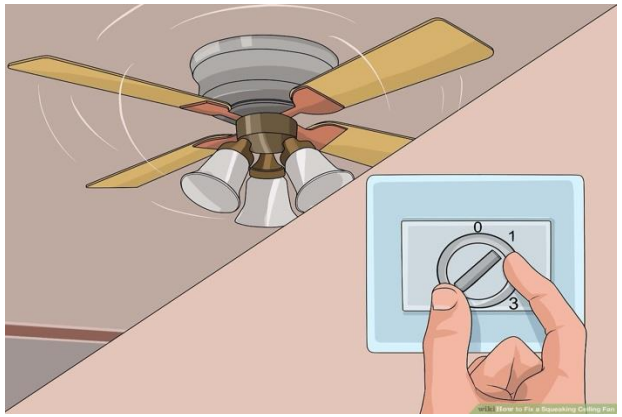
What is Operating System

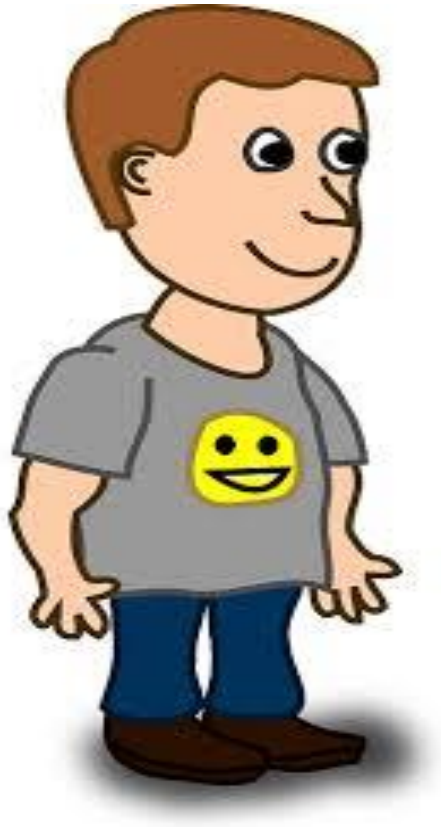
- Whatever used as an interface between the user and the core machine is OS.
  - E.g. steering of car, switch of the fan etc., Buttons over electronic devices.



- Question comes why we need an operating system?
  - To enable everybody to use h/w In a convenient and efficient manner

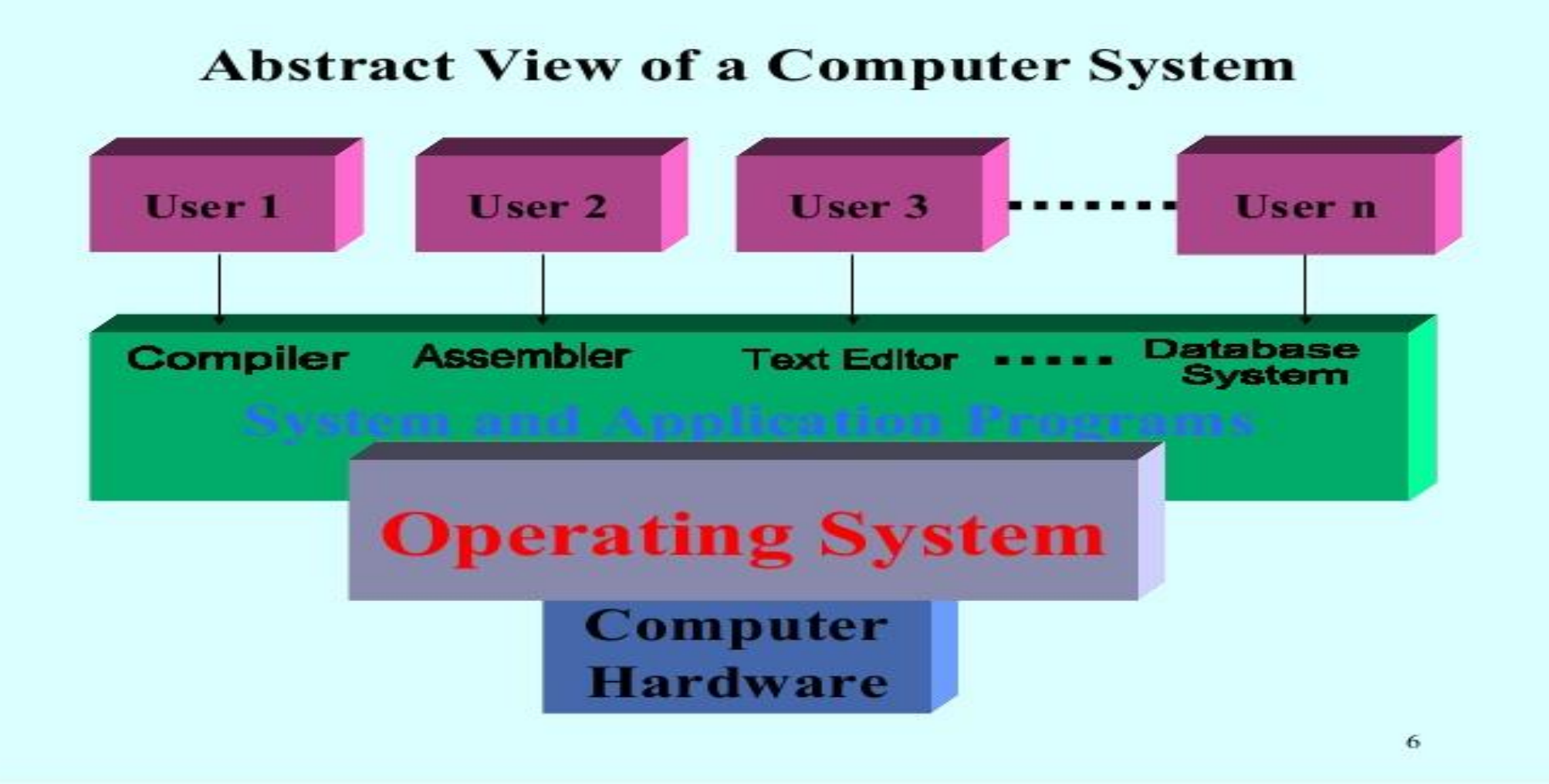
Definition of Operating System

- **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

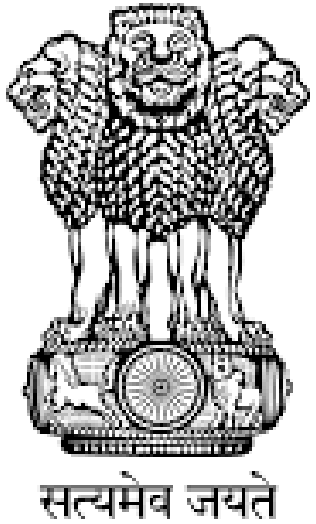


- **Resource Manager/Allocator** - Manage system resources in an unbiased fashion both h/w (mainly CPU time, memory, system buses) & s/w (access, authorization, semaphores) and provide functionality to application programs.
- OS controls and coordinates the use of resources among various application programs.

- OS provides platform on which other application programs can be installed, provides the environment within which programs are executed.



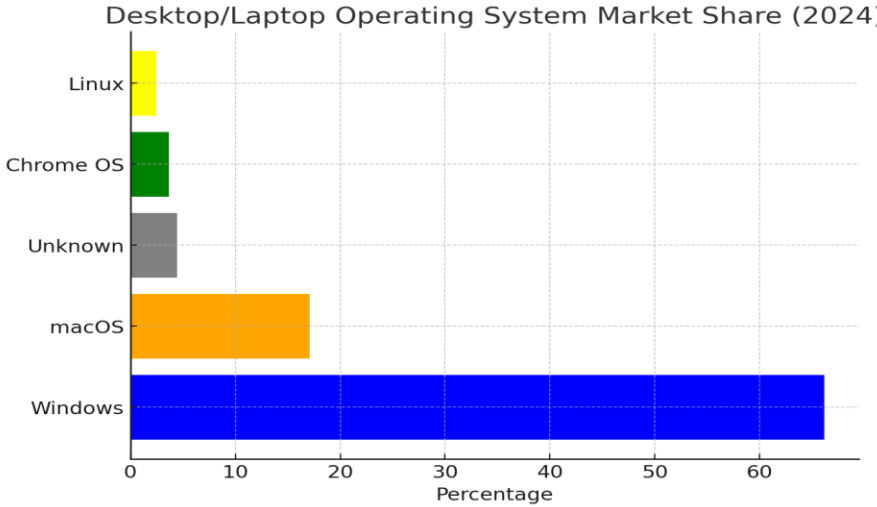
What is Operating System

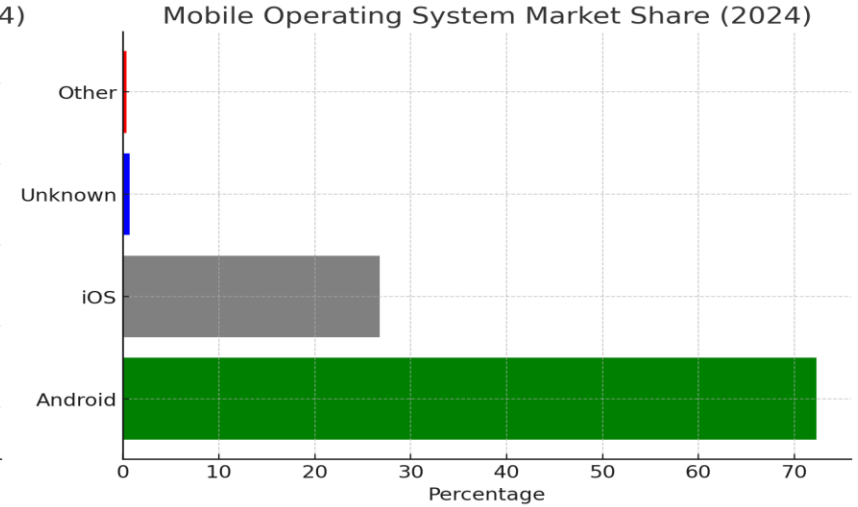


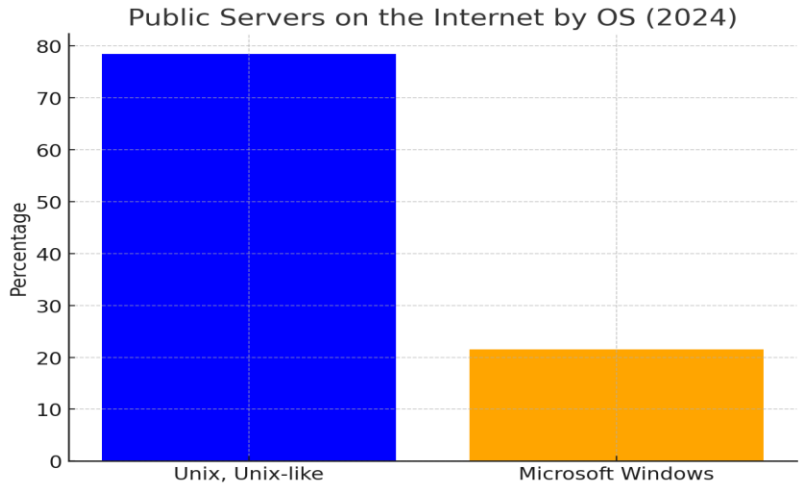
सत्यमेव जयते

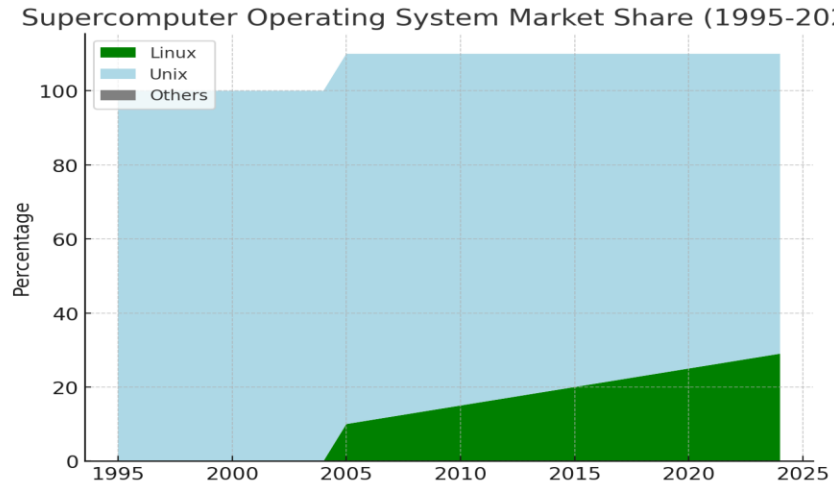
भारत सरकार
GOVERNMENT
OF INDIA

- **Windows OS**:
  - Windows 10: Popular version of Microsoft's OS.
  - Windows 11: Latest version with new features.
- **Mac OS**:
  - macOS Big Sur: Older version with improved performance.
  - macOS Monterey: Latest version with new features.
- **Linux OS**:
  - Ubuntu: Popular Linux distribution for desktops and servers.
  - Fedora: Known for cutting-edge features.
  - Debian: Highly stable Linux distribution.
- **Unix**:
  - AIX: IBM’s version of UNIX.
  - HP-UX: Hewlett Packard's version of UNIX.
- **Mobile OS**:
  - Android: Developed by Google for smartphones and tablets.
  - iOS: Apple's OS for iPhones and iPads.
- **Real-Time Operating Systems (RTOS)**:
  - VxWorks: Used in embedded systems and real-time applications.
  - RTEMS: Open-source RTOS for embedded systems.


Desktop/Laptop Operating System Market Share (2024)
Linux
Chrome OS
Unknown
macOS
Windows
0
10
20
30
40
50
60
Percentage


Mobile Operating System Market Share (2024)
Other
Unknown
iOS
Android
0
10
20
30
40
50
60
70
Percentage


Public Servers on the Internet by OS (2024)
Percentage
0
10
20
30
40
50
60
70
80
Unix, Unix-like
Microsoft Windows


Supercomputer Operating System Market Share (1995-2024)
Linux
Unix
Others
Percentage
0
20
40
60
80
100
1995
2000
2005
2010
2015
2020
2025

Q. What is an operating system? [Asked in Wipro NLTH 2021]

(A) interface between the hardware and application programs

(B) collection of programs that manages hardware resources

(C) system service provider to the application programs

(D) all of the mentioned

Answer: d

Q. Which is the Linux operating system? **[Asked in L&T InfoTech]**

(A) Private operating system

(B) Windows operating system

(C) Open-source operating system

(D) None of these

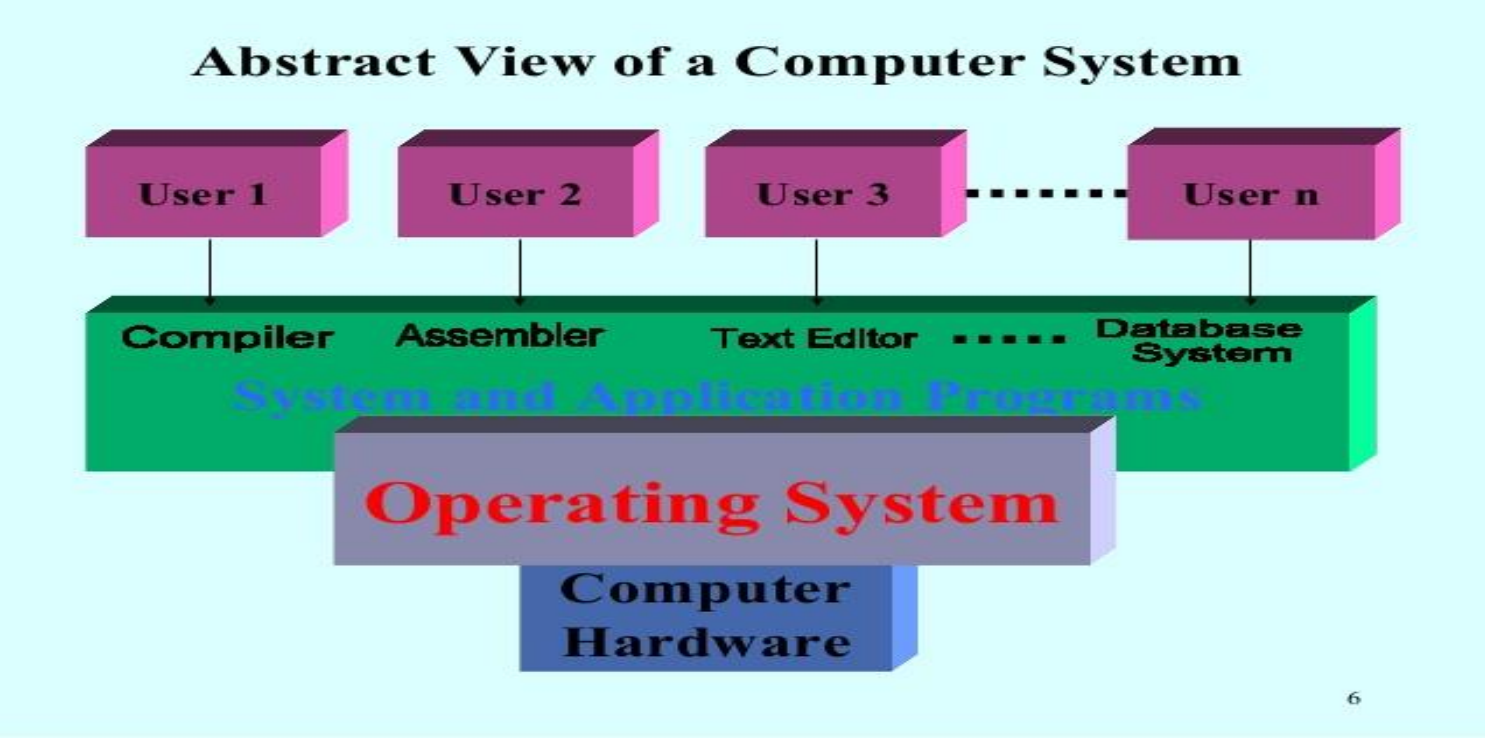
Answer: (c) Open-source operating system

- **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.
  - **Primary goals** (Convenience / user friendly)
  - **Secondary goals** (Efficiency (Using resources in efficient manner) / Reliability / maintainability)

