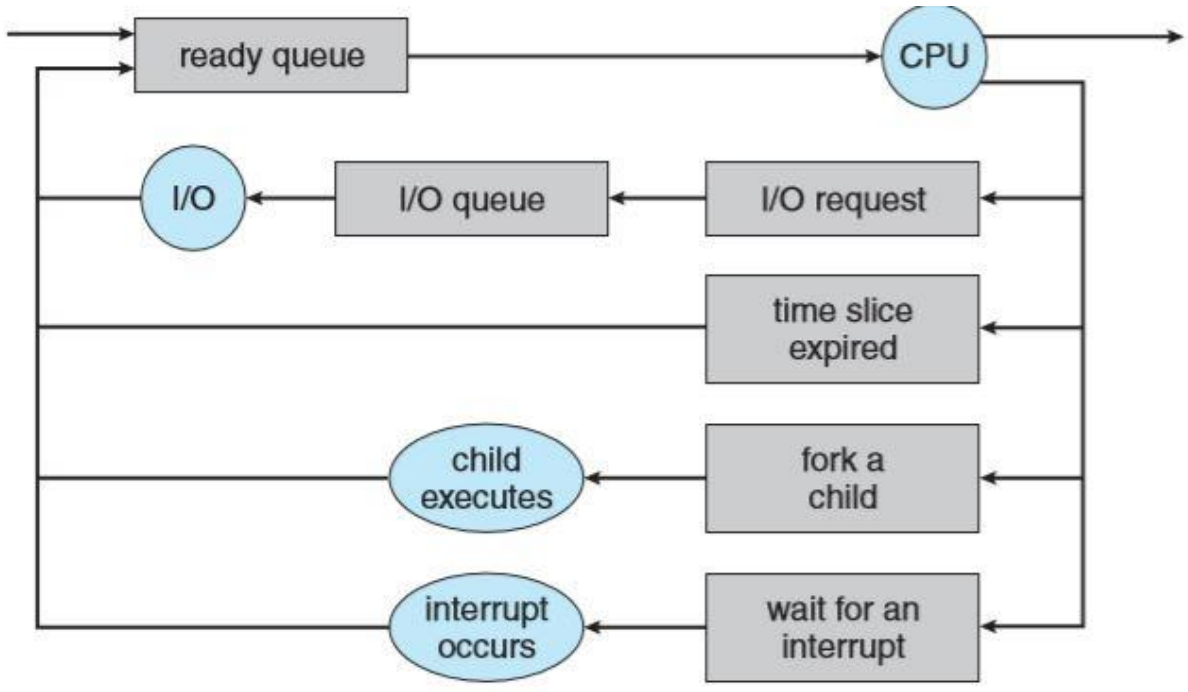
(C) Interrupt from Keyboard

(D) Interrupt from CPU temperature sensor

Answer: (D)

Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for execution on the CPU. Note: For a single-processor system, there will never be more than one running process.
- Scheduling Queues - As processes enter the system, they are put into a job queue, which consists of all processes in the system. The processes that are residing in main memory and are ready to execute are kept on a list called the ready queue. This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.
- The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.
- Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.



Queueing-diagram representation of process scheduling.

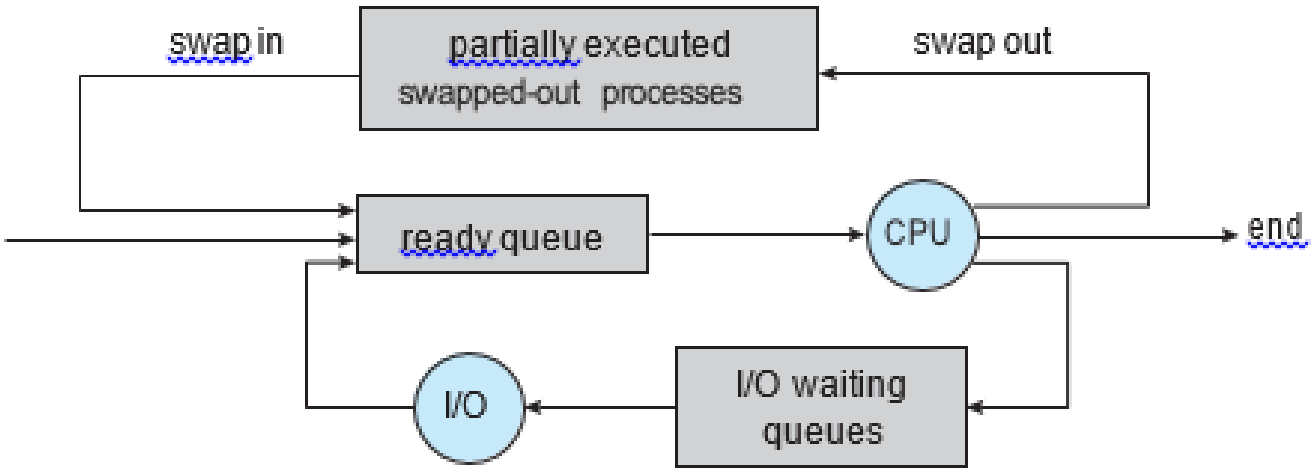
- Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues. A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched.
- Once the process is allocated the CPU and is executing, one of several events could occur:
    - The process could issue an I/O request and then be placed in an I/O queue.
    - The process could create a new child process and wait for the child's termination.
    - Time slice expired, the process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.
    - In the first two cases the process eventually switches from the waiting state to the ready state and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Q A software to create a Job Queue is called..... **(NET-DEC-2006)**
a) Linkage editor **b)** Interpreter **c)** Driver **d)** Spooler
Answer: D

Q A special software that is used to create a job queue is called **(NET-JUNE-2011)**
(A) Drive **(B)** Spooler **(C)** Interpreter **(D)** Linkage editor
Answer: B

Schedulers

- **Schedulers:** A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.
- **Types of Schedulers**
  - **Long Term Schedulers (LTS):** In multiprogramming os, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.
  - **Short Term Scheduler (STS):** The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.
  - **Difference between LTS and STS -** The primary distinction between these two schedulers lies in frequency of execution. The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Often, the short-term scheduler executes at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast. The long-term scheduler on the other hand executes much less frequently; minutes may separate the creation of one new process and the next.
- **Degree of Multiprogramming** - The number of processes in memory is known as Degree of Multiprogramming. The long-term scheduler controls the degree of multiprogramming as it is responsible for bringing in the processes to main memory. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. So, this means the long-term scheduler may need to be invoked only when a process leaves the system. Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution. It is very important that the long-term scheduler make a careful selection. Apart from these two schedulers some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.
- **Medium-term scheduler:** The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called swapping. The process is swapped out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.



- Dispatcher - The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following: Switching context, switching to user mode, jumping to the proper location in the user program to restart that program. The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Q Which module gives control of the CPU to the process selected by the short - term scheduler? (NET-NOV-2017)

a) Dispatcher b) Interrupt c) Scheduler d) Threading

Ans: a

Q A virtual memory-based memory management algorithm partially swaps out a process. This is an example of: **(NET-June-2013)**

(A) short term scheduling **(B)** long term scheduling

(C) medium term scheduling **(D)** mutual exclusion

Answer C

CPU Bound and I/O Bound Processes

- **I/O Bound Processes:** An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations.
- **CPU Bound Processes:** A CPU-bound process, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes. If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do. Similarly, if all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
- So, to have the best system performance LTS needs to select a good combination of I/O and CPU Bound processes.

Context Switch

- When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. Context-switch time is pure overhead, because the system does no useful work while switching.
- Switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). A typical speed is a few milliseconds.

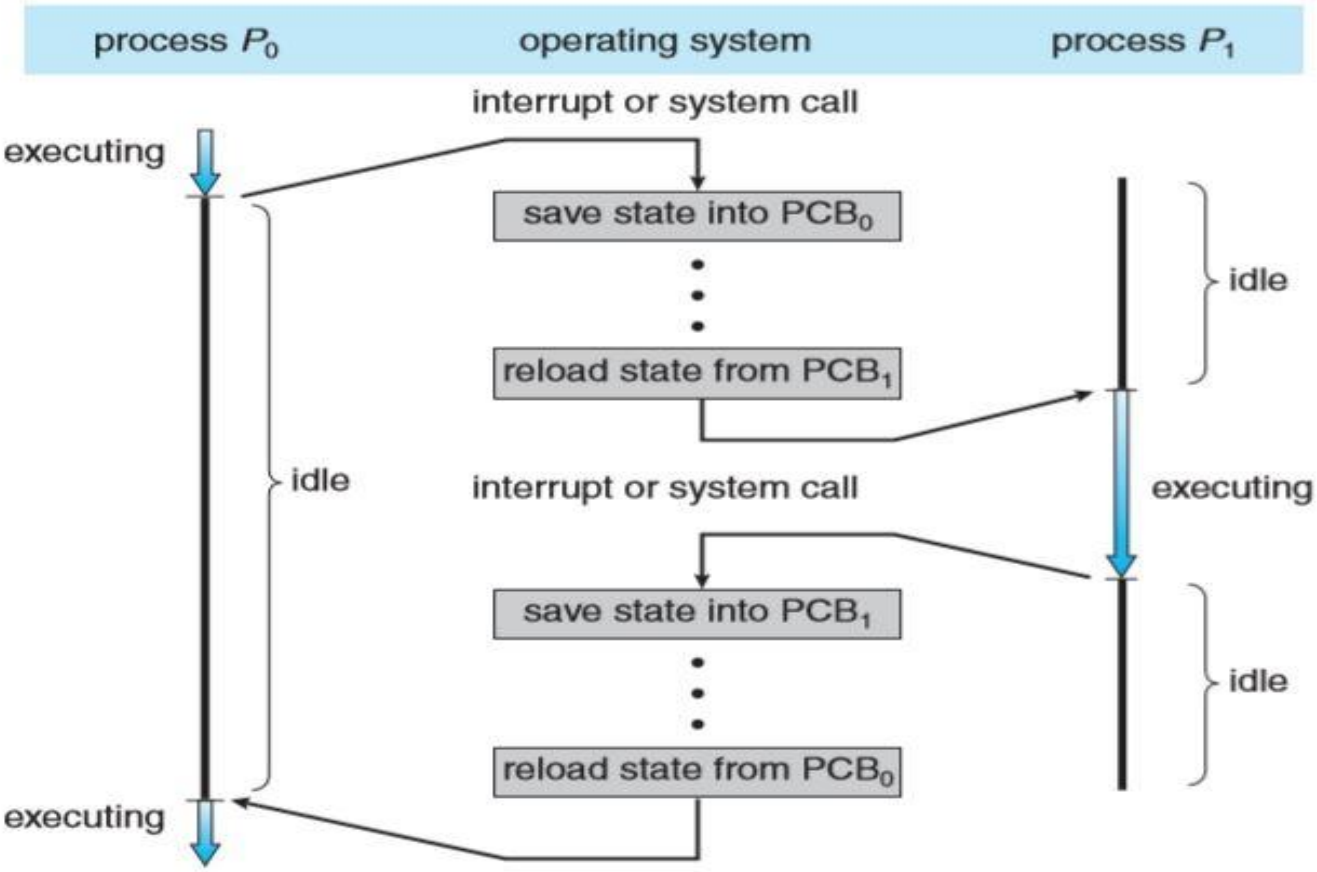


Diagram showing CPU switch from process to process.

Q What is the ready state of a process?
a) when process is scheduled to run after some execution
b) when process is using the CPU
c) when process is unable to run until some task has been completed
d) none of the mentioned
Answer: a

Q Which of the following does not interrupt a running process? **(GATE-2001) (2 Marks)**
(A) A device **(B)** Timer **(C)** Scheduler process **(D)** Power failure
Answer: (C)

Q Which is the correct definition of a valid process transition in an operating system?
a) Wake up: ready → running **b)** Dispatch: ready → running
c) Block: ready → running **d)** Timer runout: ready → running
Ans. B

Q A process stack does not contain
a) function parameters **b)** local variables
c) return addresses **d)** PID of child process
Answer: d

Q A task in a blocked state
a) is executable
b) is running
c) must still be placed in the run queues
d) is waiting for some temporarily unavailable resources
Ans. D

Q Loading operating system from secondary memory to primary memory is called..... **(NET-DEC-2006)**
(A) Compiling **(B)** Booting **(C)** Refreshing **(D)** Reassembling
Answer: B

Q N processes are waiting for I/O. A process spends a fraction p of its time in I/O wait state. The CPU utilization is given by: (NET-DEC-2008)
a) $1-p^{-N}$ b) $1-p^{N}$ **c)** p^{N} **d)** p^{-N}
Answer: B

CPU Scheduling

- A process execution consists of a cycle of CPU execution or wait and i/o execution or wait. Normally a process alternates between two states. Process execution begin with the CPU burst that may be followed by a i/o burst, then another CPU and i/o burst and so on. eventually in the last will end up on CPU burst. So, process keep switching between the CPU and i/o during execution.

Terminology

- **Arrival Time (AT):** Time at which process enters a ready state.
- **Burst Time (BT):** Amount of CPU time required by the process to finish its execution.
- **Completion Time (CT):** Time at which process finishes its execution.
- **Turn Around Time (TAT):** Completion Time (CT) – Arrival Time (AT), WT + BT
- **Waiting Time:** Turn Around Time (TAT) – Burst Time (BT)
- **Response Time:** Is the time it takes to start responding, not the time it takes to output the response.

Scheduling criteria

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another. So, in order to efficiently select the scheduling algorithms following criteria should be taken into consideration:

- **CPU utilization:** Keeping the CPU as busy as possible.
- **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput.
- **Turnaround time:** The interval from the time of submission of a process to the time of completion is the turnaround time.
- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response Time:** Is the time it takes to start responding, not the time it takes to output the response.
- Note: The CPU-scheduling algorithm does not affect the amount of time during which a process executes I/0; it affects only the amount of time that a process spends waiting in the ready queue. It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.

Q State any undesirable characteristic of the following criteria for measuring performance of an operating system: **(GATE-1998) (1 Marks)**

a) waiting time

B) turnaround time

Type of scheduling

- CPU scheduling decision may take place under the following circumstances:
- **Non-Pre-emptive:** Under Non-Pre-emptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. i.e. A process will leave the CPU willingly it can't be forced out. They are rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
  - When a process completes its execution
  - When a process leaves CPU voluntarily to perform some i/o or other operations and enters waiting state.
- **Pre-emptive**
  - If a new process enters in the ready state, in case of high priority
  - When process switches from running to ready state because of time quantum expire.

FCFS (FISRT COME FIRST SERVE)

- FCFS is the simplest scheduling algorithm, as the name suggest, the process that requests the CPU first is allocated the CPU first. Implementation is managed by FIFO Queue. It is always non pre-emptive in nature.

Example: Consider the following table of arrival time and burst time for three processes P_0, P_1, P_2, P_3, P_4. What is the average Waiting Time and Turnaround Time for the five processes?

P. No	Arrival Time (AT)	Burst Time (BT)
P_0	0	4
P_1	1	3
P_2	2	1
P_3	3	2
P_4	4	5

Ans. We make a Gantt chart (Bar chart that illustrates a particular Schedule).

P_1(0-4)	P_2(4-7)	P_3(7-8)	P_4(8-10)	P_5(10-15)

Calculating the various parameters:

P. No	AT	BT	CT	TAT	WT)
P_0	0	4	4	4	0
P_1	1	3	7	6	3
P_2	2	1	8	6	5
P_3	3	2	10	7	5
P_4	4	5	15	11	6
Total				34	19

Average TAT = 34/5 = 6.8
Average WT = 19/5 = 3.8

Example: Calculating the various parameters:

P. No	AT	BT	CT	TAT	WT
P_0	0	3			
P_1	2	2			
P_2	6	4			
Average					

4

Example: Calculating the various parameters:

P. No	AT	BT	CT	TAT	WT
P_0	2	4			
P_1	1	2			
P_2	0	3			
P_3	4	2			
P_4	3	1			
Average					

Example: Calculating the various parameters:

P. No	AT	BT	CT	TAT	WT
P_0	6	4			
P_1	2	5			
P_2	3	3			
P_3	1	1			
P_4	4	2			
P_5	5	6			
Average					

- **Advantage**
  - Easy to understand, and can easily be implemented using Queue data structure.
  - Can be used for Background processes where execution is not urgent.
- **Disadvantage**
  - FCFS suffers from convoy which means smaller process have to wait larger process, which result into large average waiting time. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.
  - This algo fails with time sharing system or interactive systems
  - Higher average waiting time and TAT compared to other algorithms.
  - The FCFS algorithm is thus particularly troublesome for time-sharing systems (due to its non-pre-emptive nature), where it is important that each user get a share of the CPU at regular intervals.

Shortest Job First (SJF)(non-pre-emptive)
Shortest Remaining Time First (SRTF)/ (Pre-emptive)

- Whenever we make a decision of selecting the next process for CPU execution, out of all available process, CPU is assigned to the process having smallest burst time requirement. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If there is a tie, FCFS is used to break tie
- It supports both version non-pre-emptive and pre-emptive (purely greedy approach)
  - In Shortest Job First (SJF)(non-pre-emptive) once a decision is made and among the available process, the process with the smallest CPU burst is scheduled on the CPU, it cannot be pre-empted even if a new process with the smaller CPU burst requirement then the remaining CPU burst of the running process enter in the system
  - In Shortest Remaining Time First (SRTF) (Pre-emptive) whenever a process enters in ready state, again we make a scheduling decision weather, this new process with the smaller CPU burst requirement then the remaining CPU burst of the running process and if it is the case then the running process is pre-empted and new process is scheduled on the CPU.
  - This version (SRTF) is also called optimal is it guarantee minimal average waiting time.

Example on SJF (Non-Pre-emptive), Calculate Average TAT and WT?

P. No	A.T	B.T
P_1	1	7
P_2	2	5
P_3	3	1
P_4	4	2
P_5	5	8

Ans. Constructing the Gantt chart

P_0 (0-1)	P_1 (1-8)	P_3 (8-9)	P_4 (9-11)	P_2 (11-16)	P_5 (16-24)

P. No	A.T	B.T	C.T	TAT	WT
P1	1	7	8	7	0
P2	2	5	16	14	9
P3	3	1	9	6	5

P4	4	2	11	7	5
P5	5	8	24	19	11
Total				53	30

Average Waiting Time: 30/5 = 6
Average Turn Around Time: 53/5 = 10.6

Example: Consider the following processes:

P. No	AT	BT
P1	0	10
P2	3	6
P3	7	1
P4	8	3

Calculate Average WT and TAT?
Ans.

P1 (0-3)	P2 (3-7)	P3 (7-8)	P2 (8-10)	P4 (10-13)	P1 (13-20)

P. No	AT	BT	CT	TAT	WT
P1	0	10	20	20	10
P2	3	6	10	7	1
P3	7	1	8	1	0
P4	8	3	13	5	2
Total				33	13

Average TAT: 33/4 = 8.25
Average WT: 13/4 = 3.25

Example:

Process	AT	BT	CT	TAT	WT
P_1	0	6			
P_2	1	4			
P_3	2	3			
P_4	3	1			
P_5	4	2			
P_6	5	1			

Example:

Process	AT	BT	CT	TAT	WT
P_1	3	4			
P_2	4	2			
P_3	5	1			
P_4	2	6			
P_5	1	8			
P_6	2	4			

Advantage:

- Pre-emptive version guarantees minimal average waiting time so some time also referred as optimal algorithm.
- Provide a standard for other algo in terms of average waiting time
- Provide better average response time compare to FCFs

Q For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time? **(GATE-2015) (2 Marks)**

Process	Arrival Time	Processing Time
A	0	3
B	1	6
C	4	4
D	6	2

(A) First Come First Serve
(B) Non-pre-emptive Shortest Job First
(C) Shortest Remaining Time
(D) Round Robin with Quantum value two
Answer: (C)

Q Consider an arbitrary set of CPU-bound processes with unequal CPU burst lengths submitted at the same time to a computer system. Which one of the following process scheduling algorithms would minimize the average waiting time in the ready queue? **(GATE - 2016) (1 Marks)**
(a) Shortest remaining time firs
(b) Round-robin with time quantum less than the shortest CPU burst
(c) Uniform random

(d) Highest priority first with priority proportional to CPU burst length
Answer: (A)

Disadvantage

- This algo cannot be implemented as there is no way to know the length of the next CPU burst.
- Here process with the longer CPU burst requirement goes into starvation.
- No idea of priority, longer process has poor response time.

- **Note: -** There are several approaches either to make SJF implementable or to improve it in terms of starvation or in other sense. Will discuss them later.
- As SJF is not implementable, we can use the one technique where we try to predict the CPU burst of the next coming process. The method is used as exponential averaging technique, where we consider the previous value and previous prediction.
- $Tau_{(n+1)}$ = alpa t_n + (1- alpa) tau_n. This idea is also more of theoretical importance as most of the time the burst requirement of the coming process may vary by a large extent and if the burst time requirement of all the process is approximately same then there is no advantage of using this scheme.
  - **Example:**

Process	t	tau
P_1	10	20
P_2	12	
P_3	14	
P_4		

Q The aging algorithm with a=0.5 is used to predict run times. The previous four runs from oldest to most recent are 40, 20, 20 and 15 msec. The prediction for the next time will be: **(NET-Dec-2007)**
(A) 15 msec **(B)** 25 msec **(C)** 39 msec **(D)** 40 msec
Answer: (B)

Q Consider the following set of processes, with the arrival times and the CPU-burst times given in milliseconds **(GATE - 2017) (2 Marks)**

Process	Arrival Time	CPU Time
P_1	0	5
P_2	1	3
P_3	2	3
P_4	4	1

What is the average turnaround time for these processes with the pre-emptive shortest remaining processing time first (SRPT) algorithm?

(A) 5.50 **(B)** 5.75 **(C)** 6.00 **(D)** 6.25
Answer: (A)

Q Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is pre-emptive shortest remaining-time first.

Process	Arrival Time	CPU Time
P_1	0	10
P_2	3	6
P_3	7	1
P_4	8	3

The average turnaround time of these processes is _____ milliseconds. **(GATE - 2016) (2 Marks)**
Answer: 8.25

Q Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below: **(GATE - 2019) (2 Marks)**

Process	Arrival Time	CPU Time
P_1	0	3
P_2	1	1
P_3	3	3
P_4	4	Z

These processes are run on a single processor using pre-emptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is _____.
Answer: 2

Q Consider the following CPU processes with arrival times (in milliseconds) and length of CPU bursts (in milliseconds) as given below: **(GATE - 2017) (1 Marks)**

Process	Arrival Time	CPU Time
P_1	0	7
P_2	3	3
P_3	5	5
P_4	6	2

If the pre-emptive shortest remaining time first scheduling algorithm is used to schedule the processes, then the average waiting time across all processes is _____ milliseconds.
Answer: 3

Q Consider the following set of processes that need to be scheduled on a single CPU. All the times are given in milliseconds? **(GATE-2014) (2 Marks)**

Process Name	Arrival Time	Execution Time
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Using the *shortest remaining time first* scheduling algorithm, the average process turnaround time (in msec) is _____.
Answer: 7.2

Q An operating system uses shortest remaining time first scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds) **(GATE-2014) (2 Marks)**

Process	Arrival Time	Burst Time
P_1	0	12
P_2	2	4
P_3	3	6
P_4	8	5

The average waiting time (in milliseconds) of the processes is _____.
Answer: 5.5

Q Consider the following table of arrival time and burst time for three processes P_0, P_1 and P_2.

Process	Arrival Time	Burst Time
P_0	0	9

P_1	1	4
P_2	2	9

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes? **(GATE-2011) (2 Marks)**

(A) 5.0 ms **(B)** 4.33 ms **(C)** 6.33 **(D)** 7.33

Answer: (A)

Q An operating system uses Shortest Remaining Time first (SRT) process scheduling algorithm. Consider the arrival times and execution times for the following processes:

Process	Arrival Time	CPU Time
P_1	0	20
P_2	15	25
P_3	30	10
P_4	45	15

What is the total waiting time for process P2? **(GATE - 2007) (2 Marks)**

(A) 5 **(B)** 15 **(C)** 40 **(D)** 55

Answer: (B)

Q Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end? **(GATE-2006) (1 Marks)**

(A) 1 **(B)** 2 **(C)** 3 **(D)** 4

Answer: (B)

Q Consider the following three processes with the arrival time and CPU burst time given in milliseconds: **(NET-July-2018)**

Process	Arrival Time	Burst Time
P1	0	7
P2	1	4
P3	2	8

The Gantt Chart for pre-emptive SJF scheduling algorithm is _____.

a)

P_1(0-7)	P_2(7-13)	P_3(13-21)

b)

P_1(0-1)	P_2(1-5)	P_1(5-11)	P_3(11-19)

c)

P_1(0-7)	P_2(7-11)	P_3(11-19)

d)

P_2(0-4)	P_3(4-12)	P_1(12-19)

Answer: (b)

Q Consider the following four processes with the arrival time and length of CPU burst given in milliseconds: **(NET-Nov-2018)**

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

The average waiting time for pre-emptive SJF scheduling algorithm is _____.

a) 6.5 **b)** 7.5 **c)** 6.75 **d)** 7.75

Answer: (A)

Q Consider three CPU intensive processes P1, P2, P3 which require 20,10 and 30 units of time, arrive at times 1,3 and 7 respectively. Suppose operating system is implementing Shortest Remaining Time first (pre-emptive scheduling) algorithm, then _____ context switches are required (suppose context switch at the beginning of Ready queue and at the end of Ready queue are not counted). **(NET-Aug-2016)**

a) 3 **b)** 2 **c)** 4 **d)** 5

Answer: (A)

Q The sequence …… is an optimal non-preemptive scheduling sequence for the following jobs which leaves the CPU idle for …… unit(s) of time. **(GATE-1995) (2 Marks)**

Job	Arrival Time	Burst Time
1	0.0	9
2	0.6	5
3	1.0	1

(a) {3,2,1),1 **(b) (2,1,3},0** **(c) {3,2,1),0** **(d) {1,2,3},5**

Answer: (A)

Priority Scheduling

- Here a priority is associated with each process. At any instance of time out of all available process, CPU is allocated to the process which possess highest priority (may be higher or lower number). Tie is broken using FCFS order. No importance to senior or burst time. It supports both non-pre-emptive and pre-emptive versions.
- In Priority (non-pre-emptive) once a decision is made and among the available process, the process with the highest priority is scheduled on the CPU, it cannot be pre-empted even if a new process with higher priority more than the priority of the running process enter in the system.
- In Priority (pre-emptive) once a decision is made and among the available process, the process with the highest priority is scheduled on the CPU, and if it a new process with higher priority more than the priority of the running process enter in the system, then we do a context switch and the processor is provided to the new process with higher priority.
- Priorities are generally indicated by some fixed range of numbers, such as 0 to 7 or 0 to 4,095. There is no general agreement on whether 0 is the highest or lowest priority, it can vary from systems to systems.

Example: We will assume in example that low numbers have higher priority. Non-Pre-emptive Priority Scheduling

P. No	AT	BT	Priority
P_1	0	10	3
P_2	0	1	1
P_3	0	2	4
P_4	0	1	5
P_5	0	5	2

Gantt chart:

P_2 (0-1)	P_5 (1-6)	P_1 (6-16)	P_3 (16-18)	P_4 (18-19)

P. No	AT	BT	Priority	CT	TAT	WT
P_1	0	10	3	16	16	6
P_2	0	1	1	1	1	0
P_3	0	2	4	18	18	16
P_4	0	1	5	19	19	18
P_5	0	5	2	6	6	1
Total					60	41

Average WT: 41/5 = 8.2 ms
Average TAT: 60/5 = 12 ms

Q Example: Pre-emptive priority scheduling

P. No	AT	BT	Priority
P_1	1	4	4
P_2	2	2	5
P_3	2	3	7
P_4	3	5	8(H)
P_5	3	1	5
P_6	4	2	6

Ans. Gantt chart

P_0 (0-1)	P_1 (1-2)	P_3 (2-3)	P_4 (3-4)	P_4 (4-8)	P_3 (8-10)	P_6 (10-12)	P_2 (12-14)	P_5 (14-15)	P_1 (15-18)

P. No	AT	BT	Priority	CT	TAT	WT
P1	1	4	4	18	17	13
P2	2	2	5	14	12	10
P3	2	3	7	10	8	5
P4	3	5	8	8	5	0
P5	3	1	5	15	12	11
P6	4	2	6	12	8	6
Total					62	45

Average TAT: 62/6 = 10.33
Average WT: 45/6: 7.5

Q Example: Pre-emptive priority scheduling

Process	AT	BT	Priority	CT	TAT	WT
P_1	0	50	4			
P_2	20	20	1(H)			
P_3	40	100	3			
P_4	60	40	2			

Q Example: Pre-emptive priority scheduling

Process	AT	BT	Priority	CT	TAT	WT
P_1	1	4	5			
P_2	2	5	2			

P_3	3	6	6			
P_4	0	1	4			
P_5	4	2	1			
P6	5	3	8(H)			

- **Advantage**
  - Gives a facility specially to system process.
  - Allow us to run important process even if it is a user process.
- **Disadvantage**
  - Here process with the smaller priority may starve for the CPU
  - No idea of response time or waiting time.
- Note: - Specially use to support system process or important user process
- Ageing: - a technique of gradually increasing the priority of processes that wait in the system for long time. E.g. priority will increase after every 10 mins

Q Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	AT	BT	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is ____. **(GATE-2017) (2 Marks)**
Note: This question appeared as Numerical Answer Type.
Answer: 29

Q The problem of indefinite blockage of low-priority jobs in general priority scheduling algorithm can be solved using: **(NET-Dec-2012)**
a) Parity bit **b)** Aging **c)** Compaction **d)** Timer
Answer: (B)

Q Some of the criteria for calculation of priority of a process are:
a. Processor utilization by an individual process.
b. Weight assigned to a user or group of users.
c. Processor utilization by a user or group of processes
In fair share scheduler, priority is calculated based on: **(NET-Jan-2017)**
a) only (a) and (b) **b)** only (a) and (c) **c)** (a), (b) and (c) **d)** only (b) and (c)
Answer: C

Q Consider a uniprocessor system executing three tasks T1, T2 and T3, each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of 3, 7 and 20 milliseconds, respectively. The priority of each task is the inverse of its period and the available tasks are scheduled in order of priority, with the highest priority task scheduled first. Each instance of T1, T2 and T3 requires an execution time of 1, 2 and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1st milliseconds and task preemptions are allowed, the first instance of T3 completes its execution at the end of _____ milliseconds. **(GATE-2015) (2 Marks)**

Answer: 12

Q Five jobs A, B, C, D and E are waiting in Ready Queue. Their expected runtimes are 9, 6, 3, 5 and x respectively. All jobs entered in Ready queue at time zero. They must run in _____ order to minimize average response time if 3 < x < 5. **(NET-Aug-2016)**

(A) B, A, D, E, C
(B) C, E, D, B, A
(C) E, D, C, B, A
(D) C, B, A, E, D

Answer: (B)

Q We wish to schedule three processes P1, P2 and P3 on a uniprocessor system. The priorities, CPU time requirements and arrival times of the processes are as shown below.