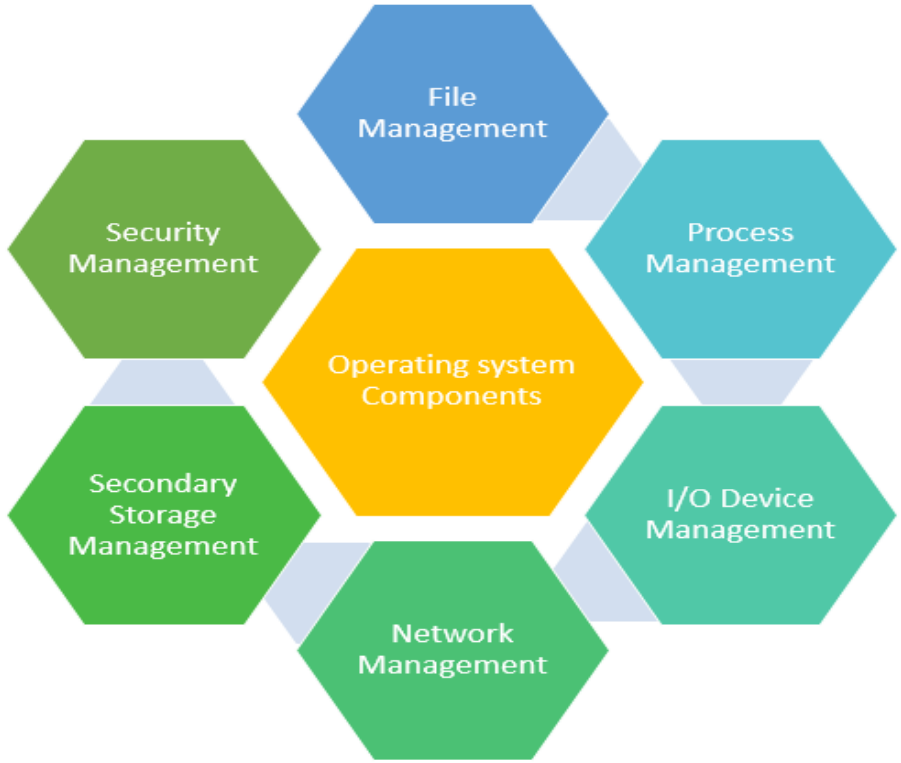
सबका साथ सबका विकास

सत्यमेव जयते

Goverment Of India

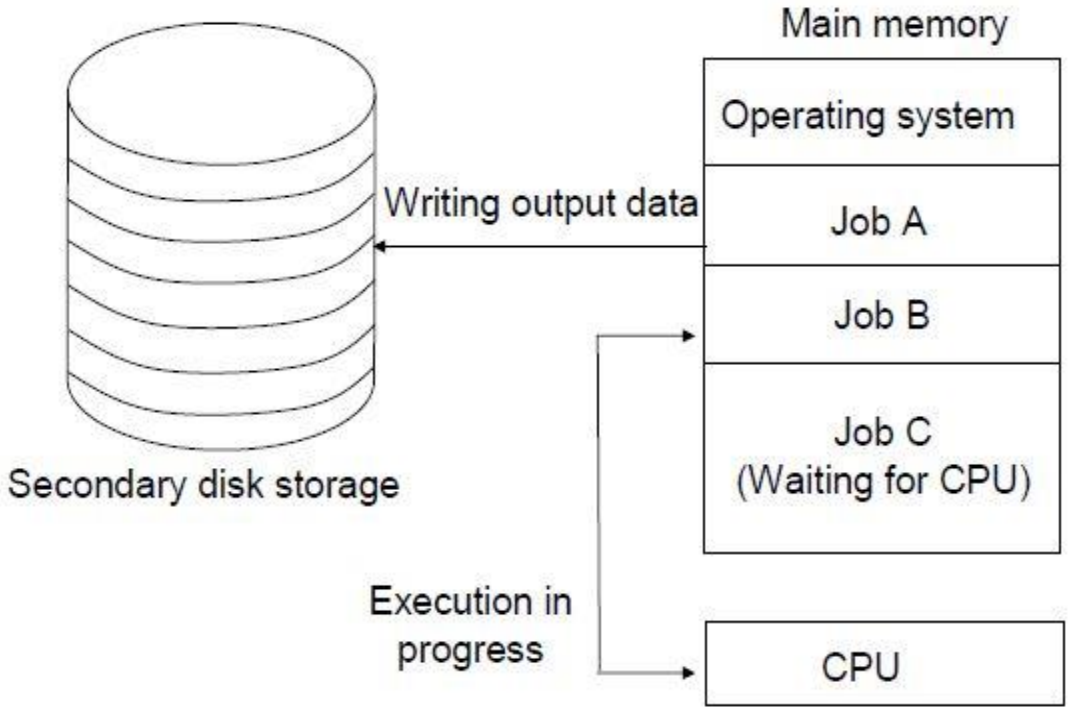
Ministries	Departments
58	93

<u>Primary goals</u> (Convenience / user friendly)



Multiprogramming Operating System

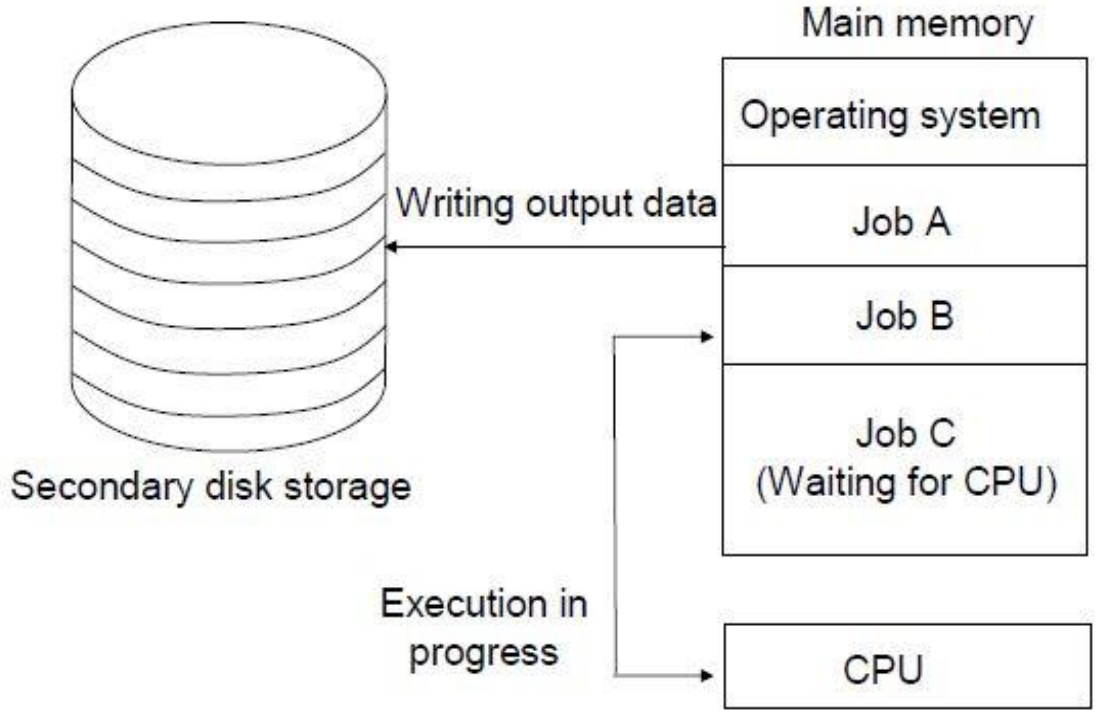
- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. The basic idea of multiprogramming operating system is it keeps several jobs in main memory simultaneously.
- The jobs are kept initially on the disk in the **job pool**. This pool consists of all processes 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.



Show must go on

- 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.



Processor किसी के लिए wait नहीं करेगा

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. In a multiprogramming environment _____ **[Asked in Wipro NLTH 2020]**

(A) the processor executes more than one process at a time

(B) the programs are developed by more than one person

(C) more than one process resides in the memory

(D) a single user can execute many programs at the same time

Answer: C

Q. What is the objective of multiprogramming? **[Asked in Hexaware 2020]**

(A) Have a process running at all time

(B) Have multiple programs waiting in a queue ready to run

(C) To increase CPU utilization

(D) None of the mentioned

Answer: C

Q. The systems which allow only one process execution at a time, are called _____ **[Asked in Tech Mahindra 2020]**

(A) uniprogramming systems

(B) uniprocessing systems

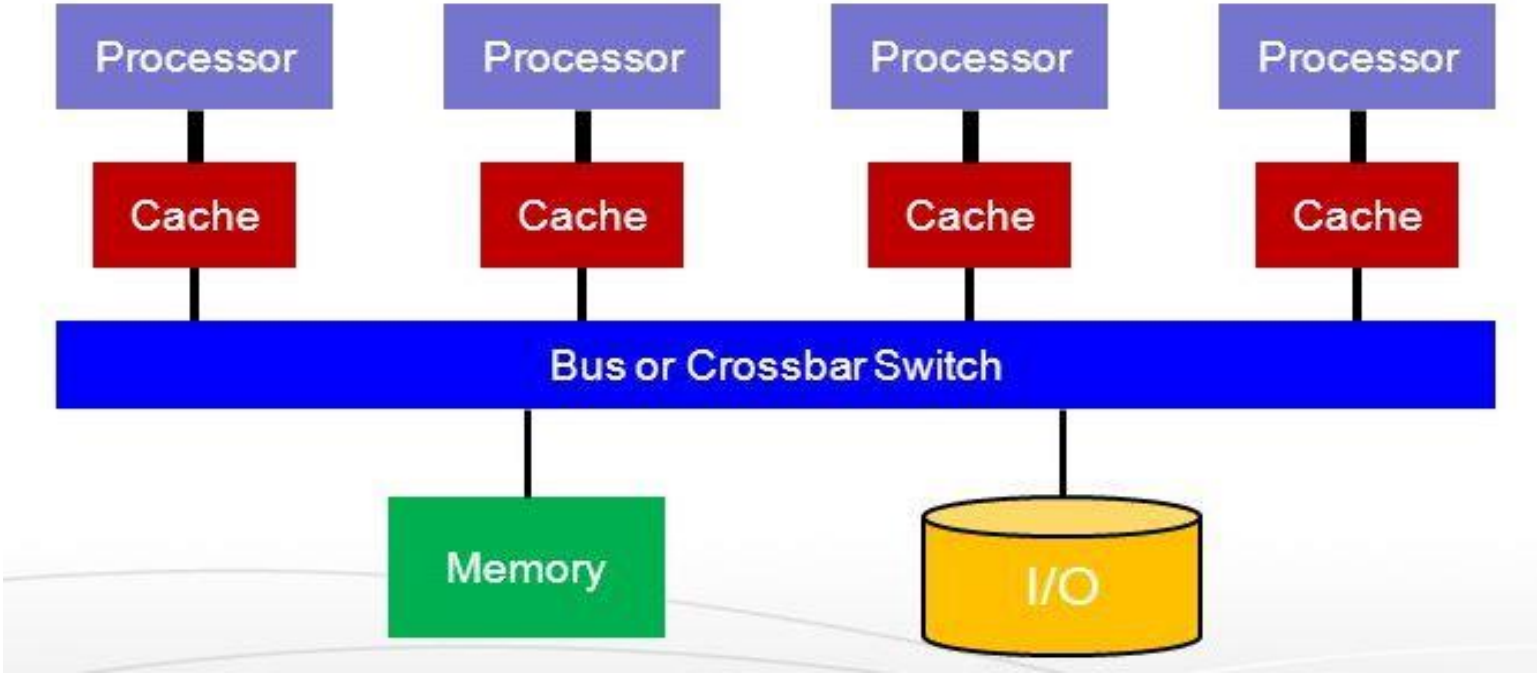
(C) unitasking systems

(D) none of the mentioned

Answer: B

Multiprocessing Operating System/ tightly coupled system

- **Definition and Purpose**: A multiprocessing operating system supports the simultaneous execution of multiple processes by utilizing two or more CPUs within a single computer system. The goal is to enhance processing speed and reliability.
- **How It Works**: The system distributes processes across multiple CPUs, allowing tasks to be executed in parallel. This parallelism increases efficiency, reduces processing time, and improves system performance.
- **Advantages**:
  - **Increased Throughput**: More tasks can be processed in a shorter time due to parallel execution.
  - **Improved Reliability**: If one CPU fails, others can continue to work, maintaining system functionality.
  - **Better Resource Utilization**: Multiple CPUs ensure that system resources are used more effectively, reducing idle time.



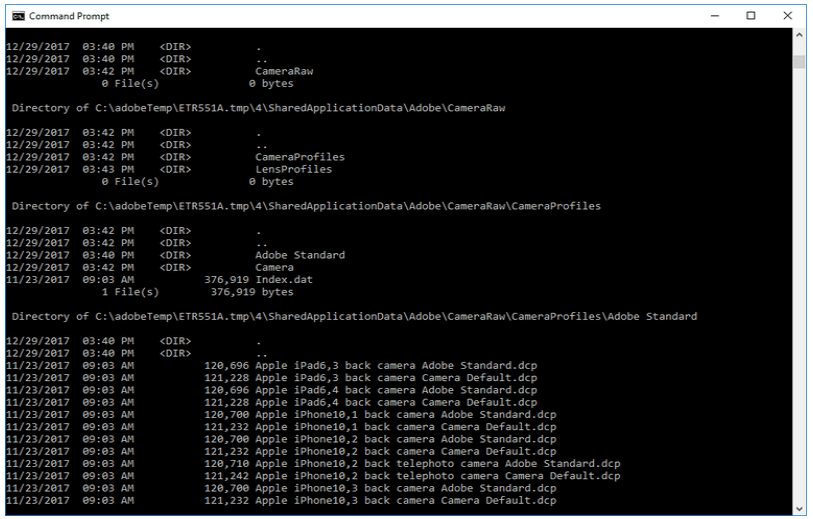
Real time Operating system

- **Definition and Purpose**: An RTOS is designed to process data and execute tasks within a strict time constraint, providing predictable and deterministic responses to real-time events. It is essential for applications requiring precise timing and reliability, such as embedded systems and industrial control.
- **Mechanism**: RTOS prioritizes tasks based on their urgency and importance, using real-time scheduling algorithms. It ensures that high-priority tasks are executed immediately, preempting lower-priority tasks if necessary, to meet critical deadlines.
- **Advantages**:
  - **Predictability**: Ensures consistent, timely responses to real-time events.
  - **Reliability**: Provides stable and dependable performance in time-sensitive applications.
  - **Resource Management**: Efficiently manages system resources to meet the demands of real-time applications, minimizing latency and maximizing throughput.



User and Operating-System Interface

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



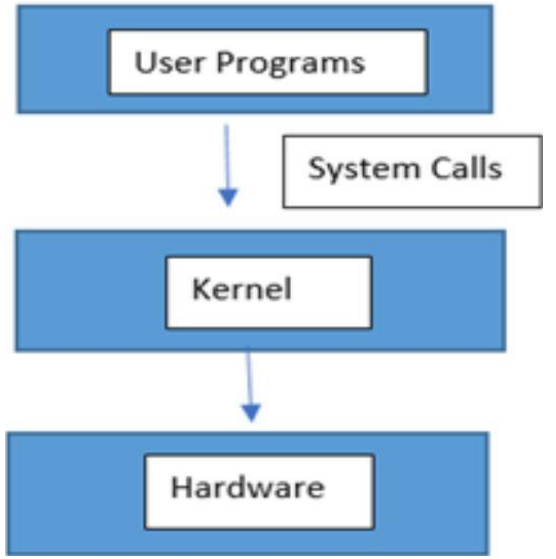
Feature	Command-Line Interface (CLI)	Graphical User Interface (GUI)	Touchscreen Interface
Interface Method	Command Interpreters	Mouse-based window and menu system	Touch-based interaction
User Interaction	Users enter text commands	Users click on icons and menus	Users make gestures on the screen
Typical Users	System administrators, power users	General users, beginners	Mobile users, tablet users
Advantages	Efficient for experienced users, faster access to tasks	User-friendly, visually intuitive	Intuitive for mobile use, supports gestures
Disadvantages	Steep learning curve, less user-friendly	Can be slower for advanced users, limited functionality compared to CLI	May lack precision, limited to touch-compatible devices

System call

- 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.

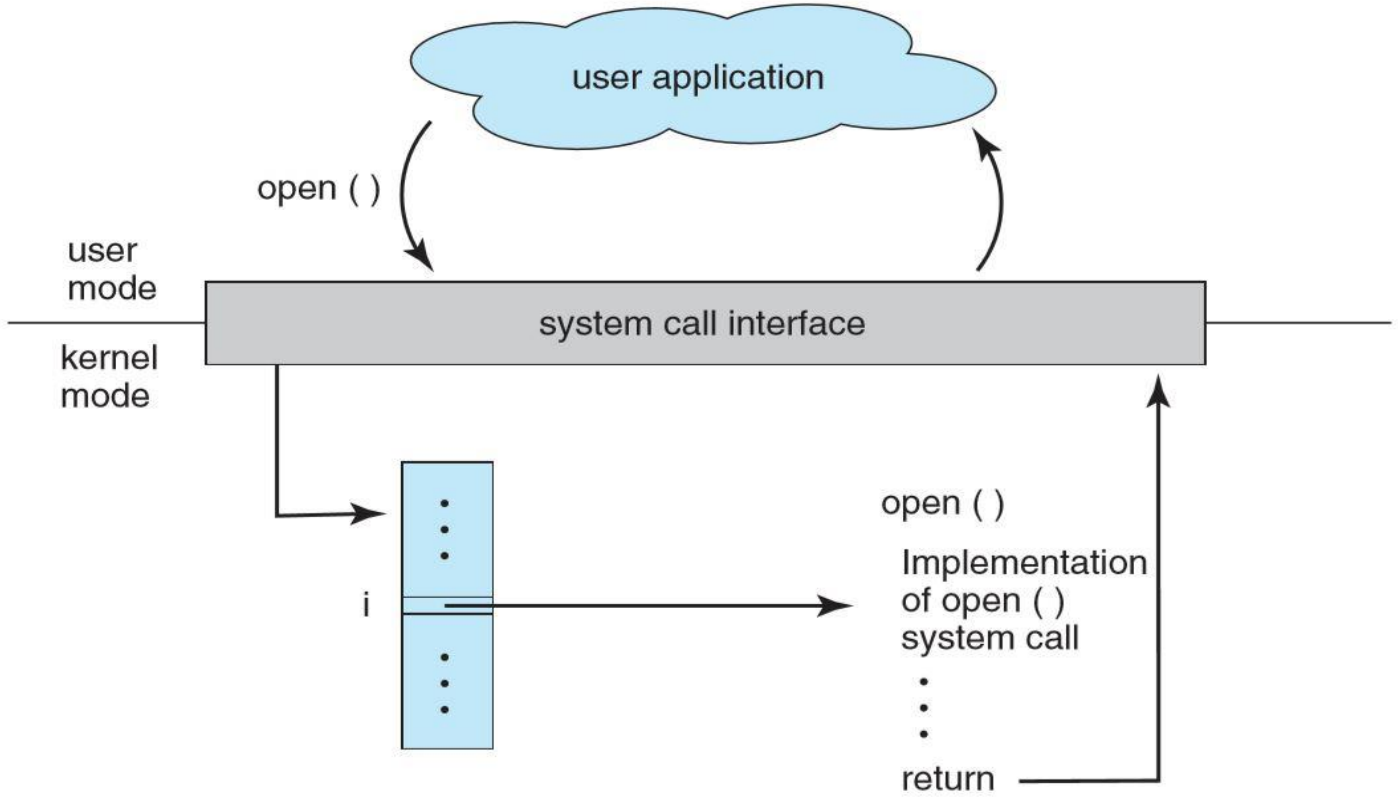
- System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++.

- 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.



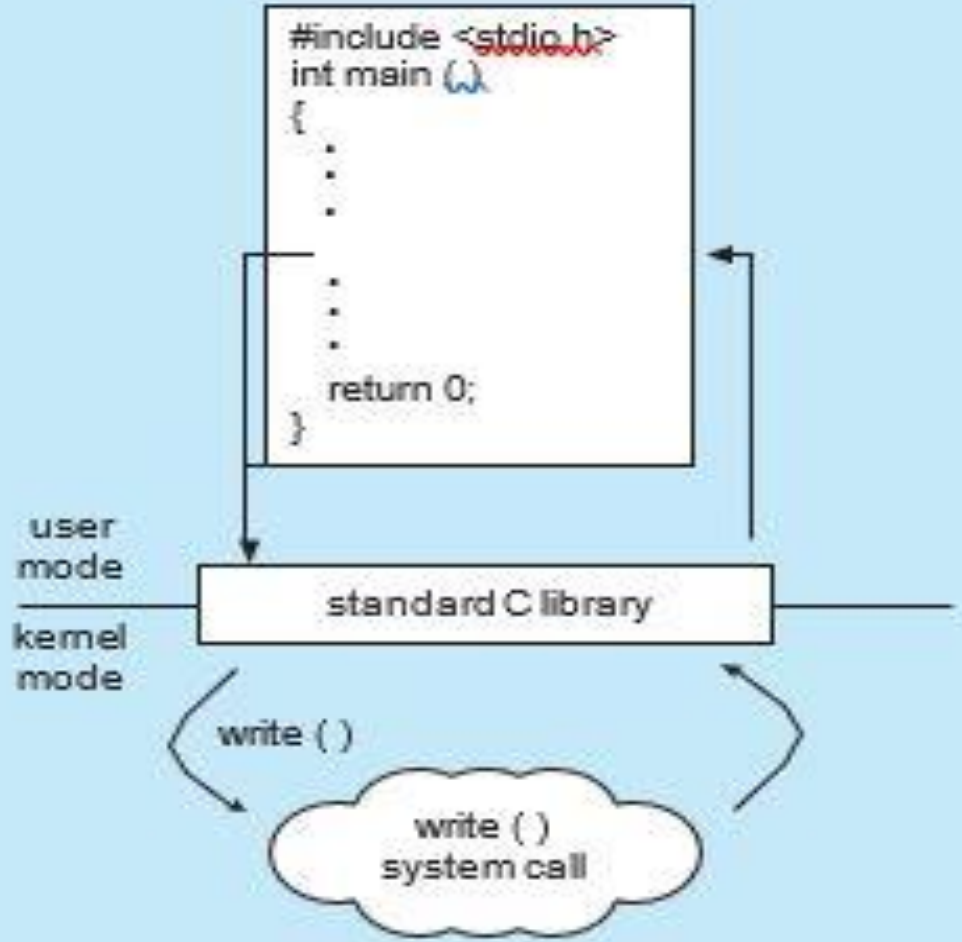
- Three of the most common APIs available to application programmers are the
  - Windows API for Windows systems
  - POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and Mac OS X)
  - The Java API for programs that run on the Java virtual machine.