Process	Priority	CPU time required	Arrival time (hh: mm: ss)
P1	10(highest)	20 sec	00:00:05
P2	9	10 sec	00:00:03
P3	8 (lowest)	15 sec	00:00:00

We have a choice of pre-emptive or non-pre-emptive scheduling. In pre-emptive scheduling, a late-arriving higher priority process can pre-empt a currently running process with lower priority. In non-pre-emptive scheduling, a late-arriving higher priority process must wait for the currently executing process to complete before it can be scheduled on the processor. What are the turnaround times (time from arrival till completion) of P2 using pre-emptive and non-pre-emptive scheduling respectively? **(GATE-2005) (2 Marks)**

(A) 30 sec, 30 sec
(B) 30 sec, 10 sec
(C) 42 sec, 42 sec
(D) 30 sec, 42 sec

Answer: (D)

Q Consider a preemptive priority-based scheduling algorithm based on dynamically changing priority. Larger priority number implies higher priority. When the process is waiting for CPU in

the ready queue (but not yet started execution), its priority changes at a rate a = 2. When it starts running, its priority changes at a rate b = 1. All the processes are assigned priority value 0 when they enter ready queue. Assume that the following processes want to execute: **(NET-Dec-2013)**

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

The time quantum q = 1. When two processes want to join ready queue simultaneously, the process which has not executed recently is given priority. The finish time of processes P1, P2, P3 and P4 will respectively be

(A) 4, 5, 7 and 8
(B) 8, 2, 7 and 5
(C) 2, 5, 7 and 8
(D) 8, 2, 5 and 7

Answer: (B)

Round robin

- This algo is designed for time sharing systems, where it is not, necessary to complete one process and then start another, but to be responsive and divide time of CPU among the process in the ready state. Here ready queue is treated as a circular queue (FIFO). It is similar to FCFS scheduling, but pre-emption is added to enable the system to switch between processes.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval equivalent 1 Time quantum (where value of TQ can be anything).
- We fix a time quantum, up to which a process can hold the CPU in one go, with in which either a process terminates or process must release the CPU and enter the ready queue and wait for the next chance.
- The process may have a CPU burst of less than given time quantum. In this case, the process itself will release the CPU voluntarily. CPU Scheduler will select the next process for execution. OR The CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.
- RR is always pre-emptive in nature.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units. Each process must wait no longer than (n - 1) x q time units until its next time quantum.
- If the time quantum is extremely large, the RR policy is the same as the FCFS policy. If the time quantum is extremely small (say, 1 millisecond), the RR approach is called processor sharing and (in theory) creates the appearance that each of n processes has its own processor running at 1/n the speed of the real processor. We also need also to consider the effect of context switching on the performance of RR scheduling.
- We have only one process of 10-time units. If the quantum is 12-time units, the process finishes in. less than 1-time quantum, with no overhead. If the quantum is 6-time units, however, the process requires 2 quanta, resulting in a context switch. If the time quantum is 1-time unit, then nine context switches will occur, slowing the execution of the process accordingly. In General, If TQ is less the number of Context Switch increases, response time will be less. If TQ is large the number of Context Switch will decrease and response time increases.
- Optimal Time Quantum, so that every problem gets CPU after t seconds: $q \leq t - ns / n-1$. A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.

Q **Example:** Let Time Quantum (TQ) = 2

P. No	AT	BT

P_1	0	4
P_2	1	5
P_3	2	2
P_4	3	1
P_5	4	6
P_6	6	3

Gantt Chart:

P1 (0-2)	P2 (2-4)	P3 (4-6)	P1 (6-8)	P4 (8-9)	P5 (9-11)	P2 (11-13)	P6 (13-15)	P5 (15-17)	P2 (17-18)	P6 (18-19)	P5 (19-21)

P. No	AT	BT	CT	TAT	WT
P_1	0	4	8	8	4
P_2	1	5	18	17	12
P_3	2	2	6	4	2
P_4	3	1	9	6	5
P_5	4	6	21	17	11
P_6	6	3	19	13	10
Total				65	44

Average TAT: 65/6 = 10.83 ms
Average WT = 44/6 = 7.33 ms

Process	Arrival Time	Burst Time	CT	TAT	WT
P_1	5	5			
P_2	4	6			
P_3	3	7			
P_4	1	9			
P_5	2	2			
P6	6	3			

- **Advantage**
  - Perform best in terms of average response time
  - Works will in case of time-sharing systems, client server architecture and interactive system

- kind of SJF implementation

- **Disadvantage**
  - Longer process may starve
  - Performance depends heavily on time quantum - If value of the time quantum is very less, then it will give lesser average response time (good but total no of context switches will be more, so CPU utilization will be less), If time quantum is very large then average response time will be more bad, but no of context switches will be less, so CPU utilization will be good.
  - No idea of priority

Q A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero? **(GATE-2013) (1 Marks)**

(A) This algorithm is equivalent to the first-come-first-serve algorithm

(B) This algorithm is equivalent to the round-robin algorithm.

(C) This algorithm is equivalent to the shortest-job-first algorithm..

(D) This algorithm is equivalent to the shortest-remaining-time-first algorithm

Answer: (B)

Q If the time-slice used in the round-robin scheduling policy is more than the maximum time required to execute any process, then the policy will **(GATE-2008) (1 Marks)**

(A) degenerate to shortest job first **(B)** degenerate to priority scheduling

(C) degenerate to first come first serve **(D)** none of the above

Answer: (C)

Q Which scheduling policy is most suitable for a time-shared operating system? **(GATE-1995) (1 Marks)**

(a) Shortest Job First **(b)** Round Robin **(c)** First Come First Serve **(d)** Elevator

Answer: (B)

Q In round robin CPU scheduling as time quantum is increased the average turnaround time **(NET-June-2012)**

(A) increases **(B)** decreases

(C) remains constant **(D)** varies irregularly

Answer: (D)

Q In the process management Round-robin method is essentially the pre-emptive version of ……**(NET-Dec-2009)**

(A) FILO **(B)** FIFO **(C)** SSF **(D)** Longest time first

Answer: (B)

Q In processor management, round robin method essentially uses the pre-emptive version of ..... (NET-JUNE-2006)

(A) FILO **(B)** FIFO **(C)** SJF **(D)** Longest time first

Answer: B

Q Consider the following set of processes with the length of CPU burst time in milliseconds (ms):

Process	Burst Time	Priority
A	6	3
B	1	1
C	2	3
D	1	4
E	5	2

Assume that processes are stored in ready queue in following order: A – B – C – D – E

Using round robin scheduling with time slice of 1 ms, the average turnaround time is **(NET-Spet-2013)**

(A) 8.4 ms **(B)** 12.4 ms **(C)** 9.2 ms **(D)** 9.4 ms

Answer: A

Q consider the following set of processes and the length of CPU burst time given in milliseconds:

Process	CPU Burst Time
P1	5
P2	7
P3	6
P4	4

Assume that processes being scheduled with round robin scheduling algorithm with time quantum 4ms. Then the waiting for p4 is _____. **(NET-DEC-2018)**

a) 0 **b) 4** **c) 6** **d) 12**

Answer: (D)

Q Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds? **(GATE-1998) (1 Marks)** **(NET-Dec-2012)**

a) $q \leq (t-ns)/(n-1)$ **b)** $q \geq (t-ns)/(n-1)$

c) $q \leq (t-ns)/(n+1)$ **d)** $q \geq (t-ns)/(n+1)$

Answer: (A)

Q Four jobs to be executed on a single processor system arrive at time 0+ in the order A, B, C, D. their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one-time unit is. **(GATE-1996) (2 Marks) (ISRO-2008)**

(a) 10 **(b) 4** **(c) 8** **(d) 9**

Answer: (D)

Q Assume that the following jobs are to be executed on a single processor system

Job Id	CPU Burst Time
P	4
Q	1
R	8
S	1
T	2

The jobs are assumed to have arrived at time 0+ and in the order p, q, r, s, t. calculate the departure time (completion time) for job p if scheduling is round robin with time slice 1. **(GATE-1993) (2 Marks)**

(a) 4 **(b)** 10 **(c)** 11 **(d)** 12

Answer: (C)

Q Which of the following statements are true? **(GATE-2010) (2 Marks)**

1) Shortest remaining time first scheduling may cause starvation

2) Pre-emptive scheduling may cause starvation

3) Round robin is better than FCFS in terms of response time

(A) 1 only **(B)** 1 and 3 only **(C)** 2 and 3 only **(D)** 1, 2 and 3

Answer: (D)

Q Pre-emptive scheduling is the strategy of temporarily suspending a running process **(NET-June-2012)**

a) before the CPU time slice expires
b) to allow starving processes to run
c) when it requires I/O
d) to avoid collision

Answer: B

Q Which of the following scheduling algorithms may cause starvation? **(NET-Jan-2017)**

a. First-come-first-served **b.** Round Robin **c.** Priority
d. Shortest process next **e.** Shortest remaining time first

a) a, c and e
b) c, d and e
c) b, d and e
d) b, c and d

Answer: B

Q If the quantum time of round robin algorithm is very large, then it is equivalent to:

a) First in first out
b) Shortest Job Next
c) Lottery scheduling
d) None of the above

Ans. A

Q A scheduling Algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (lowest priority). The scheduler reevaluates the process priority for every 'T' time units and decides next process to be scheduled. If the process has no I/O operations and all arrive at time zero, then the scheduler implements _____ criteria. **(NET-June-2016)**

a) Priority scheduling
b) Round Robin Scheduling
c) Shortest Job First
d) FCFS

Answer: B

Q Consider the 3 processes, P_1, P_2 and P_3 shown in the table. **(GATE-2012) (2 Marks)**

Process	Arrival Time	Time Unit Required
P_1	0	5
P_2	1	7
P_3	3	4

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2-time units) are

(A) FCFS: P_1, P_2, P_3 RR: P_1, P_2, P_3
(B) FCFS: P_1, P_3, P_2 RR: P_1, P_3, P_2
(C) FCFS: P_1, P_2, P_3 RR: P_1, P_3, P_2
(D) FCFS: P_1, P_3, P_2 RR: P_1, P_2, P_3

Answer: (C)

Q Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle? **(GATE-2006) (2 Marks)**

(A) 0% **(B)** 10.6% **(C)** 30.0% **(D)** 89.4%

Answer: (B)

Q Three processes A, B and C each execute a loop of 100 iterations. In each iteration of the loop, a process performs a single computation that requires t_c CPU milliseconds and then initiates a single I/O operation that lasts for $t_{i/o}$ milliseconds. It is assumed that the computer where the processes execute has sufficient number of I/O devices and the OS of the computer assigns different I/O devices to each process. Also, the scheduling overhead of the OS is negligible. The processes have the following characteristics:

Process Id	T_c	$T_{i/o}$
A	**100**	**500**
B	**350**	**500**
C	**200**	**500**

The processes A, B, and C are started at times 0, 5 and 10 milliseconds respectively, in a pure time-sharing system (round robin scheduling) that uses a time slice of 50 milliseconds. The time in milliseconds at which process C would complete its first I/O operation is _____.
(GATE-2014) (2 Mark)
Answer: 1000

Q The arrival time, priority, and duration of the CPU and I/O bursts for each of three processes P_1, P_2 and P_3 are given in the table below. Each process has a CPU burst followed by an I/O burst followed by another CPU burst. Assume that each process has its own I/O resource.
(GATE-2006) (2 Marks)

Process	Arrival Time	Priority	Burst duration, CPU, I/O CPU
A	0	2	1,5,3
B	2	3(L)	3,3,1
C	3	1(H)	2,3,1

The multi-programmed operating system uses preemptive priority scheduling. What are the finish times of the processes P_1, P_2 and P_3 ?

(A) 11, 15, 9 **(B)** 10, 15, 9 **(C)** 11, 16, 10 **(D)** 12, 17, 11

Answer: (B)

Q A uniprocessor computer system only has two processes, both of which alternate 10 ms CPU bursts with 90 ms I/O bursts. Both the processes were created at nearly the same time. The I/O of both processes can proceed in parallel. Which of the following scheduling strategies will result in the least CPU utilization (over a long period of time) for this system? **(GATE-2003) (2 Marks)**

(A) First come first served scheduling

(B) Shortest remaining time first scheduling

(C) Static priority scheduling with different priorities for the two processes

(D) Round robin scheduling with a time quantum of 5 ms

Answer: (D)

Q In which of the following scheduling criteria, context switching will never take place? **(NET-July-2018)**

a) Round Robin **b)** Pre-emptive SJF

c) Non-Pre-emptive SJF **d)** Preemptive priority

Answer: (c)

Q .....is one of pre-emptive scheduling algorithm. **(NET-Dec-2006)**

(A) Shortest-Job-first **(B)** Round-robin

(C) Priority based **(D)** Shortest-Job-next

Answer: B

Q Consider the following processes with time slice of 4 milliseconds (I/O requests are ignored):

Process	Arrival Time	Processing Time
A	0	8
B	1	4
C	2	9
D	3	5

The average turnaround time of these processes will be **(NET-June-2013)**

(A) 19.25 milliseconds **(B)** 18.25 milliseconds

(C) 19.5 milliseconds **(D)** 18.5 milliseconds

Answer: B

Q ……. is one of pre-emptive scheduling algorithm. **(NET-June-2010)**

(A) RR **(B)** SSN **(C)** SSF **(D)** Priority based

Answer: A

Q An example of a non-preemptive CPU scheduling algorithm is: **(NET-June-2008)**

(A) Shortest job first scheduling **(B)** Round robin scheduling.

(C) Priority scheduling **(D)** Fair share scheduling.

Answer: A

Q Four jobs are waiting to be run. Their expected run times are 6,3,5 and x. In what order should they be run to minimize the average response time? **(GATE-1998) (2 Marks)**

Longest Job First

- Process having longest Burst Time will get scheduled first.
- It can be both pre-emptive and non-pre-emptive in nature.
- The pre-emptive version is referred to as Longest Remaining Time First (LRTF) Scheduling Algorithm.

Example:

P. No	AT	BT
P_1	0	3
P_2	1	2
P_3	2	4
P_4	3	5
P_5	4	6

P_1 (0-3)	P_4 (3-8)	P_5 (8-14)	P_3 (14-18)	P_2 (18-20)

P. No	AT	BT	CT	TAT	WT
P_1	0	3	3	3	0
P_2	1	2	20	19	17
P_3	2	4	18	16	12
P_4	3	5	8	5	0
P_5	4	6	14	10	4
Total				53	33

Average TAT: 53/5 = 10.6 ms
Average WT: 33/5 = 6.6 ms

Longest Remaining Time First (LRTF)

- It is pre-emptive in nature.
- Longest Burst time process will get scheduled first.

Example:

P. No	AT	BT
P_1	0	2
P_2	0	4
P_3	0	8

P_1 (0-1)	P_3 (1-2)	P_3 (2-3)	P_3 (3-4)	P_3 (4-5)	P_2 (5-6)	P_3 (6-7)	P_2 (7-8)

P_3 (8-9)	P_2 (9-10)	P_3 (10-11)	P_1 (11-12)	P_2 (12-13)

P. No	AT	BT	CT	TAT	WT
P_1	0	2	12	12	10
P_2	0	4	13	13	9
P_3	0	8	14	14	6

Average TAT: 39/3 = 13 ms
Average WT: 25/3 = 8.33 ms

Q Consider three processes (process id 0, 1, 2 respectively) with compute time bursts 2, 4- and 8-time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with the lowest process id. The average turnaround time is? **(GATE-2006) (2 Marks)**

(A) 13 units **(B)** 14 units **(C)** 15 units **(D)** 16 units

Answer: (A)

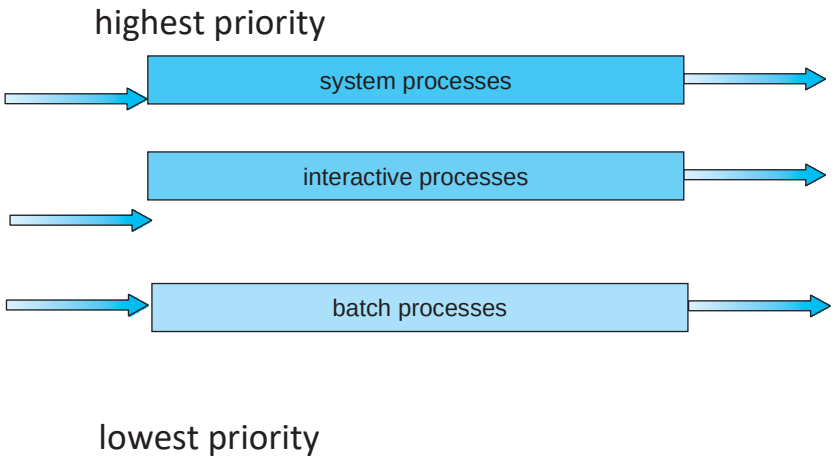
Highest response ratio next (HRRN)

- Scheduling is a non-pre-emptive discipline, similar to shortest job next (SJN), in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting.
- Jobs gain higher priority the longer they wait, which prevents indefinite waiting or in other words what we say starvation. In fact, the jobs that have spent a long time waiting compete against those estimated to have short run times.
- Response Ratio = (W + S)/S
- Here, **W** is the waiting time of the process so far and **S** is the Burst time of the process.
- So, the conclusion is it gives priority to those processes which have less burst time (or execution time) but also takes care of the waiting time of longer processes, thus preventing starvation.

Q highest response ratio next scheduling policy favours ------- jobs, but it also limits the waiting time of ---------- jobs. **(GATE-1990) (1 Marks)**
Answer: "shorter, longer"

MULTIVEL QUEUE SCGEDULING

- Another class of scheduling algorithm has been created for situations in which processes are easily classified into different groups. For example, a common division is made between foreground (interactive) processes and background (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm. For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. For example, the foreground queue may have absolute priority over the background queue.



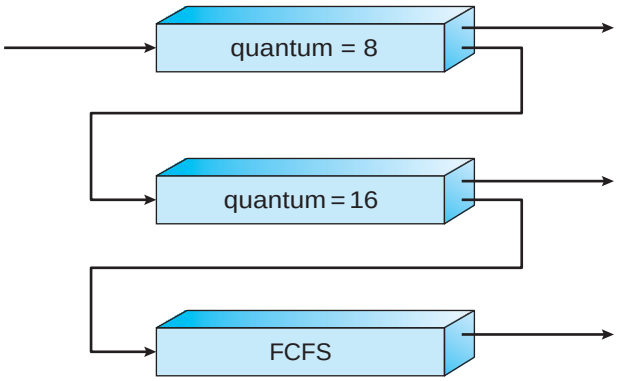
- Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.
- Another possibility is to time-slice among the queues. Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes. For instance, in the foreground– background queue example, the foreground queue can be given 80 percent of the CPU time for RR scheduling among its processes, while the background queue receives 20 percent of the

CPU to give to its processes on an FCFS basis.

Q Consider the following justifications for commonly using the two-level CPU scheduling:
I. It is used when memory is too small to hold all the ready processes.
II. Because its performance is same as that of the FIFO.
III. Because it facilitates putting some set of processes into memory and a choice is made from that.
IV. Because it does not allow to adjust the set of in-core processes.
Which of the following is true? **(NET-Dec-2014)**
(A) I, III and IV **(B)** I and II **(C)** III and IV **(D)** I and III
Answer: D

MULTIVEL QUEUE SCHEDULING WITH FEEDBACK

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature. This setup has the advantage of low scheduling overhead, but it is inflexible.
- The multilevel feedback queue scheduling algorithm, in contrast, allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues. In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



- A process entering the ready queue is put in queue 0.A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- This scheduling algorithm gives highest priority to any process with a CPU burst of 8 milliseconds or less. Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/O burst. Processes that need more than 8 but less than 24 milliseconds are also served quickly, although with lower priority than shorter processes. Long processes automatically sink to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.
- In general, a multilevel feedback queue scheduler is defined by the following

parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher-priority queue
- The method used to determine when to demote a process to a lower-priority queue
- The method used to determine which queue a process will enter when that process needs service

- The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it is also the most complex algorithm since defining the best scheduler requires some means by which to select values for all the parameters.

Q Which of the following statements is not true for Multi-Level Feedback Queue processor scheduling algorithm? **(NET-June-2015)**
a) Queues have different priorities
b) Each queue may have different scheduling algorithm
c) Processes are permanently assigned to a queue
d) This algorithm can be configured to match a specific system under design
Answer: C

Q An example of a non-pre-emptive scheduling algorithm is: **(NET-Dec-2008)**
(A) Round Robin
(B) Priority Scheduling
(C) Shortest job first
(D) 2 level scheduling
Answer: C

Q Which of the following scheduling algorithms is non-pre-emptive? **(GATE-2001) (1 Marks)**
(A) Round Robin
(B) First-In First-Out
(C) Multilevel Queue Scheduling
(D) Multilevel Queue Scheduling with Feedback
Answer: (B)

Q Which of the following scheduling algorithms is non-preemptive? **(GATE-2002) (1 Marks)**

(A) Round Robin **(B)** First-In First-Out

(C) Multilevel Queue Scheduling **(D)** Multilevel Queue Scheduling with Feedback

Answer: (B)

Q Consider a set of n tasks with known runtimes r_1, r_2, …. r_n to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput? **(GATE-2001) (1 Marks)**

(A) Round-Robin **(B)** Shortest-Job-First

(C) Highest-Response-Ratio-Next **(D)** First-Come-First-Served

Answer: (B)

Process Synchronization

- As we understand in a multiprogramming environment a good number of processes compete for limited number of resources.
- Concurrent access to shared data at same time may result in data inconsistency for e.g.

```
P ()
{
        read ( i );
        i = i + 1;
        write( i );
}
```

- **Race Condition** - if we change the order of execution of different process with respect to other process the output may change.
- That is why we need some kind of synchronization to eliminate the possibility of data inconsistency.

Q The following two functions P_1 and P_2 that share a variable B with an initial value of 2 execute concurrently.

$P_1()$	$P_2()$
{	{
C = B – 1;	D = 2 * B;
B = 2 * C;	B = D - 1;
}	}

The number of distinct values that B can possibly take after the execution is **(GATE-2015) (1 Mark)**

Ans. 3

Q Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100. **(GATE-2019) (2 Marks)**

P_1	P_2	P_3
.	.	.

.	.	.
D = D + 20	**D = D - 50**	**D = D + 10**
.	.	.
.	.	.

The process are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then the value of Y–X is _____.

Q When the result of a computation depends on the speed of the processes involved, there is said to be **(GATE-1998) (1 Marks)**

a) cycle stealing b) race condition c) a time lock d) a deadlock

Ans: b

Q Consider the following C code for process P1 and P2. a=4, b=0, c=0 (initialization)?

P_1()

```
{
    if (a < 0)
        c = b - a;
    else
        c = b + a;
}
```

P_2()

```
{
    b=10;
    a=-3;
}
```

If the processes P1 and P2 executes concurrently (shared variables a, b and c), which of the following cannot be the value of 'c' after both processes complete?

a) 4 **b)** 7 **c)** 10 **d)** 13

Answer: C

Critical Section Problem

General Structure of a process

- **Initial Section:** Where process is accessing private resources.
- **Entry Section:** Entry Section is that part of code where, each process request for permission to enter its critical section.
- **Critical Section:** Where process is access shared resources.
- **Exit Section:** It is the section where a process will exit from its critical section.
- **Remainder Section:** Remaining Code.

```
Pi()
{
    While(T)
    {
        Initial Section
        Entry Section
        Critical Section
        Exit Section
        Remainder Section
    }
}
```

Criterion to Solve Critical Section Problem

- **Mutual Exclusion:** No two processes should be present inside the critical section at the same time, i.e. only one process is allowed in the critical section at an instant of time.

- **Progress:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next(means other process will participate which actually wish to enter), and there should be no deadlock.

- **Bounded Waiting:** There exists a bound or a limit on the number of times a process is allowed to enter its critical section and no process should wait indefinitely to enter the CS.

Some Points to Remember:

- Mutual Exclusion and Progress are mandatory requirements that needs to be followed in order to write a valid solution for critical section problem.
- Bounded waiting is optional criteria, if not satisfied then it may lead to starvation.

Q Suppose P, Q and R are co-operating processes satisfying Mutual Exclusion condition. Then, if the process Q is executing in its critical section then **(NET-DEC-2018)**
a) Both 'P' and 'R' execute in critical section
b) Neither 'P' nor 'R' executes in their critical section
c) 'P' executes in critical section
d) 'R' executes in critical section
Ans: b

Q Which of the following conditions does not hold good for a solution to a critical section problem? **(NET-DEC-2014)**
(A) No assumptions may be made about speeds or the number of CPUs.
(B) No two processes may be simultaneously inside their critical sections.
(C) Processes running outside its critical section may block other processes.
(D) Processes do not wait forever to enter its critical section if bounded wait is satisfied.
Answer: C

Q Part of a program where the shared memory is accessed and which should be executed indivisibly, is called: **(NET-JUNE-2007)**
a) Semaphores **b)** Directory **c)** Critical Section d) Mutual exclusion
Answer: C

Solutions to Critical Section Problem:

We generally have the following solutions to a Critical Section Problems:

1. Two Process Solution
   (a) Using Boolean variable turn
   (b) Using Boolean array flag
   (c) Peterson's Solution

2. Operating System Solution
   (a) Counting Semaphore
   (b) Binary Semaphore

3. Hardware Solution
   (a) Test and Set Lock

4. Computer and Programming Support Type
   (a) Monitors

Two Process Solution

- In general, it will be difficult to write a valid solution in the first go to solve critical section problem among multiple processes, so it will be better to first attempt two process solution and then generalize it to N-Process solution.
- There are 3 Different idea to achieve valid solution, in which some are invalid while some are valid.

  1- Using Boolean variable turn

  2- Using Boolean array flag

  3- Peterson's Solution

- Here we will use a Boolean variable turn, which is initialize randomly (0/1)

P_0	P_1

while (1)
{
while (turn! = 0);
Critical Section
turn = 1;
Remainder section
}
	while (1)
	{
	while (turn! = 1);
	Critical Section
	turn = 0;
	Remainder Section
	}

- The solution follows **Mutual Exclusion** as the two processes cannot enter the CS at the same time.
- The solution does not follow the **Progress,** as it is suffering from the strict alternation. We never asked the process whether it wants to enter the CS or not?

- Here we will use a Boolean array flag with two cells, where each cell is initialized to 0.

P_0	P_1

while (1)
{
flag [0] = T;
while (flag [1]);
Critical Section
flag [0] = F;
Remainder section
}
	while (1)
	{
	flag [1] = T;
	while (flag [0]);
	Critical Section
	flag [1] = F;
	Remainder Section
	}

- This solution follows the **Mutual Exclusion** Criteria.
- But in order to achieve the progress the system ended up being in a **deadlock state**.

Peterson's Solution

- It is a classic software-based solution to the critical-section problem.
- Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections.
- The processes are numbered P_0 and P_1.

We will be using two data items: **int turn and Boolean flag**.

P_0	P_1

while (1)
{
flag [0] = T;
turn = 1;
while (turn = = 1 && flag [1] = = T);
Critical Section
flag [0] = F;
Remainder section
}
	while (1)
	{
	flag [1] = T;
	turn = = 0;
	while (turn = = 0 && flag [0] = =T);
	Critical Section
	flag [1] = F;
	Remainder Section
	}

- This solution ensures **Mutual Exclusion, Progress and Bounded Wait.**

Q Consider the methods used by processes P_1 and P_2 for accessing their critical sections whenever needed, as given below. The initial values of shared Boolean variables S_1 and S_2 are randomly assigned. **(GATE-2010) (1 Marks)** **(NET-JUNE-2012)**

$P_1()$	$P_2()$
While (S_1==S_2);	While (S_1!= S_2);
Critical section	Critical section
S_1= S_2;	SS_2=not (S_1);

Which one of the following statements describes the properties achieved?

(A) Mutual exclusion but not progress
(B) Progress but not mutual exclusion
(C) Neither mutual exclusion nor progress
(D) Both mutual exclusion and progress

Answer: (A)

Q Two processes, P_1 and P_2, need to access a critical section of code. Consider the following synchronization construct used by the processes: Here, $wants_1$ and $wants_2$ are shared

variables, which are initialized to false. Which one of the following statements is TRUE about the above construct? **(GATE-2007) (2 Mark)**

$P_1()$	$P_2()$
While(t)	While(t)
{	{
$wants_1$ = T	$wants_1$ = T
While ($wants_1$== T);	While ($wants_1$== T);
Critical section	Critical section
$wants_1$ = F	$wants_1$ = F
Remainder section	Remainder section
}	}

(A) It does not ensure mutual exclusion.
(B) It does not ensure bounded waiting.
(C) It requires that processes enter the critical section in strict alternation.
(D) It does not prevent deadlocks, but ensures mutual exclusion.
Answer: (D)

Q Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes.
Here, varP and varQ are shared variables and both are initialized to false. Which one of the following statements is true? **(GATE-2015) (1 Mark)**

Process X	Process Y
While(t)	While(t)
{	{
varP = T;	varQ = T;
While (varQ == T)	While (varP == T)
{	{
Critical section	Critical section
varP = F;	varQ = F;
}	}
}	}

(A) The proposed solution prevents deadlock but fails to guarantee mutual exclusion
(B) The proposed solution guarantees mutual exclusion but fails to prevent deadlock

(C) The proposed solution guarantees mutual exclusion and prevents deadlock
(D) The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion
Answer: (A)

Q Consider the following two-process synchronization solution. **(GATE-2016) (2 Marks)**

Process 0	Process 1
Entry: loop while (turn == 1);	Entry: loop while (turn == 0);
(critical section)	(critical section)
Exit: turn = 1;	Exit: turn = 0;

The shared variable turn is initialized to zero. Which one of the following is TRUE?
(a) This is a correct two-process synchronization solution.
(b) This solution violates mutual exclusion requirement
(c) This solution violates progress requirement.
(d) This solution violates bounded wait requirement
Answer: (C)

Q Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below. **(GATE-2001) (2 Mark)**

Repeat
flag[i] = T;
turn = j;
while(P) do no-op;
Enter critical section, perform actions, then critical section
flag[i] = f;
Perform other non-critical section actions
Until false;

For the program to guarantee mutual exclusion, the predicate P in the while loop should be.
(A) flag[j] = true and turn = I
(B) flag[j] = true and turn = j
(C) flag[i] = true and turn = j
(D) flag[i] = true and turn = i
Answer: (B)

Operating System Solution

- Semaphores are synchronization tools using which we will attempt n-process solution.
- A semaphore S is a simple integer variable that, but apart from initialization it can be accessed only through two standard atomic operations: wait(S) and signal(S).
- The wait(S) operation was originally termed as P(S) and signal(S) was originally called V(S).

```
Wait(S)
{
    while(s<=0);
    s--;
}
```

```
Signal(S)
{
    s++;
}
```

- Peterson's Solution was confined to just two processes, and since in a general system can have n processes, Semaphores provides n-processes solution.
- While solving Critical Section Problem only we initialize semaphore S = 1.

```
Pi()
{
    While(T)
    {
        Initial Section
        wait(s)
        Critical Section
        signal(s)
        Remainder Section
    }
}
```

- Semaphores are going to ensure Mutual Exclusion and Progress but does not ensures bounded waiting.
- It is critical that semaphore operations be executed atomically. We must guarantee that no two processes can execute wait() and signal() operations on the same semaphore at the

same time. This is a critical-section problem; and in a single-processor environment, we can solve it by simply inhibiting interrupts during the time the wait() and signal() operations are executing. This scheme works in a single-processor environment because, once interrupts are inhibited, instructions from different processes cannot be interleaved. Only the currently running process executes until interrupts are reenabled and the scheduler can regain control.