- 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:



Q. To access the services of the operating system, the interface is provided by the _____
[Asked in Hexaware]

(A) Library

(B) System calls

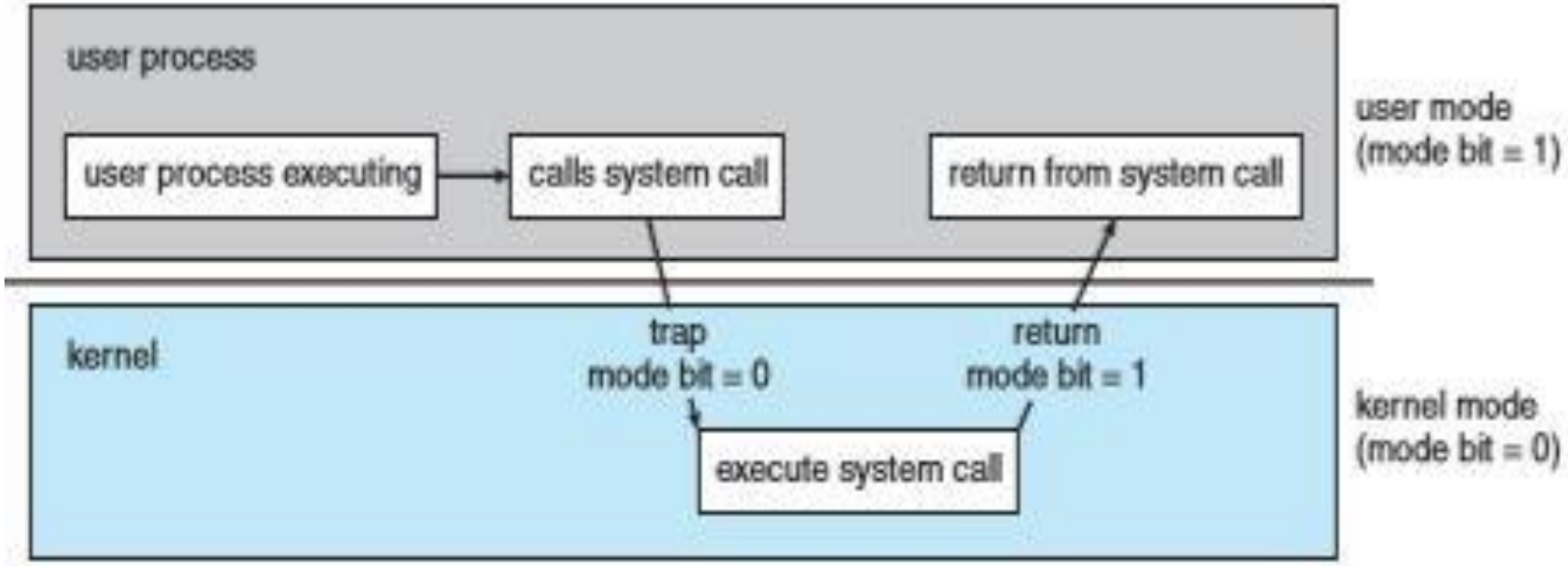
(C) Assembly instructions

(D) API

Answer: b

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.



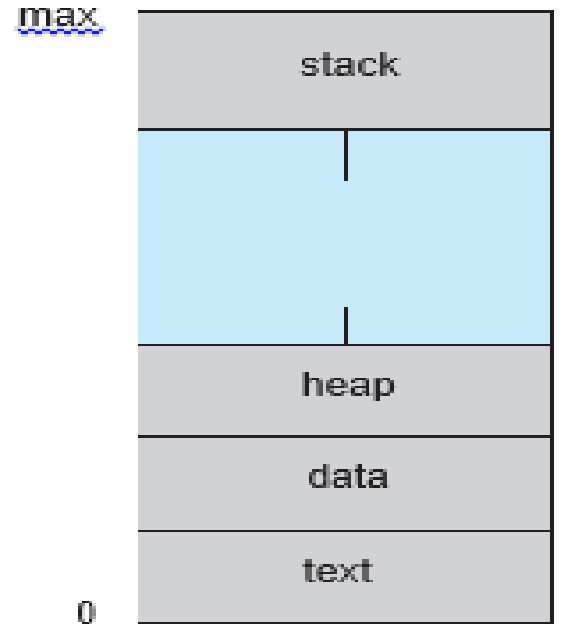
User Mode

Kernel Mode

Process

- In general, a process is a program in execution.
- A program is not inherently a process. A program is a passive entity, meaning it is a file containing a list of instructions stored on disk (secondary memory) and is often referred to as an executable file.
- A program becomes a process when the executable file is loaded into main memory, and its Process Control Block (PCB) is created.
- Conversely, a process is an active entity that requires resources like main memory, CPU time, registers, system buses, etc. Even if two processes are associated with the same program, they are considered separate execution sequences and are entirely different processes.
- For instance, if a user has multiple copies of a web browser program running, each copy will be treated as a separate process. Although the text section is the same, 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. A process stack does not contain _____ **[Asked in Cognizant]**

(A) Function parameters

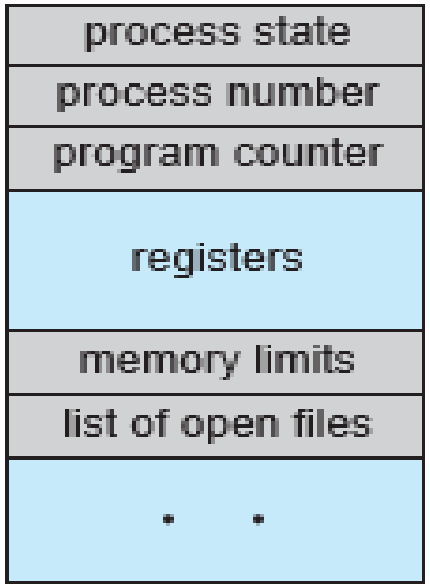
(B) Local variables

(**C)** Return addresses

(D) PID of child process

Answer: B

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.

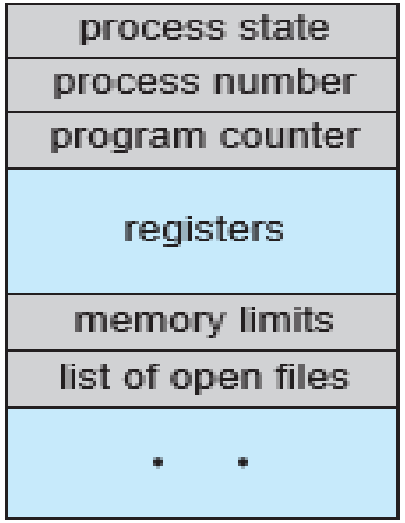
  - **CPU registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when 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.



Q. In operating system, each process has its own _____ **[Asked in Wipro NLTH]**

(A) address space and global variables

(B) open files

(**C)** pending alarms, signals and signal handlers

(**D)** all of the mentioned

Answer: D

Q. The address of the next instruction to be executed by the current process is provided by the _____ **[Asked in L&T InfoTech (LTI)]**

(A) CPU registers

(B) Program counter

(**C)** Process stack

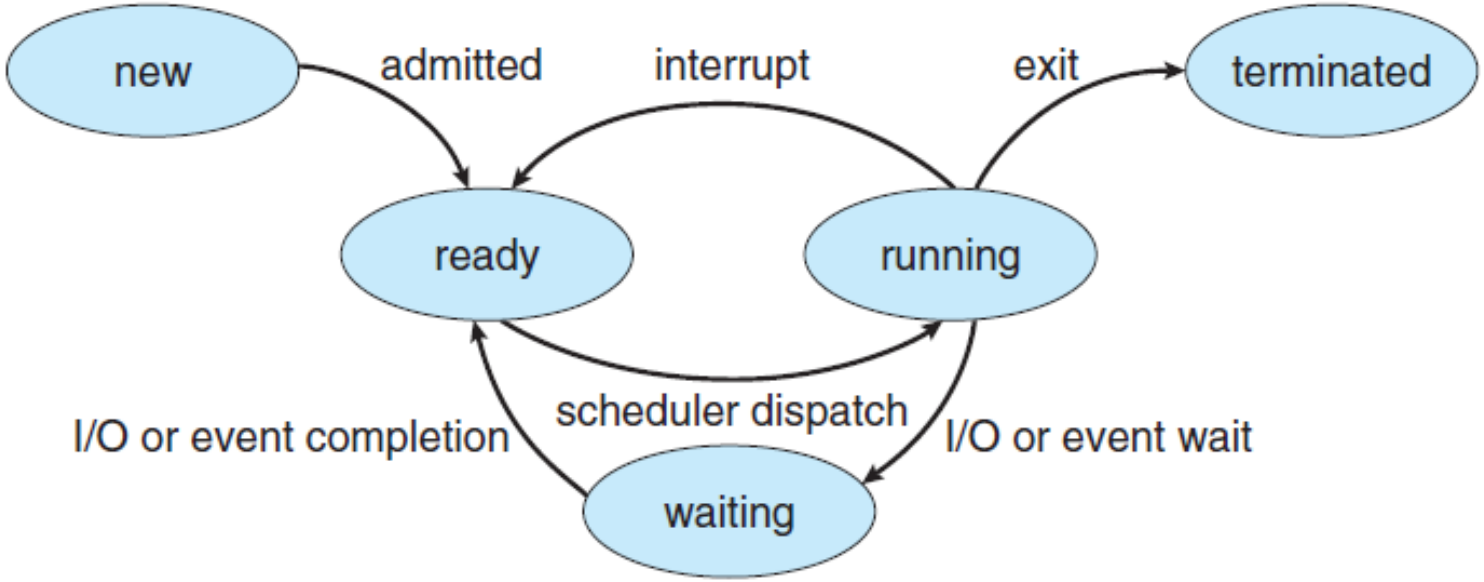
(**D)** Pipe

Answer: B

The Human Life Cycle

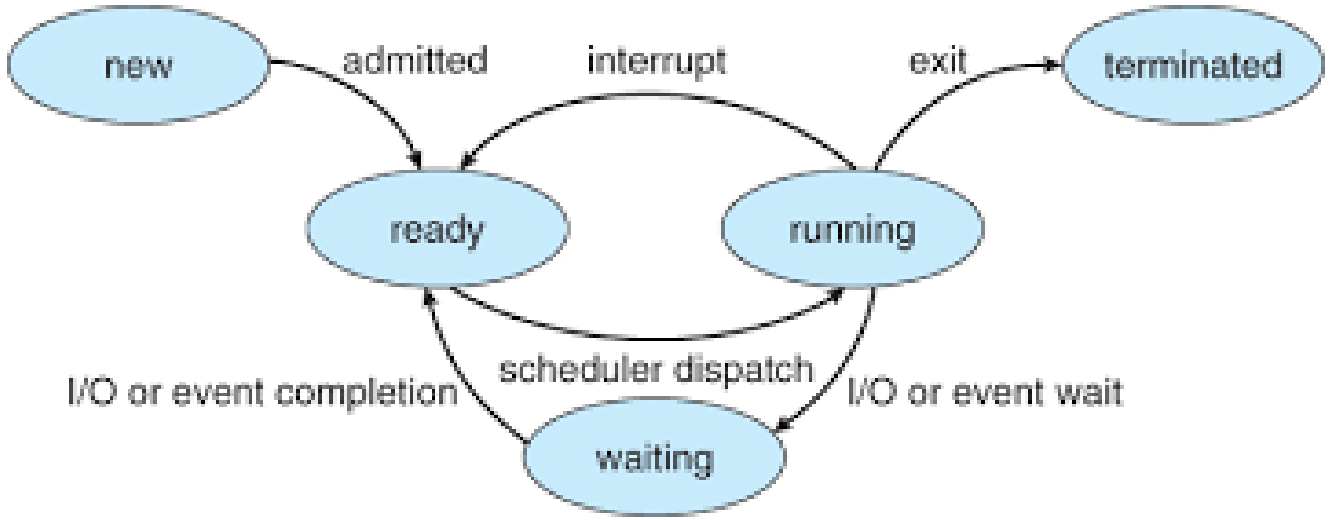
Process States

- 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 What is the minimum and maximum number of processes that can be in the ready, run, and blocked states, if total number of process is n?

	Min	Max
Ready		
Run		
Block		



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.

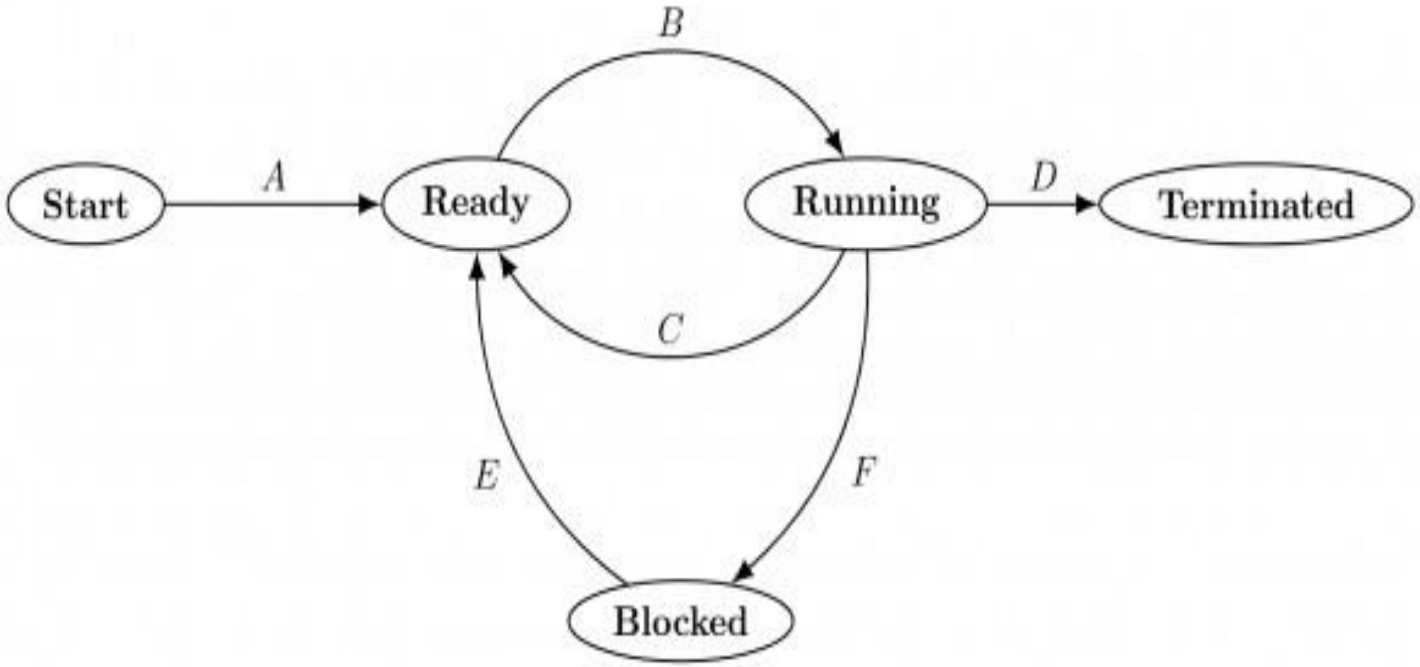
Which of the above statements are TRUE?

a) I and II

b) I and III

c) II and III

d) II and IV



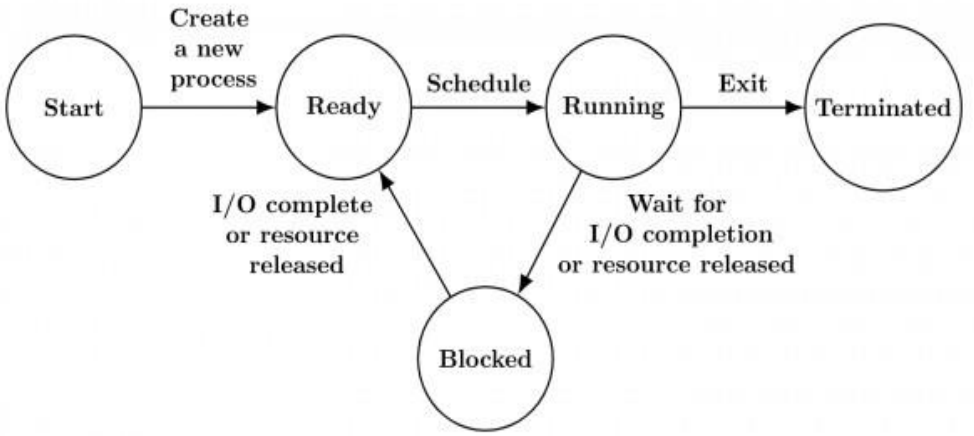
Q The process state transition diagram of an operating system is as given below. Which of the following must be FALSE about the above operating system?

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



Q Match the following:

List – I	List - II
Process state	Reason for transition
a) Ready → Running	i. Request made by the process is satisfied or an event for which it was waiting occurs.
b) Blocked → Ready	ii. Process wishes to wait for some action by another process.
c) Running → Blocked	iii. The process is dispatched.
d) Running → Ready	iv. The process is pre-empted.

	a	b	c	d
a)	iii	i	ii	iv
b)	iv	i	iii	ii
c)	iv	iii	i	ii
d)	iii	iii	ii	i

Q. What is the ready state of a process? **[Asked in Amcat]**

(A) when process is scheduled to run after some execution

(B) when process is unable to run until some task has been completed

(C) when process is using the CPU

(**D)** none of the mentioned

Answer: D

Q. Suppose that a process is in “Blocked” state waiting for some I/O service. When the service is completed, it goes to the _____ **[Asked in Accenture]**

(A) Running state

(B) Ready state

(C) Suspended state

(D) Terminated state

Answer: B

Q. A process can be terminated due to _____ **[Asked in Cognizant]**

(A) normal exit

(B) fatal error

(**C)** killed by another process

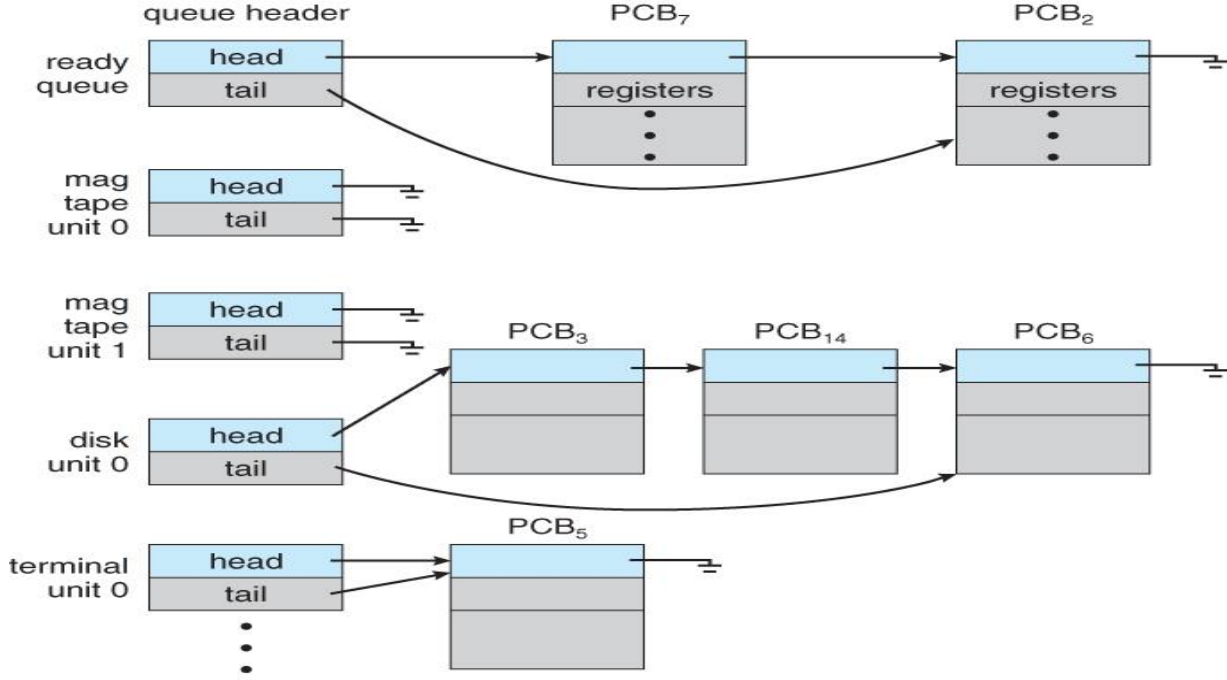
(**D)** all of the mentioned

Answer: D

- **Job Queue**: Contains all processes in the system. It is the initial queue where processes are placed as they enter the system.