- In a multiprocessor environment, interrupts must be disabled on every processor. Otherwise, instructions from different processes (running on different processors) may be interleaved in some arbitrary way. Disabling interrupts on every processor can be a difficult task and furthermore can seriously diminish performance. Therefore, SMP systems must provide alternative locking techniques —such as compare and swap() or spinlocks —to ensure that wait() and signal() are performed atomically.

Q..... synchronize critical resources to prevent deadlock. **(NET-JUNE-2006)**
a) P-operator **b)** V-operator **c)** Semaphores **d)** Hard disk
Answer: C

Q Two atomic operations permissible on Semaphores are _____ and _____. (NET-NOV-2017)
a) wait, stop **b)** wait, hold **c)** hold, signal **d)** wait, signal
Answer: D

Q A relationship between processes such that each has some part (critical section) which must not be executed while the critical section of another is being executed, is known as (NET-JUNE-2011)
a) Semaphore **b)** Mutual exclusion
b) Multiprogramming **d)** Message passing
Answer: B

Q In order to allow only one process to enter its critical section, binary semaphore are initialized to (NET-JUNE-2010)
a) 0 **b)** 1 **c)** 2 **d)** 3
Answer: B

Q Semaphores are used to: **(NET-DEC-2004)**
a) Synchronise critical resources to prevent deadlock

b) Synchronise critical resources to prevent contention
c) Do I/o
d) Facilitate memory management
Answer: A

Q A critical section is a program segment? **(GATE-1996) (2 Mark)**
(a) which should run in a certain specified amount of time
(b) which avoids deadlocks
(c) where shared resources are accessed
(d) which must be enclosed by a pair of semaphore operations, P and V
Answer: (A)

Q A critical region is **(GATE-1987) (1 Marks)**
a) One which is enclosed by a pair of P and V operations on semaphores.
c) A program segment that has not been proved bug-free.
c) A program segment that often causes unexpected system crashes.
d) A program segment where shared resources are accessed.
Answer: (D)

Q Suppose S and Q are two semaphores initialized to 1. P1 and P2 are two processes which are sharing resources. **(NET-JUNE-2013)**

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
Critical section 1;	critical-section 2;
signal(S);	signal(Q);
signal(Q);	signal(S);

Their execution may sometimes lead to an undesirable situation called
(A) Starvation
(B) Race condition
(C) Multithreading
(D) Deadlock

Q Given below is a program which when executed spawns two concurrent processes:
semaphore X : = 0 ;
/* Process now forks into concurrent processes P1 & P2 */

P_1	P_2
repeat forever	repeat forever

V (X) ; **Compute ;** **P(X) ;**	P(X) ; Compute ; V(X) ;

Consider the following statements about processes P1 and P2:

1. It is possible for process P1 to starve.
2. It is possible for process P2 to starve.

Which of the following holds? **(GATE-2005) (2 Marks)**

(A) Both I and II are true
(B) I is true but II is false
(C) II is true but I is false
(D) Both I and II are false

Answer: (A)

Q Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below.

Process P	Process Q
While(t)	While(t)
{	{
W:	Y:
print '0';	print '0';
print '0';	print '0';
X:	Z:
}	}

Synchronization statements can be inserted only at points W, X, Y and Z.

Which of the following will always lead to an output staring with '001100110011' ? **(GATE-2004) (2 Marks)**

(A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1
(B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0
(C) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
(D) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S initially 1, and T initially 0

Answer: (B)

Q Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd? **(GATE-2004) (2 Marks)**

(A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1

(B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1
(C) P(S) at W, V(S) at X, P(S) at Y, V(S) at Z, S initially 1
(D) V(S) at W, V(T) at X, P(S) at Y, P(T) at Z, S and T initially 1
Answer: (C)

Q Two concurrent processes P_1 and P_2 use four shared resources R_1, R_2, R_3 and R_4, as shown below.

P_1	P_2
Compute: Use R_1; Use R_2; Use R_3; Use R_4;	Compute; Use R_1; Use R_2; Use R_3;. Use R_4;

Both processes are started at the same time, and each resource can be accessed by only one process at a time The following scheduling constraints exist between the access of resources by the processes:

- P2 must complete use of R_1 before P_1 gets access to R_1
- P1 must complete use of R_2 before P_2 gets access to R_2.
- P2 must complete use of R_3 before P_1 gets access to R_3.
- P1 must complete use of R_4 before P_2 gets access to R_4.

There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed? **(GATE-2005) (2 Marks)**
(A) 1 **(B)** 2 **(C)** 3 **(D)** 4
Answer: (B)

Q A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution? **(GATE-2013) (2 Marks)**
(A) -2 **(B)** -1 **(C)** 1 **(D)** 2
Answer: (D)

Q Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d, and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the V operation (i.e., signal) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes? **(GATE-2013) (1 Marks)**

(a) X: P(a)P(b)P(c) Y: P(b)P(c)P(d) Z: P(c)P(d)P(a)
(b) X: P(a)P(b)P(c) Y: P(b)P(c)P(d) Z: P(c)P(d)P(a)
(c) X: P(b)P(a)P(c) Y: P(c)P(b)P(d) Z: P(a)P(c)P(d)
(d) X: P(a)P(b)P(c) Y: P(c)P(b)P(d) Z: P(c)P(d)P(a)

Answer: (B)

Q Consider two processes P1 and P2 accessing the shared variables X and Y protected by two binary semaphores SX and SY respectively, both initialized to 1. P and V denote the usual semaphone operators, where P decrements the semaphore value, and V increments the semaphore value. The pseudo-code of P1 and P2 is as follows: **(GATE-2004) (2 Marks)**

P_0	P_1
While true do {	While true do {
L1 : …….	L3 : …….
L2 : …….	L4 : …….
X = X + 1;	Y = Y + 1;
Y = Y – 1;	X = Y – 1;
V(SX);	V(SX);
V(SY);	V(SY);
}	}

In order to avoid deadlock, the correct operators at L1, L2, L3 and L4 are respectively

(A) P(SY), P(SX); P(SX), P(SY)
(B) P(SX), P(SY); P(SY), P(SX)
(C) P(SX), P(SX); P(SY), P(SY)
(D) P(SX), P(SY); P(SX), P(SY)

Answer: (D)

Classical Problems on Synchronization

- There are number of actual industrial problem we try to solve in order to improve our understand of Semaphores and their power of solving problems.
- Here in this section we will discuss a number of problems like
    - Producer consumer problem/ Bounder Buffer Problem
    - Reader-Writer problem
    - Dining Philosopher problem

Producer-Consumer Problem

- Problem Definition – There are two process Producer and Consumers, producer produces information and put it into a buffer which have n cell, that is consumed by a consumer.
- Both Producer and Consumer can produce and consume only one article at a time.

Producer()	Consumer()
{	{
while(T)	while(T)
{	{
Produce()	wait(F)//UnderFlow
wait(E)//OverFlow	wait(S)
wait(S)	pick()
append()	signal(S)
signal(S)	wait(E)
wait(F)	consume()
}	}
}	}

- Producer-Consumer Problem needs to sort out three major issues:
- A producer needs to check whether the buffer is overflowed or not after producing an item before accessing the buffer.
- Similarly, a consumer needs to check for an underflow before accessing the buffer and then consume an item.
- Also, *the producer and consumer must be synchronized, so that once a producer and consumer it accessing the buffer the other must wait.*

Solution Using Semaphores

Now to solve the problem we will be using three semaphores:

Semaphore S = 1

Semaphore E = n

Semaphore F = 0

Total three resources are used

- semaphore E take count of empty cells and over flow
- semaphore F take count of filled cells and under flow
- Semaphore S take care of buffer

Q The Bounded buffer problem is also known as _____. (NET-NOV-2017)
a) Producer - consumer problem
b) Reader - writer problem
c) Dining Philosophers problem
d) Both (2) and (3)
Answer: A

Q Producer consumer problem can be solved using: **(NET-DEC-2005)**
a) semaphores **b)** event counters **c**) monitors d) all the above
Answer: D

Q The semaphore variables full, empty and mutex are initialized to 0, n and 1, respectively. Process P_1 repeatedly adds one item at a time to a buffer of size n, and process P_2 repeatedly removes one item at a time from the same buffer using the programs given below. In the programs, K, L, M and N are unspecified statements. **(GATE-2004) (2 Marks)**

```
P1
while (1) {
   K;
   P(mutex);
   Add an item to the buffer;
   V(mutex);
    L;
 }

P2
while (1) {
    M;
    P(mutex);
    Remove an item from the buffer;
    V(mutex);
    N;
}
```

The statements K, L, M and N are respectively
(A) P(full), V(empty), P(full), V(empty) **(B)** P(full), V(empty), P(empty), V(full)

(C) P(empty), V(full), P(empty), V(full) **(D)** P(empty), V(full), P(full), V(empty)
Answer: (D)

Q Consider the procedure below for the Producer-Consumer problem which uses semaphores:
Semaphore n = 0;
Semaphore s = 1;

Void Producer ()	Void Consumer ()
{	{
While(true)	While(true)
{	{
Produce ();	semWait(s);
SemWait(s);	semWait(n);
addToBuffer();	RemovefromBuffer();
semSignal(s);	semSignal(s);
SemSignal(n);	consume();
}	}
}	}

Which one of the following is TRUE? **(GATE-2014) (2 Marks)**
(A) The producer will be able to add an item to the buffer, but the consumer can never consume it.
(B) The consumer will remove no more than one item from the buffer.
(C) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.
(D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.
Answer: (C)

Q Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores *empty, full* and *mutex* are defined with respective initial values of 0, N and 1. Semaphore *empty* denotes the number of available slots in the buffer, for the consumer to read from. Semaphore *full* denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either *empty* or *full*. The valid semaphore operations are: *wait()* and *signal()*. **(GATE-2018) (2 Marks)**

Producer:	Consumer:
Do{	Do{
Wait(P);	Wait(R);
Wait(mutex);	Wait(mutex);
//Add item to buffer	//Consume item from buffer
Signal(mutex);	Signal(mutex);
Signal(Q);	Signal(S);
} while(1);	} while(1);

Which one of the following assignments to P, Q, R and S will yield the correct solution?

(A) P: *full*, Q: *full*, R: *empty*, S: *empty*

(B) P: *empty*, Q: *empty*, R: *full*, S: *full*

(C) P: *full*, Q: *empty*, R: *empty*, S: *full*

(D) P: *empty*, Q: *full*, R: *full*, S: *empty*

Answer: (C)

Reader-Writers Problem

- Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database (readers), whereas others may want to update (that is, to read and write) the database(writers). The former are referred to as readers and to the latter as writers.
- If two readers access the shared data simultaneously, no adverse effects will result. But, if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.
- To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database.
- Points that needs to be taken care for generating a Solutions:
- The solution may allow more than one reader at a time, but should not allow any writer.
- The solution should strictly not allow any reader or writer, while a writer is performing a write operation.
- Solution using Semaphores
- The reader processes share the following data structures:
- semaphore mutex = 1, wrt =1; // Two semaphores
- int readcount = ; // Variable
- Three resources are used
- Semaphore Wrt is used for synchronization between WW, WR, RW
- Semaphore reader is used to synchronize between RR
- Readcount is simple int variable which keep counts of number of readers

Writer()	Reader()
Wait(wrt)	Wait(mutex)
CS //Write	Readcount++
Signal(wrt)	If(readcount ==1)
	wait(wrt)
	signal(mutex)
	CS //Read
	Wait(mutex)
	Readcount--
	If(readcount ==0)
	signal(wrt)
	signal(mutex)

Q To overcome difficulties in Readers-Writers problem, which of the following statement/s is/are true? **(NET-DEC-2018)**

1) Writers are given exclusive access to shared objects

2) Readers are given exclusive access to shared objects

3) Both readers and writers are given exclusive access to shared objects.

Choose the correct answer from the code given below:

a) 1 **b)** 2 **c)** 3 **d)** 2 and 3 only

Ans: a

Q Let P_i and P_j two processes, R be the set of variables read from memory, and W be the set of variables written to memory. For the concurrent execution of two processes pi and Pj which of the conditions are not true? **(NET-JUNE-2015)**

a) $R(Pi) \cap W(Pj) = \Phi$ **b)** $W(Pi) \cap R(Pj) = \Phi$

c) $R(Pi) \cap R(Pj) = \Phi$ **d)** $W(Pi) \cap W(Pj) = \Phi$

Q Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization **(GATE-2007) (2 Marks)**

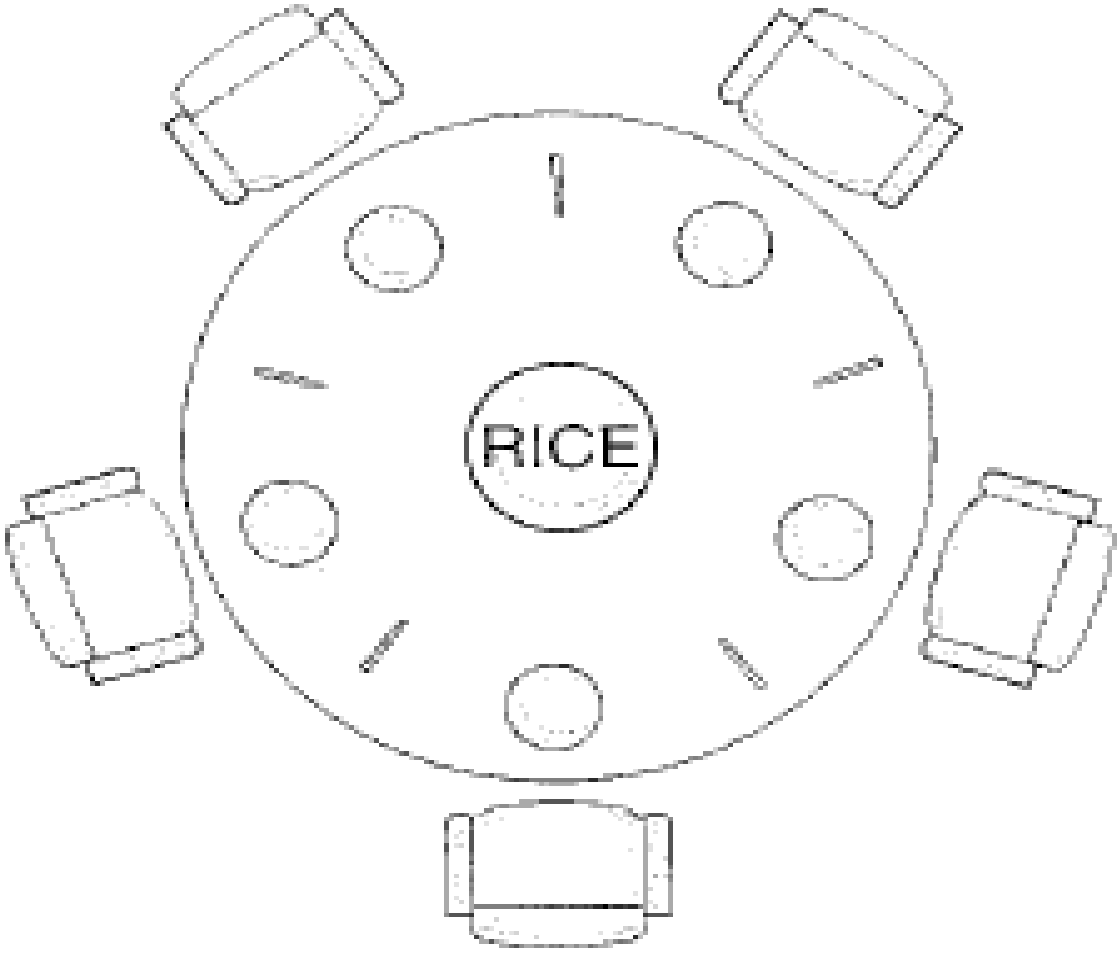
wait (wrt)
writing is performed

signal (wrt)
wait (mutex)
readcount = readcount + 1
if readcount = 1 then S1
S2
reading is performed
S3
readcount = readcount – 1
if readcount = 0 then S4
signal (mutex)
The values of S1, S2, S3, S4, (in that order) are
(A) signal (mutex), wait (wrt), signal (wrt), wait (mutex)
(B) signal (wrt), signal (mutex), wait (mutex), wait (wrt)
(C) wait (wrt), signal (mutex), wait (mutex), signal (wrt)
(D) signal (mutex), wait (mutex), signal (mutex), wait (mutex)
Answer: (C)

Dining-Philosopher Problem

Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher.

- In the center of the table is a bowl of rice, and the table is laid with five single chopsticks.
- When a philosopher thinks, she does not interact with her colleagues.
- From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).
- A philosopher may pick up only one chopstick at a time. Obviously, she can't pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again.



Code that needs to be followed by philosophers is:

```
Void Philosopher (void)
{
        while ( T )
        {
                Thinking ( ) ;
                wait(chopstick [i]);
                wait(chopstick([(i+1)%5]);
                Eat( );
                signal(chopstick [i]);
                signal(chopstick([(i+1)%5]);
        }
}
```

- Here we have used an array of semaphores called chopstick[]
- Solution is not valid because there is a possibility of deadlock.
- Here we have used an array of semaphores called chopstick[]
- Solution is not valid because there is a possibility of deadlock.
- The proposed solution for deadlock problem is
    - Allow at most four philosophers to be sitting simultaneously at the table.
    - Allow six chopstick to be used simultaneously at the table.
    - Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in a critical section).
    - One philosopher picks up his right chopstick first and then left chop stick, i.e. reverse the sequence of any philosopher.
    - Odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Q Dining Philosopher's problem is a: (NET-JUNE-2015)

a) Producer - consumer problem
b) Classical IPC problem
c) Starvation problem
d) Synchronization primitive

Q A solution to the Dining Philosophers Problem which avoids deadlock is: **(GATE-1996) (2 Marks)**

(A) ensure that all philosophers pick up the left fork before the right fork
(B) ensure that all philosophers pick up the right fork before the left fork
(C) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork
(D) None of the above
Answer: (C)

Q Let m[0]…m[4] be mutexes (binary semaphores) and P[0] …. P[4] be processes.
Suppose each process P[i] executes the following:

wait (m[i]); wait(m[(i+1) mode 4]);
---
release (m[i]); release (m[(i+1)mod 4]);

This could cause: **(GATE-2000) (1 Marks)**

(A) Thrashing
(B) Deadlock
(C) Starvation, but not deadlock
(D) None of the above
Answer: (B)

Types of Semaphore

There are of two types of semaphores:

1. Binary Semaphores: The value of a binary semaphore can range only between 0 and 1.

2. Counting Semaphores: can range over an unrestricted domain Counting semaphore can range over an unrestricted domain. i.e. -∞ to +∞. Counting semaphores can be used to control access to a given resource consisting of a finite number of instances. The semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a wait() operation on the semaphore (thereby decrementing the count). When a process releases a resource, it performs a signal() operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0.

Implementation of semaphore

- This simple implementation of semaphore with this wait(S) and signal(S) function suffer from busy waiting.

```
Wait(S)
{
    while(s<=0);
    s--;
}
```

```
Signal(S)
{
    s++;
}
```

- To overcome the need for busy waiting, we can modify the definition of the wait() and signal() operations as follows: When a process executes the wait() operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.
- A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal() operation. The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue. (The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU-scheduling algorithm.)

To implement semaphores under this definition, we define a semaphore as follows:

```
typedef struct{
    int value;
    struct  process *list;
} semaphore;
```

Each semaphore has an integer value and a list of processes list.

Now, the wait() semaphore operation can be defined as

```
wait(semaphore *S){
    S->value--;
    if  (S->value  < 0){
        add this process to S->list;
        block();
    }
}
```

Signal() semaphore operation can be defined as

```
signal(semaphore  *S){
    S->value++;
    if  (S->value  <= 0){
        remove a process P from S->list;
        wakeup(P);
    }
}
```

- The block() operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls.
- Note that in this implementation, semaphore values may be negative, whereas semaphore values are never negative under the classical definition of semaphores with busy waiting. If a semaphore value is negative, its magnitude is the number of processes waiting on that semaphore. This fact results from switching the order of the decrement and the test in the implementation of the wait() operation.
- The list of waiting processes can be easily implemented by a link field in each process control block (PCB). Each semaphore contains an integer value and a pointer to a list of PCBs. One way to add and remove processes from the list so as to ensure bounded waiting is to use a FIFO queue, where the semaphore contains both head and tail pointers to the queue.

Q At a particular time of computation the value of a counting semaphore is 7. Then 20 P operations and 15V operations were completed on this semaphore. If the new value of semaphore is will be **(GATE-1992) (1 Marks)**

(A) 42 **(B)** 2 **(C)** 7 **(D)** 12

Answer: (B)

Q A counting semaphore was initialized to 10. Then 6P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is **(GATE-1998) (1 Marks)**

a) 0 b) 8 c) 10 d) 12

Ans: b

Q At a particular time of computation, the value of a counting semaphore is 10. Then 12 P operations and "x" V operations were performed on this semaphore. If the final value of semaphore is 7, x will be: (NET-JULY-2018)

a) 8 b) 9 c) 10 d) 11

Q There are three processes P1, P2 and P3 sharing a semaphore for synchronizing a variable. Initial value of semaphore is one. Assume that negative value of semaphore tells us how many processes are waiting in queue. Processes access the semaphore in following order : (NET-JAN-2017)

(a) P2 needs to access (b) P1 needs to access

(c) P3 needs to access (d) P2 exits critical section

(e) P1 exits critical section

The final value of semaphore will be:

a) 0 b) 1 c) -1 d) -2

Answer: A

Q A semaphore count of negative n means (s=–n) that the queue contains …….. waiting processes. (NET-DEC-2009)

a) n + 1 b) n c) n – 1 d) 0

Answer: B

Q Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 20 P(S) operations and 12 V(S) operations are issued in some order. The largest initial value of S for which at least one P(S) operation will remain blocked is _____. **(GATE-2016) (2 Marks)**

(A) 7 **(B)** 8 **(C)** 9 **(D)** 10

Answer: (A)

Hardware Type Solution Test and Set Lock (TSL)

software-based solutions such as Peterson's are not guaranteed to work on modern computer architectures. In the following discussions, we explore several more solutions to the critical-section problem using techniques ranging from hardware to software, all these solutions are based on the premise of locking —that is, protecting critical regions through the use of locks.

The critical-section problem could be solved simply in a single-processor environment if we could prevent interrupts from occurring while a shared variable was being modified.

```
Boolean test and set(Boolean *target){
    Boolean rv = *target;
    *target  = true;
    return rv;
}



While(1){
    while  (test and set(&lock));
    /* critical section */
   lock = false;
   /* remainder section */
}
```

Unfortunately, this solution is not as feasible in a multiprocessor environment. Disabling interrupts on a multiprocessor can be time consuming, since the message is passed to all the processors. This message passing delays entry into each critical section, and system efficiency decreases.

Many modern computer systems therefore provide special hardware instructions that allow us either to test and modify the content of a word atomically —that is, as one uninterruptible unit. We can use these special instructions to solve the critical-section problem in a relatively simple manner.

The important characteristic of this instruction is that it is executed atomically. Thus, if two test and set() instructions are executed simultaneously (each on a different CPU), they will be executed sequentially in some arbitrary order.

Q The enter_CS() and leave_CS() functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
   while test-and-set(X) ;
}

void leave_CS(X)
{
  X = 0;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements: **(GATE-2009) (2 Marks)**

I. The above solution to CS problem is deadlock-free
II. The solution is starvation free.
III. The processes enter CS in FIFO order.
IV More than one process can enter CS at the same time.

Which of the above statements is TRUE?

(A) I only **(B)** I and II **(C)** II and III **(D)** IV only

Answer: (A)

Q Fetch_And_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock(L){
       while (Fetch_And_Add(L,1))
            L = 1;
  }

 ReleaseLock(L){
       L = 0;
  }
```

This implementation **(GATE-2012) (2 Marks)**

(A) fails as L can overflow
(B) fails as L can take on a non-zero value when the lock is actually available
(C) works correctly but may starve some processes
(D) works correctly without starvation

Answer: (B)

Q The atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x In y without allowing any intervening access to the memory location x. consider the following implementation of P and V functions on a binary semaphore S.

```
void P (binary_semaphore *s)
{
   unsigned y;
   unsigned *x = &(s->value);
   do
   {
      fetch-and-set x, y;
   }
   while (y);
}
void V (binary_semaphore *s)
{
   S->value = 0;
}
```

Which one of the following is true? **(GATE-2006) (2 Marks)**

(A) The implementation may not work if context switching is disabled in P
(B) Instead of using fetch-and –set, a pair of normal load/store can be used
(C) The implementation of V is wrong
(D) The code does not implement a binary semaphore
Answer: (A)

Disable interrupt

- This could be a hardware solution where process have a privilege instruction, i.e. before entering into critical section, process will disable all the interrupts and at the time of exit, it again enables interrupts.
- This solution is only used by OS, as if some user process enter into critical section, then can block the entire system.

```
Pi()
{
    While(T)
    {
        Initial Section
        Entry Section//Disable interrupt
        Critical Section
        Exit Section//Enable interrupt
        Remainder Section
    }
}
```

Q In an operating system, indivisibility of operation means: **(NET-DEC-2015)**

(A) Operation is interruptible
(B) Race – condition may occur
(C) Processor cannot be pre-empted
(D) All of the above

Answer: (C)

Q The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0. **(GATE-2010) (2 Marks)**

Process P_0	**Process P_1**	**Process P_2**
While(true){	Wait(S_1);	Wait(S_2);
Wait(S_0);	Release(S_0);	Release(S_0);
Print'0'		
Release(S_1);		

Release(S_2);		
}		

How many times will process P0 print '0'?

(A) At least twice **(B)** Exactly twice **(C)** Exactly thrice **(D)** Exactly once

Answer: (A)

Q Each Process P_i, i= 1…….9 is coded as follows **(GATE-1997) (1 Marks)**

```
repeat
   P(mutex)
   {Critical section}
   V(mutex)
forever
```

The code for P_{10} is identical except it uses V(mutex) in place of P(mutex). What is the largest number of processes that can be inside the critical section at any moment?

(A) 1 **(B)** 2 **(C)** 3 **(D)** None of above

Answer: (D)

Q Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on left.

```
void barrier (void) {
1:   P(S);
2:   process_arrived++;
3.   V(S);
4:   while (process_arrived !=3);
5:   P(S);
6:   process_left++;
7:   if (process_left==3) {
8:      process_arrived = 0;
9:      process_left = 0;
10:  }
11:  V(S);
}
```

The variables process_arrived and process_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

Q The above implementation of barrier is incorrect. Which one of the following is true? **(GATE-**

2006) (2 Marks)

a) The barrier implementation is wrong due to the use of binary semaphore S.

b) The barrier implementation may lead to a deadlock if two barrier in invocations are used in immediate succession.

c) Lines 6 to 10 need not be inside a critical section.

d) The barrier implementation is correct if there are only two processes instead of three.

Answer: (B)

Q Which one of the following rectifies the problem in the implementation? **(GATE-2006) (2 Marks)**

(A) The barrier implementation is wrong due to the use of binary semaphore S

(B) The barrier implementation may lead to a deadlock if two barrier in invocations are used in immediate succession.

(C) Lines 6 to 10 need not be inside a critical section

(D) The barrier implementation is correct if there are only two processes instead of three.

Answer: (B)

Q The P and V operations on counting semaphores, where s is a counting semaphore, are defined as follows:

```
P(s) : s =  s - 1;
  if (s  < 0) then wait;

V(s) : s = s + 1;
  if (s <= 0) then wakeup a process waiting on s;
```

Assume that P_b and V_b the wait and signal operations on binary semaphores are provided. Two binary semaphores X_b and Y_b are used to implement the semaphore operations P(s) and V(s) as follows:

```
P(s) : Pb(Xb);
  s = s - 1;
  if (s < 0) {
   Vb(Xb) ;
   Pb(Yb) ;
  }
  else Vb(Xb);

V(s) : Pb(Xb) ;
  s = s + 1;
  if (s <= 0) Vb(Yb) ;
  Vb(Xb) ;
```

The initial values of X_b and Y_b are respectively **(GATE-2008) (2 Marks)**

(A) 0 and 0 **(B)** 0 and 1 **(C)** 1 and 0 **(D)** 1 and 1

Answer: (C)

Q Suppose a processor does not have any stack pointer register. Which of the following statements is true? **(GATE-2001) (1 Marks)**

(A) It cannot have subroutine call instruction

(B) It can have subroutine call instruction, but no nested subroutine calls

(C) Nested subroutine calls are possible, but interrupts are not

(D) All sequences of subroutine calls and also interrupts are possible

Answer: (A)

Q A certain computation generates two arrays a and b such that a[i]=f(i) for $0 \le i < n$ and b[i]=g(a[i]) for $0 \le i < n$. Suppose this computation is decomposed into two concurrent processes X and Y such that X computes the array a and Y computes the array b. The processes employ two binary semaphores R and S, both initialized to zero. The array a is shared by the two processes. The structures of the processes are shown below. **(GATE-2013) (2 Marks)**

Process X:	Process X:
private i;	private i;
for (i=0; i < n; i++) {	for (i=0; i < n; i++) {
a[i] = f(i);	EntryY(R, S);
ExitX(R, S);	b[i]=g(a[i]);
}	}

ExitX(R, S) {
P(R);
V(S);
}
EntryY(R, S) {
P(S);
V(R);
}

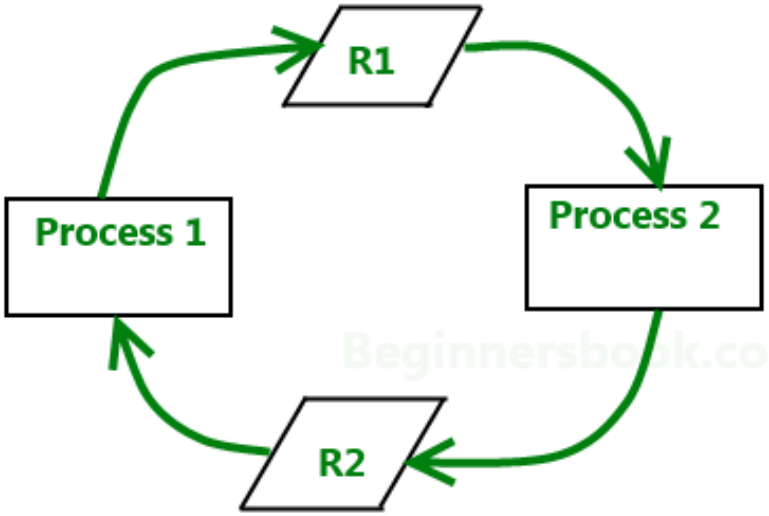
```
ExitX(R, S) {
P(S);
V(R);
}
EntryY(R, S) {
    V(S);
    P(R);
}
```