- **Ready Queue**: Holds processes that are in main memory and ready to execute. This queue is typically implemented as a linked list, with a ready-queue header containing pointers to the first and last process control blocks (PCBs) in the list. Each PCB has a pointer to the next PCB in the ready queue.

- **Device Queues**: These are used when processes require I/O operations and the requested I/O device is busy. Each device (like a disk) has its own queue holding processes that are waiting for it to become available.



Q. When the process issues an I/O request _____ **[Asked in Wipro NLTH]**

(A) It is placed in an I/O queue

(B) It is placed in a waiting queue

(C) It is placed in the ready queue

(D) It is placed in the Job queue

Answer: A

Q. What will happen when a process terminates? **[Asked in Infosys]**

(A) It is removed from all queues

(B) It is removed from all, but the job queue

(C) Its process control block is de-allocated

(D) Its process control block is never de-allocated

Answer: A

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)/Spooler:** 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.

Q. What is a long-term scheduler? **[Asked in Capgemini]**

(A) It selects processes which have to be brought into the ready queue

(B) It selects processes which have to be executed next and allocates CPU

(C) It selects processes which heave to remove from memory by swapping

(D) None of the mentioned

Answer: A

Q. What is a short-term scheduler? **[Asked in Infosys]**

(A) It selects which process has to be brought into the ready queue

(B) It selects which process has to be executed next and allocates CPU

(C) It selects which process to remove from memory by swapping

(D) None of the mentioned

Answer: B

Q. The primary distinction between the short term scheduler and the long term scheduler is _____ **[Asked in Hexaware]**

(A) The length of their queues

(B) The type of processes they schedule

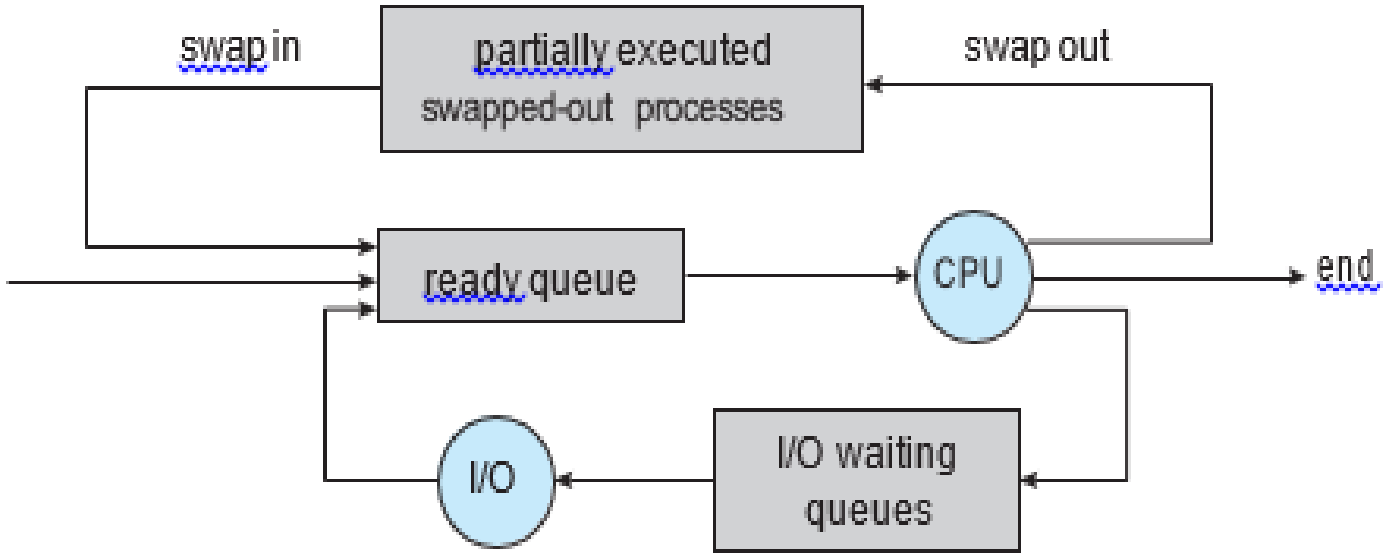
(C) The frequency of their execution

(D) None of the mentioned

Answer: C

Degree of Multiprogramming - The number of processes in memory is known as Degree of Multiprogramming.

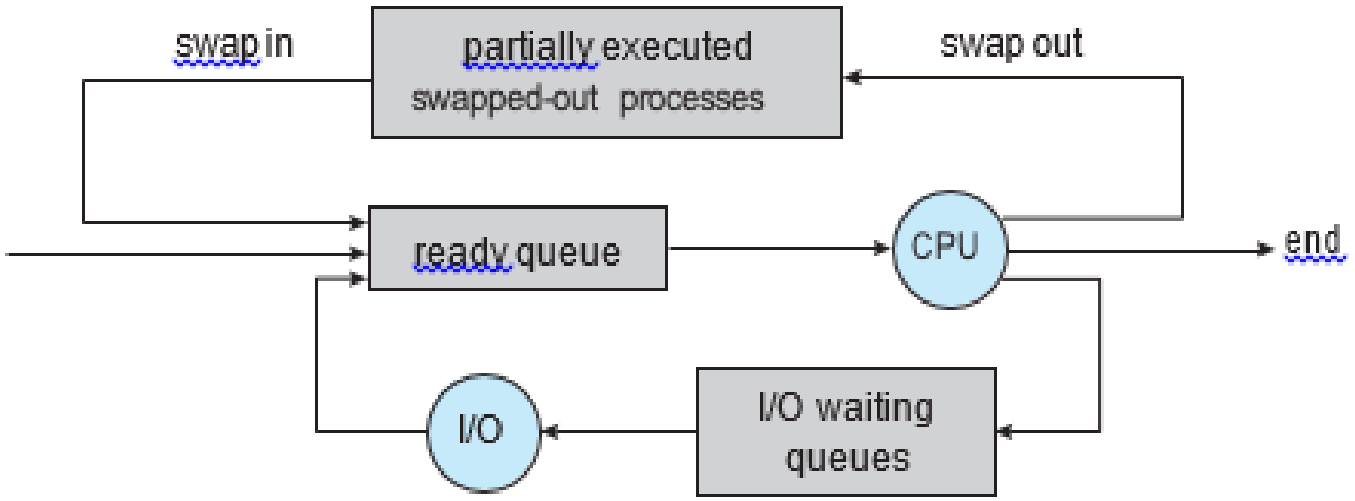
- The long-term scheduler controls the degree of multiprogramming as it is responsible for bringing in the processes to main memory. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. So, 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.



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 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, requiri memory to be freed up.



Q. What is a medium-term scheduler? **[Asked in Cognizant]**

(A) It selects which process has to be brought into the ready queue

(B) It selects which process has to be executed next and allocates CPU

(C) It selects which process to remove from memory by swapping

(D) None of the mentioned

Answer: C

Q. If all processes I/O bound, the ready queue will almost always be _____ and the Short term Scheduler will have a _____ to do. **[Asked in Tech Mahindra]**

(A) full, little
(B) full, lot
(C) empty, little
(D) empty, lot

Answer: C

- **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.

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.

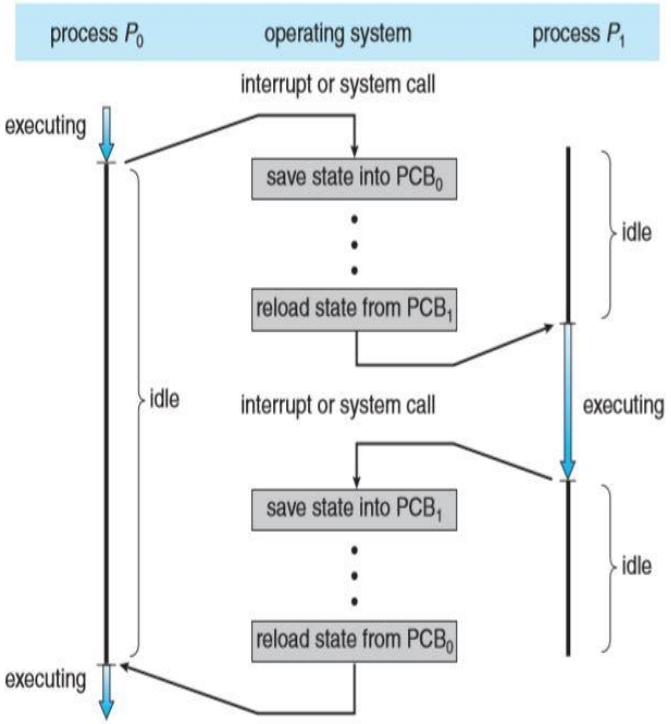


Diagram showing CPU switch from process to process.

CPU Bound and I/O Bound Processes

- 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.

- **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.

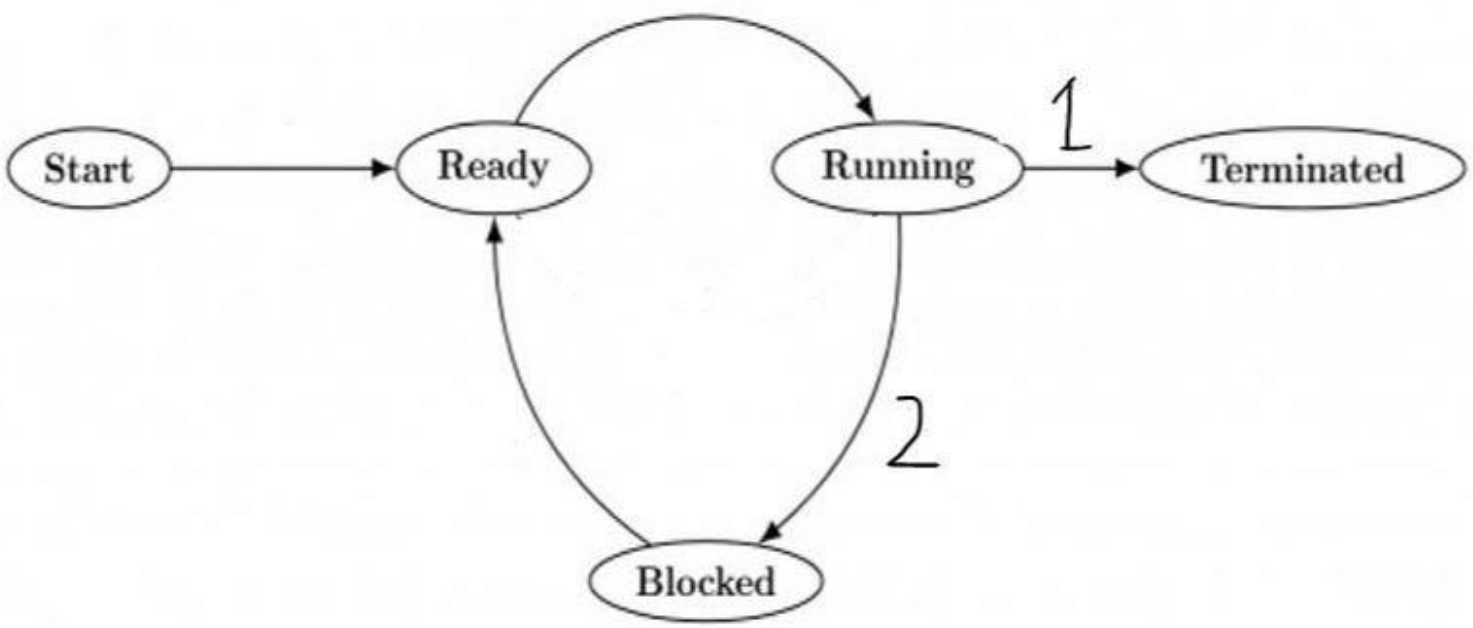
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes. If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.

- 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.

Type of scheduling

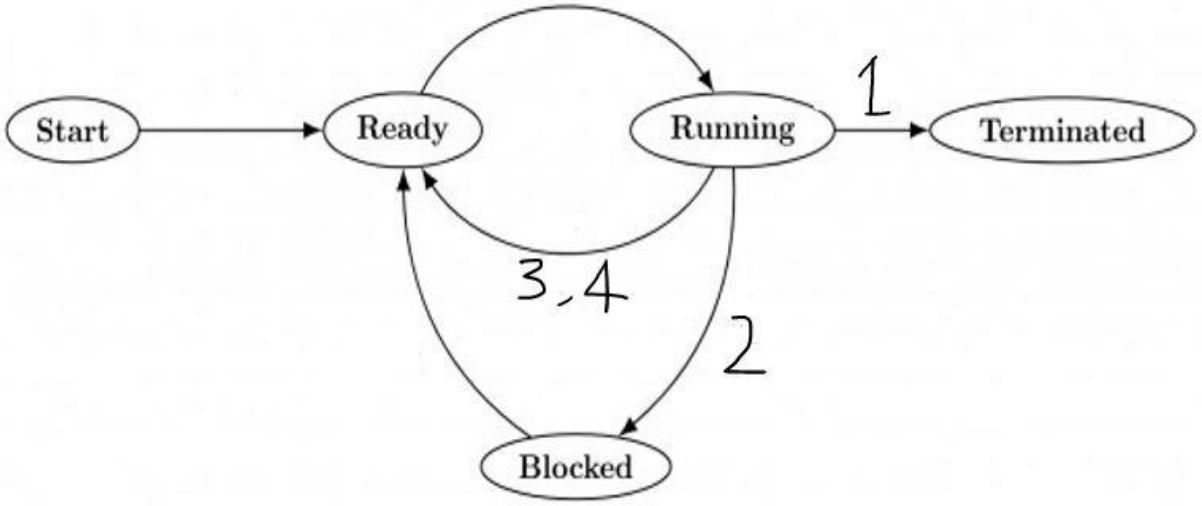
- **Non-Pre-emptive**: Under Non-Pre-emptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU willingly.

- A process will leave the CPU only
    1. When a process completes its execution (Termination state)
    2. When a process wants to perform some i/o operations(Blocked state)



Pre-emptive

- Under Pre-emptive scheduling, once the CPU has been allocated to a process, A process will leave the CPU willingly or it can be forced out. So it will leave the CPU
  1. When a process completes its execution
  2. When a process leaves CPU voluntarily to perform some i/o operations
  3. If a new process enters in the ready states (new, waiting), in case of high priority
  4. When process switches from running to ready state because of time quantum expire.



- **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:

- **<u>CPU utilization</u>:** Keeping the CPU as busy as possible.

- **<u>Throughput</u>:** 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.

- **<u>Waiting time</u>:** Waiting time is the sum of the periods spent waiting in the ready queue.

- **<u>Response Time</u>:** 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.

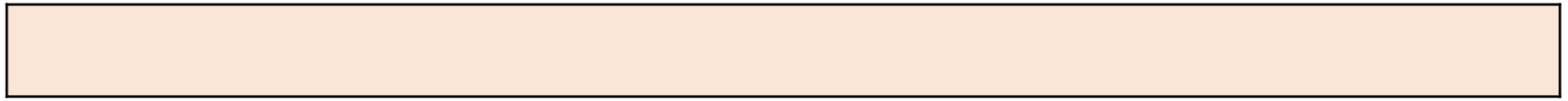
Terminology

- **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), Waiting Time + Burst Time (BT)
- **Waiting Time:** Turn Around Time (TAT) – Burst Time (BT)

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.

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P_0	2	4			
P_1	1	2			
P_2	0	3			
P_3	4	2			
P_4	3	1			
Average					



P. No	AT	BT	TAT=CT-AT	WT=TAT -BT
P_0	0	100		
P_1	1	2		
Average				

P. No	AT	BT	TAT=CT-AT	WT=TAT -BT
P_0	1	100		
P_1	0	2		
Average				

Convoy Effect

- If the smaller process have to wait more for the CPU because of Larger process then this effect is called Convoy Effect, it result into more average waiting time.

- Solution, smaller process have to be executed before longer process, to achieve less average waiting time.

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.
- 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.
- Higher average waiting time and TAT compared to other algorithms.

Shortest Job First (SJF)(non-pre-emptive)
Shortest Remaining Time First (SRTF)/ (Shortest Next CPU Burst) (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.

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P_0	1	7			
P_1	2	5			
P_2	3	1			
P_3	4	2			
P_4	5	8			
Average					

- **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
- **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.

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	CT	TAT	WT
P_1	0	20			
P_2	15	25			
P_3	30	10			
P_4	45	15			

What is the total waiting time for process P_2?

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

Priority scheduling

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.

- if it a new process with 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.

P. No	AT	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
P_0	1	4	4			
P_1	2	2	5			
P_2	2	3	7			
P_3	3	5	8(H)			
P_4	3	1	5			
P_5	4	2	6			
Average						

P. No	AT	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
P_0	0	50	4			
P_1	20	20	1(h)			
P_2	40	100	3			
P_3	60	40	2			
Average						

- **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	Arrival Time	Burst Time	Priority	CT	TAT	WT
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 ____.

Q What is not a disadvantage of priority scheduling in operating systems? **[Asked in TCS NQT]**

a) A low priority process might have to wait indefinitely for the CPU

b) If the system crashes, the low priority systems may be lost permanently

c) Interrupt handling

d) Indefinite blocking

Q The problem of indefinite blockage of low-priority jobs in general priority scheduling algorithm can be solved using:**[Asked in TCS NQT]**

(A) Dirty bit

(B) Compaction

(C) Aging

(D) Swapping

Answer : C

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.
- 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.

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P_0	0	4			
P_1	1	5			
P_2	2	2			
P_3	3	1			
P_4	4	6			
P_5	6	3			
Average					

P. No	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) = CT - AT	Waiting Time (WT) = TAT - BT
P_0	5	5			
P_1	4	6			
P_2	3	7			
P_3	1	9			
P_4	2	2			
P_5	6	3			
Average					

- 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.

- **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 Round Robin Scheduling is most suitable for _____. **[Asked in IBM]**

A) Batch OS

B) Hard RTOS

C) Soft RTOS

D) Time Sharing OS

Q. Consider a set of n tasks with known runtimes r1,r2,.....rn to be run on a uniprocessor machine. which of the following processor scheduling algorithms will result in the maximum throughput? **[Asked in Wipro NLTH]**

(A) Round - robin

(B) Shortest job first

(C) FCFS

(D) Priority scheduling

Answer: B

Q. The real-time operating system, which of the following is the most suitable scheduling scheme? **[Asked in E-Litmus]**

(A) Round robin

(B) First come first serve

(**C)** Pre-emptive

(**D)** Random scheduling

Answer : C

Q Which of the following statements are true?

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

Q Consider the following. The completion order of the three processes under the policies FCFS and RR (with CPU quantum of 2 time units) are:**[Asked in Infosys]**

A) FCFS: P_1, P_2, P_3 and RR: P_1, P_3, P_2

B) FCFS: P_1, P_2, P_3 and RR: P_1, P_2, P_3

C) FCFS: P_1, P_2, P_3 and RR: P_3, P_2, P_1

D) FCFS: P_1, P_3, P_2, and RR: P_1, P_2 ,P_3

Process	Arrival Time	Time Unit Required
P_1	0	5
P_2	2	7
P_3	3	3

Q. Consider the following set of processes, assumed to have arrived at time 0. Consider the CPU scheduling algorithms Shortest Job First (SJF) and Round Robin (RR). For RR, assume that the processes are scheduled in the order P1, P2, P3, P4.

Processes	P_1	P_2	P_3	P_4
Burst time (in ms)	8	7	2	4

If the time quantum for RR is 4 ms, then the absolute value of the difference between the average turnaround times (in ms) of SJF and RR (round off to 2 decimal places) is _____. **[Asked in Capgemini]**

(A) 5.25

(B) 3.78

(C) 0.0

(D) 4.5

Answer : A

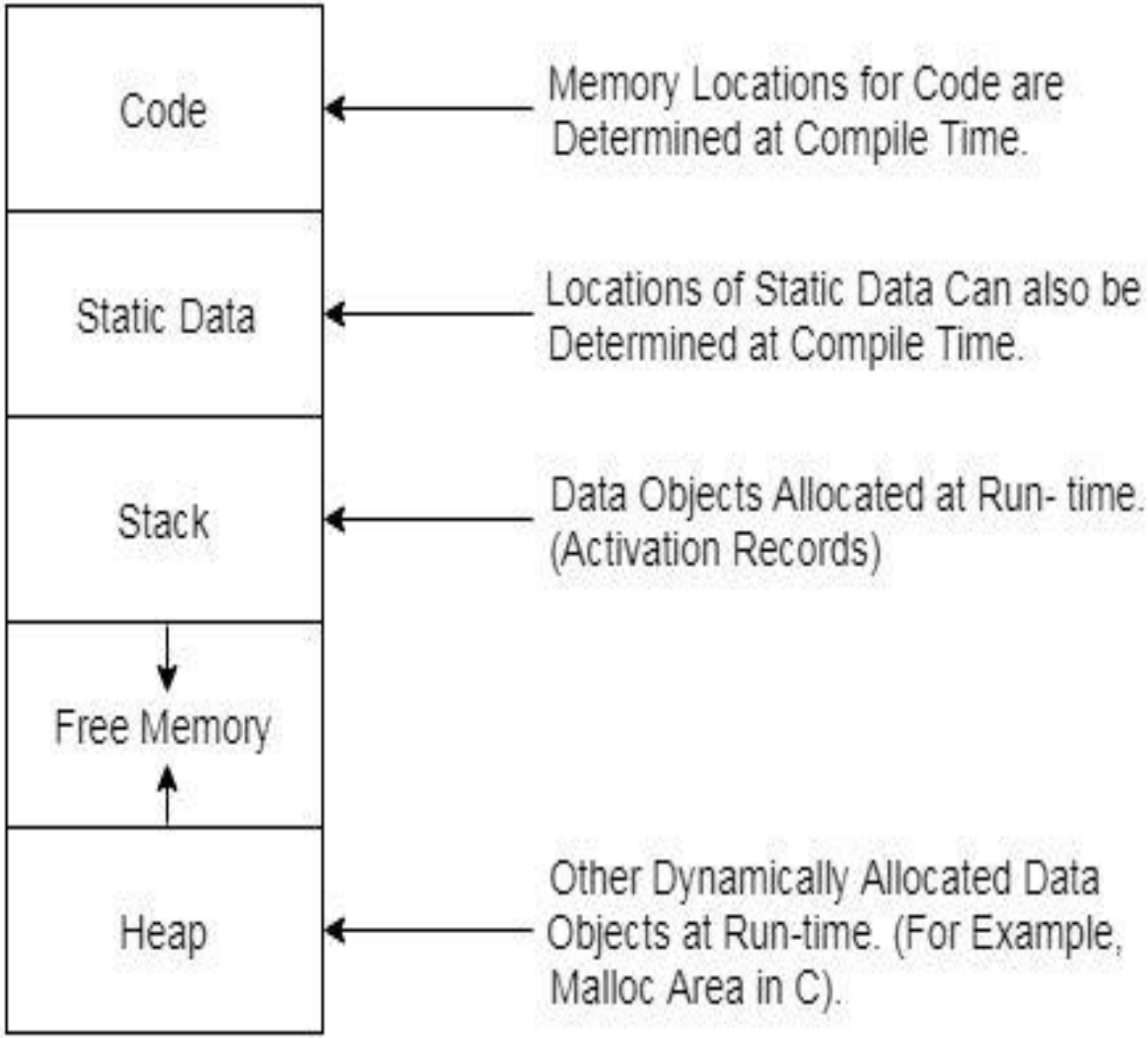
Fork(Requirement)

- 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 means we have the multiple copy of the same data waiting the main memory


Code
Static Data
Stack
Free Memory
Heap
Memory Locations for Code are Determined at Compile Time.
Locations of Static Data Can also be Determined at Compile Time.
Data Objects Allocated at Run- time. (Activation Records)
Other Dynamically Allocated Data Objects at Run-time. (For Example, Malloc Area in C).

Q. _____ system call creates new process in Unix. **[Asked in Syntel]**

(A) fork

(B) fork new

(C) Create

(D) create new

Answer : A

Q. In Unix, Which system call creates the new process? **[Asked in L&T InfoTech (LTI)]**

(A) Fork

(B) Create

(C) New

(D) none of the mentioned

Answer: A

Q. fork() is a function used in a **[Asked in Hexaware]**

(A) OS system call

(B) Procedure call of OS

(C) Process of OS

(D) Discard the current process

(E) All of these

Answer : A

Q A process executes the code

```
fork();
fork();
fork();
```

the total number of child processes created is **[Asked in L&T InfoTech]**

a) 3

b) 4

c) 7

d) 8

Q A process executes the following code

```
for (i=0; i<n; i++)
fork();
```

a) n

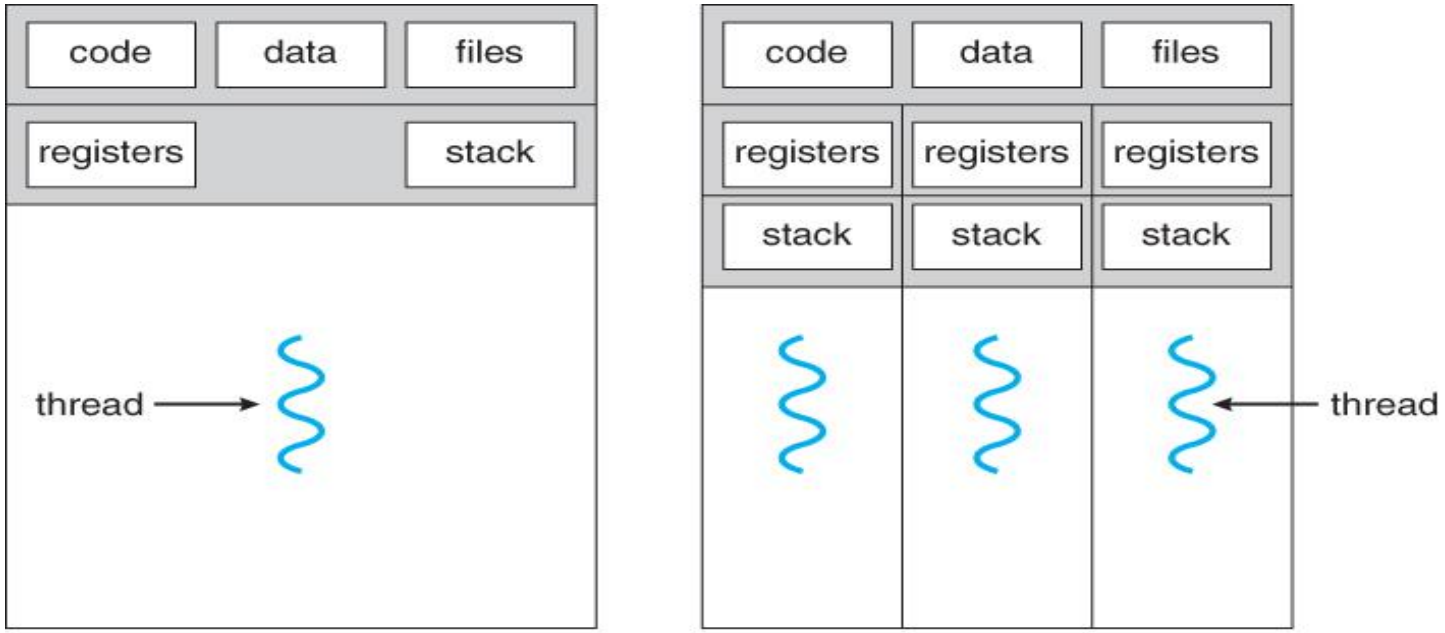
b) $(2^n) - 1$

c) 2^n

d) $(2^{n+1}) - 1$

- 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. 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.)



single-threaded process

multithreaded process

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.

Q Which of the following is/are shared by all the threads in a process?

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

Q Threads of a process share

(A) global variables but not heap

(B) heap but not global variables

(C) neither global variables nor heap

(D) both heap and global variables

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?

(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

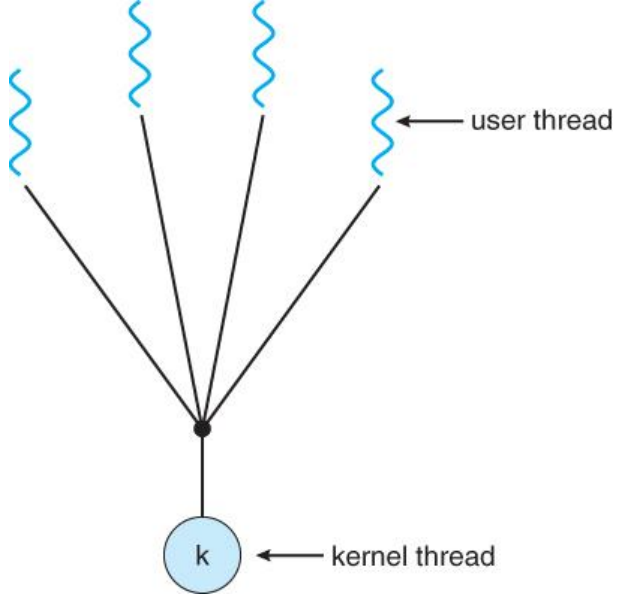
(D) On per-thread basis, the OS maintains only scheduling and accounting information

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 OS support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

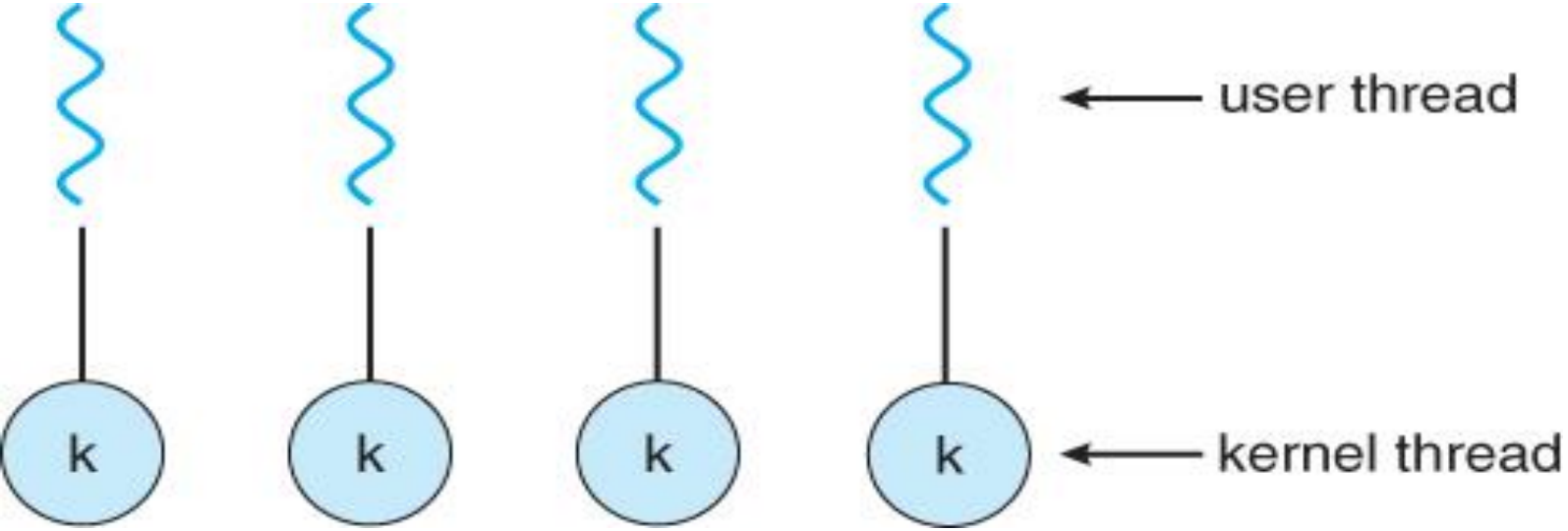
Many-To-One Model

- 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.



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.

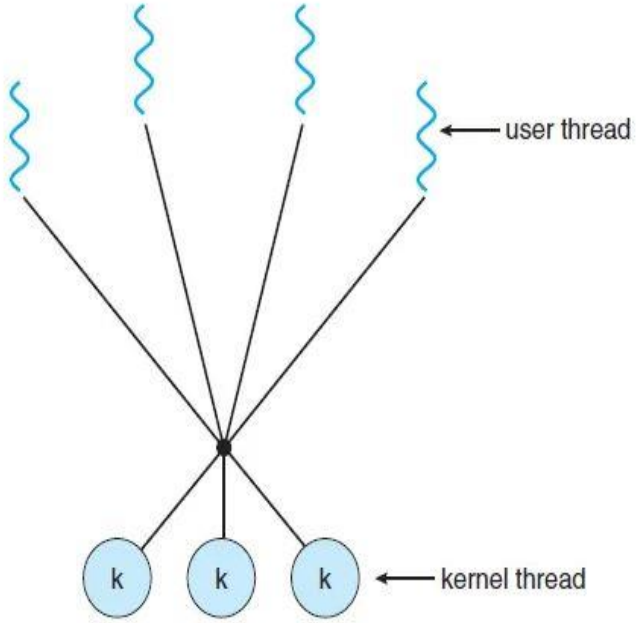


Many-To-Many Model

- 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.



Q Which one of the following is FALSE?

(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

Q Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE?

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

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?

(A) (ii), (iii) and (iv) only

(B) (ii) and (iii) only

(C) (i) and (iii) only

(D) (i) and (ii) only

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 some time may result in data inconsistency for e.g.

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

- **Race Condition:** The condition in which the output of a process depends on the execution sequence of process. i.e. 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. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called _____
[Asked in TCS NQT]

(A) dynamic condition

(B) race condition

(C) essential condition

(D) critical condition

Answer: b

Q The following two functions P_1 and P_2 that share a variable B with an initial value of 2 execute concurrently. The number of distinct values that B can possibly take after the execution is

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

$P_1()$	$P_2()$
{	{
(I_{11}) C = B – 1;	(I_{21}) D = 2 * B;
(I_{12}) B = 2 * C;	(I_{22}) B = D - 1;
}	}

$Case_1$	$Case_2$	$Case_3$	$Case_4$	$Case_5$	$Case_6$
I_{11}C=1	I_{21}D=4	I_{11}C=1	I_{21}D=4	I_{11}C=1	I_{21}D=4
I_{12}B=2	I_{22}B=3	I_{21}D=4	I_{11}C=1	I_{21}D=4	I_{11}C=1
I_{21}D=4	I_{11}C=2	I_{22}B=3	I_{12}B=2	I_{12}B=2	I_{22}B=3
I_{22}B=3	I_{12}B=4	I_{12}B=2	I_{22}B=3	I_{22}B=3	I_{12}B=2

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.

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

Q A critical section is a program segment?

(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

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). 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.
- 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 Part of a program where the shared memory is accessed and which should be executed indivisibly, is called:

a) Semaphores

b) Directory

c) Critical Section

d) Mutual exclusion

Q. If a process is executing in its critical section, then no other processes can be executing in their critical section. What is this condition called? **[Asked in Cognizant]**

(A) mutual exclusion

(B) critical exclusion

(C) synchronous exclusion

(D) asynchronous exclusion

Answer: A

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**

- Peterson's solution is a classic Software-based solution to the critical-section problem for two process. This solution ensures **Mutual Exclusion, Progress and Bounded Wait.**

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
	}

Operation 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).
- The definition of wait () is as follows:

```
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.

- Semaphores are going to ensure Mutual Exclusion and Progress but does not ensures bounded waiting.

$P_i()$
{
While(T)
{
Initial Section
wait(s)
Critical Section
signal(s)
Remainder Section
}
}

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

Signal(S)
{
s++;
}

Q Two atomic operations permissible on Semaphores are _____ and _____.

a) wait, stop

b) wait, hold

c) Hold, signal

d) wait, signal

Q. Which one of the following is a synchronization tool?

(A) Thread

(B) Pipe

(C) Semaphore

(D) socket

Answer: C

Q. A semaphore is a shared integer variable _____ **[Asked in Infosys]**

(A) that can not drop below zero

(B) that can not be more than zero

(C) that can not drop below one

(D) that can not be more than one

Answer: A

Q Using Semaphores ensure that the order of execution of there concurrent process p_1, p_2 and p_3 must be $p_2 \rightarrow p_3 \rightarrow p_1$?

$P_1()$	$P_2()$	$P_3()$
code	code	code

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 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 // CS
Semaphore E = n // Count Empty cells
Semaphore F = 0 // Count Filled cells

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

Producer()	Consumer()
{	{
while(T)	while(T)
{	{
// Produce an item	wait(F)//UnderFlow
wait(E)//OverFlow	wait(S)
wait(S)	// Pick item from buffer
// Add item to buffer	signal(S)
signal(S)	signal(E)
signal(F)	Consume item
}	}
}	}

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

Dining Philosopher Problem

- **Scenario Setup**: Five philosophers are seated around a circular table, each with a bowl of rice in the center. The table has five chairs and five single chopsticks placed between each pair of philosophers.
- **Activity Cycle**: Philosophers alternate between thinking and eating. They do not interact with each other while thinking.
- **Eating Process**:
  - A philosopher becomes hungry and attempts to pick up the two closest chopsticks — one between her and the philosopher on her left, and one between her and the philosopher on her right.
  - Each philosopher can pick up only one chopstick at a time.
  - A philosopher cannot pick up a chopstick if it is already being held by a neighbor.
  - Once a philosopher has both chopsticks, she eats without releasing the chopsticks.
- **Post-Eating**: After eating, the philosopher puts both chopsticks back on the table and resumes thinking.

Solution for Dining Philosophers

```
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.
- 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 her 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.

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 enable interrupts.

- This solution is only used by OS, as if some user process enter into critical section, then can block the entire system.

- 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.

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

Hardware Type Solution Test and Set

- 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.

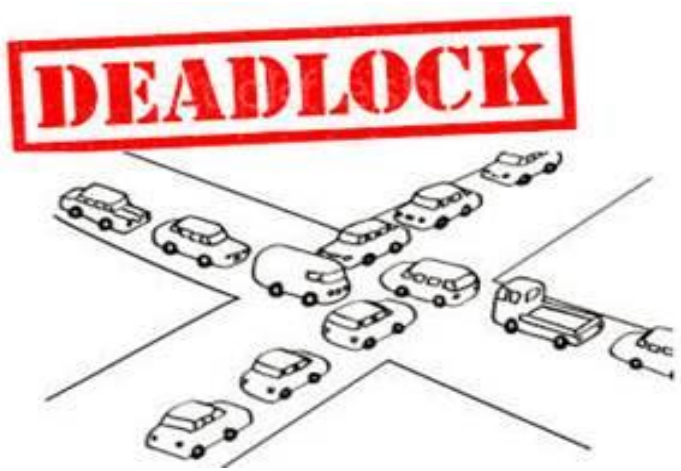
- The critical-section problem could be solved simply in a single-processor environment if we could prevent interrupts from occurring while a shared variable was being modified.

```
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 */
}
```

Basics of Dead-Lock

- In a multiprogramming environment, several processes may compete for a finite number of resources.
- In these scenario there is a possibility of Deadlock



P_1 P_2

R_1 R_2

Tax

Services

- 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. Starvation is long waiting but deadlock is infinite waiting

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.

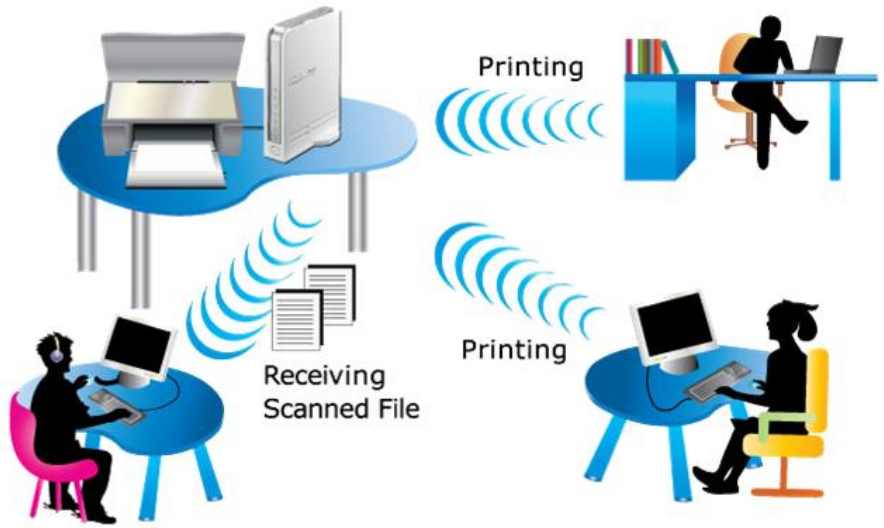
Necessary conditions for deadlock

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

- Mutual exclusion
- Hold and wait
- No pre-emption
- Circular wait

- **<u>Mutual exclusion</u>**: - 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.