```
ExitX(R, S) {
V(R);
V(S);
}
EntryY(R, S) {
    P(R);
    P(S);
}
```

```
ExitX(R, S) {
V(R);
P(S);
}
EntryY(R, S) {
    V(S);
    P(R);
}
```

Basics of Dead-Lock

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.
- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. E.g. nawab, nuclear bomb
- Starvation is long waiting but deadlock is infinite waiting



Process P1 holds resource R2 and requires R1 while Process P2 holds resource R1 and requires R2.

System model

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request**. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- **Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release**. The process releases the resource.

Necessary conditions for deadlock

A deadlock can occur if all these 4 conditions occur in the system simultaneously.

- **Mutual exclusion**: - At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. And the resource Must be desired by more than one process. i.e. one by one e.g. Printer. E.g. sects and money, board
- **Hold and wait**: - A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes. E.g. Plate and spoon
- **No pre-emption**: - Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait**: - A set P_0, P_1, ..., P_n of waiting processes must exist such that P_0 is waiting for a 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 by P_0.

  We emphasize that all four conditions must hold for a deadlock to occur.

Deadlock Handling methods

- **Prevention**: - Design such protocols that there is no possibility of deadlock.
- **Avoidance**: - Try to avoid deadlock in run time so ensuring that the system will never enter a deadlocked state.
- **Detection**: - We can allow the system to enter a deadlocked state, then detect it, and recover.
- **Ignorance**: - We can ignore the problem altogether and pretend that deadlocks never occur in the system.

Last solution is the one used by most operating systems, including Linux and Windows. It is then up to the application developer to write programs that handle deadlocks.

Some researchers have argued that none of the basic approaches alone is appropriate for the entire spectrum of resource-allocation problems in operating systems. The basic approaches can be combined, however, allowing us to select an optimal approach for each class of resources in a system.

Polio, health, coaching / hospitals, frequency + severity

Q A Dead-lock in an Operating System is **(NET-DEC-2010)**

(A) Desirable process
(B) Undesirable process
(C) Definite waiting process
(D) All of the above

Answer: C

Prevention

It means designing such systems where there is no possibility of existence of deadlock. For that we have to remove one of the four necessary condition of deadlock.

Mutual exclusion: -In prevention approach, there is no solution for mutual exclusion as resource can't be made sharable as it is a hardware property and process also can't be convinced to do some other task. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.

Hold & wait: -

- In conservative approach, process is allowed to run if & only if it has acquired all the resources.
- An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.
- Wait time outs we place a max time outs up to which a process can wait. After which process must release all the holding resources & exit.

No pre-emption: -

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are pre-empted. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we pre-empt the desired resources from the waiting process and allocate them to the requesting process. If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be pre-empted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were pre-empted while it was waiting.

Circular wait: -

- We can eliminate circular wait problem by giving a natural number mapping to every resource and then any process can request only in the increasing order and if a process wants a lower number, then process must first release all the resource larger than that number and then give a fresh request.

Q Resources are allocated to the process on non-sharable basis is **(NET-JUNE-2012)**

(A) mutual exclusion
(B) hold and wait
(C) no pre-emption
(D) circular wait

Ans: a

Q An operating system implements a policy that requires a process to release all resources before making a request for another resource. Select the TRUE statement from the following: **(GATE-2008) (2 Marks)**

(A) Both starvation and deadlock can occur
(B) Starvation can occur but deadlock cannot occur
(C) Starvation cannot occur but deadlock can occur
(D) Neither starvation nor deadlock can occur

Answer: (B)

Q Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

I) Process should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.

II) The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers

III) The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers

IV) The resources are numbered uniquely. A process is allowed to request for resources only for a resource with resource number larger than its currently held resources

Which of the above policies can be used for preventing deadlock? **(GATE-2015) (2 Marks)**

(A) Any one of I and III but not II or IV
(B) Any one of I, III and IV but not II
(C) Any one of II and III but not I or IV
(D) Any one of I, II, III and IV

Answer: (D)

Q Which of the following is scheme to deal with deadlock? (NET-JUNE-2012)

(A) Time out
(B) Time in
(C) Both (A) & (B)
(D) None of the above

Ans: a

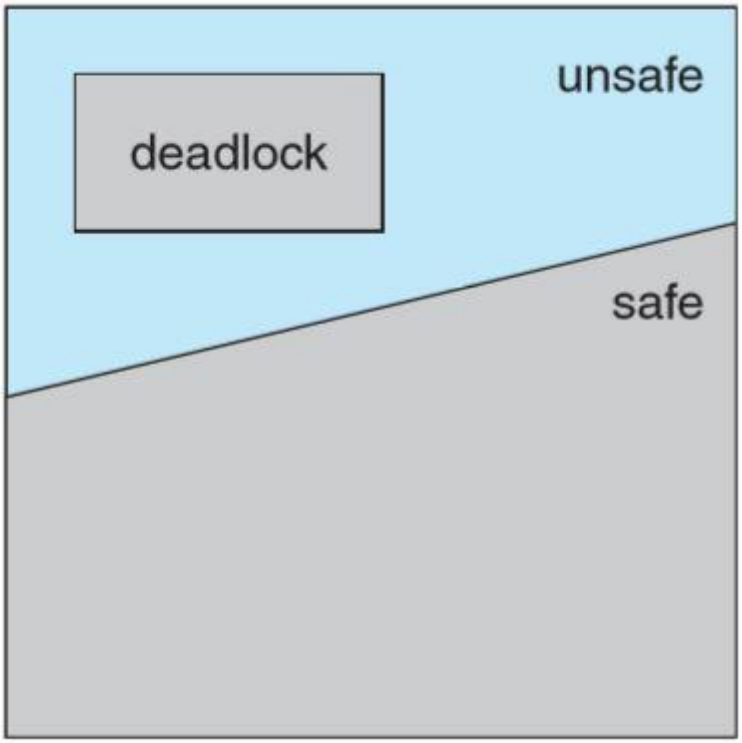
Q Which of the following is NOT a valid deadlock prevention scheme? **(GATE-2000) (2 Marks)**

(A) Release all resources before requesting a new resource
(B) Number the resources uniquely and never request a lower numbered resource than the last one requested.
(C) Never request a resource after releasing any resource
(D) Request and all required resources be allocated before execution.

Answer: (C)

Avoidance

- **Problem with Prevention: -** Different deadlock Prevention approach put different type of restrictions or conditions on the processes and resources Because of which system becomes slow and resource utilization and reduced system throughput.
- An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested. which resources a process will request and use during its lifetime i.e. maximum number of resources of each type that it may need.
- With this additional knowledge, the operating system can decide for each request whether process should wait or not.
- So, in order to avoid deadlock in run time, System try to maintain some books for recorder and like a banker, whenever someone ask for a loan(resource), it is granted only when the books allow. Let's understand by an example
- Here we use the idea of how a bank perform its operation, bank don't have its own money for business. But some people put their money in account while other borrow money from the bank, and with the difference in the interest bank operate. Whenever someone ask for a loan first bank will check whether it has sufficient funds to allow the operation.
- A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist. The resource- allocation state is defined by the number of available and allocated resources and the maximum demands of the processes before allowing that request first, we check, if there exist "some sequence in which we can satisfies demand of every process without going into deadlock, if yes, this sequence is called safe sequence" and request can be allowed. Otherwise there is a possibility of going into deadlock.
- More formally, a system is in a safe state only if there exists a safe sequence. A sequence of processes $< P_1, P_2, ..., P_n>$ is a safe sequence for the current allocation state if, for each P_i, the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j, with $j < i$. In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished. When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe.
- Given the concept of a safe state, we can define avoidance algorithms that ensure that the system will never deadlock. The idea is simply to ensure that the system will always remain in a safe state. Initially, the system is in a safe state. Whenever a process requests a resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in a safe state.


unsafe
deadlock
safe

Banker's Algorithm

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

- **Available:** A vector of length m indicates the number of available resources of each type. If Available[j] equals k, then k instances of resource type R_j are available.
- **Max:** An n*m matrix defines the maximum demand of each process. If Max[i][j] equals k, then process P_i may request at most k instances of resource type R_j.
- **Allocation:** An n*m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process P_i is currently allocated k instances of resource type R_j.
- **Need/Demand/Requirement:** An n*m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process P_i may need k more instances of resource type R_j to complete its task. Note that Need[i][j] = Max[i][j] − Allocation[i][j].
- These data structures vary over time in both size and value.
- We can treat each row in the matrices Allocation and Need as vectors and refer to them as $Allocation_i$ and $Need_i$. The vector $Allocation_i$ specifies the resources currently allocated to process P_i; the vector $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Safety Algorithm

- We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

1- Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available and Finish[i] = false for i = 0, 1, ..., n − 1.
2- Find an index i such that both
Finish[i] == false
Needi ≤ Work
If no such i exists, go to step 4.
3- Work = Work + Allocationi
Finish[i] = true
Go to step 2.
4- If Finish[i] == true for all i, then the system is in a safe state.

- This algorithm may require an order of $m*n^2$ operations to determine whether a state is safe.

Resource-Request Algorithm

- Next, we describe the algorithm for determining whether requests can be safely granted. Let $Request_i$ be the request vector for process P_i. If $Request_i$ [j] k, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1- If Requesti <= Needi, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2- If Requesti <= Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
3- Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

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

- If the resulting resource-allocation state is safe, the transaction is completed, and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for Requesti, and the old resource-allocation state is restored.

Q Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a maximum of K instances. Resource instances can be requested and released only one at a time. The larges value of K that will always avoid deadlock is _____. **(GATE-2018) (1 Marks)**
Answer: 2

Q A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is **(GATE-2014) (2 Marks)**
(A) 6 **(B)** 7 **(C)** 8 **(D)** 9
Answer: (B)

Q Consider a system having 'm' resources of the same type. These resources are shared by three processes P1, P2and P3 which have peak demands of 2, 5 and 7 resources respectively. For what value of 'm' deadlock will not occur? **(NET-AUG-2016)**
a) 70 **b)** 14 **c)** 13 **d)** 7
Answer: B

Q Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur? **(NET-DEC-2012)**
(A) 7 **(B)** 9 **(C)** 10 **(D)** 13
Answer: d

Q An operating system contains 3 user processes each requiring 2 units of resource *R*. The minimum number of units of *R* such that no deadlocks will ever arise is **(GATE-1997) (2 Marks)**

(A) 3 **(B)** 5 **(C)** 4 **(D)** 6

Answer: (C)

Q Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur? **(GATE-1993) (2 Marks)**

(a) 7 **(b) 9** **(c) 10** **(d) 13** **(e) 15**

Answer: (D) & (E)

Q A system has 6 identical resources and N processes competing for them. Each process can request at most 2 resources. Which one of the following values of N could lead to a deadlock? **(GATE-2015) (1 Marks)**

a) 1 **b)** 2 **c)** 3 **d)** 6

Answer: D

Q A computer system has 6 tape drives, with n process completing for them. Each process may need 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock free is **(GATE-1992) (2 Marks)**

(a) 2 **(b)** 3 **(c)** 4 **(d)** 1

Answer: (A)

Q A computer has 6 tape drives with 'n' processes competing for them. Each process may need two drives. For which values of 'n' is the system deadlock free? **(NET-JUNE-2008)**

(A) 1 **(B)** 2 **(C)** 3 **(D)** 6

Answer: C

Q A computer has six tape drives, with n processes competing for them. Each process may need two drives. What is the maximum value of n for the system to be deadlock free? **(GATE-1998) (2 Marks)**

(A) 6 **(B)** 5 **(C)** 4 **(D)** 3

Answer: (B)

Q Suppose n processes, P1, …. Pn share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process Pi is Si, where Si>0. Which one of the following is a sufficient condition for ensuring that deadlock does not occur? **(GATE-2005) (2 Marks)**

a) $\forall_i$, $S_i < m$ **b)** $\forall_i$, $S_i < n$

c) $\sum_{i=1}^{n} si < (m + n)$ **d)** $\sum_{i=1}^{n} si < (m * n)$

Answer: (C)

Q A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for 3 processes are shown below:

Process	Current Allocation	Maximum Requirement
P_1	3	7
P_2	1	6
P_3	3	5

Which of the following best describes the current state of the system? **(GATE-2014) (2 Marks)**

(A) Safe, Deadlocked
(B) Safe, Not Deadlocked
(C) Not Safe, Deadlocked
(D) Not Safe, Not Deadlocked

Answer: (B)

Q Suppose there are four processes in execution with 12 instances of a Resource R in a system. The maximum need of each process and current allocation are given below:

Process	Max Need	Allocation
P_1	8	3
P_2	9	4
P_3	5	2
P_4	3	1

With reference to current allocation, is system safe? If so, what is the safe sequence? **(NET-JULY-2016)**

a) No **b)** Yes, P1P2P3P4 **c)** Yes, P4P3P1P2 **d)** Yes, P2P1P3P4

Answer: (c)

Q Consider a system with twelve magnetic tape drives and three processes P_1, P_2 and P_3. Process P_1 requires maximum ten tape drives, process P_2 may need as many as four tape drives and P_3 may need up to nine tape drives. Suppose that at time t_1, process P_1 is holding five tape drives, process P_2 is holding two tape drives and process P_3 is holding three tape drives. At time t_1, system is in: **(NET-DEC-2015)**

A) safe state
b) unsafe state
c) deadlocked state
d) starvation state

Ans b

Q An operating system has 13 tape drives. There are three processes P_1, P_2 & P_3. Maximum requirement of P_1 is 11 tape drives, P_2 is 5 tape drives and P_3 is 8 tape drives. Currently, P_1 is allocated 6 tape drives, P_2 is allocated 3 tape drives and P_3 is allocated 2 tape drives. Which of the following sequences represent a safe state? **(NET-DEC-2014)**

(A) P_2 P_1 P_3 **(B)** P_2 P_3 P_1 **(C)** P_1 P_2 P_3 **(D)** P_1 P_3 P_2

ans a

Q In a system, there are three types of resources: E, F and G. Four processes P_0, P_1, P_2 and P_3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example, Max [P_2, F] is the maximum number of instances of F that P_2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.
Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of E and 3 instances of F are the only resources available.

	Allocation			**Max**		
	E	F	G	E	F	G
P_0	1	0	1	4	3	1
P_1	1	1	2	2	1	4
P_2	1	0	3	1	3	3
P_3	2	0	0	5	4	1

From the perspective of deadlock avoidance, which one of the following is true? (GATE-2018) (2 Marks)
(A) The system is in *safe* state
(B) The system is not in *safe* state, but would be *safe* if one more instance of E were available
(C) The system is not in *safe* state, but would be *safe* if one more instance of F were available
(D) The system is not in *safe* state, but would be *safe* if one more instance of G were available
Answer: (A)

Q A single processor system has three resource types X, Y and Z, which are shared by three processes. There are 5 units of each resource type. Consider the following scenario, where the column alloc denotes the number of units of each resource type allocated to each process, and the column request denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST? **(GATE-2007) (2 Marks)**

	Allocation			**request**		
	X	Y	Z	X	Y	Z
P_0	1	2	1	1	0	3
P_1	2	0	1	0	1	2
P_2	2	2	1	1	2	0

(A) P_0 **(B)** P_1
(C) P_2 **(D)** None of the above, since the system is in a deadlock
Answer: (C)

Q An operating system using banker's algorithm for deadlock avoidance has ten dedicated devices (of same type) and has three processes P_1, P_2 and P_3 with maximum resource requirements of 4, 5 and 8 respectively. There are two states of allocation of devices as follows: **(NET-JUNE-2013)**

State 1	Processes	P_1	P_2	P_3
	Device allocated	2	3	4
State 2	Processes	P_1	P_2	P_3
	Device allocated	0	2	4

Which of the following is correct ?
(A) State 1 is unsafe and state 2 is safe. **(B)** State 1 is safe and state 2 is unsafe.
(C) Both, state 1 and state 2 are safe. **(D)** Both, state 1 and state 2 are unsafe.
Ans: a

Q An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P_0, P_1, and P_2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			**Max**		
	X	Y	Z	X	Y	Z
P_0	0	0	1	8	4	3
P_1	3	2	0	6	2	0
P_2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the
current state:
REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z
REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z
Which one of the following is TRUE? **(GATE-2014) (2 Marks)**
(A) Only REQ1 can be permitted. **(B)** Only REQ2 can be permitted.
(C) Both REQ1 and REQ2 can be permitted. **(D)** Neither REQ1 nor REQ2 can be permitted
Answer: (B)

Q A system has four processes and five resources. The current allocation and maximum needs are as follows: **(NET-DEC-2015)**

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	2	2	0	0	2			

The smallest value of x for which the above system in safe state is _____.

a) 1 **b)** 3 **c)** 2 **d)** Not safe for any value of x

Ans: c

Q Banker's algorithm is used for ..... purpose. **(NET-DEC-2005)**

(A) Deadlock avoidance **(B)** Deadlock removal

(C) Deadlock prevention **(D)** Deadlock continuations

Answer: A

Q Bankers algorithm is for. **(NET-JUNE-2005)**

(A) Dead lock Prevention **(B)** Dead lock Avoidance

(C) Dead lock Detection **(D)** Dead lock creation

Answer: B

Q Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes?**(GATE-2008) (2 Marks)**

(A) In deadlock prevention, the request for resources is always granted if the resulting state is safe

(B) In deadlock avoidance, the request for resources is always granted if the result state is safe

(C) Deadlock avoidance is less restrictive than deadlock prevention

(D) Deadlock avoidance requires knowledge of resource requirements a priori

Answer: (A)

Q Consider the following snapshot of a system running n processes. Process P_i is holding X_i instances of a resource R, 1 <= i <= n. currently, all instances of R are occupied. Further, for all i, process i has placed a request for an additional Y_i instances while holding the X_i instances it already has. There are exactly two processes p and q such that $Y_p = Y_q = 0$. Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock? **(GATE-2006) (2 Mark)**

(A) min (X_p, X_q) < max (Y_k) where k != p and k != q

(B) $X_p + X_q$ >= min (Y_k) where k != p and k != q

(C) max (X_p, X_q) > 1
(D) min (X_p, X_q) > 1
Answer: (B)

Q Consider the following snapshot of a system running n concurrent processes. Process ii is holding Xi instances of a resource RR, 1≤i≤n. Assume that all instances of R are currently in use. Further, for all i, process i can place a request for at most Yi additional instances of R while holding the Xi instances it already has. Of the n processes, there are exactly two processes p and q such that Yp=Yq=0. Which one of the following conditions guarantees that no other process apart from p and q can complete execution? **(GATE-2019) (2 Marks)**
a) $X_p + X_q < Min\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
b) $X_p + X_q < Max\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
c) $Min(X_p, X_q) \ge Min\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
d) $Min(X_p, X_q) \le Max\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$
Ans: a

Q Consider a system which have 'n' number of processes and 'm' number of resource types. The time complexity of the safety algorithm, which checks whether a system is in safe state or not, is of the order of: **(NET-AUG-2016)**
a) O(mn) **b)** $O(m^2n^2)$ **c)** $O(m^2n)$ **d)** $O(mn^2)$
Ans: d

Q Consider a system with five processes P0 through P4 and three resource types A, B and C. Resource type A has seven instances, resource type B has two instances and resource type C has six instances suppose at time T0we have the following allocation.

	Allocation			**Request**			**Available**		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	2	2	0	0	2			

If we implement Deadlock detection algorithm, we claim that system is _____. **(NET-NOV-2017)**
(1) Semaphore **(2)** Deadlock state **(3)** Circular wait **(4)** Not in deadlock state
a) (1), (2) and (3) **b)** (1) and (3)
c) (3) and (4) **d)** All are correct
Answer: D

Deadlock detection and recovery

- When should we invoke the detection algorithm? It depends on two factors:
  - How often is a deadlock likely to occur?
  - How many processes will be affected by deadlock when it happens?
- **Active approach:** If deadlocks occur frequently, then the detection algorithm should be invoked frequently. Of course, invoking the deadlock-detection algorithm for every resource request will incur considerable overhead in computation time.
- **Lazy approach:** A less expensive alternative is simply to invoke the algorithm at defined intervals—for example, once per hour or whenever CPU utilization drops below 40 percent.

Recovery from Deadlock

- When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock.
- **Process Termination:** One is simply to abort one or more processes to break the circular wait.
- **Pre-empt Resource:** The other is to pre-empt some resources from one or more of the deadlocked processes.

Process Termination

- **Abort all deadlocked processes**: This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.
- **Abort one process at a time**: until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.
- Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

How to choose a victim

If the partial termination method is used, then we must determine which deadlocked process (or processes) should be terminated. This determination is a policy decision, similar to CPU-scheduling decisions. The question is basically an economic one; we should abort those processes whose termination will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may affect which process is chosen, including:

- What the priority of the process is
- How long the process has computed and how much longer the process will compute before completing its designated task
- How many and what types of resources the process has used (for example, whether the resources are simple to pre-empt)
- How many more resources the process needs in order to complete
- How many processes will need to be terminated?
- Whether the process is interactive or batch

Resource Pre-emption

- To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then three issues need to be addressed:
- **Selecting a victim** Which resources and which processes are to be preempted? As in process termination, we must determine the order of preemption to minimize cost. Cost factors may include such parameters as the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed.
- **Rollback**. If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must roll back the process to some safe state and restart it from that state. Since, in general, it is difficult to determine what a safe state is, the simplest solution is a total rollback: abort the process and then restart it. Although it is more effective to roll back the process only as far as necessary to break the deadlock, this method requires the system to keep more information about the state of all running processes.
- **Starvation**. How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process? In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation any practical system must address. Clearly, we must ensure that a process can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

Q Consider a system with 4 types of resources R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-pre-emptive resource allocation policy is used. At any given instance, a request is not entertained if it cannot be completely satisfied. Three processes P1, P2, P3 request the sources as follows if executed independently. **(GATE-2009) (2 Marks)**

Process P1:	Process P2:	Process P3:
t=0: requests 2 units of R2	t=0: requests 2 units of R3	t=0: requests 1 unit of R4
t=1: requests 1 unit of R3	t=2: requests 1 unit of R4	t=2: requests 2 units of R1
t=3: requests 2 units of R1	t=4: requests 1 unit of R1	t=5: releases 2 units of R1
t=5: releases 1 unit of R2 and 1 unit of R1.	t=6: releases 1 unit of R3	t=7: requests 1 unit of R2
t=7: releases 1 unit of R3	t=8: Finishes	t=8: requests 1 unit of R3
t=8: requests 2 units of R4		t=9: Finishes
t=10: Finishes		

Which one of the following statements is TRUE if all three processes run concurrently starting at time t=0?

(A) All processes will finish without any deadlock

(B) Only P1 and P2 will be in deadlock.

(C) Only P1 and P3 will be in a deadlock.

(D) All three processes will be in deadlock

Answer: (A)

Q Consider a system with seven processes A through G and six resources R through W. Resource ownership is as follows : process A holds R and wants T process B holds nothing but wants T process C holds nothing but wants S process D holds U and wants S & T process E holds T and wants V process F holds W and wants S process G holds V and wants U Is the system deadlocked ? If yes, _____ processes are deadlocked. **(NET-AUG-2016)**

a) No **b)** Yes, A, B, C **c)** Yes, D, E, G **d)** Yes, A, B, F

Ans: c

Q Consider a system with five processes P0 through P4 and three resource types R1 R and R3. Resource type R1 has 10 instances, R2 has 5 instances and R3 has 7 instances. Suppose that at time T0, the following snapshot of the system has been taken: **(NET-JUNE-2014)**

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	3	9	0	2			

P3	2	1	1	2	2	2			
P4	0	2	2	4	3	3			

Assume that now the process P1 requests one additional instance of type R1 and two instances of resource type R3. The state resulting after this allocation will be

(A) Ready state **(B)** Safe state **(C)** Blocked state **(D)** Unsafe state

Ans: b

Q Simplest way of deadlock recovery is **(NET-DEC-2013)**

a) Roll back **b)** Pre-empt resource

c) Lock one of the processes **d)** Kill one of the processes

Ans: d

Q A system has n resources $R_0,\ldots,R_{n-1}$,and k processes $P_0,\ldots.P_{k-1}$.The implementation of the resource request logic of each process P_i is as follows (GATE-2010) (2 Mark)

```
if (i % 2 == 0) {
    if (i < n) request Ri
    if (i+2 < n) request Ri+2
}
else {
    if (i < n) request Rn-i
    if (i+2 < n) request Rn-i-2