- **<u>Hold and wait</u>**: - 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.

- **<u>Circular wait</u>**: - 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.

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.

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.

Polio vaccine

- **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 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 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?
(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

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:

(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

Q Which of the following is NOT a valid deadlock prevention scheme?

(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 all required resources be allocated before execution.

Q. Consider a system having m resources of the same type. These resources are shared by 3 processes A, B, C, which have peak time demands of 3, 4, 6 respectively. The minimum value of m that ensures that deadlock will never occur is **[Asked in Wipro NLTH]**

(A) 11

(B) 12

(C) 13

(D) 14

Answer : A

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 largest value of K that will always avoid deadlock is ____. **[Asked in Cognizant]**

(A) 10

(B) 2

(C) 5

(D) 1

Answer :B

- **<u>Problem with Prevention</u>: -** 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.
- So, in order to avoid deadlock in run time, System try to maintain some books like a banker, whenever someone ask for a loan(resource), it is granted only when the books allow.

Avoidance

- To avoiding deadlocks we 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.

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

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

	Current Need		
	E	**F**	**G**
P_0	3	3	0
P_1	1	0	2
P_2	0	3	0
P_3	3	4	1

System Max		
E	F	G
8	4	6

Available		
E	F	G
3	3	0

- A deadlock-avoidance algorithm dynamically examines the resource-allocation state.

- 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.

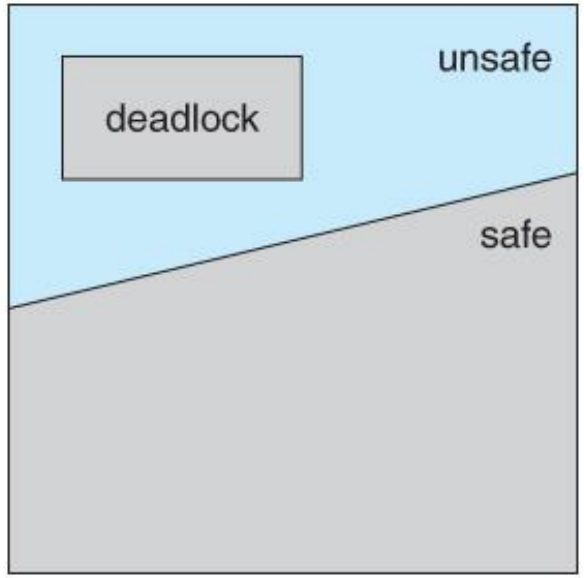
- We check, if there exist “some sequence in which we can satisfies demand of every process without going into deadlock, if yes, this sequence is called safe sequence” and request can be allowed. Otherwise there is a possibility of going into deadlock.

- 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.



Q. A process said to be in _____ state, if it was waiting for an event that will never occur. **[Asked in Amcat]**

(A) Safe

(B) Unsafe

(C) Starvation

(D) deadlock

Answer : D

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?

(A) P_0

(B) P_1

(C) P_2

	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

(D) None of the above, since the system is in a deadlock

Q A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for 3 processes are shown below: Which of the following best describes the current state of the system?

(A) Safe, Deadlocked

(B) Safe, Not Deadlocked

(C) Not Safe, Deadlocked

(D) Not Safe, Not Deadlocked

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

Q Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes?

(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

Q. Which of the following is not true with respect to deadlock prevention and deadlock avoidance schemes? **[Asked in Tech Mahindra]**

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

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

(C) Deadlock avoidance requires knowledge of resource requirements a prior

(D) Deadlock prevention is more restrictive than deadlock avoidance

Answer : A

Q. Banker's algorithm is used? **[Asked in L&T InfoTech]**

(A) To avoid deadlock

(B) To deadlock recovery

(C) To solve the deadlock

(D) None of these

Answer: (a)

Deadlock detection and recovery

- Here we do not check safety and where any process request for some resources then these resources are allocated immediately, if available.

- Here there is a possibility of deadlock, which must be detected using different approaches.
- **Active approach:** Here we simply invoke the algorithm at defined intervals—for example, once per hour. or whenever CPU utilization drops below 40 percent.

- **Lazy approach:** whenever CPU utilization drops below 40 percent or some unusual performance is there, we go for Detection.

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

- **Abort All Deadlocked Processes**: This direct method guarantees the deadlock will be broken by terminating all involved processes. However, it is costly since all the computation done by these processes is lost and must be repeated later.

- **Abort One Process at a Time**: This more gradual approach involves aborting one deadlocked process at a time and then running a deadlock-detection algorithm to check if the deadlock still persists. This method can be resource-intensive due to the repeated checks needed after each process termination.

- **Challenges with Process Abortion**: Terminating a process can be problematic, especially if the process was engaged in critical operations like updating files or printing. Abrupt termination could leave resources in an inconsistent state, requiring additional steps to restore them to a correct state.

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.

- Operating System behaves like there is no concept of deadlock.

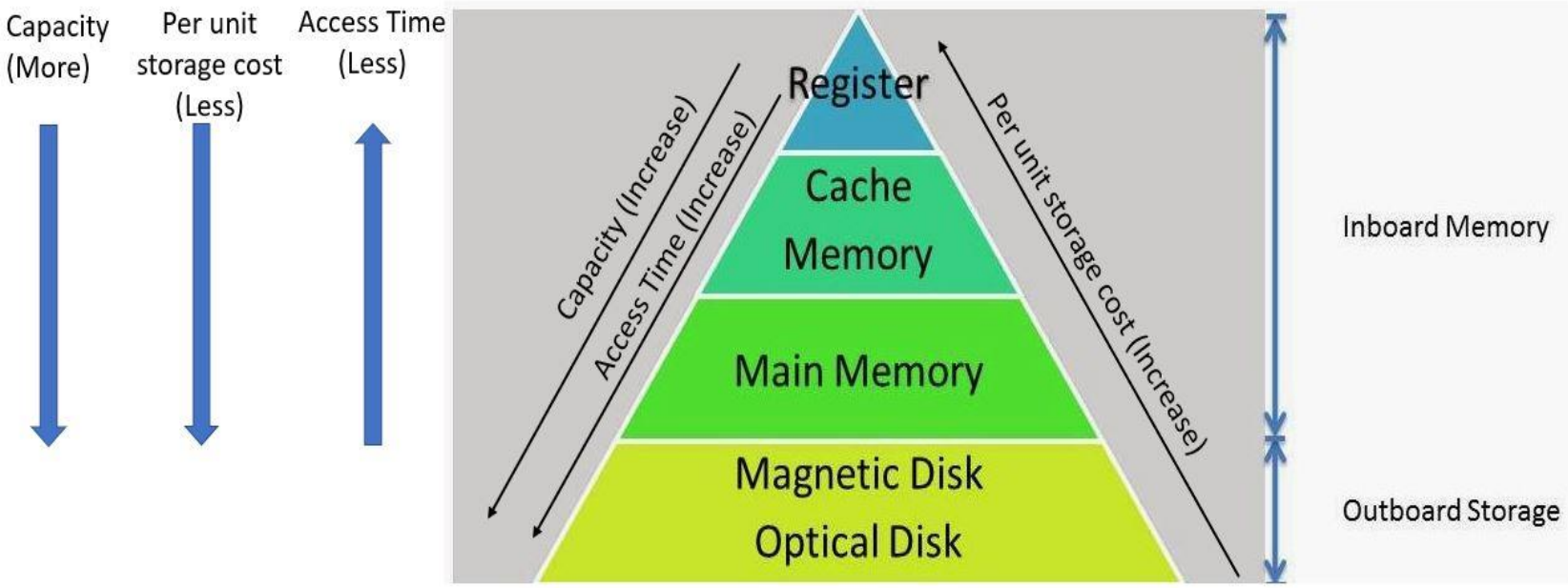
Ostrich Algorithm

Memory Hierarchy

- Let first understand what we need from a memory
  - Large capacity
  - Less per unit cost
  - Less access time(fast access)

Memory Hierarchy

- The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.
- Let first understand what we need from a memory
  - Large capacity
  - Less per unit cost
  - Less access time(fast access)



Cycle

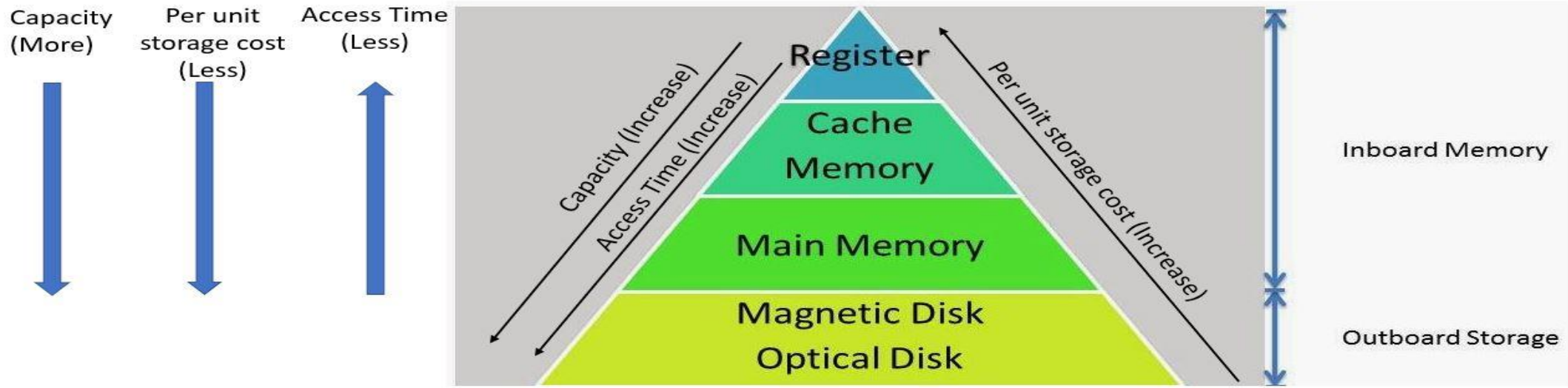
Car

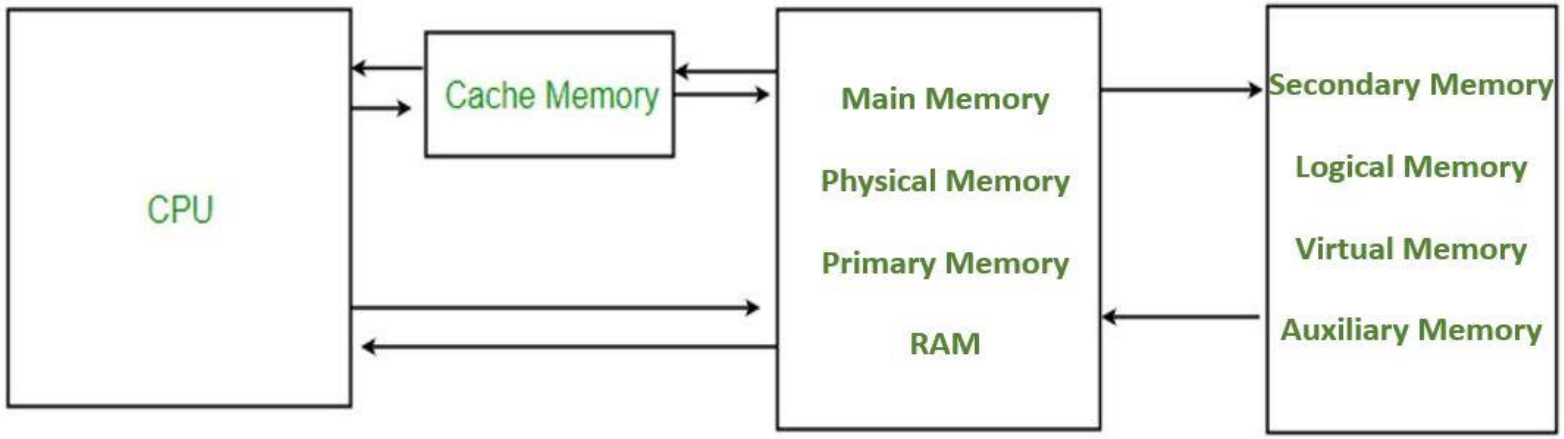
Airbus

Lathi

303

AK-47


Capacity (More)
Per unit storage cost (Less)
Access Time (Less)
Register
Cache Memory
Main Memory
Magnetic Disk
Optical Disk
Capacity (Increase)
Access Time (Increase)
Per unit storage cost (Increase)
Inboard Memory
Outboard Storage


CPU
Cache Memory
Main Memory
Physical Memory
Primary Memory
RAM
Secondary Memory
Logical Memory
Virtual Memory
Auxiliary Memory

Showroom

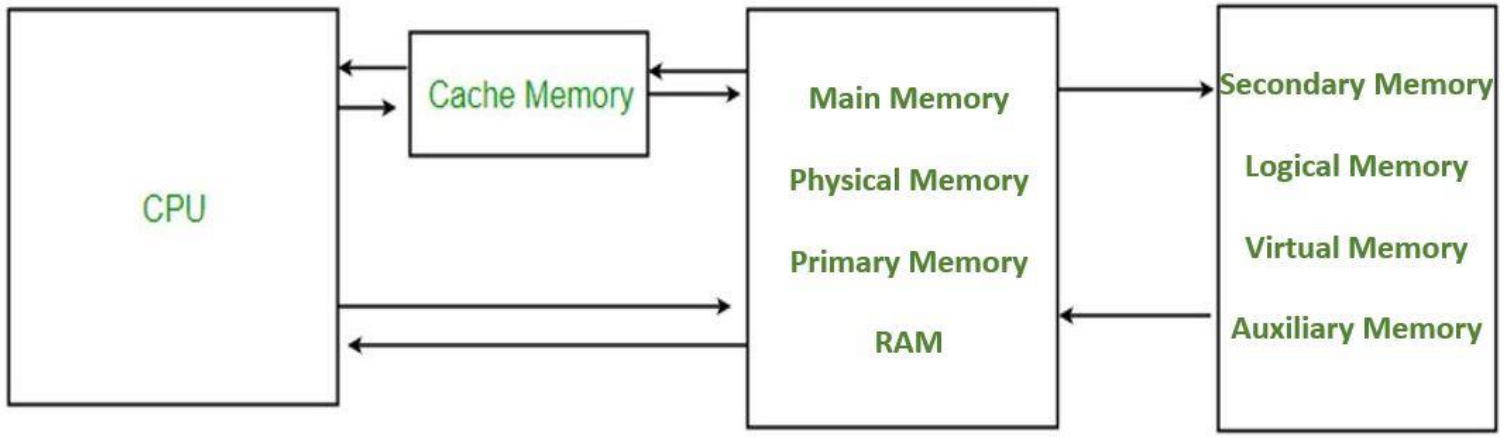
Go down

Factory

- If we want this hierarchy to work fine, then most of the time when CPU needs a data it must be present in Cache, if not possible main memory, worst case Secondary memory. But this is a difficult task to do as my computer has, 8 TB of Secondary Memory, 32 GB of Main Memory, but only 768KB, 4MB, 16 MB of L_1, L_2, L_3 cache respectively. If somehow we can estimate what data CPU will require if future we can prefetch in Cache and Main Memory. Locality of reference Helps we to perform this estimation.

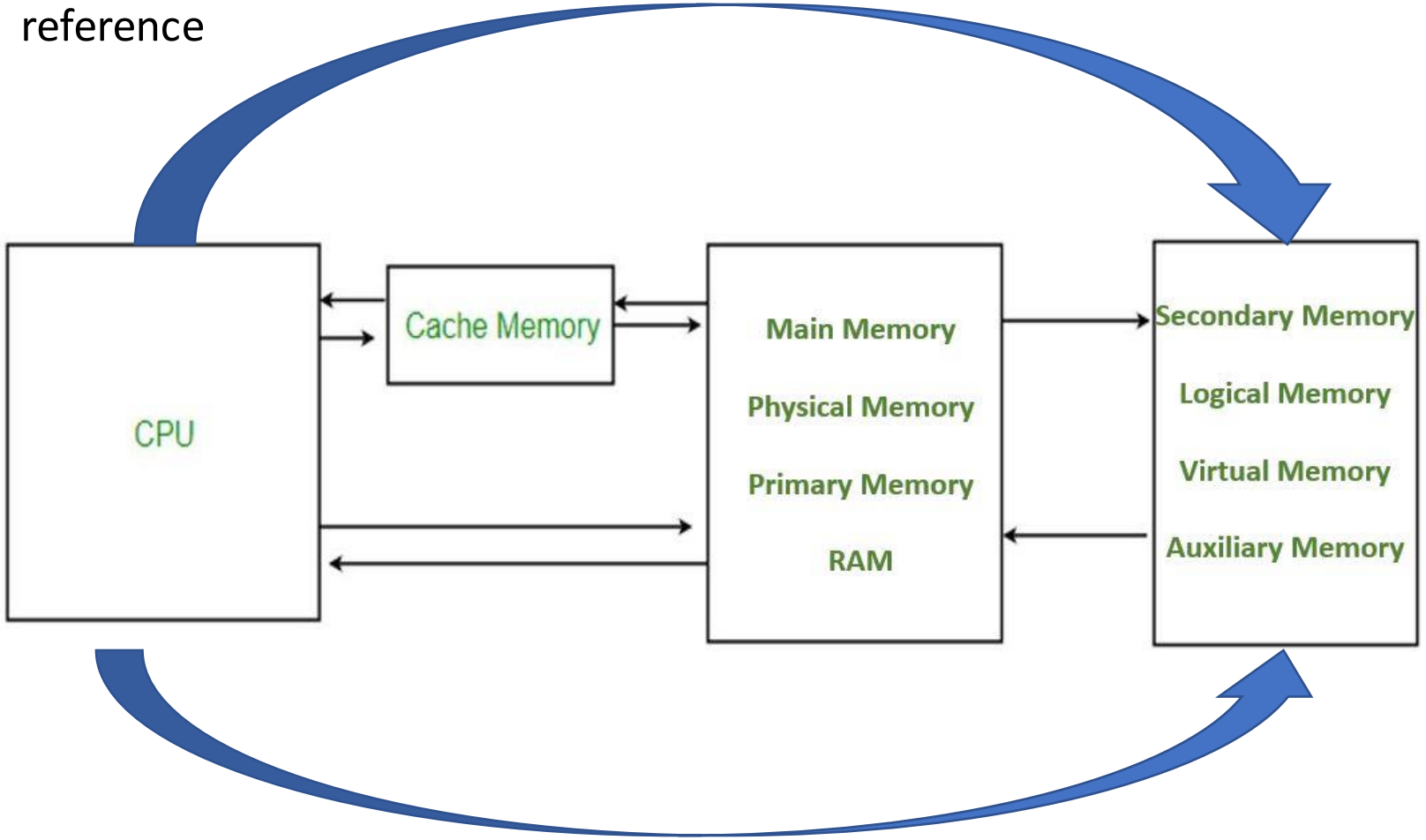
- The main memory occupies a central position by being able to communicate directly with the CPU. When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.

- A special very-high speed memory called a Cache is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.