}
```

In which one of the following situations is a deadlock possible?

(A) n=40, k=26 **(B)** n=21, k=12 **(C)** n=20, k=10 **(D)** n=41, k=19

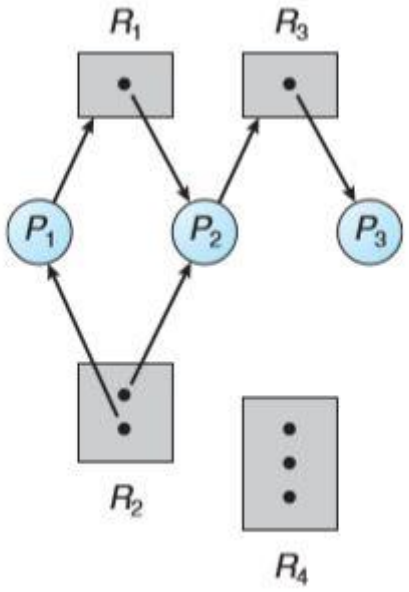
Answer: (B)

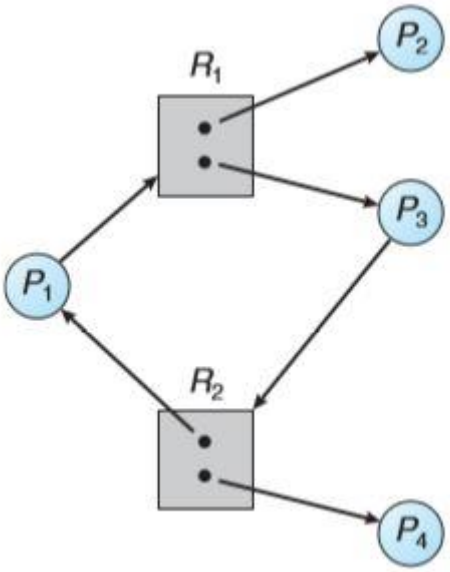
Ignorance

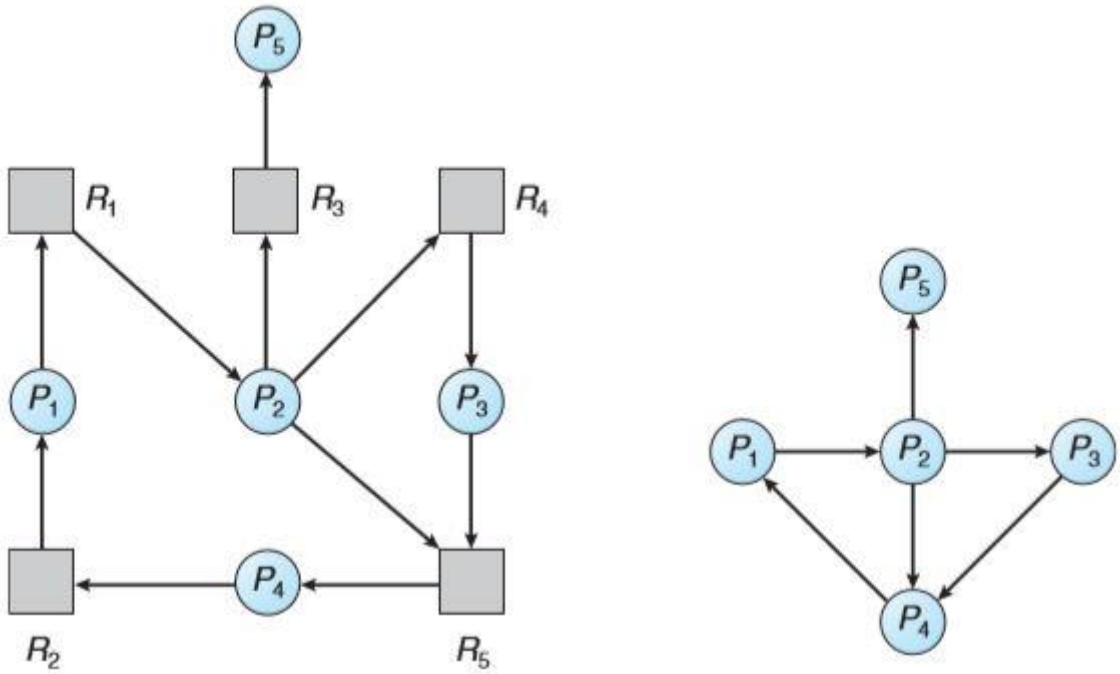
- In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened. In this case, the undetected deadlock will cause the system's performance to deteriorate, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state. Eventually, the system will stop functioning and will need to be restarted manually.
- Although this method may not seem to be a viable approach to the deadlock problem, it is nevertheless used in most operating systems. Expense is one important consideration. Ignoring the possibility of deadlocks is cheaper than the other approaches. Since in many systems, deadlocks occur infrequently (say, once per year), the extra expense of the other methods may not seem worthwhile. In addition, methods used to recover from other conditions may be put to use to recover from deadlock. In some circumstances, a system is in a frozen state but not in a deadlocked state. We see this situation, for example, with a real-time process running at the highest priority (or any process running on a nonredemptive scheduler) and never returning control to the operating system. The system must have manual recovery methods for such conditions and may simply use those techniques for deadlock recovery.

Resource Allocation Graph

- Deadlock 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 two different types of nodes: $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the active processes in the system, and $R = \{R_1, R_2, ..., R_m\}$, 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 $P_i \rightarrow R_j$; 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 $R_j \rightarrow P_i$; it signifies that an instance of resource type R_j has been allocated to process P_i. A directed edge $P_i \rightarrow R_j$ is called a request edge; a directed edge $R_j \rightarrow P_i$ is called an assignment edge.
- Pictorially, we represent each process P_i as a circle and 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. Note that a request edge points to only the rectangle R_j, whereas an assignment edge must also designate one of the dots in the rectangle.
- 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.







Cycle is resource allocation graph

Virtual Memory

One important goal in now-a-days computing environment is to keep many processes in memory simultaneously to follow multiprogramming, and use resources efficiently especially main memory.

Pure Demand Paging/Demand Paging

- we can start executing a process with no pages in memory. When the operating system sets the instruction pointer to the first instruction of the process, which is on a non-memory-resident page, the process immediately faults for the page. After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that it can execute with no more faults. This scheme is **Pure Demand Paging:** never bring a page into memory until it is required.
- The **Locality of Reference** helps Demand Paging in performing reasonably.
- A demand-paging system is extension to a paging system with swapping.
- But rather than swapping the entire process into memory, we use a **Lazy Swapper**. A lazy swapper never swaps a page into memory unless that page will be needed.
- We use the term **Pager** instead of **swapper** in demand paging.

Q Moving Process from main memory to disk is called: **(NET-JUNE-2005)**
(A) Caching **(B)** Termination **(C)** Swapping **(D)** Interruption
Answer: (C)

Q If the executing program size is greater than the existing RAM of a computer, it is still possible to execute the program if the OS supports: **(NET-DEC-2008)**
(A) multitasking **(B)** virtual memory **(C)** paging system **(D)** none of the above
Answer: (B)

Q Page making process from main memory to disk is called **(NET-DEC-2010)**
(A) Interruption **(B)** Termination **(C)** Swapping **(D)** None of the above
Answer: (C)

Q Which of the following is incorrect for virtual memory? **(NET-JAN-2017)**
a) Large programs can be written **b)** More I/O is required
c) More addressable memory available **d)** Faster and easy swapping of process
Answer: (B)

Advantage

- A program would no longer be constrained by the amount of physical memory that is available, Allows the execution of processes that are not completely in main memory, i.e. process can be larger than main memory.
- More programs could be run at the same time as use of main memory is less.
- Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.
- Virtual memory also allows processes to share files easily and to implement shared memory.

Disadvantages

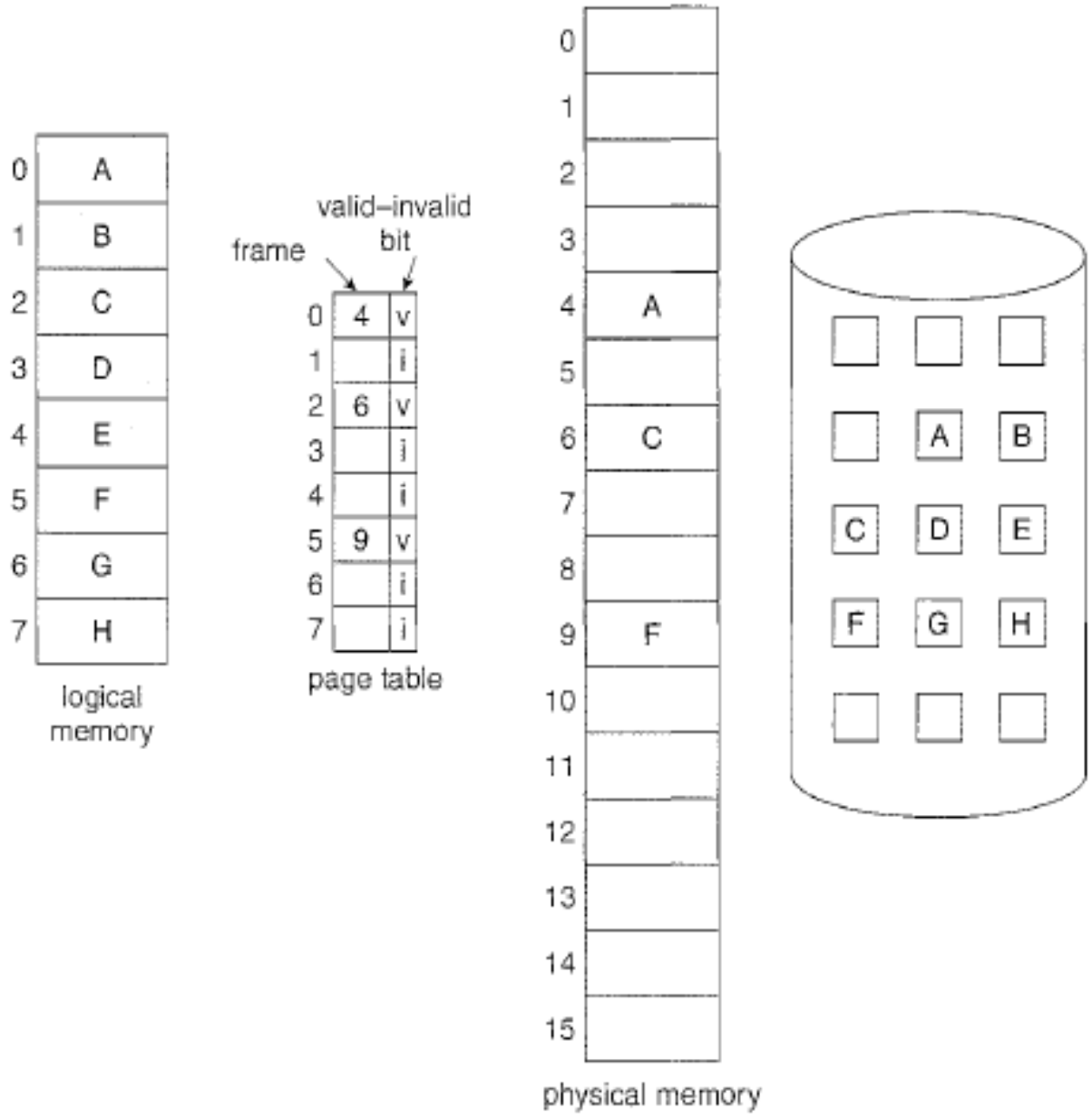
- Virtual memory is not easy to implement.
- It may substantially decrease performance if it is used carelessly (Thrashing)

Q Which of the following statements is false? **(GATE-2001) (1 Marks)**
(A) Virtual memory implements the translation of a program's address space into physical memory address space
(B) Virtual memory allows each program to exceed the size of the primary memory
(C) Virtual memory increases the degree of multiprogramming
(D) Virtual memory reduces the context switching overhead
Answer: (A)

Q Which of the following memory allocation scheme suffers from external fragmentation? **(NET-DEC-2012)**
(A) Segmentation **(B)** Pure demand paging **(C)** Swapping **(D)** Paging
Answer: (A)

Basic Concepts

- When a process is started execution, no page is loaded into main memory before demand so, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.
- Now we need some hardware to distinguish between which pages are in memory and which are not, so **valid -invalid bit scheme** can be used for it.
- This time, however, when this bit is set to "**valid**" the associated page is both legal and in memory. If the bit is set to "**invalid**" the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.



- The page-table entry for a page that is brought into memory is set as usual but the page-table entry for a page that is not currently in memory is either simply marked invalid.

Page Fault: - When a process tries to access a page that is not in main memory then a **Page Fault Occurs.**

Steps to handle Page Fault

- If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
- We find a free frame (by taking one from the free-frame list, for example).
- We schedule a disk operation to read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

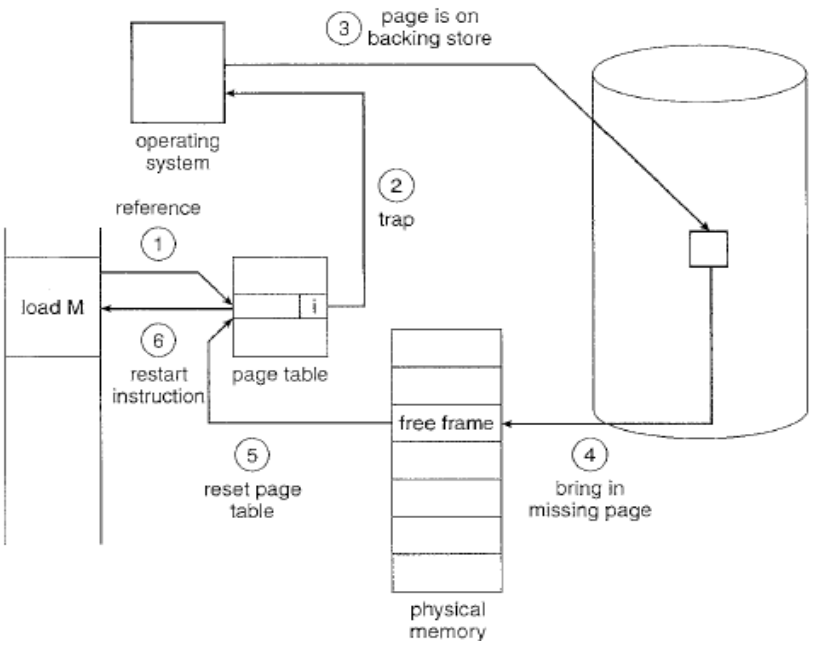


Figure: Page Fault Handling

- A crucial requirement for demand paging is the ability to restart any instruction after a page fault. Because we save the state (registers, condition code, instruction counter) of the interrupted process when a page fault occurs, we must be able to restart the process in exactly the same place and state.

Q A page fault ..... **(NET-JUNE-2006)**
(A) is an error in specific page
(B) is an access to the page not currently in main memory
(C) occurs when a page program accesses a page of memory
(D) is reference to the page which belongs to another program
Answer: (B)

Q A page fault **(NET-DEC-2009)**
(A) is an error specific page.
(B) is an access to the page not currently in memory.
(C) occur when a page program occurs in a page memory.
(D) page used in the previous page reference.
Answer: B

Performance of Demand Paging

- **Effective Access time for Demand Paging:** (1 - p) x ma + p x page fault service time.
- Here, p: Page fault rate or probability of a page fault.
- ma is memory access time.

Q Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one-page fault is generated for every 10^6 memory accesses, what is the effective access time for the memory? **(GATE-2011) (1 Marks)**

(A) 21ns **(B)** 30ns **(C)** 23ns **(D)** 35ns

Answer: (B)

Q In a demand paging memory system, page table is held in registers. The time taken to service a page fault is 8 msec. if an empty frame is available or if the replaced page is not modified, and it takes 20 msec., if the replaced page is modified. What is the average access time to service a page fault assuming that the page to be replaced is modified 70 of the time? **(NET-DEC-2014)**

(A) 11.6 msec. **(B)** 16.4 msec. **(C)** 28 msec. **(D)** 14 msec.

Answer: (B)

Q Suppose the time to service a page fault is on the average 10 milliseconds, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in average memory access time of? (GATE-2000) (1 Marks).

(A)1.9999 milliseconds **(B)**1 millisecond

(C)9.999 microseconds **(D)**1.9999 microseconds

Answer: (D)

Q In a paged memory, the page hit ratio is 0.40. The time required to access a page in secondary memory is equal to 120 ns. The time required to access a page in primary memory is 15 ns. The average time required to access a page is _____. **(NET-JULY-2018)**

(a) 105 **(b)** 68 **(c)** 75 **(d)** 78

Answer: (D)

Q The memory allocation scheme subjected to "external" fragmentation is: **(NET-JUNE-2006)**

(A) Segmentation **(B)** Swapping

(C) Demand paging **(D)** Multiple contiguous fixed partitions

Answer: (A)

Q Virtual memory is **(NET-JUNE-2011)**

(A) related to virtual reality
(B) a form of ROM
(C) a form of RAM
(D) None of the above
Answer: (C)

Q Consider a process executing on an operating system that uses demand paging. The average time for a memory access in the system is M units if the corresponding memory page is available in memory, and D units if the memory access causes a page fault. It has been experimental measured that the average time taken for a memory access in the process is X units.
Which one of the following is the correct expression for the page fault rate experienced by the process? **(GATE-2018) (2 Marks)**
(A) (D – M) / (X – M)
(B) (X – M) / (D – M)
(C) (D – X) / (D – M)
(D) (X – M) / (D – X)
Answer: (B)

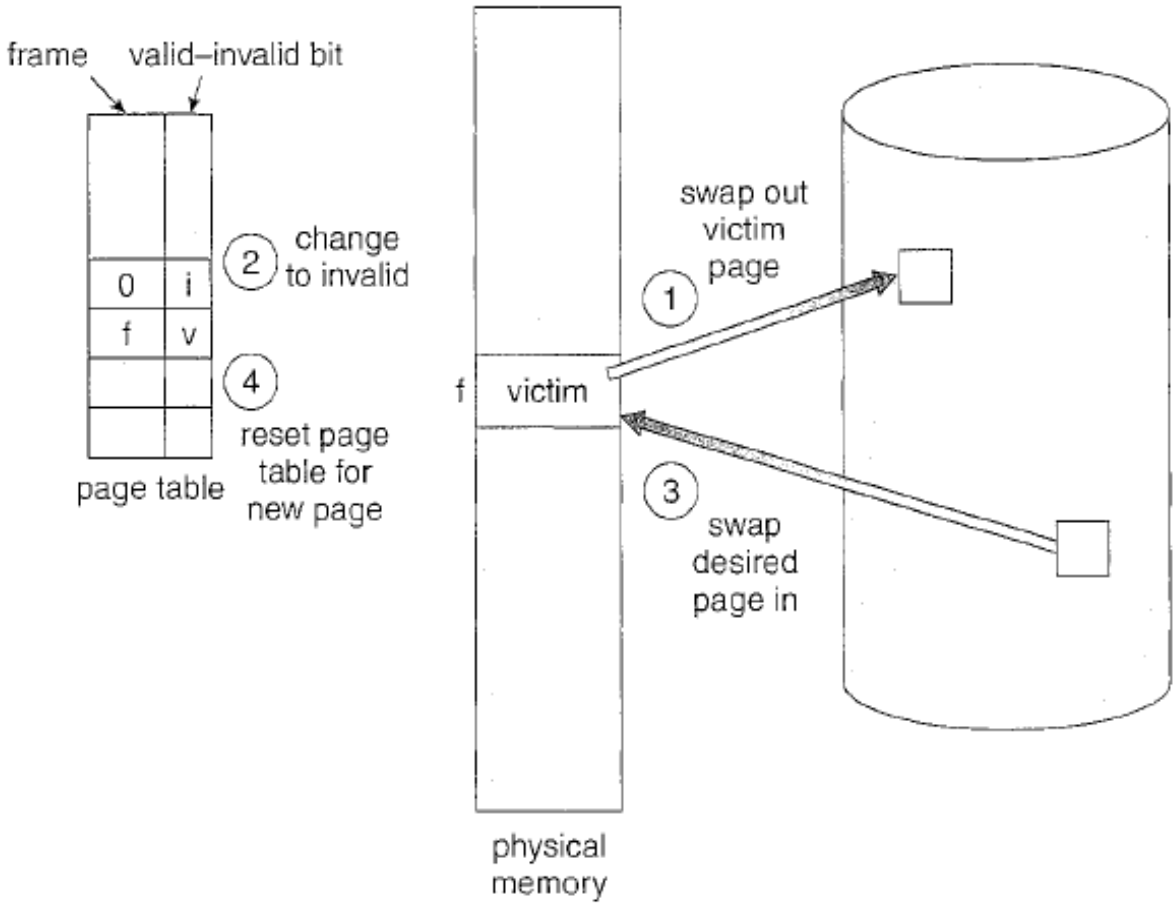
Q Suppose that the number of instructions executed between page fault is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Further, consider that a normal instruction takes one microsecond, but if a page fault occurs, it takes 2001 microseconds. If a program takes 60 sec to run, during which time it gets 15,000-page faults, how long would it take to run if twice as much memory were available**? (NET-DEC-2015)**
A) 60 sec **b)** 30 sec **c)** 45 sec **d)** 10 sec
Answer: (C)

Page Replacement

With the increase in multiprogramming each process will get less amount of space in main memory so, the rate of page faults may rise.

thus, to reduce the degree of multiprogramming the operating system swaps out processes from the memory freeing the frames and thus the process that requires to execute can now execute.

- If no frames are free, two-page transfers (one out and one in) are required. This situation effectively doubles the page-fault service time and increases the effective access time accordingly.
- We can reduce this overhead by using a **Modify bit or Dirty Bit**. When this scheme is used, each page or frame has a modify bit associated with it in the hardware.
- The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.
- **If the bit is set**: the page has been modified since it was read in from the disk. In this case, we must write the page to the disk.
- **If the modify bit is not set**: however, the page has not been modified since it was read into memory. In this case, we need not write the memory page to the disk: it is already there.



Now, we must solve two major problems to implement demand paging: We must be able to develop a **frame allocation algorithm** and a **page replacement algorithm**.

- Frame Allocation will decide how much frames to allocate to a process.
- Page Replacement will decide which page to replace next.

Q Dirty bit is used to show the **(NET-DEC-2018)**
a) Wrong Page
b) Page with corrupted data
c) Page with low frequency occurrence
d) Page that is modified after being loaded into cache memory
Answer: (D)

Q A virtual memory has a page size of 1K words. There are eight pages and four blocks. The associative memory page table contains the following entries:

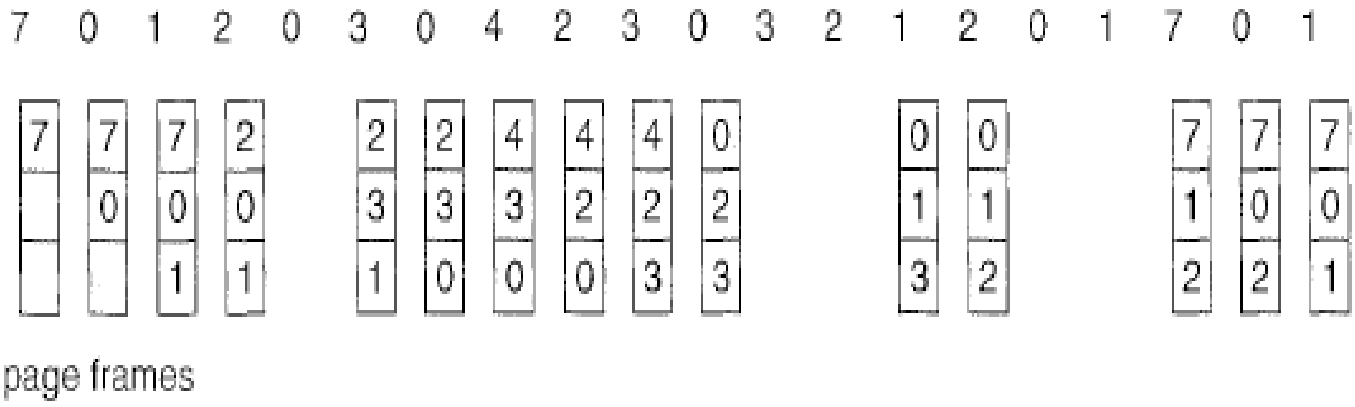
Page	Block
0	3
2	1
5	2
7	0

Which of the following list of virtual addresses (in decimal) will not cause any page fault if referenced by the CPU? **(NET-DEC-2015)**
a) 1024, 3072, 4096, 6144
b) 1234, 4012, 5000, 6200
c) 1020, 3012, 6120, 8100
d) 2021, 4050, 5112, 7100
Answer: (C)

Page Replacement Algorithms

First In First Out Page Replacement Algorithm: - A FIFO replacement algorithm associates with each page, the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. i.e. the first page that came into the memory will be replaced first.

Example:



- In the above example the number of page fault is 15.
- The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good.
- **Belady's Anomaly:** for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

Q Consider the reference string 0 1 2 3 0 1 4 0 1 2 3 4 If FIFO page replacement algorithm is used, then the number of page faults with three-page frames and four-page frames are _____ and _____ respectively. **(NET-JUNE-2016)**

a) 10, 9 **b)** 9, 9 **c)** 10, 10 **d)** 9,10

Answer: (D)

Q In which one of the following page replacement policies, Belady's anomaly may occur? **(GATE-2009) (1 Marks)**

(A) FIFO **(B)** Optimal **(C)** LRU **(D)** MRU

Answer: (A)

Q Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will (GATE-2001) (1 Marks)

(A) always decrease the number of page faults

(B) always increase the number of page faults

(C) sometimes increase the number of page faults
(D) never affect the number of page faults
Answer: (C)

Q A virtual memory system uses First in First Out (FIFO) page replacement policy and allocates a fixed number of frames to a process. Consider the following statements: **(GATE-2007) (2 Marks)**
P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.
Q: Some programs do not exhibit locality of reference.
Which one of the following is TRUE?
(A) Both P and Q are true, and Q is the reason for P
(B) Both P and Q are true, but Q is not the reason for P.
(C) P is false, but Q is true
(D) Both P and Q are false
Answer: (B)

Q. In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases? (GATE-2016) (1 Marks)
(1) LRU (Least Recently Used
(2) OPT (Optimal Page Replacement)
(3) MRU (Most Recently Used)
(4) FIFO (First In First Out)
Answer: (**D)**

Q A system uses FIFO policy for page replacement. It has 4-page frames with no pages loaded to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur? **(GATE-2010) (1 Marks)**
(A) 196 **(B)** 192 **(C)** 197 **(D)** 195
Answer: (A)

Q Which page replacement policy suffers from Belady's anomaly? **(NET-JUNE-2007)**
(A) LRU **(B)** LFU **(C)** FIFO **(D)** OPTIMAL
Answer: (C)

Q Suppose for a process P, references to pages occur in order are 1, 2, 4, 5, 2, 1, 2, 4. Assume that the main memory can accommodate 3 pages and the main memory already has the pages 1 and 2 in the order 1-first, 2-second. At this moment, assume fifo page replacement algorithm is used then the number of page faults that occur to complete the

execution of process P is (NET-DEC-2018)

(A) 4 **(B)** 3 **(C)** 5 **(D)** 6

Answer: (B)

Q A program has five virtual pages, numbered from 0 to 4. If the pages are referenced in the order 012301401234, with three-page frames, the total number of page faults with FIFO will be equal to: **(NET-DEC-2007)**

(A) 0 **(B)** 4 **(C)** 6 **(D)** 9

Answer: (D)

Q Consider the following page trace: 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5. Percentage of page fault that would occur if FIFO page replacement algorithm is used with number of frames for the JOB m = 4 will be **(NET-JUNE-2012)**

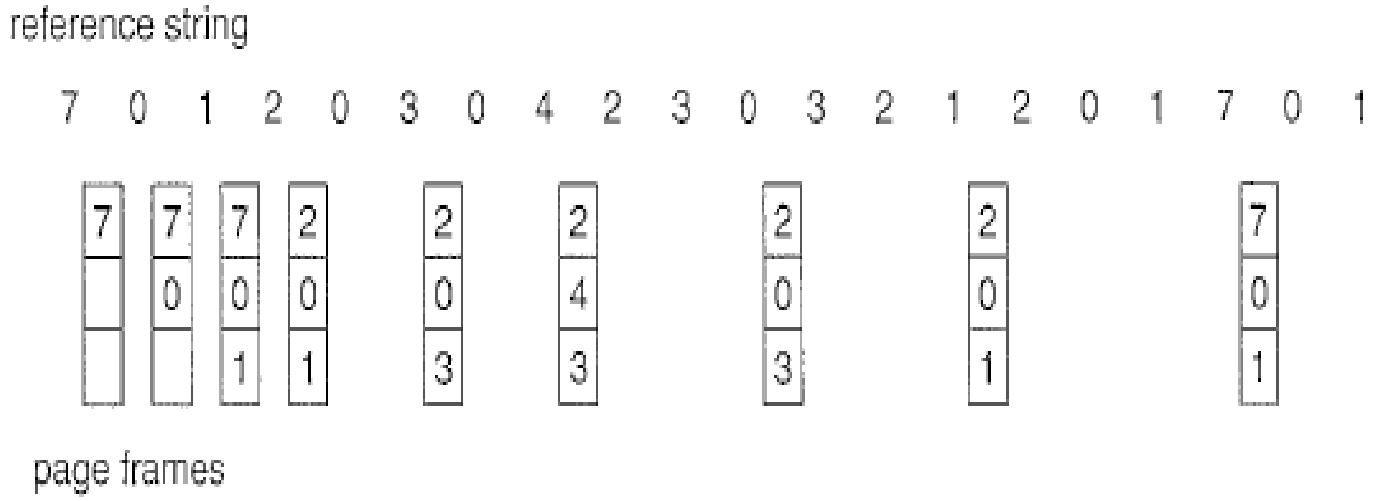
(A) 8 **(B)** 9 **(C)** 10 **(D)** 12

Answer: (C)

Optimal Page Replacement Algorithm

- Replace the page that will not be used for the longest period of time.
- It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

Example:



- The number of page faults occurred: 9
- Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.
- It is mainly used for comparison studies.

Q The optimal page replacement algorithm will select the page that(GATE-2002) (1 Marks)
(A) Has not been used for the longest time in the past.
(B) Will not be used for the longest time in the future.
(C) Has been used least number of times.
(D) Has been used most number of times.
Answer: (B)

Q Assume that there are 3-page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____. **(GATE-2014) (2 Marks)**
Answer: 7

Q A process has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, 1. If optimal page replacement policy is used, how many page faults occur for the above reference string? **(GATE-2007) (2 Marks)**

(A) 7 **(B)** 8 **(C)** 9 **(D)** 10

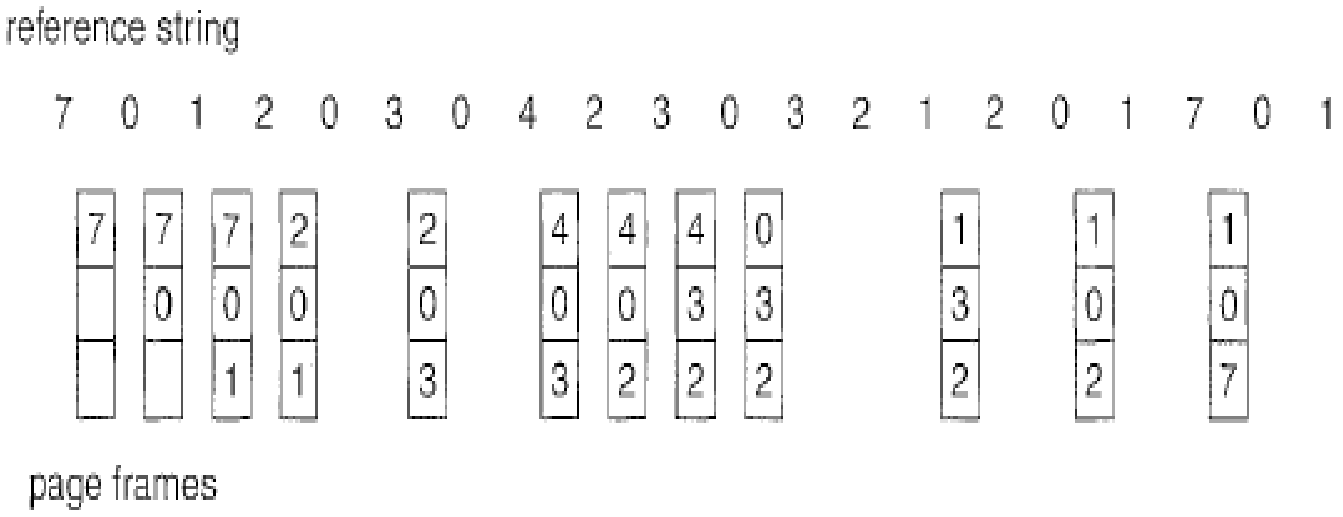
Answer: (A)

Q Assume that there are 3-page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____. **(GATE-2014) (2 Marks)**

Answer: 7

Least Recently Used (LRU) Page Replacement Algorithm

- Replace the page that has not been used for the longest period of time.
- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.



- LRU gives us 12-page faults.
- LRU is much better than FIFO replacement.
- The LRU policy is often used as a page-replacement algorithm and is considered to be good.
- LRU also does not suffer from Belady's Anomaly.
- The major problem is how to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance.
- LRU and OPT belong to a class of page-replacement algorithms, called **stack algorithms** and can never exhibit Belady's anomaly.
- A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for *n* frames is always a subset of the set of pages that would be in memory with *n + 1* frames.

Q Consider a virtual page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Suppose a demand paged virtual memory system running on a computer system such that the main memory has 3 page frames. Then _____ page replacement algorithm has minimum number of page faults. **(NET-NOV-2017)**

A) FIFO **b)** LIFO **c)** LRU **d)** Optimal

Answer: (D)

Q Recall that Belady's anomaly is that the page-fault rate may increase as the number of allocated frames increases. Now, consider the following statements:

S1: Random page replacement algorithm (where a page chosen at random is replaced) suffers from Belady's anomaly
S2: LRU page replacement algorithm suffers from Belady's anomaly
Which of the following is CORRECT? **(GATE-2017) (1 Marks)**
(1)S1 is true, S2 is true **(2)**S1 is true, S2 is false
(3)S1 is false, S2 is true **(4)**S1 is false, S2 is false
Answer: (B)

Q A system uses 3-page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below?
4, 7, 6, 1, 7, 6, 1, 2, 7, 2 (**GATE-2014) (2 Marks)**
Answer: 6

Q A process, has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1,2,1,3,7,4,5,6,3,1
Least Recently Used (LRU) page replacement policy is a practical approximation to optimal page replacement. For the above reference string, how many more page faults occur with LRU than with the optimal page replacement policy? **(GATE-2007) (2 Marks)**
(A) 0 **(B)** 1 **(C)** 2 **(D)** 3
Answer: (C)

Q Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. Which one of the following is true with respect to page replacement policies First-In-First Out (FIFO) and Least Recently Used (LRU)? **(GATE-2015) (2 Marks)**
(A) Both incur the same number of page faults
(B) FIFO incurs 2 more-page faults than LRU
(C) LRU incurs 2 more-page faults than FIFO
(D) FIFO incurs 1 more page faults than LRU
Answer: (A)

Q Consider the virtual page reference string
1, 2, 3, 2, 4, 1, 3, 2, 4, 1
On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then

(GATE-2012) (2 Marks)

(A) OPTIMAL < LRU < FIFO
(B) OPTIMAL < FIFO < LRU
(C) OPTIMAL = LRU
(D) OPTIMAL = FIFO

Answer: (B)

Q A job has four pages A, B, C, D and the main memory has two-page frames only. The job needs to process its pages in following order: ABACABDBACD. Assuming that a page interrupt occurs when a new page is brought in the main memory, irrespective of whether the page is swapped out or not. The number of page interrupts in FIFO and LRU page replacement algorithms are **(NET-JUNE-2013)**

(A) 9 and 7 **(B)** 7 and 6 **(C)** 9 and 8 **(D)** 8 and 6

Answer: (C)

Q Consider a program that consists of 8 pages (from 0 to 7) and we have 4-page frames in the physical memory for the pages. The page reference string is: 1 2 3 2 5 6 3 4 6 3 7 3 1 5 3 6 3 4 2 4 3 4 5 1. The number of page faults in LRU and optimal page replacement algorithms are respectively (without including initial page faults to fill available page frames with pages): **(NET-JUNE-2014)**

(A) 9 and 6 **(B)** 10 and 7 **(C)** 9 and 7 **(D)** 10 and 6

Answer: (B)

Q A LRU page replacement is used with four page frames and eight pages. How many page faults will occur with the reference string 0172327103 if the four frames are initially empty? **(NET-JUNE-2015)**

a) 6 **b)** 7 **c)** 8 **d)** 5

Answer: (B)

Q Consider the following page reference string : 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6 Which of the following options, gives the correct number of page faults related to LRU, FIFO, and optimal page replacement algorithms respectively, assuming 05 page frames and all frames are initially empty ? **(NET-AUG-2016)**

a) 10, 14, 8 **b)** 8, 10, 7 **c)** 7, 10, 8 **d)** 7, 10, 7

Answer: (B)

Q Consider a virtual page reference string 1, 2, 3, 2, 4, 2, 5, 2, 3, 4. Suppose LRU page replacement algorithm is implemented with 3-page frames in main memory. Then the number of page faults are_____. **(NET-JULY-2018)**

(a) 5 **(b)** 7 **(c)** 9 **(d)** 10

Answer: (B)

Q Suppose that the virtual Address space has eight pages and physical memory with four-page frames. If LRU page replacement algorithm is used, _____ number of page faults occur with the reference string.

0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1 **(NET-AUG-2016)**

(A) 13 **(B)** 12 **(C)** 11 **(D)** 10

Answer: (A)

Solution: Two implementations are feasible:

- **Counters**. In the simplest case, we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter. The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock registers are copied to the time-of-use field in the page-table entry for that page. In this way, we always have the "time" of the last reference to each page. We replace the page with the smallest time value.

- **Stack:** Another approach to implementing LRU replacement is to keep a stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on the top. In this way, the most recently used page is always at the top of the stack and the least recently used page is always at the bottom.

Counting-Based Page Replacement

We keep a counter of the number of references that have been made to each page and develop the following two schemes.

- The **least frequently used (LFU)** page-replacement algorithm requires that the page with the smallest count be replaced.
- The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.
- One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.
- **The most frequently used (MFU)** page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- Neither MFU nor LFU replacement is common. The implementation of these algorithms is expensive, and they do not approximate OPT replacement well.