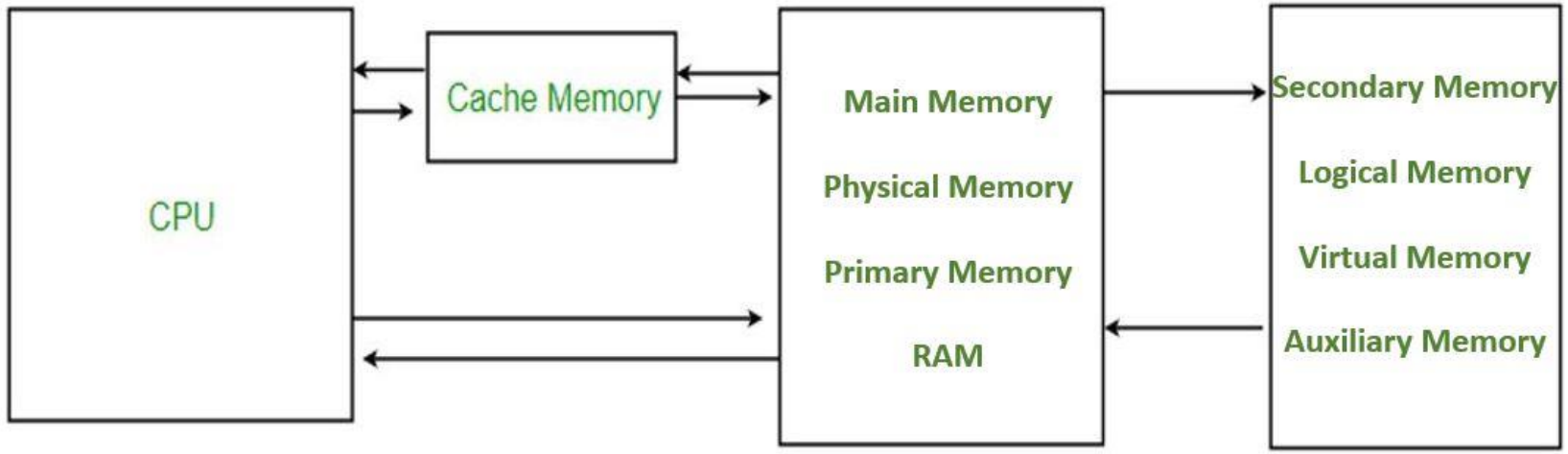
Locality of Reference

- The references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. There are two types of locality of reference



- **Spatial Locality:** Spatial locality refers to the use of data elements in the nearby locations.

- **Temporal Locality:** Temporal locality refers to the reuse of specific data, and/or resources, within a relatively small-time duration. Or, the most frequently used items will be needed soon. (LRU is used for temporal locality)



Duty of Operating System

- Operating system is responsible for the following activities in connection with memory management:
  - Address translation from Logical address to Physical address.
  - Deciding which processes (or parts of processes) and data to move into and out of memory. Allocating and deallocating memory space as needed.
  - Keeping track of which parts of memory are currently being used and who is using them.
  - The first duty of OS is to translate this logical address into physical address. There can be two approaches for storing a process in main memory.
    - Contiguous allocation policy
    - Non-contiguous allocation policy

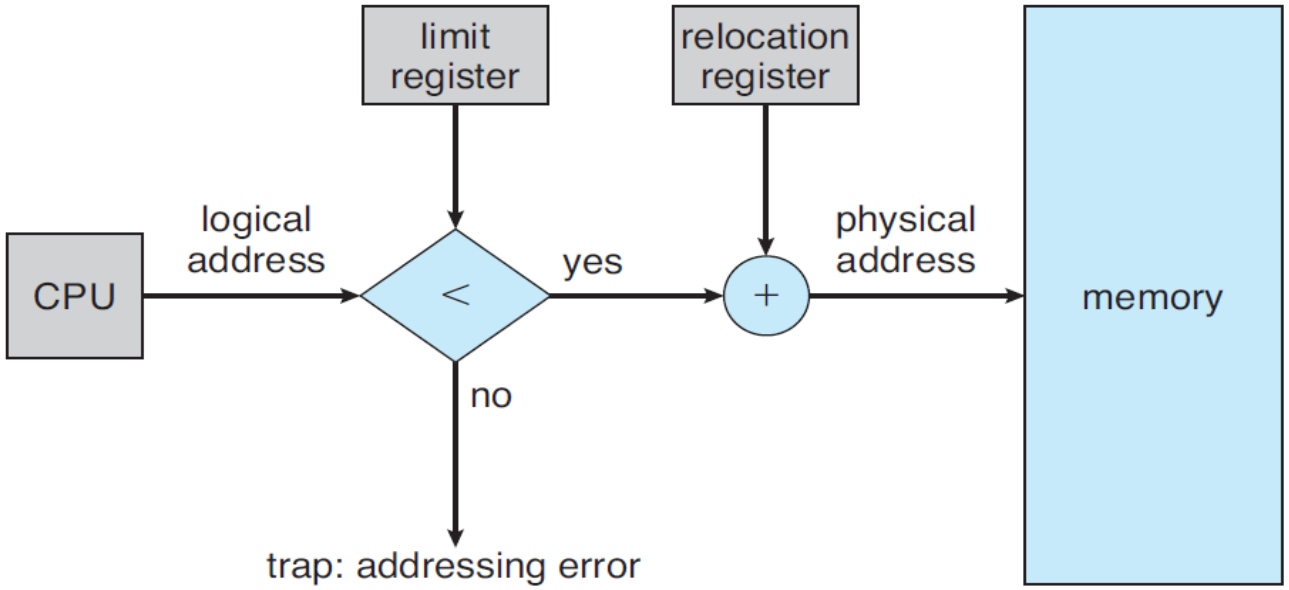
Contiguous allocation policy

- We know that when a process is required to be executed it must be loaded to main memory, by policy has two implications.
  - It must be loaded to main memory completely for execution.

  - Must be stored in main memory in contiguous fashion.

- In order to check whether address generated to CPU is valid(with in range) or invalid, we compare it with the value of limit register, which contains the max no of instructions in the process.

- So, if the value of logical address is less than limit, then it means it's a valid request and we can continue with translation otherwise, it is a illegal request which is immediately trapped by OS.

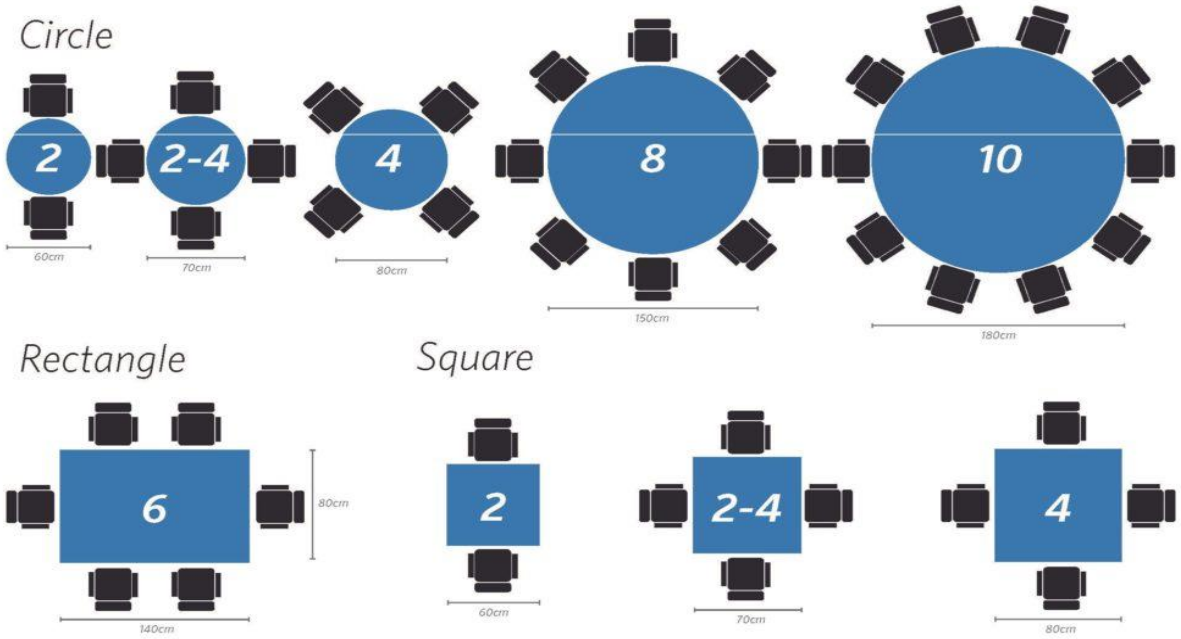


LA	PA
0	0 + r
Max	Max + r
Min	Min + r

Space Allocation Method in Contiguous Allocation

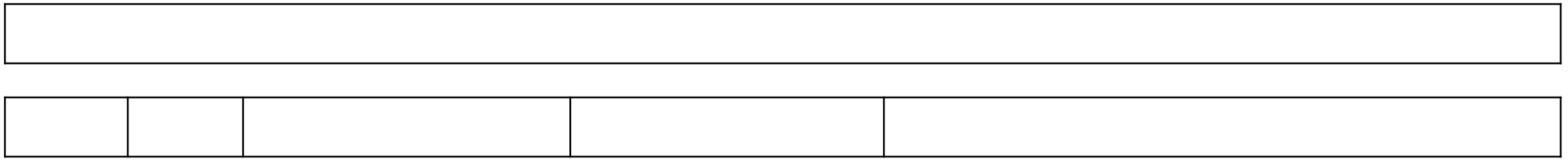
- **Variable size partitioning:** -In this policy, in starting, we treat the memory as a whole or a single chunk & whenever a process request for some space, exactly same space is allocated if possible and the remaining space can be reused again.

- **<u>Fixed size partitioning</u>:** - here, we divide memory into fixed size partitions, which may be of different sizes, but here if a process request for some space, then a partition is allocated entirely if possible, and the remaining space will be waisted internally.



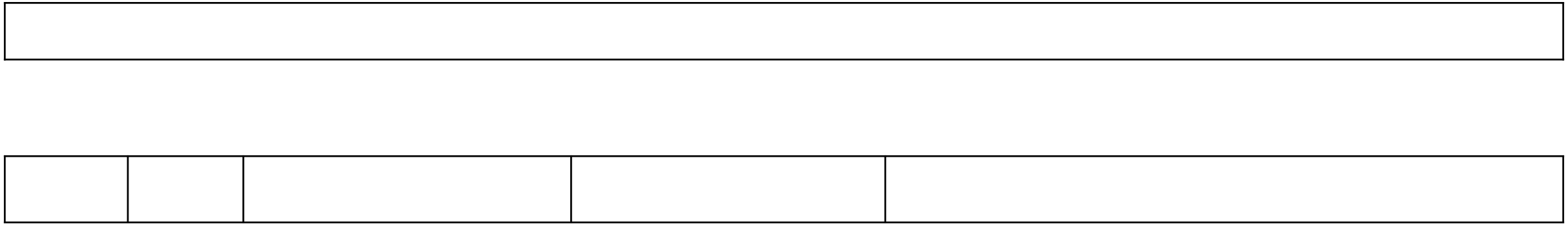
- **First fit policy:** - as the name implies, it states searching the memory from the base and will allocate first partition which is capable enough.
  - **Advantage:** - simple, easy to use, easy to understand
  - **Disadvantage**: -poor performance, both in terms of time and space

- **Best fit policy:** - Here, we search the entire memory and will allocate the smallest partition which is capable enough.
  - **Advantage**: - perform best in fix size partitioning scheme.
  - **Disadvantage**: - difficult to implement, perform worst in variable size partitioning as the remaining spaces which are of very small size.



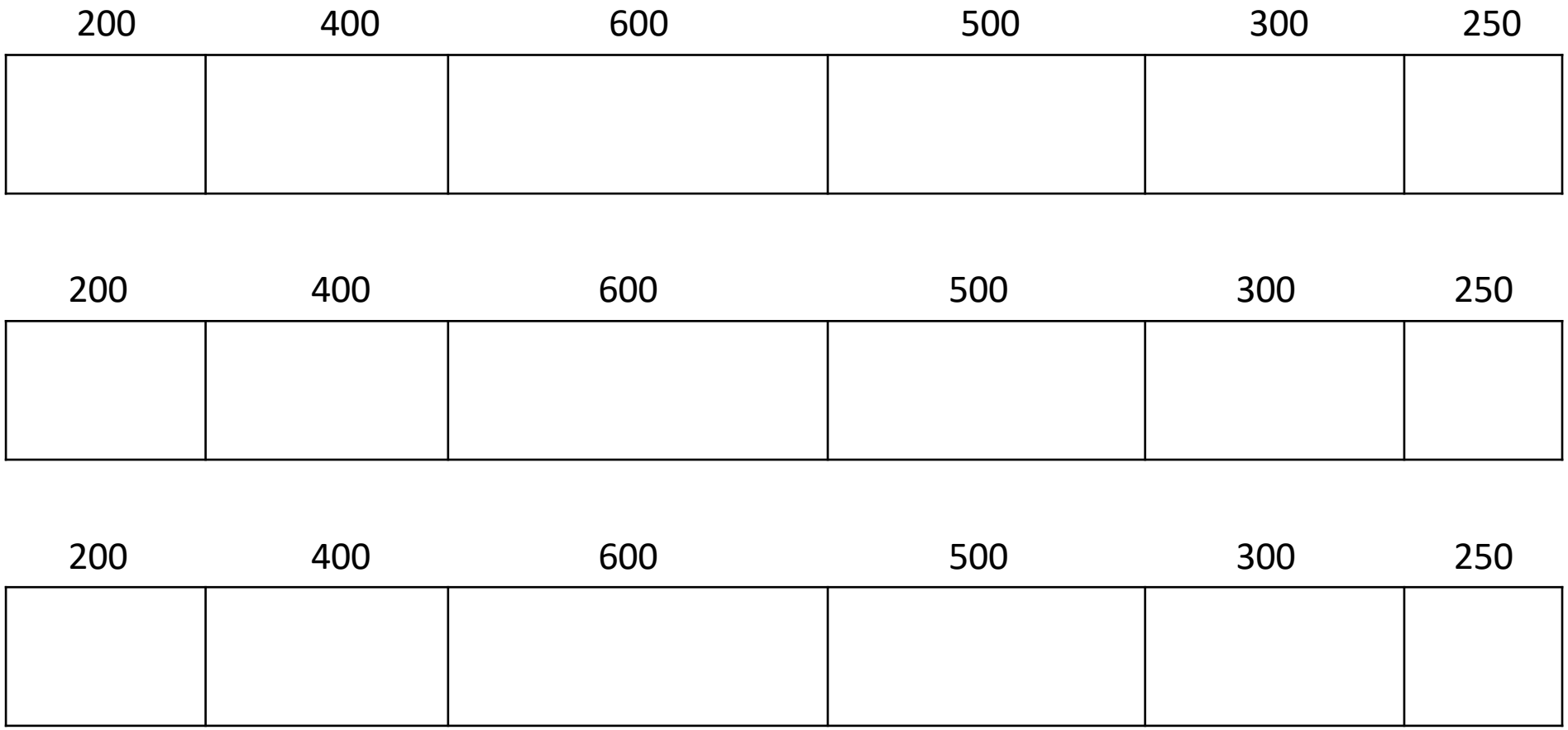
- **Worst fit policy:** - it also searches the entire memory and allocate the largest partition possible.
  - **Advantage:** - perform best in variable size partitioning
  - **Disadvantage:** - perform worst in fix size partitioning, resulting into large internal fragmentation.

- **<u>Next fit policy:</u>** - next fit is the modification in the best fit where, after satisfying a request, we start satisfying next request from the current position.



Q Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process?

(A) 200 KB and 300 KB **(B)** 200 KB and 250 KB **(C)** 250 KB and 300 KB **(D)** 300 KB and 400 KB



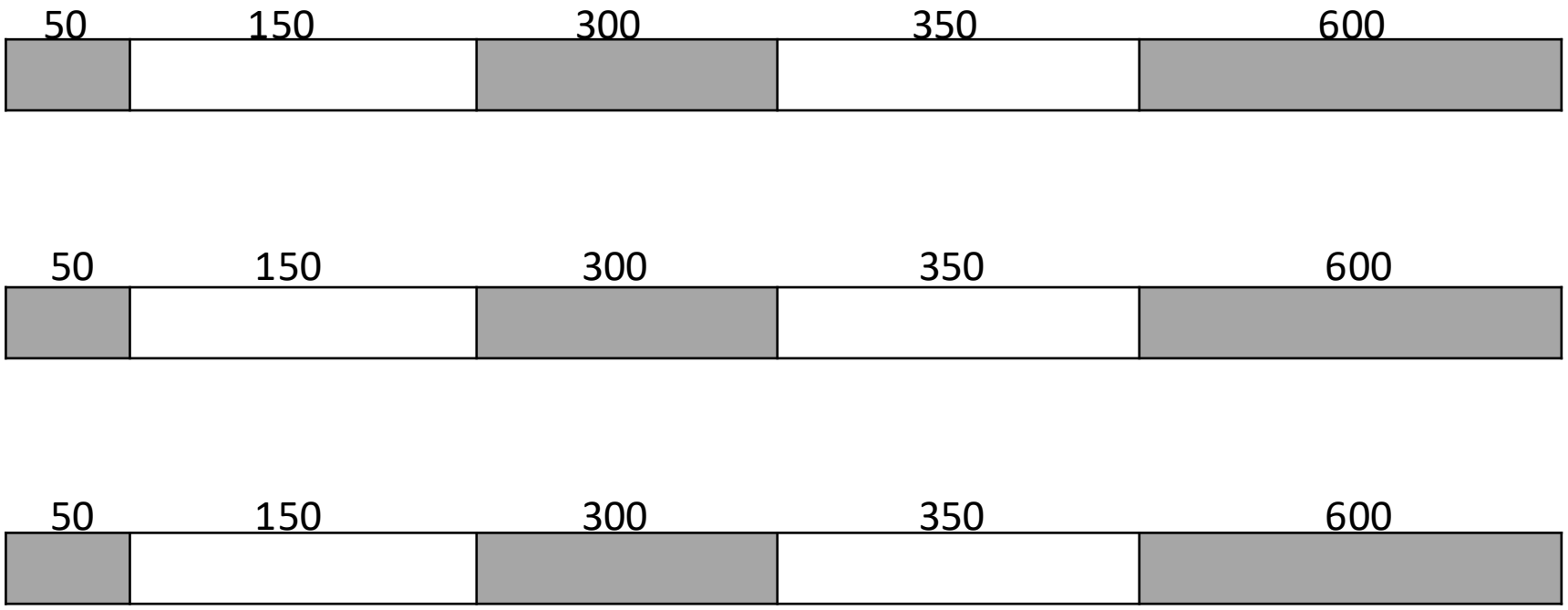
Q Consider the following heap (figure) in which blank regions are not in use and hatched region are in use. The sequence of requests for blocks of size 300, 25, 125, 50 can be satisfied if we use

(a) either first fit or best fit policy (any one)

(b) first fit but not best fit policy

(c) best fit but first fit policy

(d) None of the above



Q A program is located in the smallest available hole in the memory is ……

(A) best – fit

(B) first – bit

(C) worst – fit

(D) buddy

Q In which of the following storage replacement strategies, is a program placed in the largest available hole in the memory?

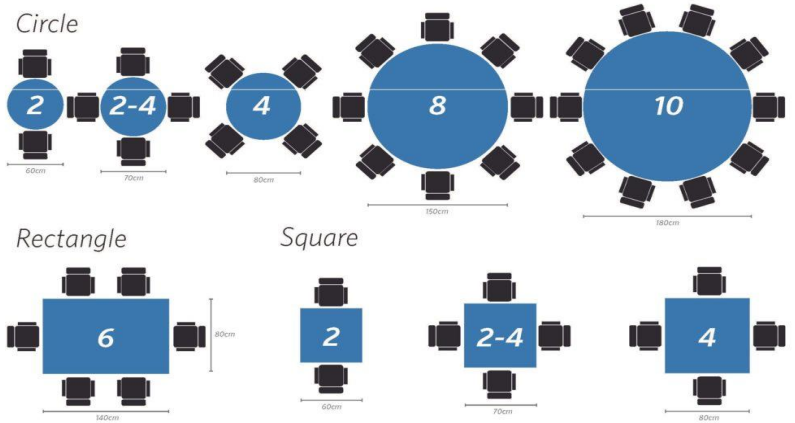
(A) Best fit

(B) First fit

(C) Worst fit

(D) Buddy

- **<u>External fragmentation:</u>** - External fragmentation is a function of contiguous allocation policy. The space requested by the process is available in memory but, as it is not being contiguous, cannot be allocated this wastage is called external fragmentation.