Q Increasing the RAM of a computer typically improves performance because **(GATE-2005) (1 Marks) (ISRO-2015)**

(A) Virtual memory increases
(B) Larger RAMs are faster
(C) Fewer page faults occur
(D) Fewer segmentation faults occur

Answer: (C)

Q Consider a computer system with ten physical page frames. The system is provided with an access sequence (a_1,a_2,…,a_{20},a_1,a_2,…,a_{20}), where each a_i is a distinct virtual page number. The difference in the number of page faults between the last-in-first-out page replacement policy and the optimal page replacement policy is _____. **(GATE-2016) (1 Marks)**

Answer: 1

Q Consider a main memory with 3 page frames for the following page reference string : 5, 4, 3, 2, 1, 4, 3, 5, 4, 3, 4, 1, 4. Assuming that the execution of process is initiated after loading page 5 in memory, the number of page faults in FIFO and second chance replacement respectively are **(NET-SEP-2013)**

(A) 8 and 9 **(B)** 10 and 11 **(C)** 7 and 9 **(D)** 9 and 8

Answer: (D)

Q A computer has twenty physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, …, 100 in that order, and repeats the access sequence THRICE. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacement policy for this program? **(GATE-2014) (2 Marks)**

(A) Least-recently-used

(B) First-in-first-out

(C) Last-in-first-out

(D) Most-recently-used

Answer: (D)

Frame Allocation Algorithms

Minimum Number of frames to be allocated to each process depends on Instruction Set Architecture, and the maximum number of allocation of frames depends on the size of the process.

Equal Allocation: Frames will be equally allocated to each process. Ex: if we have 30 frames and 3 processes, each will get 10 regardless of their size.

Weighted / Proportional Allocation: Frames will be allocated according to the weights of the process. The allocation of frames per process: **a(i)= s(i)/S x m.**

Where, a(i) is the number of allocated frames, s(i) is the process and S is the size of the virtual memory in general, **S = Σ s(i),** and m is the total number of frames available.

Example: If we have 3 processes of size 20 k, 30k and 40 k then frames will be allocated as:

P1 gets: 20/100*30 = 6 frames, similarly P2 and P3 will get 9 and 15 frames respectively.

Some points to remember

- If the multiprogramming level is increased, each process will lose some frames to provide the memory needed for the new process.
- Conversely, if the multiprogramming level decreases, the frames that were allocated to the departed process can be spread over the remaining processes.

Global vs Local Frame Allocation

- With multiple processes competing for frames, we can classify page-replacement algorithms into two broad categories:
    - **Global Replacement**
    - **Local Replacement**

Global Replacement: Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another.

- With global replacement, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it.
- One problem with a global replacement algorithm is that a process cannot control its own page-fault rate. As The set of pages in memory for a process depends not only on the paging behaviour of that process but also on the paging behaviour of other processes.
- Global replacement generally results in greater system throughput and is therefore the more common method.

Local Replacement: Local replacement requires that each process select from only its own set of allocated frames.

- With a local replacement strategy, the number of frames allocated to a process does not change.
- Under local replacement, the set of pages in memory for a process is affected by the paging behaviour of only that process.
- Local replacement might hinder a process, however, by not making available to it other, less used pages of memory.

Thrashing

- A process is thrashing if it is spending more time paging than executing. High paging activity is called Thrashing.

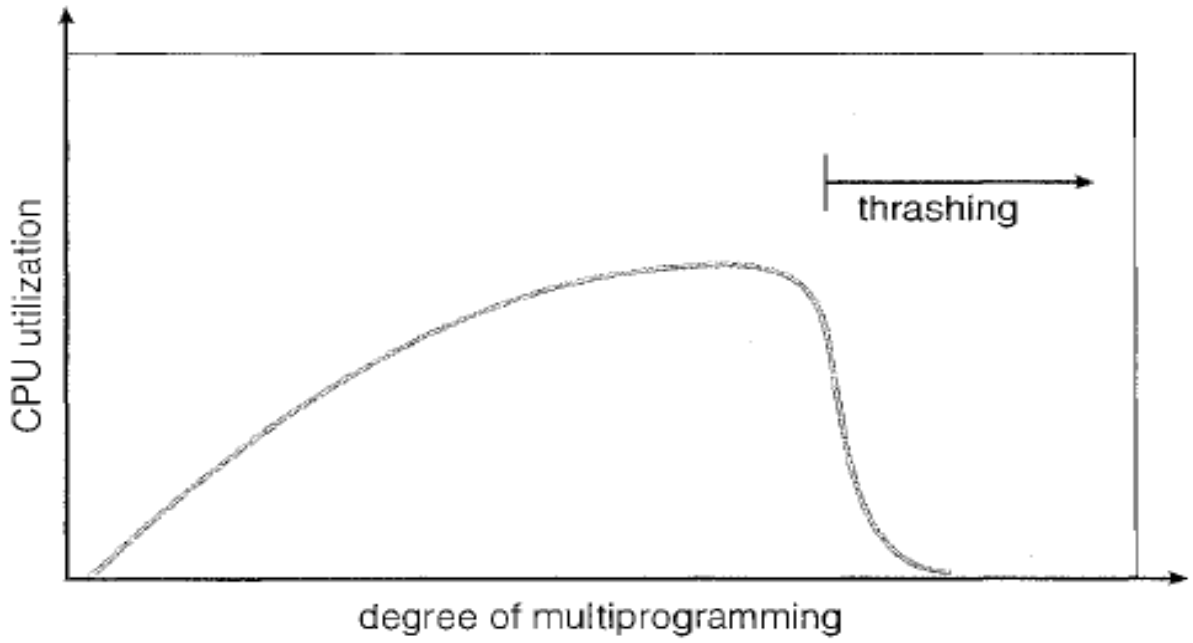


Figure: Thrashing

Causes of Thrashing

High Degree of Multiprogramming: If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system.

A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong. Now if that a process enters a new phase in its execution and needs more frames. It will start faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred, and system throughput plunges. The page fault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging.

Solution

- We can limit the effects of thrashing by using a **Local Replacement Algorithm.**

- With local replacement, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well.
- To prevent thrashing, we must provide a process with as many frames as it needs.
- We decide how to allocate the number of frames to a process by **the working-set strategy.**

The Working Set Strategy

- This approach defines the **locality model** of process execution.
- The locality model states that, as a process executes, it moves from locality to locality. A locality is a set of pages that are actively used together
- A program is generally composed of several different localities, which may overlap.

Working Set Model

- This model uses a parameter Δ, to define the **working set window.**
- The set of pages in the most recent Δ page references is the working set.
- If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set Δ time units after its last reference.
- The working set is an approximation of the program's locality.
- The accuracy of the working set depends on the selection of Δ . If Δ is too small, it will not encompass the entire locality; if Δ is too large, it may overlap several localities.
- If we can calculate the Working-Set Size (WSS) for each process then: D = Σ WSSi, where D is D is the total demand for frames.
- If the total demand is greater than the total number of available frames (D > m), thrashing will occur, because some processes will not have enough frames.

Page-Fault Frequency

- It is a much more direct strategy to control **Page-Fault Frequency.**
- Thrashing has a high page-fault rate and thus we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
- **We establish upper and lower bounds on the desired page-fault rate.**
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame; if the page-fault rate falls below the lower limit, we remove a frame from the process. **With the working-set strategy, we may have to suspend a process. If the page-fault rate increases and no free frames are available, we must select some process and suspend it.**

Q Working set model is used in memory management to implement the concept of **(NET-JUNE-2013)**
(A) Swapping **(B)** Principal of Locality **(C)** Segmentation **(D)** Thrashing
Answer: (B)

File-System Management

- File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).
- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields). Clearly, the concept of a file is an extremely general one.
- The operating system implements the abstract concept of a file by managing mass-storage media, such as tapes and disks, and the devices that control them.
- The operating system is responsible for the following activities in connection with file management:
    - Creating and deleting files
    - Creating and deleting directories to organize files
    - Supporting primitives for manipulating files and directories
    - Mapping files onto secondary storage
    - Backing up files on stable (nonvolatile) storage media

Mass-Storage Management

- As we have already seen, because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the principal on-line storage medium for both programs and data. Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory. They then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system. The operating system is responsible for the following activities in connection with disk management:

    - Free-space management
    - Storage allocation
    - Disk scheduling
- Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may hinge on the speeds of the disk subsystem and the algorithms that manipulate that subsystem.
- There are, however, many uses for storage that is slower and lower in cost (and sometimes of higher capacity) than secondary storage. Backups of disk data, storage of seldom-used data, and long-term archival storage are some examples. Magnetic tape drives and their tapes and CD and DVD drives and platters are typical tertiary storage devices. The media (tapes and optical platters) vary between WORM (write-once, read-many-times) and RW (read– write) formats.
- Tertiary storage is not crucial to system performance, but it still must be managed. Some operating systems take on this task, while others leave tertiary-storage management to application programs. Some of the functions that operating systems can provide include mounting and unmounting media in devices, allocating and freeing the devices for exclusive use by processes, and migrating data from secondary to tertiary storage.

Disk scheduling

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, efficiency means having fast access time and large disk bandwidth.

The disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:

- Whether this operation is input or output
- What is the disk address for the transfer
- What is the memory address for the transfer
- What is the number of sectors to be transferred
- If the desired disk drive and controller are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive. Thus, when one request is completed, the operating system chooses which pending request to service next. How does the operating system make this choice? Any one of several disk-scheduling algorithms can be used.

FCFS (First Come First Serve)

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. In FCFS, the requests are addressed in the order they arrive in the disk queue. This algorithm is intrinsically fair, but it generally does not provide less average seek time.

Advantages:

- Easy to understand easy to use
- Every request gets a fair chance
- no starvation (may suffer from convoy effect)

Disadvantages:

- Does not try to optimize seek time (extra seek movements)
- May not provide the best performance based on different parameters

Q If the disk head is located initially at 32, find the number of disk moves required with FCFS if the disk queue of I/O blocks requests are 98, 37, 14, 124, 65, 67. **(NET-DEC-2012)**

(A) 239 **(B)** 310 **(C)** 321 **(D)** 325

Answer: (C)

Q Assuming that the disk head is located initially at 32, find the number of disk moves required with FCFS if the disk queue of I/O block requests are 98, 37, 14, 124, 65, 67: **(NET-JUNE-2013)**

(A) 310 **(B)** 324 **(C)** 320 **(D)** 321

Answer: (D)

Q If the Disk head is located initially at track 32, find the number of disk moves required with FCFS scheduling criteria if the disk queue of I/O blocks requests are: 98, 37, 14, 124, 65, 67 **(NET-JULY-2016)**

a) 320 **b)** 322 **c)** 321 **d)** 319

Answer: (C)

SSTF Scheduling

It seems reasonable to service all the requests close to the current head position before moving the head far to service other REQUESTS. This assumption is the basis for the SSTF algorithm.

In SSTF (Shortest Seek Time First), the request nearest to the disk arm will get executed first i.e. requests having shortest seek time are executed first.

Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal. In the example, we can do better by moving the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122, 124, and 183. This strategy reduces the total head movement to 208 cylinders.

Advantages:

- Seek movements decreases
- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate the closest request.
- Can cause Starvation for a request which is far from the current location of the header
- High variance of response time as SSTF favours only some requests

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause starvation of some requests.

Q Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence 4, 34, 10, 7, 19, 73, 2, 15, 6, 20. Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1ms to move from one cylinder to adjacent one and shortest seek time first policy is used? (GATE-2009) (1 Marks)

(A) 95 ms **(B)** 119 ms **(C)** 233 ms **(D)** 276 ms

Answer: (B)

Q Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time, the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing _____ number of

requests. **(GATE-2014) (1 Marks)**

Answer: 3

Q Consider an imaginary disk with 40 cylinders. A request come to read a block on cylinder 11. While the seek to cylinder 11 is in progress, new requests come in for cylinders 1, 36, 16, 34, 9 and 12 in that order. The number of arm motions using shortest seek first algorithm is **(NET-DEC-2014)**

(A) 111 **(B)** 112 **(C)** 60 **(D)** 61

Answer: (D)

Q Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. Suppose SSTF disk scheduling algorithm implemented to meet the requests then the total number of head movements are _____ if the disk head is initially at 53. **(NET-NOV-2017)**

a) 224 **b)** 248 **c)** 236 **d)** 240

Answer: (C)

Q A disk drive has 100 cylinders, numbered 0 to 99. Disk requests come to the disk driver for cylinders 12, 26, 24, 4, 42, 8 and 50 in that order. The driver is currently serving a request at cylinder 24. A seek takes 6 msec per cylinder moved. How much seek time is needed for shortest seek time first (SSTF) algorithm? **(NET-JUNE-2015)**

a) 0.984 sec **b)** 0.396 sec **c)** 0.738 sec **d)** 0.42 sec

Answer: (D)

SCAN

The disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each track, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the Elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

Assuming a uniform distribution of requests for cylinders, consider the density of requests when the head reaches one end and reverses direction. At this point, relatively few requests are immediately in front of the head since these cylinders have recently been serviced. The heaviest density of requests is at the other end of the disk These requests have also waited the longest.

Advantages:

- Simple easy to understand and use
- No starvation but more wait for some random process
- Low variance and Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm.
- Unnecessary move to the end of the disk, even if there is no request.

Q On a disk with 1000 cylinders (0 to 999) find the number of tracks, the disk arm must move to satisfy all the requests in the disk queue. Assume the last request service was at track 345 and the head is moving toward track 0. The queue in FIFO order contains requests for the following tracks: 123, 874, 692, 475, 105, 376 (Assume SCAN algorithm) **(NET-DEC-2012)**

(A) 2013 **(B)** 1219 **(C)** 1967 **(D)** 1507

Answer: (B)

Q Consider a disk queue with I/O requests on the following cylinders in their arriving order: 6, 10, 12, 54, 97, 73, 128, 15, 44, 110, 34, 45 The disk head is assumed to be at cylinder 23 and moving in the direction of decreasing number of cylinders. Total number of cylinders in

the disk is 150. The disk head movement using SCAN-scheduling algorithm is: **(NET-JAN-2017)**

a) 172 **b)** 173 **c)** 227 **d)** 228

Answer: (all options are incorrect 151)

Q Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks **(GATE-2015) (2 Marks)**

Answer: 10

Q Consider a disk queue with request for input/output to block on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. Assume that disk head is initially positioned at cylinder 53 and moving towards cylinder number 0. The total number of head movements using Shortest Seek Time First (SSTF) and SCAN algorithms are respectively **(NET-DEC-2013)**

(A) 236 and 252 cylinders **(B)** 640 and 236 cylinders

(C) 235 and 640 cylinders **(D)** 235 and 252 cylinders

Answer: (all options are incorrect)

C-SCAN Scheduling

Circular-scan is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

Advantages:

- Provides more uniform wait time compared to SCAN
- Better response time compared to scan

Disadvantage:

- More seeks movements in order to reach starting position

Q Consider the situation in which the disk read/write head is currently located at track 45 (of tracks 0-255) and moving in the positive direction. Assume that the following track requests have been made in this order: 40, 67, 11, 240, 87. What is the order in which optimized C-SCAN would service these requests and what is the total seek distance?**(GATE-2015) (2 Marks)**

(A) 600 **(B)** 810 **(C)** 505 **(D)** 550

Answer: (D)

LOOK Scheduling

It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Advantage: -

- Better performance compared to SCAN
- Should be used in case to less load

Disadvantage: -

- Overhead to find the last request
- Should not be used in case of more load.

C LOOK

As LOOK is similar to SCAN algorithm, in similar way, C-LOOK is similar to C-SCAN disk scheduling algorithm. In C-LOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Advantage: -

- Provides more uniform wait time compared to LOOK
- Better response time compared to LOOK

Disadvantage: -

- Overhead to find the last request and go to initial position is more
- Should not be used in case of more load.

Q Which of the following statements is not true about disk-arm scheduling algorithms? **(NET-JUNE-2014)**

(A) SSTF (shortest seek time first) algorithm increases performance of FCFS.

(B) The number of requests for disk service are not influenced by file allocation method.

(C) Caching the directories and index blocks in main memory can also help in reducing disk arm movements.

(D) SCAN and C-SCAN algorithms are less likely to have a starvation problem.

Answer: (B)

Q In _____ disk scheduling algorithm, the disk head moves from one end to other end of the disk, serving the requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk without serving any requests on the return trip. **(NET-NOV-2017)**

a) LOOK **b)** SCAN **c)** C-LOOK **d)** C-SCAN

Answer: (D)

Q Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks(GATE-2015) (2 Marks)

Answer: 10

Q Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____. **(GATE-2018) (2 Marks)**

Answer: 346

Conclusion

Selection of a Disk-Scheduling Algorithm

- Given so many disk-scheduling algorithms, how do we choose the best one? SSTF is common and has a natural appeal because it increases performance over FCFS. SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem.
- With any scheduling algorithm, however, performance depends heavily on the number and types of requests. For instance, suppose that the queue usually has just one outstanding request. Then, all scheduling algorithms behave the same, because they have only one choice of where to move the disk head: they all behave like FCFS scheduling.
- Requests for disk service can be greatly influenced by the file-allocation method. A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.
- The location of directories and index blocks is also important. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently. Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. In this case, the disk head has to move the entire width of the disk. If the directory entry were on the middle cylinder, the head would have to move only one-half the width. Caching the directories and index blocks in main memory can also help to reduce disk-arm movement, particularly for read requests.
- Because of these complexities, the disk-scheduling algorithm should be written as a separate module of the operating system, so that it can be replaced with a different algorithm if necessary. Either SSTF or LOOK is a reasonable choice for the default algorithm.
- The scheduling algorithms described here consider only the seek distances. For modern disks, the rotational latency can be nearly as large as the average seek time. It is difficult for the operating system to schedule for improved rotational latency, though, because modern disks do not disclose the physical location of logical blocks. Disk manufacturers have been alleviating this problem by implementing disk-scheduling algorithms in the controller hardware built into the disk drive. If the operating system sends a batch of requests to the controller, the controller can queue them and then schedule them to improve both the seek time and the rotational latency.
- If I/O performance were the only consideration, the operating system would gladly turn over the responsibility of disk scheduling to the disk hard- ware. In practice, however, the operating system may have other constraints on the service order for requests. For instance, demand paging may take priority over application I/O, and writes are more urgent than reads if the cache is running out of free pages.
- Also, it may be desirable to guarantee the order of a set of disk writes to make the file

system robust in the face of system crashes. Consider what could happen if the operating system allocated a disk page to a file and the application wrote data into that page before the operating system had a chance to flush the file system metadata back to disk. To accommodate such requirements, an operating system may choose to do its own disk scheduling and to spoon-feed the requests to the disk controller, one by one, for some types of I/O.

- SSTF is common and has a natural appeal because it increases performance over FCFS.
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem.
- With any scheduling algorithm however, performance depends heavily on the number and types of requests.
- For instance, suppose that the queue usually has just one outstanding request. Then all scheduling algorithms behave the same because they have only one choice of where to move the disk head: they all behave like FCFS scheduling.
- Requests for disk service can be greatly influenced by the file-allocation method.
- Because of these complexities, the disk-scheduling algorithm should be written as a separate module of the operating system, so that it can be replaced with a different algorithm if necessary. Either SSTF or LOOK is a reasonable choice for the default algorithm. The scheduling algorithms described here consider only the seek distances

Q Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs? **(GATE-2004) (1 Marks)**

(A) 50% **(B)** 40% **(C)** 25% **(D)** 0%

Answer: (D)

Q Which of the following disk scheduling strategies is likely to give the best throughput? **(GATE-1999) (1 Marks)**

(a) Farthest cylinder next **(b)** Nearest cylinder next

(c) First come first served **(d)** Elevator algorithm

Answer: (B)

Q Match the following **(NET-DEC-2004)**

(a) Disk scheduling	(1) Round robin
(b) Batch processing	(2) Scan
(c) Time sharing	(3) LIFO
(d) Interrupt processing	(4) FIFO

Codes:

(A)	a-3	b-4	c-2	d-1
(B)	a-4	b-3	c-2	d-1
(C)	a-2	b-4	c-1	d-3
(D)	a-3	b-4	c-1	d-2

Answer: (C)

Q The maximum amount of information that is available in one portion of the disk access arm for a removal disk pack (without further movement of the arm with multiple heads) **(NET-DEC-2011)**

(A) a plate of data **(B)** a cylinder of data

(C) a track of data **(D)** a block of data

Answer: (B)

Q Consider the input/output (I/O) requests made at different instants of time directed at a hypothetical disk having 200 tracks as given in the following table

Serial No	1	2	3	4	5
Track No	12	85	40	100	75
Time of arrival	65	80	110	100	175

Assume that:
Current head position is at track no. 65 Direction of last movement is towards higher numbered tracks Current clock time is 160 milliseconds Head movement time per track is 1 millisecond. “look” is a variant of “SCAN” disk- arm scheduling algorithm. In this algorithm, if no more I/O requests are left in current direction, the disk head reverses its direction. The seek times in Shortest Seek First (SSF) and “look” disk-arm scheduling algorithms respectively are: **(NET-SEP-2013)**

(A) 144 and 123 milliseconds
(B) 143 and 123 milliseconds
(C) 149 and 124 milliseconds
(D) 256 and 186 milliseconds

Answer: (B)

Q An operating system supports a paged virtual memory, using a central processor with a cycle time of one microsecond. It costs an additional one microsecond to access a page other than the current one. Pages have 1000 words, and the paging device is a drum that rotates at 3000 revolutions per minute and transfers one million words per second. Further, one percent of all instructions executed accessed a page other than the current page. The instruction that accessed another page, 80% accessed a page already in memory and when a new page was required, the replaced page was modified 50% of the time. What is the effective access time on this system, assuming that the system is running only one process and the processor is idle during drum transfers? **(NET-AUG-2016)**

A) 30 microseconds **b)** 34 microseconds **c)** 60 microseconds **d)**68 microseconds

Answer: (B)

Q Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, …, 199), and 256 sectors per track (numbered as 0, 1, … 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:
[120, 72, 2], [180, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 16, 1]
Currently head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts. Power dissipation associated with rotational latency and switching of head between different platters is negligible.
The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithm is _____. **(GATE-2018) (2 Marks)**

Answer: 85

Q Consider a hard disk with 16 recording surfaces (0-15) having 16384 cylinders (0-16383) and each cylinder contains 64 sectors (0-63). Data storage capacity in each sector is 512 bytes. Data are organized cylinder-wise and the addressing format is <cylinder no., surface no., sector no.>. A file of size 42797 KB is stored in the disk and the starting disk location of the file is <1200, 9, 40>. What is the cylinder number of the last sector of the file, if it is stored in a contiguous manner? (**GATE-2013) (2 Marks)**

Answer: 1284

Q For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to **(GATE-2008) (1 Marks)**

a) non-uniform distribution of requests

b) arm starting and stopping inertia

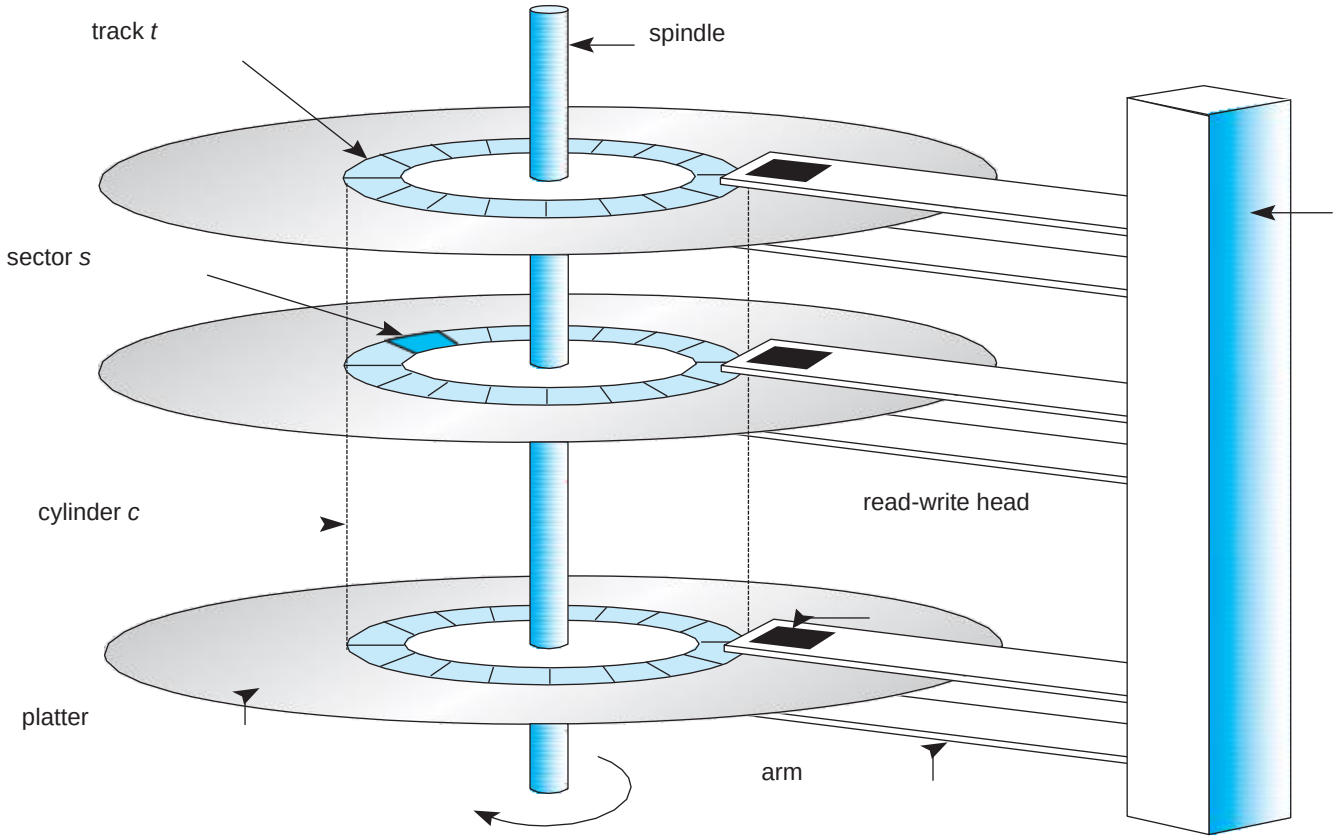
c) higher capacity of tracks on the periphery of the platter

d) use of unfair arm scheduling policies

Answer B

Transfer time

- Magnetic disks provide the bulk of secondary storage for modern computer systems. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.



- A read–write head "flies" 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 subdivided 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 is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute **(RPM)**. Common drives spin at 5,400, 7,200, 10,000, and 15,000 RPM.
- Total transfer time has two parts. The **positioning time**, or **random-access time**, consists of two parts: the time necessary to move the disk arm to the desired cylinder, called the **seek time**, and the time necessary for the desired sector to rotate to the disk head, called the **rotational latency**.

Q Consider a disk where there are 512 tracks, each track is capable of holding 128 sector and each sector holds 256 bytes, find the capacity of the track and disk and number of bits required to reach correct track, sector and disk.

Q consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?

Q Consider a system with 8 sector per track and 512 bytes per sector. Assume that disk rotates at 3000 rpm and average seek time is 15ms standard. Find total time required to transfer a file which requires 8 sectors to be stored.
a) Assume contiguous allocation
b) Assume Non- contiguous allocation

Total Transfer Time = Seek Time + Rotational Latency + Transfer Time

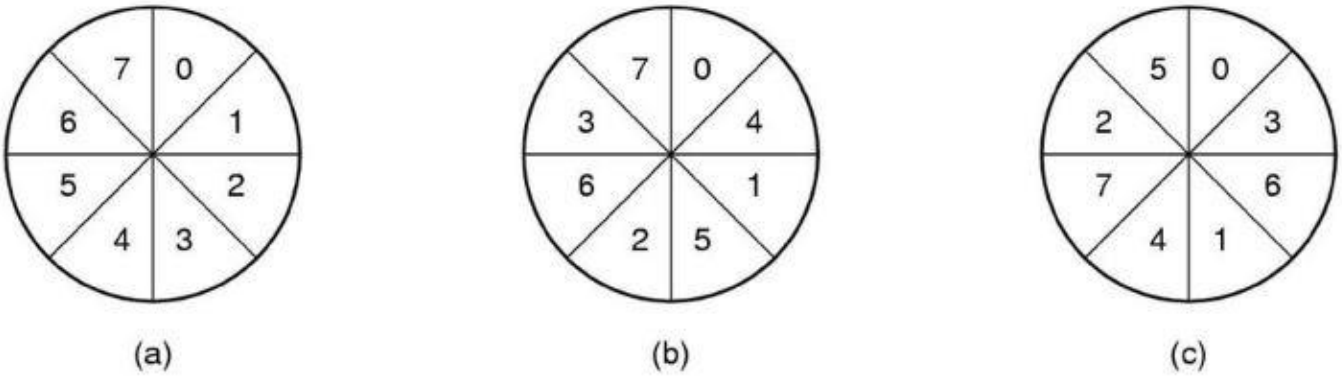
Seek Time: - It is a time taken by Read/Write header to reach the correct track. (Always given in question)

Rotational Latency: - It is the time taken by read/Write header during the wait for the correct sector. In general, it's a random value, so far average analysis, we consider the time taken by disk to complete half rotation.

Transfer Time: - it is the time taken by read/write header either to read or write on a disk. In general, we assume that in 1 complete rotation, header can read/write the either track, so total time will be

(File Size/Track Size) *time taken to complete one revolution.

Disk interleaving



- No interleaving
- Single interleaving
- Double interleaving

Single interleaving: - in 2 rotation we read 1 track
Double interleaving: - in 2.75 rotation we read 1 track

Q Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively(GATE-2006) (2 Marks)

(A) 256 Mbyte, 19 bits
(B) 256 Mbyte, 28 bits
(C)512 Mbyte, 20 bits
(D) 64 Gbyte, 28 bit

Answer: (A)

Q Consider a disk with 16384 bytes per track having a rotation time of 16 msec and average seek time of 40 msec. What is the time in msec to read a block of 1024 bytes from this disk? **(NET-DEC-2015)**

a) 57 sec **b)** 49 sec **c)** 48 sec **d)** 17 sec

Answer: (B)

Q An application loads 100 libraries at start-up. Loading each library requires exactly one disk access. The seek time of the disk to a random location is given as 10 milli second. Rotational speed of disk is 6000 rpm. If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries? (The time to transfer data from the disk block once the head has been positioned at the start of the block may be neglected)

(GATE-2011) (2 Marks)

(A) 0.50 s **(B)** 1.50 s **(C)** 1.25 s **(D)** 1.00 s

Answer: (B)

Q Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is _____. (GATE-2015) (2 Marks)

Answer: 14020

Q Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50×10^6 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is _____ (GATE-2015) (2 Marks)

Answer: 6.1

Q For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to(GATE-2008) (2 Marks)

(A) non-uniform distribution of requests

(B) arm starting and stopping inertia

(C) higher capacity of tracks on the periphery of the platter

(D) use of unfair arm scheduling policies

Answer: (B)

Q Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively(GATE-2006) (2 Marks)

(A) 256 Mbyte, 19 bits **(B)** 256 Mbyte, 28 bits

(C) 512 Mbyte, 20 bits **(D)** 64 Gbyte, 28 bit

Answer: (A)

Q Using a larger block size in a fixed block size file system leads to **(GATE-2003) (1 Marks)**

(A) better disk throughput but poorer disk space utilization

(B) better disk throughput and better disk space utilization

(C) poorer disk throughput but better disk space utilization

(D) poorer disk throughput and poorer disk space utilization

Answer: (A)

File allocation methods

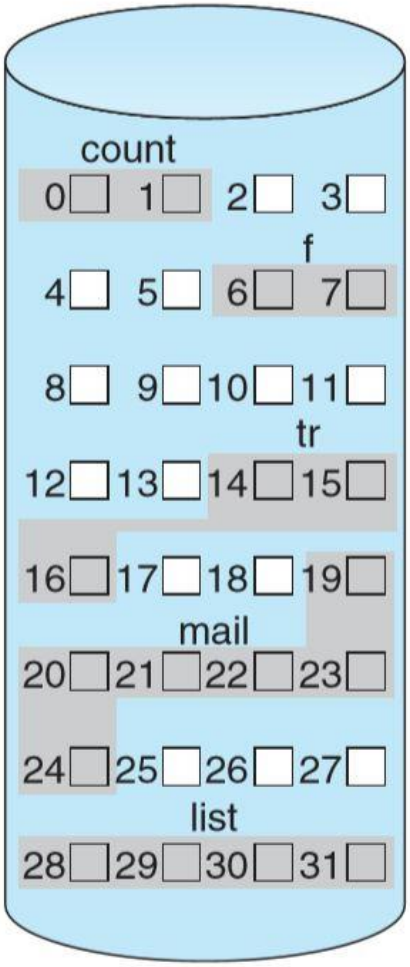
The main aim of file allocation problem is how disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use:

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

Each method has advantages and disadvantages. Although some systems support all three, it is more common for a system to use one method for all files.

Contiguous Allocation

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block b + 1 after block b normally requires no head movement.



directory

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

- Advantage
  - Accessing a file that has been allocated contiguously is easy. Thus, both sequential and direct access can be supported by contiguous allocation.
  -
- Disadvantage
  - Suffer from the problem of external fragmentation.
  - Suffer from huge amount of external fragmentation.
  - Another problem with contiguous allocation file modification

Q Suppose there are six files F1, F2, F3, F4, F5, F6 with corresponding sizes 150 KB, 225 KB, 75 KB, 60 KB, 275 KB and 65 KB respectively. The files are to be stored on a sequential device in such a way that optimizes access time. In what order should the files be stored? **(NET-NOV-2017)**

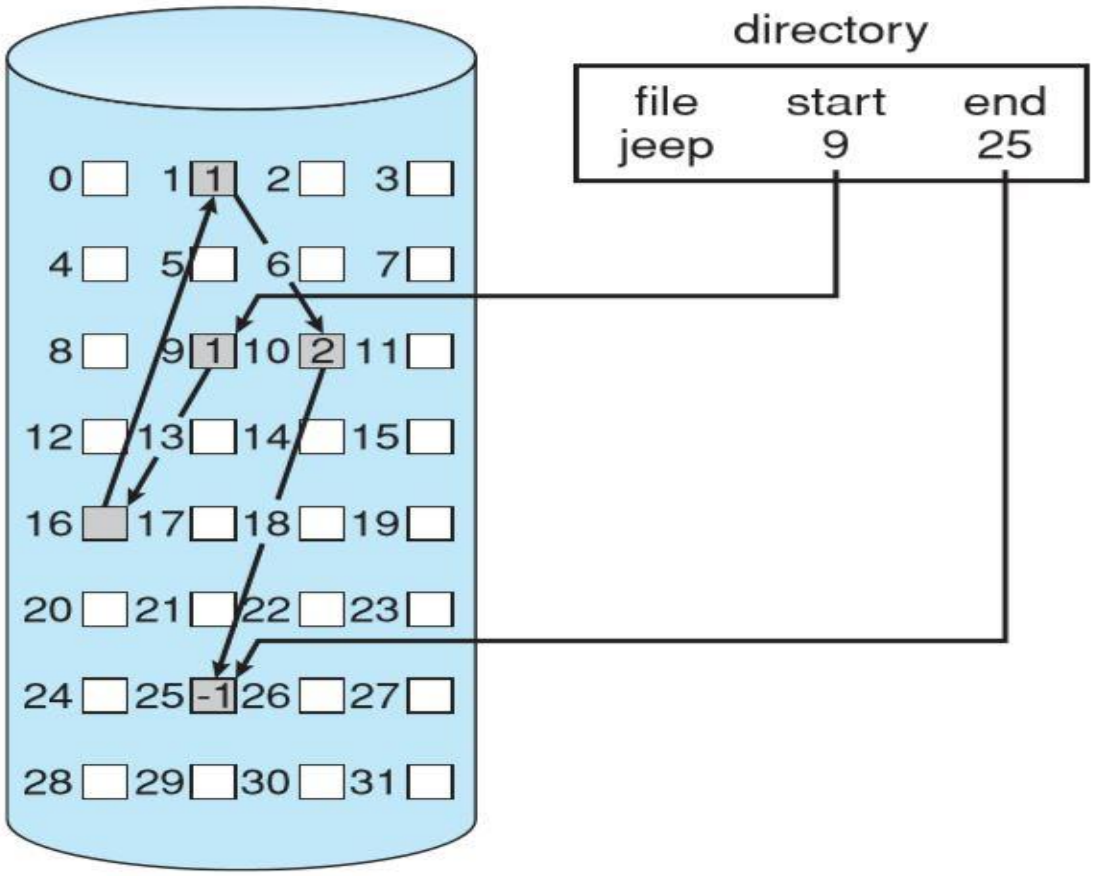
A) F5, F2, F1, F3, F6, F4 **b)** F4, F6, F3, F1, F2, F5

c) F1, F2, F3, F4, F5, F6 **d)** F6, F5, F4, F3, F2, F1

Answer: (B)

Linked Allocation

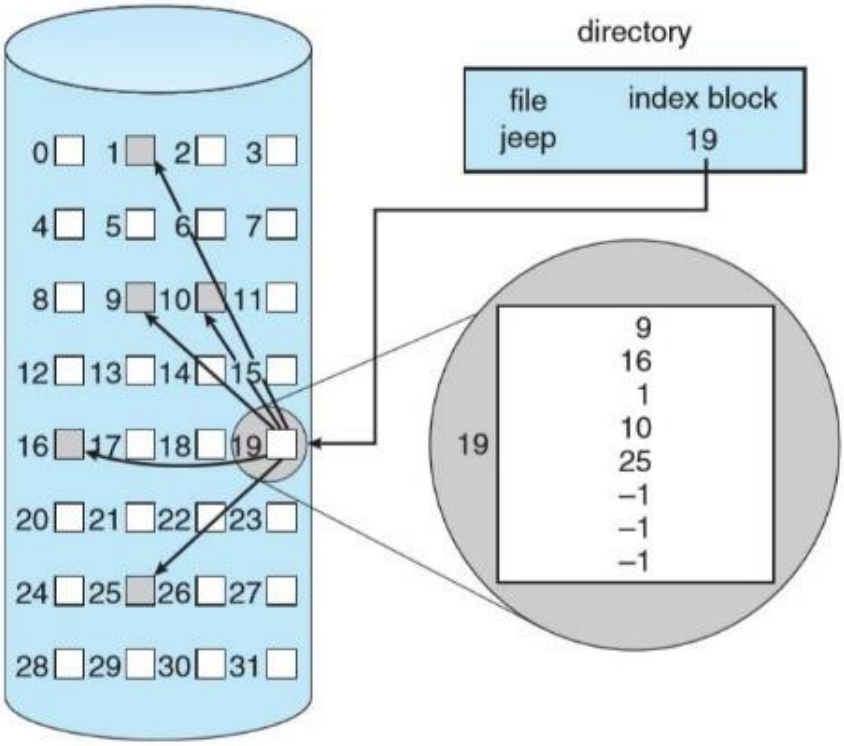
- Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

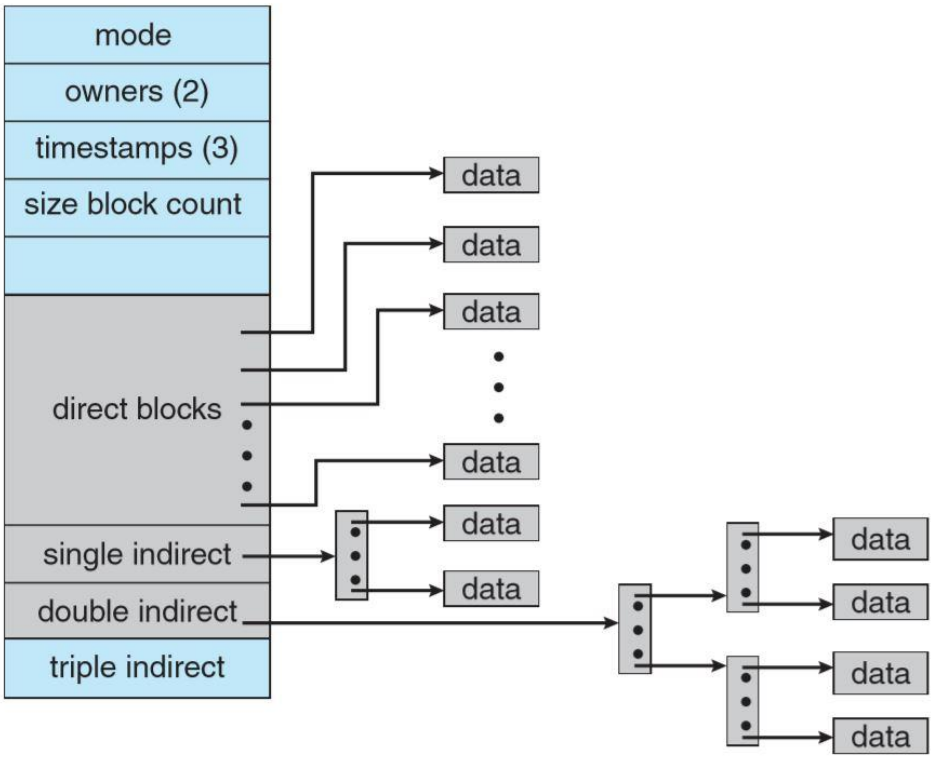


- Advantage: -
    - To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. To read a file, we simply read blocks by following the pointers from block to block. The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available.
    - There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.
- **Disadvantage: -**
    - To find the i^{th} block of a file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block. Each access to a pointer requires a disk read.
    - Another disadvantage is the space required for the pointers, so each file requires slightly more space than it would otherwise.
    - Yet another problem is reliability. Recall that the files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged. One partial solution is to use doubly linked lists, and another is to store the file name and relative block number in each block. However, these schemes require even more overhead for each file.

Indexed Allocation

- Indexed allocation solves problems of contiguous and linked allocation, by bringing all the pointers together into one location: the index block.





- Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the ith block of the file. The directory entry contains the address of the index block. To find and read the ith block, we use the pointer in the ith index-block entry.
- When the file is created, all pointers in the index block are set to null. When the ith block is first written, a block is obtained from the free-space manager, and its address is put in the ith index-block entry.
- This point raises the question of how large the index block should be. Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file.
- Linked scheme: To allow for large files, we can link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is null (for a small file) or is a pointer to another index block (for a large file).
- Multilevel index. A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size.
- Combined scheme. Another alternative, used in UNIX-based file systems, is to keep the first, say, 15 pointers of the index block in the file's inode. The first 12 of these

pointers point to direct blocks; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files do not need a separate index block. The next three pointers point to indirect blocks. The first points to a single indirect block, which is an index block containing not data but the addresses of blocks that do contain data. The second points to a double indirect block, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The last pointer contains the address of a triple indirect block.

- Advantage
  - Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.
- Disadvantage
  - Indexed allocation does suffer from wasted space, however. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Q In a file allocation system, which of the following allocation scheme(s) can be used if no external fragmentation is allowed? **(GATE-2017) (1 Marks)**

I. Contiguous **II.** Linked **III.** Indexed

(1) I and III only **(2)** II only **(3)** III only **(4)** II and III only

Q The data blocks of a very large file in the Unix file system are allocated using **(GATE-2008) (1 Marks)**

(A) contiguous allocation **(B)** linked allocation

(C) indexed allocation **(D)** an extension of indexed allocation

Answer: (D)

Q In the index allocation scheme of blocks to a file, the maximum possible size of the file depends on **(GATE-2002) (1 Marks)**

(a) the size of the blocks, and the size of the address of the blocks.

(b) the number of blocks used for the index, and the size of the blocks.

(c) the size of the blocks, the number of blocks used for the index, and the size of the address of the blocks.

(d) None of the above

Answer: (C)

Q Match the following: **(NET-JUNE-2014)**

List-I	List-II
a) **Contiguous allocation**	**i)** This scheme supports very large file sizes.
b) **Linked allocation**	**ii)** This allocation technique supports only sequential files.
c) **Indexed allocation**	**iii)** Number of disks required to access file is minimal.
d) **Multi-level indexed**	**iv)** This technique suffers from maximum wastage of space in storing pointers.

Codes:

	(a)	(b)	(c)	(d)
(a)	(iii)	(iv)	(ii)	(i)
(b)	(iii)	(ii)	(iv)	(i)
(c)	(i)	(ii)	(iv)	(iii)
(d)	(i)	(iv)	(ii)	(iii)

Answer: (B)

Q A file system with 300 Gbyte disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 Bytes and the size of each disk block address is 8 Bytes. The maximum possible file size in this file system is **(GATE-2012) (2 Marks)**

(A) 3 Kbytes
(B) 35 Kbytes
(C) 280 Bytes
(D) Dependent on the size of the disk

Answer: (B)

Q A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointer. Disk block size is 1 Kbyte, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size? **(GATE-2004) (2 Marks)**

(A) 2^{24} bytes **(B)** 2^{32} bytes **(C)** 2^{34} bytes **(D)** 2^{48} bytes

Answer: (C)

Q The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4 kB, and the disk block address is 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____ GB. **(GATE-2014) (2 Marks)**

Answer: 4

Q Consider a file currently consisting of 50 blocks. Assume that the file control block and the index block is already in memory. If a block is added at the end (and the block information to be added is stored in memory), then how many disk I/O operations are required for indexed (single-level) allocation strategy? **(NET-NOV-2016)**

A) 1 **b)** 101 **c)** 27 **d)** 0

Ans: a

Free-Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks—those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Q How many disk blocks are required to keep list of free disk blocks in a 16 GB hard disk with 1 kB block size using linked list of free disk blocks? Assume that the disk block number is stored in 32 bits. **(NET-DEC-2014)**

(A) 1024 blocks **(B)** 16794 blocks **(C)** 20000 blocks **(D)** 1048576 blocks

Ans none of above

Bit Vector

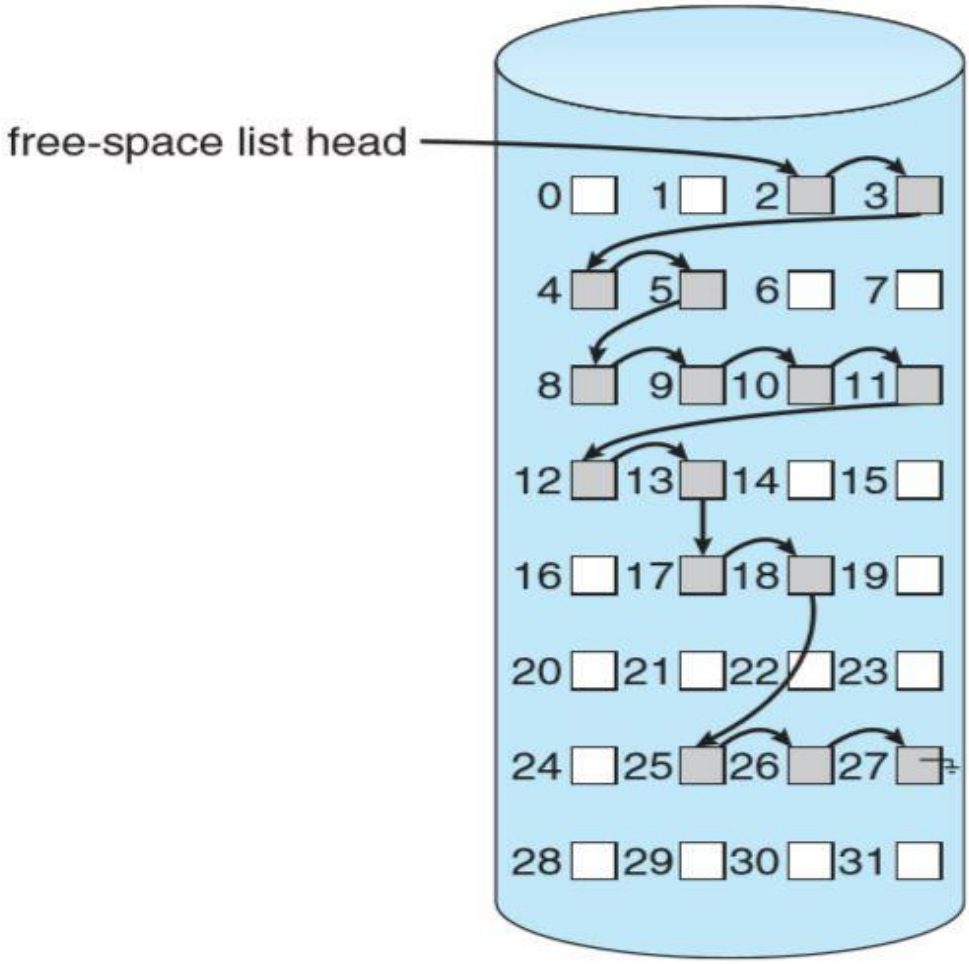
- Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be 001111001111110001100000011100000 ...
- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.
- Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory. Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.
- A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks.
- A 1-TB disk with 4-KB blocks requires 256 MB to store its bit map. Given that disk size constantly increases, the problem with bit vectors will continue to escalate as well.

Q How much space will be required to store the bit map of a 1.3 GB disk with 512 bytes block size? (**NET-DEC-2013)**

(A) 332.8 KB **(B)** 83.6 KB **(C)** 266.2 KB **(D)** 256.6 KB

Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on. Recall our earlier example, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.
- In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.



- Fortunately, however, traversing the free list is not a frequent action. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.

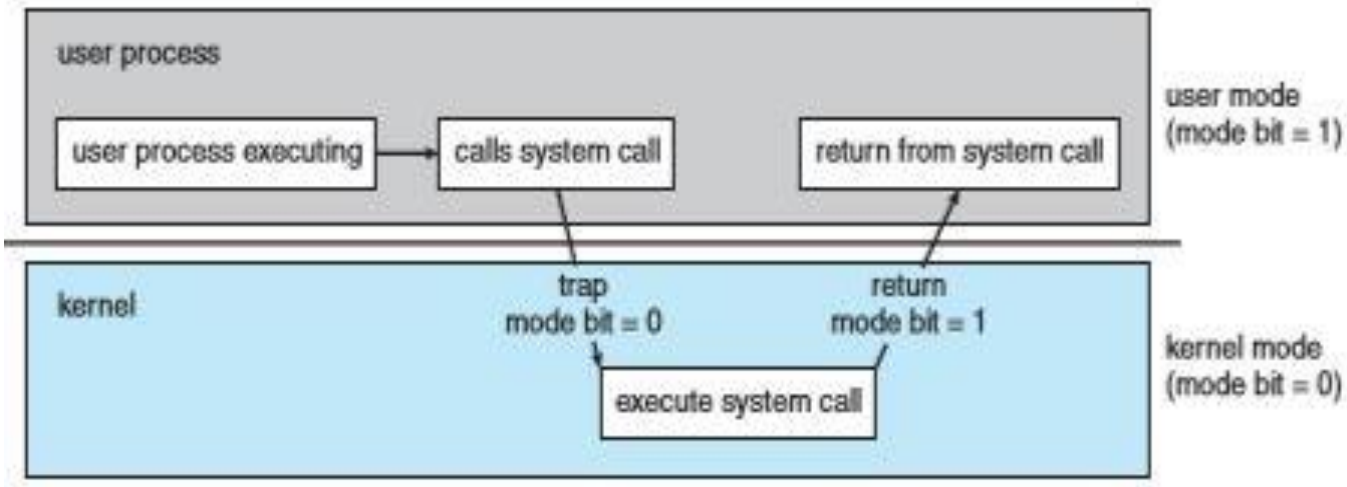
Q A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a 100 x 10^6 bytes disk on which the file system is stored and data block size is 10^3 bytes, the maximum size of a file that can be stored on this disk in units of 10^6 bytes is _____. **(GATE-2014) (2 Marks)**
Answer: 99.55 to 99.65

Q An experimental file server is up 75% of the time and down for 25% of the time due to bugs. How many times does this file server have to be replicated to give an availability of at least 99%? **(NET-NOV-2016)**
a) 2 **b)** 4 **c)** 8 **d)** 16
Ans: a

System call and Mode bit

- 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.
- At the very least, we need two separate *modes* of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged 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).
- 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), the system 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).
- 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/O control, timer management, and interrupt management.

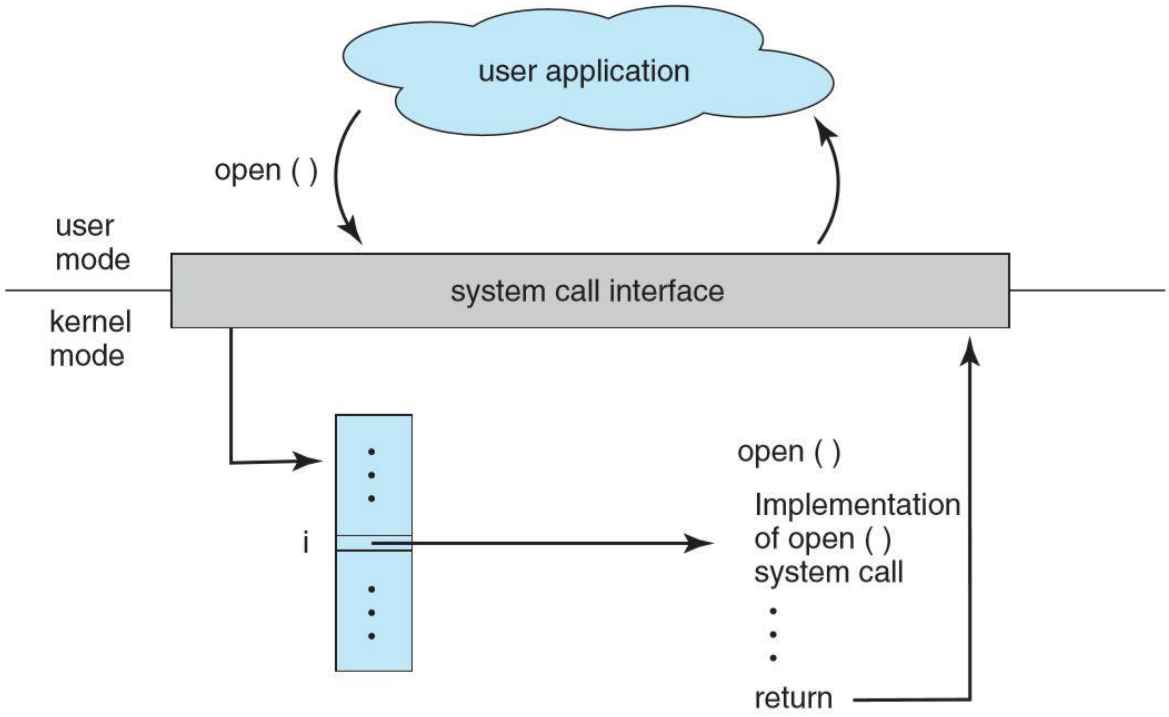


- System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.

- **System calls** provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language instructions.

- The API specifies a set of functions that are available to an application programmer,

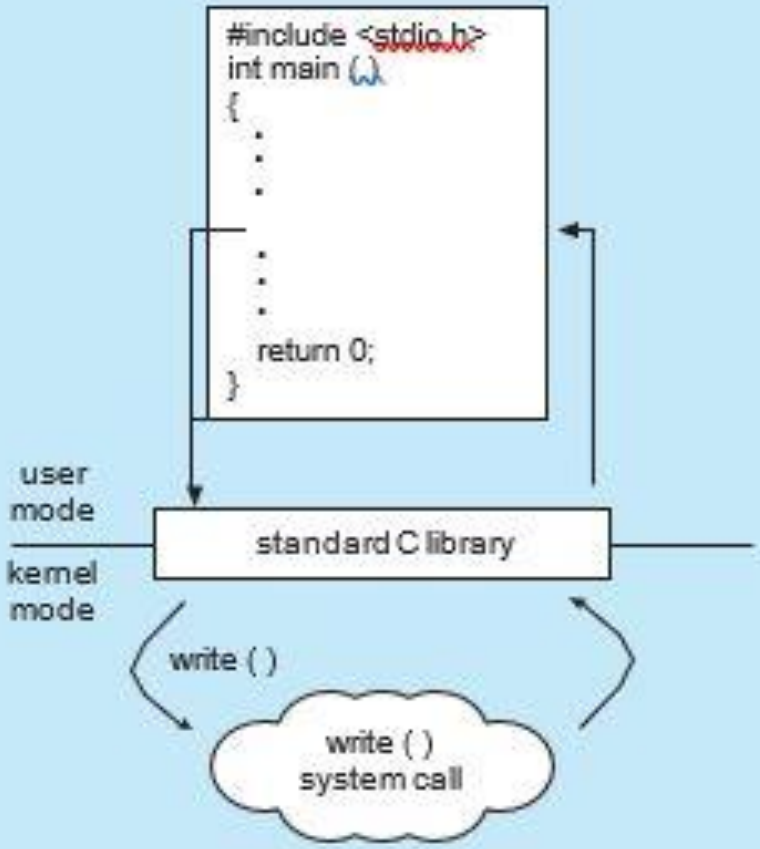
including the parameters that are passed to each function and the return values the programmer can expect. Three of the most common APIs available to application programmers are the Windows API for Windows systems, the POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and Mac OS X), and the Java API for programs that run on the Java virtual machine.



- The caller need know nothing about how the system call is implemented or what it does during execution. Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.

EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the printf() statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the write() system call. The C library takes the value returned by write() and passes it back to the user program. This is shown below:



Types of System Calls - System calls can be grouped roughly into six major categories: **process control**, **file manipulation**, **device manipulation**, **information maintenance**, **communications**, and **protection**.

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

User and Operating-System Interface

- there are several ways for users to interface with the operating system. Here, we discuss two fundamental approaches. One provides a command-line interface, or **command interpreter**, that allows users to directly enter commands to be performed by the operating system. The other allows users to interface with the operating system via a graphical user interface, or GUI.

- Command Interpreters - Some operating systems include the command interpreter in the kernel. Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (on interactive systems). On systems with multiple command interpreters to choose from, the interpreters are known as **shells**. For example, on UNIX and Linux systems, a user may choose among several different shells, including the ***Bourne shell***, ***C shell***, ***Bourne-Again shell***, ***Korn shell***, and others.

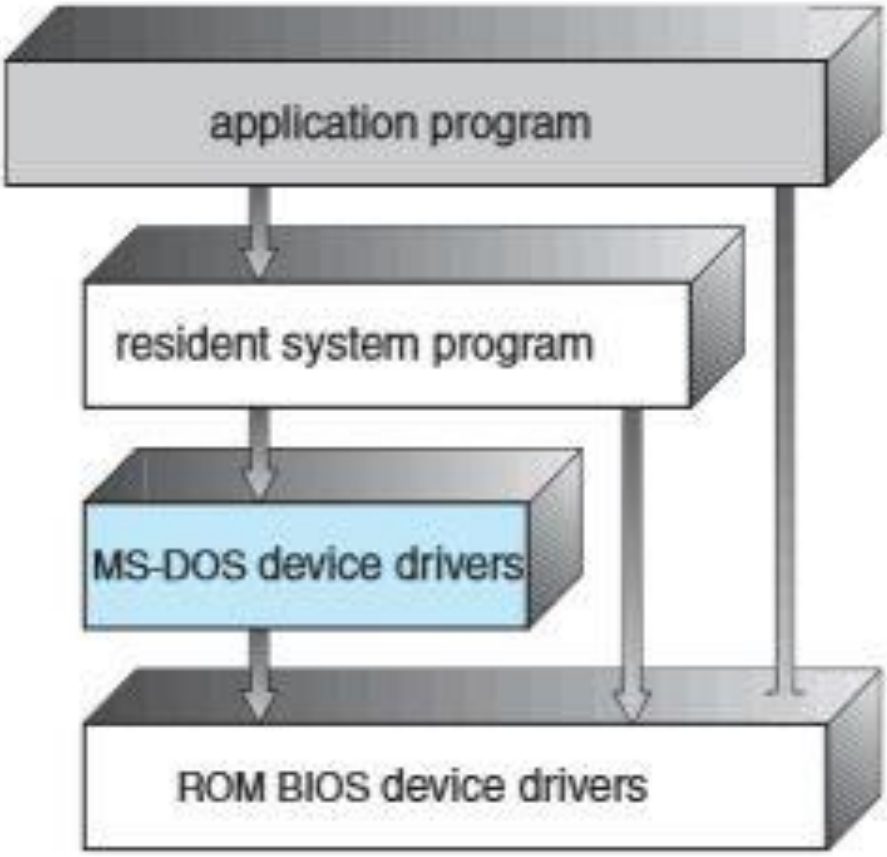
- Graphical User Interfaces - A second strategy for interfacing with the operating system is through a user- friendly graphical user interface, or GUI. Here, users employ a mouse-based window- and-menu system characterized by a **desktop**. The user moves the mouse to position its pointer on images, or **icons**, on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a **folder** —or pull down a menu that contains commands.

- Because a mouse is impractical for most mobile systems, smartphones and handheld tablet computers typically use a touchscreen interface. Here, users interact by making **gestures** on the touchscreen—for example, pressing and swiping fingers across the screen.

- The choice of whether to use a command-line or GUI interface is mostly one of personal preference. **System administrators** who manage computers and **power users** who have deep knowledge of a system frequently use the command-line interface. For them, it is more efficient, giving them faster access to the activities they need to perform. Indeed, on some systems, only a subset of system functions is available via the GUI, leaving the less common tasks to those who are command-line knowledgeable.

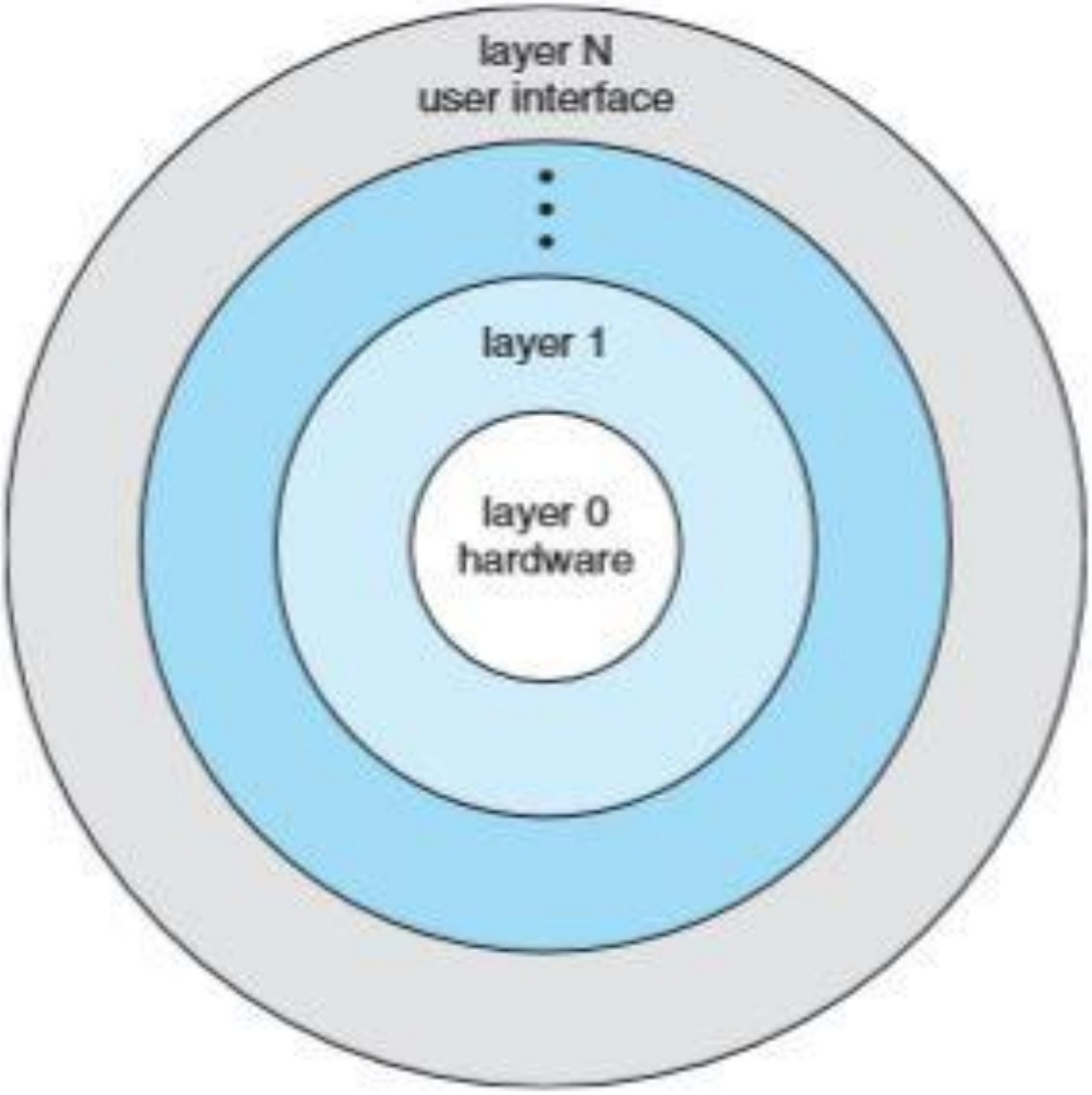
Structure of Operating System

- A common approach is to partition the task into small components, or modules, rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.
- Simple Structure - Many operating systems do not have well-defined structures. Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system.



MS-DOS layer structure.

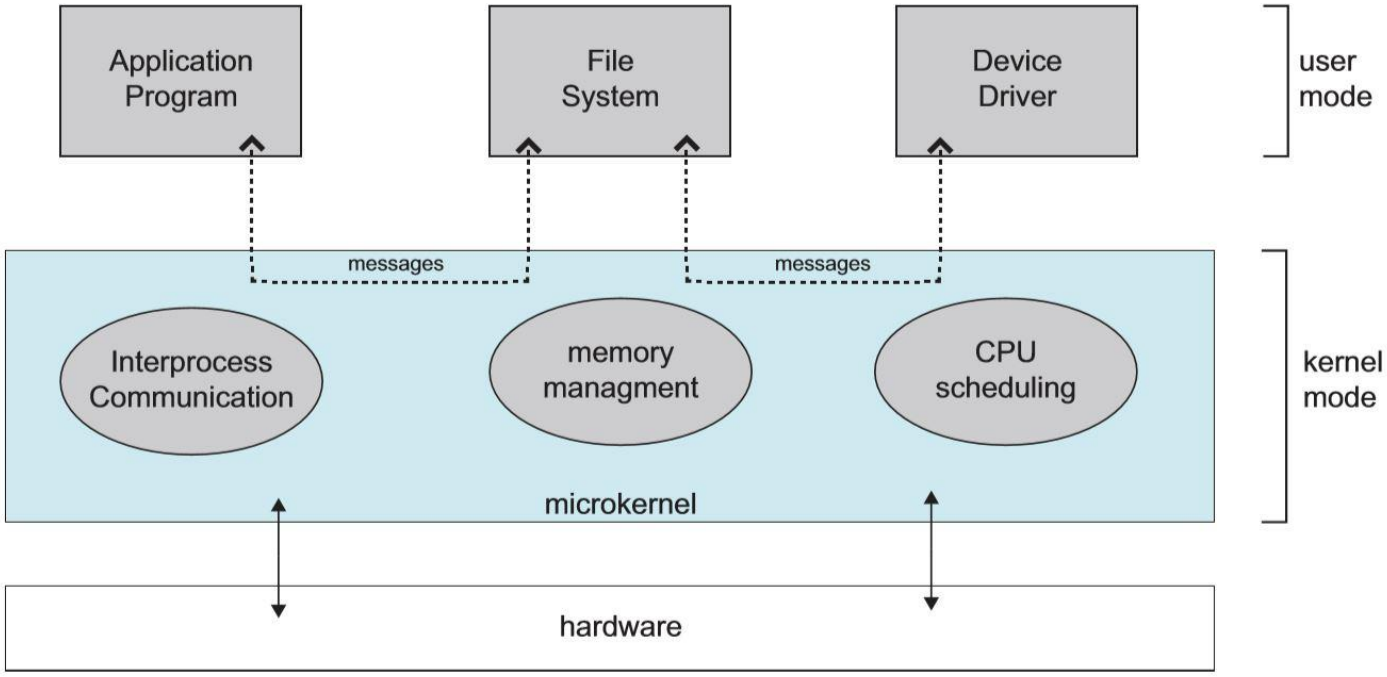
- Layered Approach - With proper hardware support, operating systems can be broken into pieces. The operating system can then retain much greater control over the computer and over the applications that make use of that computer. Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems.
- Under a top-down approach, the overall functionality and features are determined and are separated into components. Information hiding is also important, because it leaves programmers free to implement the low-level routines as they see fit.
- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer *N*) is the user interface.



A layered operating system.

Micro-Kernel approach

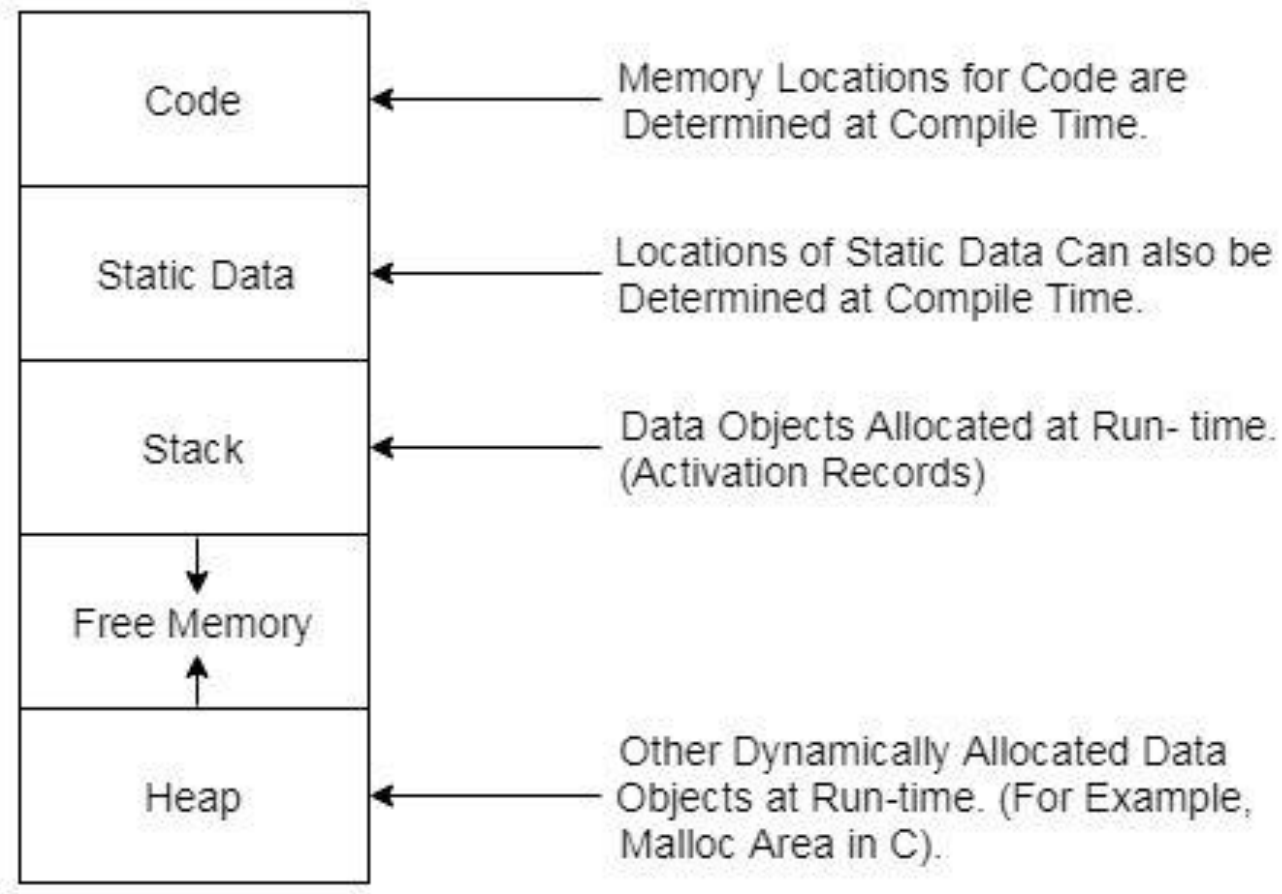
In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach. This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel.



One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.

Fork

- Requirement of Fork command
  - In number of applications specially in those where work is of repetitive nature, like web server i.e. with every client we have to run similar type of code. Have to create a separate process every time for serving a new request.
  - So it must be a better solution that instead to creating a new process every time from scratch we must have a short command using which we can do this logic.
- Idea of fork command
  - Here fork command is a system command using which the entire image of the process can be copied and we create a new process, this idea help us to complete the creation of the new process with speed.
  - After creating a process, we must have a mechanism to identify weather in newly created process which one is child and which is parent.
- Implementation of fork command
  - In general, if fork return 0 then it is child and if fork return 1 then it is parent, and then using a programmer level code we can change the code of child process to behave as new process
- Advantages of using fork commands
  - Now it is relatively easy to create and manage similar types of process of repetitive nature with the help of fork command.
- Disadvantage
  - To create a new process by fork command we have to do system call as, fork is system function
    - Which is slow and time taking
    - Increase the burden over Operating System
  - Different image of the similar type of task have same code part which manes we have the multiple copy of the same data waiting the main memory.



So, in general 2^n times a statement will get printed, where n is the number of times fork() system call was executed. 2^n-1 Child processes and 1 parent process.

The number of child processes created will be 2^n-1.

Q Fork is **(ISRO 2008)**

a) the creation of a new job
b) the dispatching of a task
c) increasing the priority of a task
d) the creation of a new process

Ans. D

Q A process executes the following code **(GATE - 2008) (2 Marks)**

```
for (i=0; i<n; i++)
fork();
```

a) n **b)** $(2^n) - 1$ **c)** 2^n **d)** $(2^{n+1}) - 1$

ANSWER B

Q a process ecexutes the following segment of code:

```
fork(i=1;i<=n;i++)
```

fork();
the number of new processes created is **(GATE-2004) (1 Marks)**
a) n **b)** (n(n+1))/2 **c)** 2^n-1 **d)** 3^n-1
Answer: (C)

Q A process executes the code
fork();
fork();
fork();
the total number of child processes created is **(GATE - 2012) (1 Marks)**
a) 3 **b)** 4 **c)** 7 **d)** 8
Answer: (C)

Q The following C program is executed on a Unix/Linux system:

```
#include <unistd.h>
int main ()
{
    int i ;
    for (i=0; i<10; i++)
        if (i%2 == 0) fork ( ) ;
    return 0 ;
}
```

The total number of child processes created is _____(**GATE-2019) (1 Marks)**
Answer: 31

Q

```
if (fork() == 0)
{
    a = a + 5;
    printf ("%d, %d /n", a, &a);
}
else
{
    a = a –5;
    printf("%d, %d /n", a, &a);
}
```

Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE? **(GATE-2005) (2 Marks)**

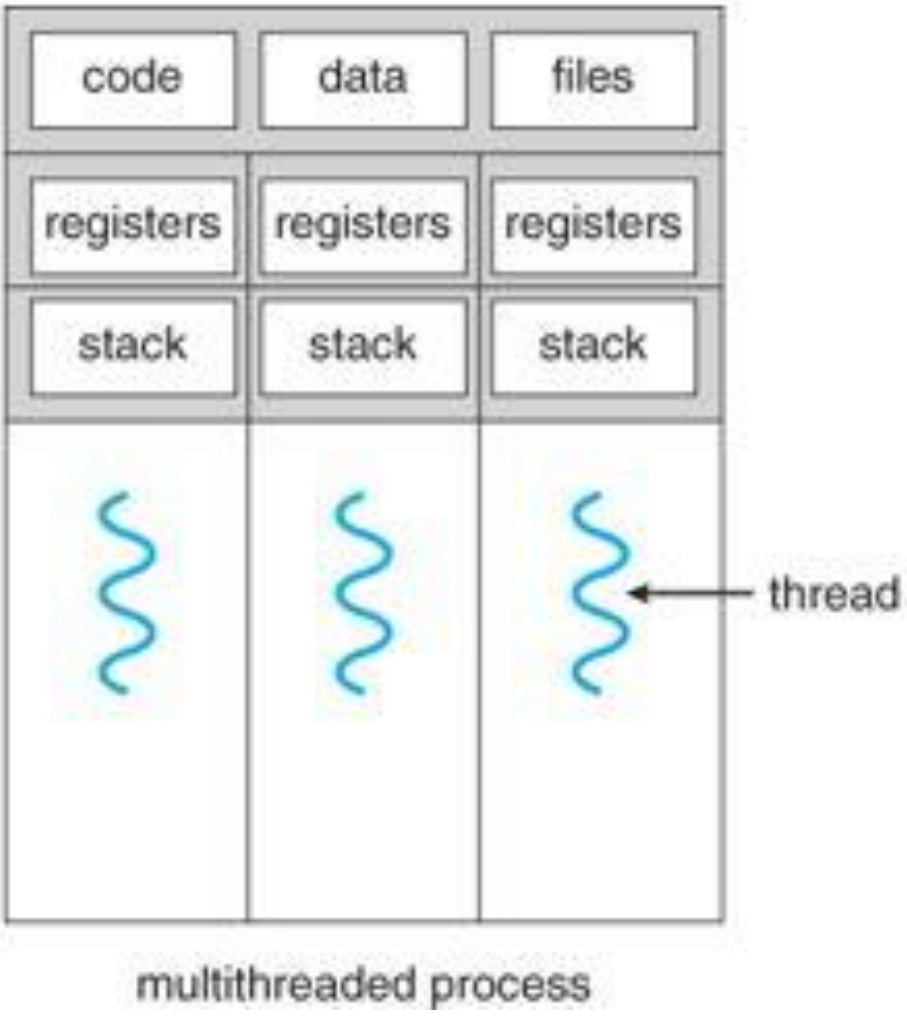
a) u = x + 10 and v = y
b) u = x + 10 and v != y
c) u + 10 = x and v = y
d) u + 10 = x and v != y
Ans. C (Geeksforgeeks) / D (Made Easy book)

Q What is the output of the following program?

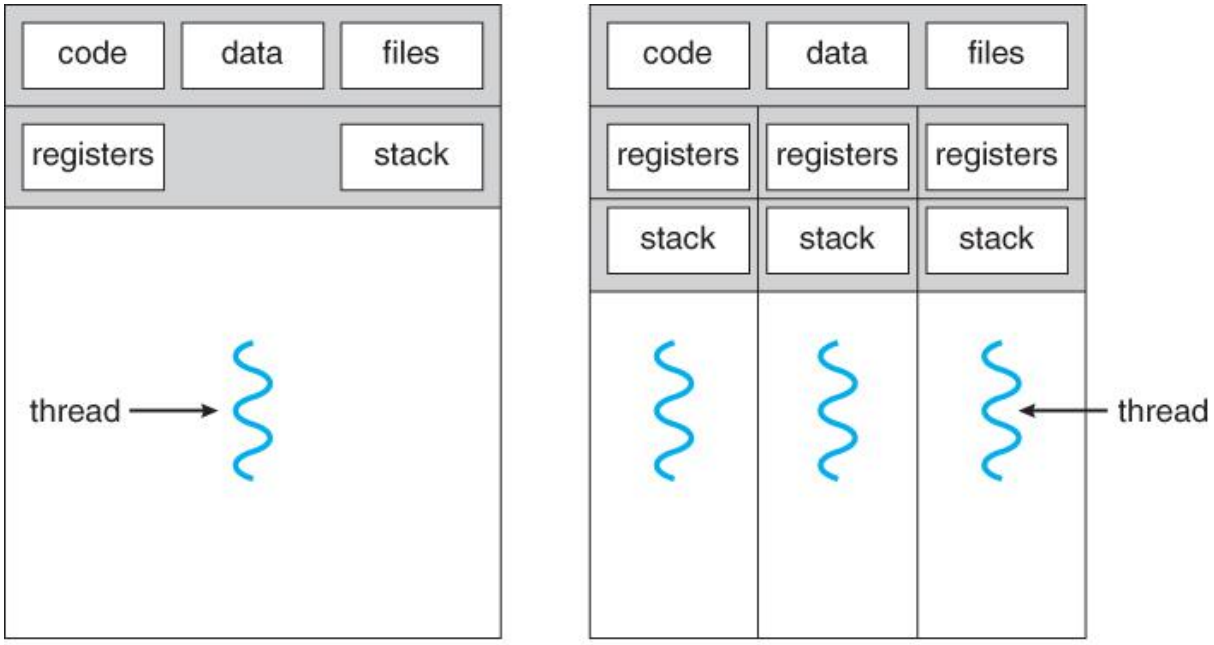
```
main( )
{
    int a = 10;
    if ((fork ( ) == 0))
        a++;
    printf (“%dn”, a );
}
```

a) 10 and 11 **b)** 10 **c)** 11 **d)** 11 and 11
Ans. A

Threads



- A ***thread*** is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.)
- Traditional (heavyweight) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.
- multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.



Motivation

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.
- This is particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking.
- For example in a word processor, a background thread may check spelling and grammar while a foreground thread processes user input (keystrokes), while yet a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.
- Another example is a web server - Multiple threads allow for multiple requests to be satisfied simultaneously, without having to service requests sequentially or to fork off separate processes for every incoming request. (The latter is how this sort of thing was done before the concept of threads was developed. A daemon would listen at a port, fork off a child for every incoming request to be processed, and then go back to listening to the port.)

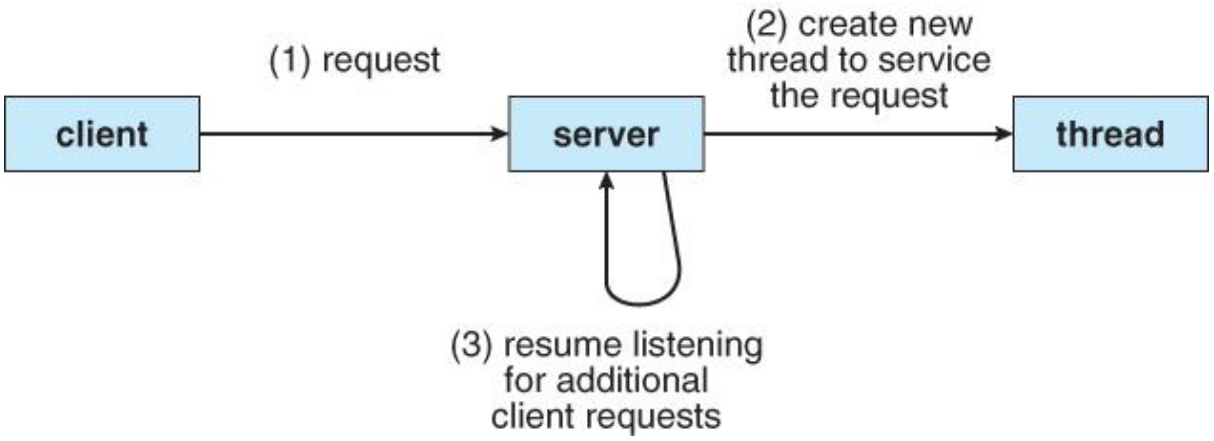


Figure 4.2 - Multithreaded server architecture

There are four major categories of benefits to multi-threading:

1. Responsiveness - One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
2. Resource sharing - By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
3. Economy - Creating and managing threads (and context switches between them) is much faster than performing the same tasks for processes.
4. Scalability, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors. (Note that single threaded processes can still benefit from multi-processor architectures when there are multiple processes contending for the CPU, i.e. when the load average is above some certain threshold.)

Multithreading Models

- There are two types of threads to be managed in a modern system: User threads and kernel threads.
- User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
- Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.
- In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

Many-To-One Model

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- However, if a blocking system call is made, then the entire process blocks, even if the other user threads would otherwise be able to continue.
- Because a single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.
- Green threads for Solaris implement the many-to-one model in the past, but few systems continue to do so today.

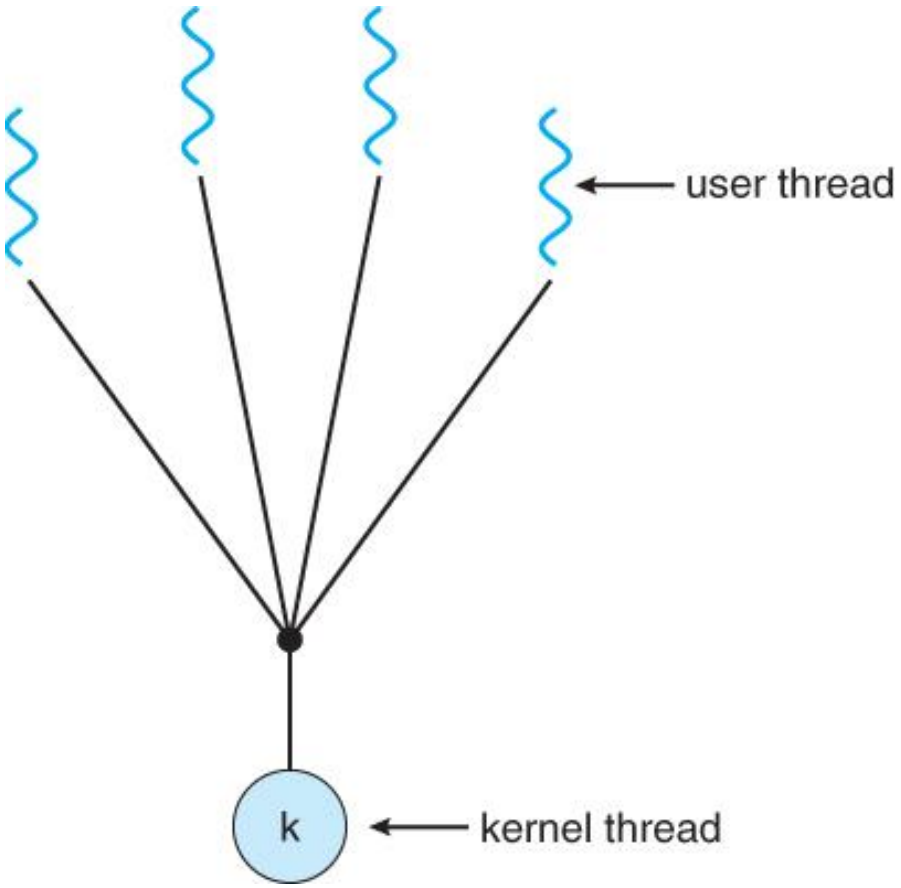


Figure 4.5 - Many-to-one model

One-To-One Model

- The one-to-one model creates a separate kernel thread to handle each user thread.
- One-to-one model overcomes the problems listed above involving blocking system calls and the splitting of processes across multiple CPUs.
- However, the overhead of managing the one-to-one model is more significant, involving more overhead and slowing down the system.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.

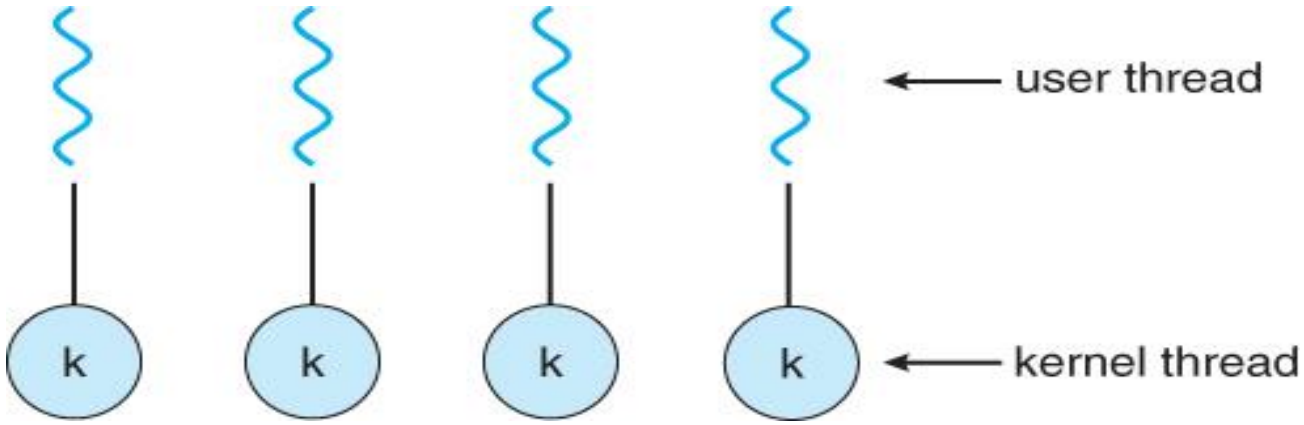


Figure 4.6 - One-to-one model

Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors.

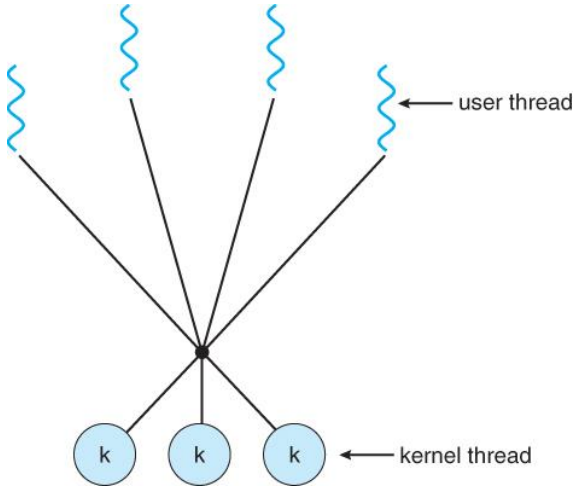


Figure 4.7 - Many-to-many model

Q A thread is usually defined as a "light weight process" because an operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is TRUE? **(GATE - 2011) (1 Marks)**
(A) On per-thread basis, the OS maintains only CPU register state
(B) The OS does not maintain a separate stack for each thread
(C) On per-thread basis, the OS does not maintain virtual memory state
(D) On per-thread basis, the OS maintains only scheduling and accounting information
Answer: (C)

Q Which of the following is/are shared by all the threads in a process? **(GATE - 2017) (1 Marks)**
I. Program Counter **II.** Stack **III.** Address space **IV.** Registers
(A) I and II only **(B)** III only **(C)** IV only **(D)** III and IV only
Answer: (B)

Q Threads of a process share **(GATE - 2017) (1 Marks)**
(A) global variables but not heap **(B)** heap but not global variables
(C) neither global variables nor heap **(D)** both heap and global variables
Answer: (D)

Q A thread is a light weight process. In the above statement, weight refers to **(NET-DEC-2012)**
(A) time **(B)** number of resources
(C) speed **(D)** All the above
Answer: (B)

Q Let the time taken to switch between user and kernel modes of execution be t_1 while the time taken to switch between two processes be t_2. Which of the following is TRUE? (**GATE-2011) (1 Marks)**
(A) t1 > t2 **(B)** t1 = t2
(C) t1 < t2 **(D)** nothing can be said about the relation between t1 and t2
Answer: (C)

Q Let the time taken to switch between user mode and kernel mode of execution be T1 while time taken to switch between two user processes be T2. Which of the following is correct? **(NET-JUNE-2013)**
a) T1 < T2 **b)** T1 > T2
c) T1 = T2 **d)** Nothing can be said about the relation between T1 and T2.

Q Which one of the following is FALSE? **(GATE - 2014) (1 Marks)**
(A) User level threads are not scheduled by the kernel.
(B) When a user level thread is blocked, all other threads of its process are blocked.
(C) Context switching between user level threads is faster than context switching between kernel level threads.
(D) Kernel level threads cannot share the code segment
Answer: (D)

Q Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE? **(GATE - 2007) (1 Marks)**
A) Context switch time is longer for kernel level threads than for user level threads.
B) User level threads do not need any hardware support.
C) Related kernel level threads can be scheduled on different processors in a multi-processor system
D) Blocking one kernel level thread blocks all related threads
ANSWER D

Q Consider the following statements with respect to user-level threads and kernel supported threads
i. context switch is faster with kernel-supported threads
ii. for user-level threads, a system call can block the entire process
iii. Kernel supported threads can be scheduled independently
iv. User level threads are transparent to the kernel
Which of the above statements are true? **(GATE-2004) (1 Marks)**
(A) (ii), (iii) and (iv) only
(B) (ii) and (iii) only
(C) (i) and (iii) only
(D) (i) and (ii) only
Answer: (A)

Q One of the disadvantages of user level threads compared to Kernel level threads is **(NET-JAN-2017)**
a) If a user level thread of a process executes a system call, all threads in that process are blocked.
b) Scheduling is application dependent.
c) Thread switching doesn't require kernel mode privileges.
d) The library procedures invoked for thread management in user level threads are local procedures.
Answer: (A)

Q User level threads are threads that are visible to the programmer and are unknown to the kernel. The operating system kernel supports and manages kernel level threads. Three different types of models relate user and kernel level threads. Which of the following statements is/are true? **(NET-NOV-2017)**

(a) (i) The Many - to - one model maps many user threads to one kernel thread
(ii) The one - to - one model maps one user thread to one kernel thread
(iii) The many - to - many model maps many user threads to smaller or equal kernel threads

(b) (i) Many - to - one model maps many kernel threads to one user thread
(ii) One - to - one model maps one kernel thread to one user thread
(iii) Many - to - many model maps many kernel threads to smaller or equal user threads

A) (a) is true; (b) is false
B) (a) is false; (b) is true
C) Both (a) and (b) are true
D) Both (a) and (b) are false

Answer: A