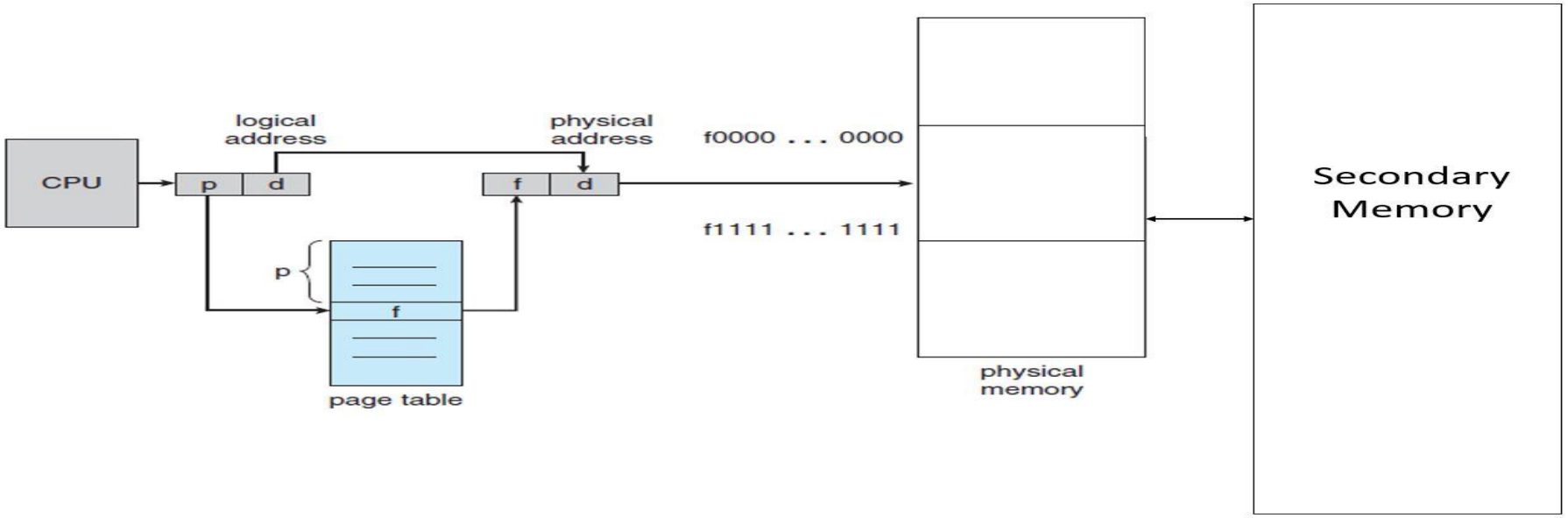


- **<u>Internal fragmentation:</u>** - Internal fragmentation is a function of fixed size partition which means, when a partition is allocated to a process. Which is either the same size or larger than the request then, the unused space by the process in the partition Is called as internal fragmentation

- How can we solve external fragmentation?
    - We can also swap processes in the main memory after fixed intervals of time & they can be swapped in one part of the memory and the other part become empty(Compaction, defragmentation). This solution is very costly in respect to time as it will take a lot of time to swap process when system is in running state.
    - Either we should go for non-contiguous allocation, which means process can be divided into parts and different parts can be allocated in different areas.

Non-Contiguous Memory allocation(Paging)

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentation

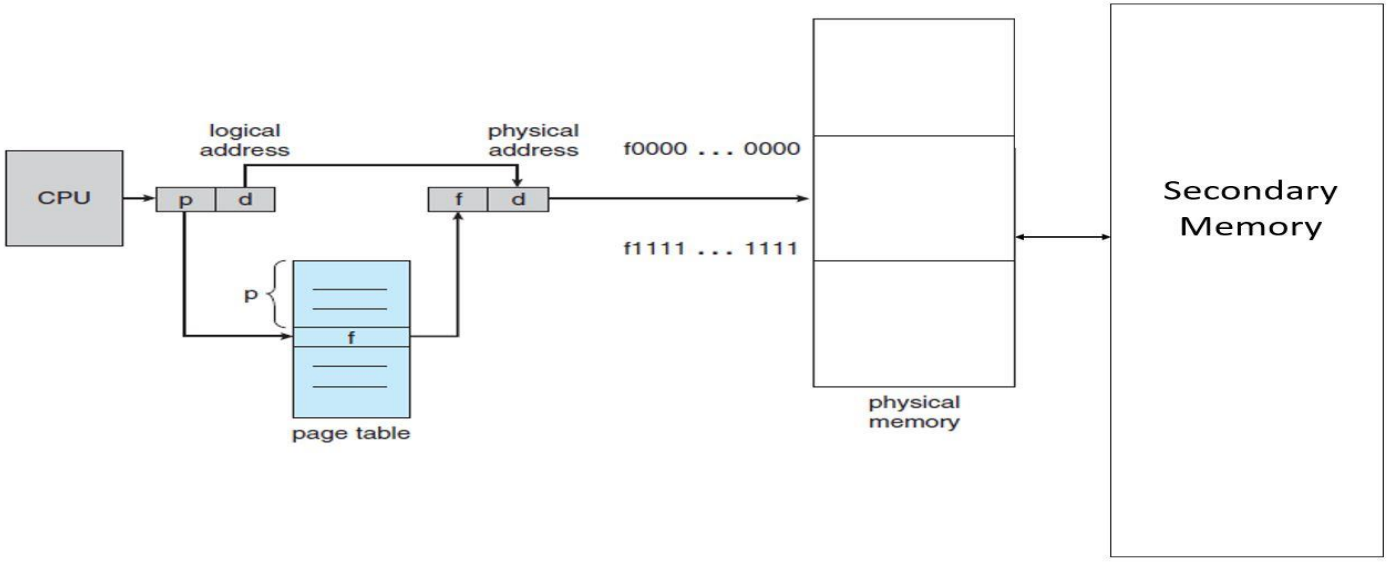


- Secondary memory is divides into fixed size partition(because management is easy) all of them of same size called pages(easy swapping and no external fragmentation).

- Main memory is divided into fix size partitions (because management is easy), each of them having same size called frames(easy swapping and no external fragmentation).

- Size of frame = size of page

- In general number of pages are much more than number of frames (approx. 128 time)



Translation process

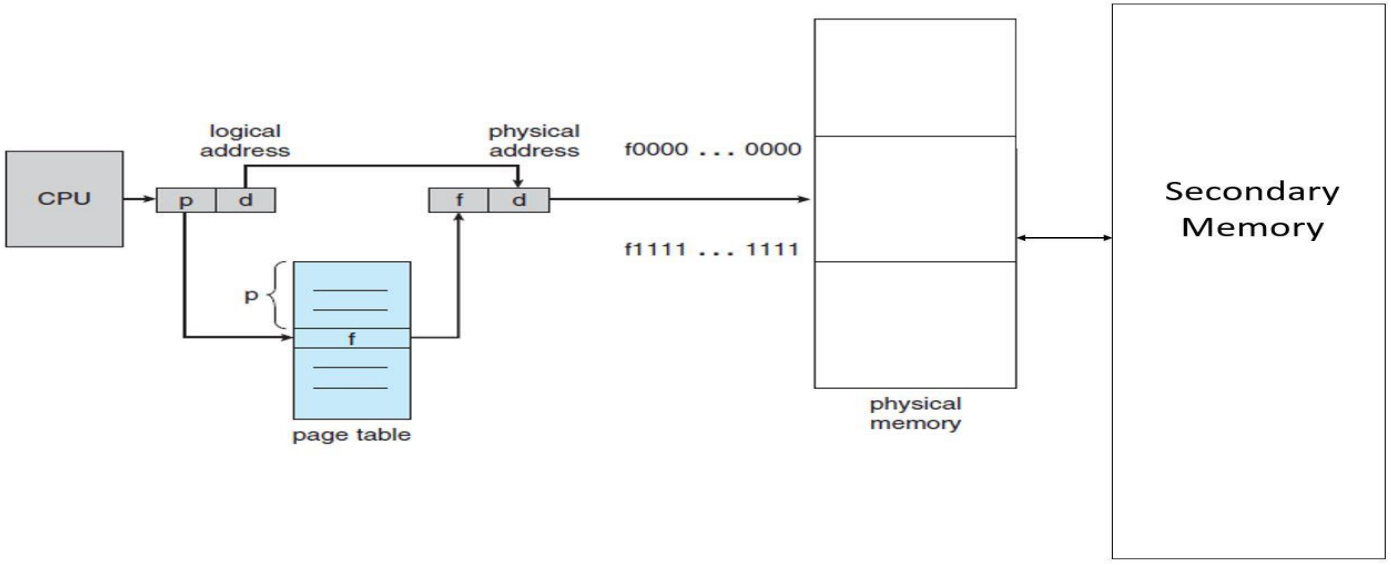
1. CPU generate a logical address is divided into two parts - **p** and **d**
    1. where p stands for page no and d stands for instruction offset.

2. The **page number(p)** is used as an index into a **Page table**

3. **Page table base register(PTBR)** provides the base of the page table and then the corresponding page no is accessed using p.

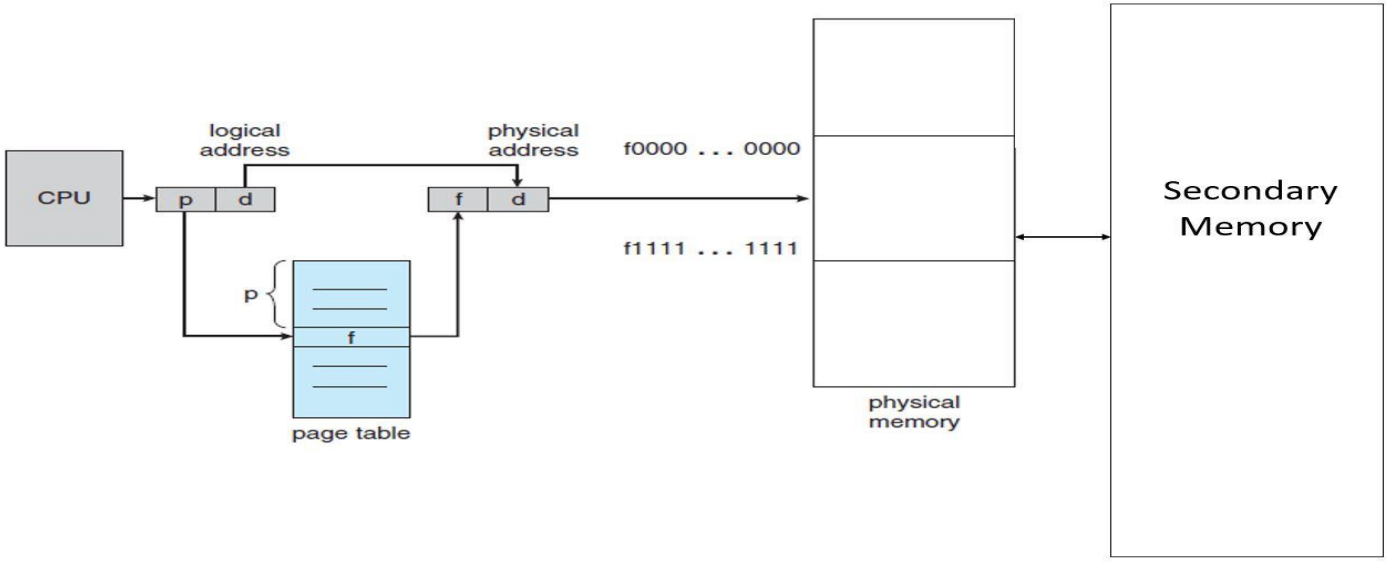
4. Here we will finds the corresponding frame no (the base address of that frame in main memory in which the page is stored)

5. Combine corresponding frame no with the instruction offset and get the physical address. Which is used to access main memory.



Page Table

- Page table is a data structure not hardware.
- Every process have a separate page table.
- Number of entries a process have in the page table is the number of pages a process have in the secondary memory.
- Size of each entry in the page table is same it is corresponding frame number.
- Page table is a data structure which is it self stored in main memory.



- **Advantage**
  - Removal of External Fragmentation

- **Disadvantage**
  - Translation process is slow as Main Memory is accessed two times(one for page table and other for actual access).
  - A considerable amount of space a waisted in storing page table(meta data).
  - System suffers from internal fragmentation(as paging is an example of fixed size partition).

Q. When does page fault occur? **[Asked in L&T InfoTech]**

(A) The page is present in memory.

(B) The deadlock occurs.

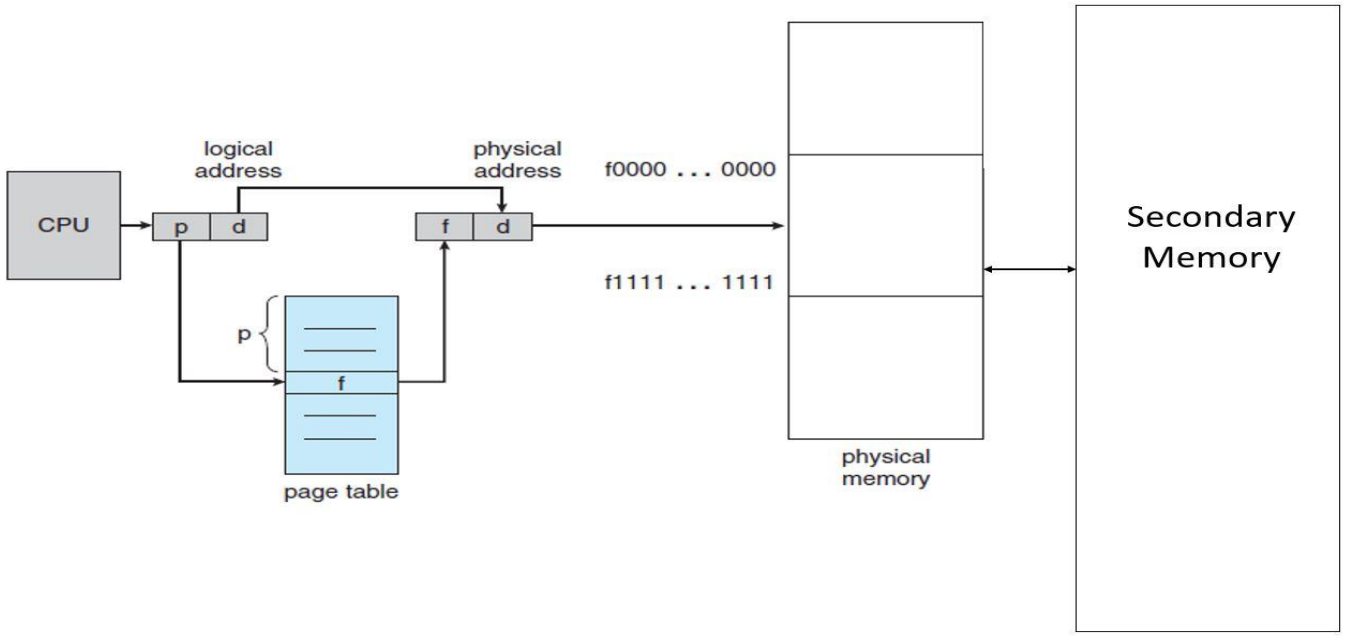
(C) The page does not present in memory.

(D) The buffering occurs.

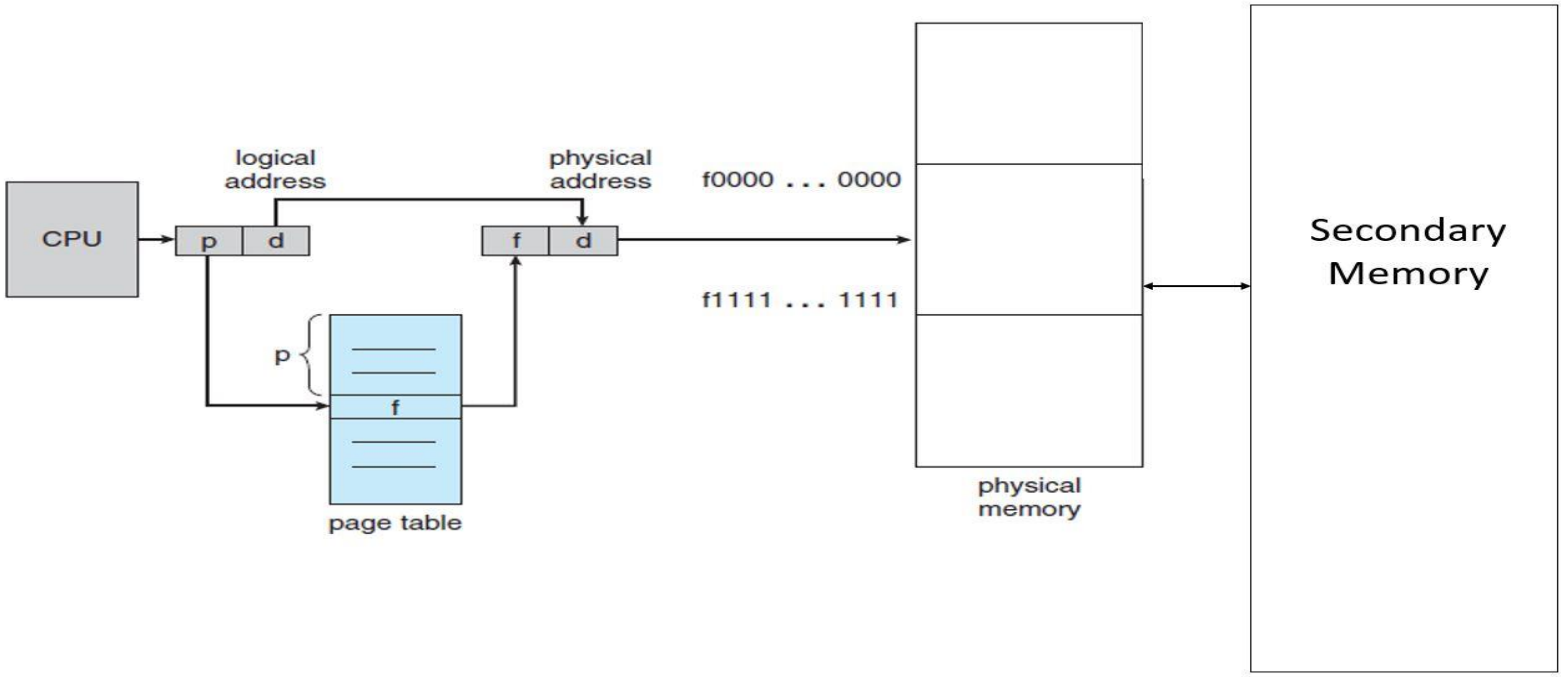
Answer: (c)

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.



- **<u>Pure Demand Paging/Demand Paging:</u>** 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.



Q Which of the following statements is false?

(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

Q. Which of the following are the advantage of Demand Paging? **[Asked in Tech Mahindra 2018]**

(A) Large virtual memory.

(B) More efficient use of memory.

(C) There is no limit on degree of multiprogramming.

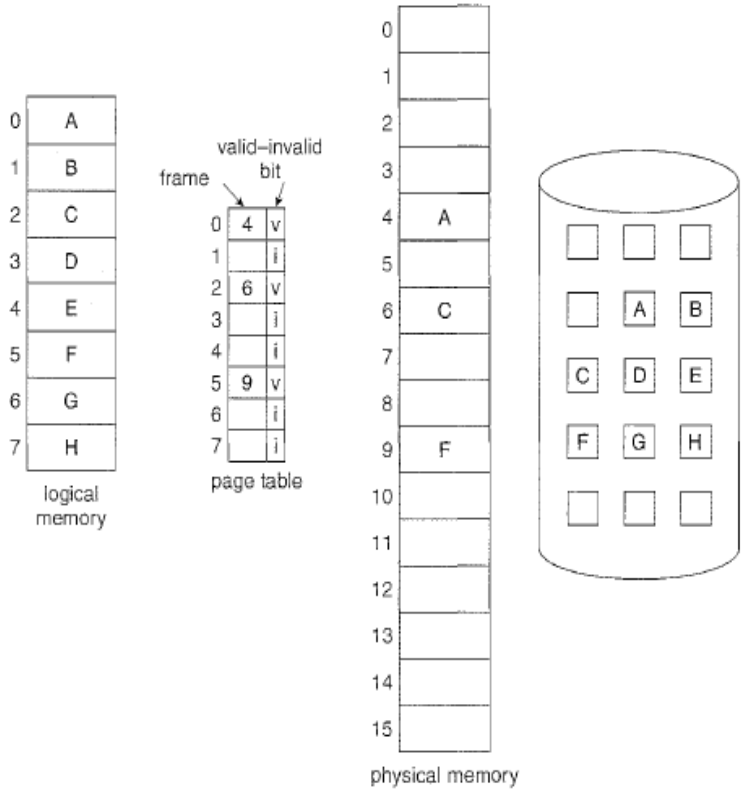
(D) All of the above

Ans : D

Valid -invalid bit scheme

- 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.

- 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.



Q. Which one of the following statements is not correct regarding the usage of virtual memory? **[Asked in Hexaware 2021]**

(A) To free user programs from the need to carry out storage allocation and to permit efficient sharing of the available memory space among different users

(B) To make program independent of the configuration and capacity of the physical memory for their execution

(C) To achieve higher CPU performance

(D) To achieve the very low access time and cost per bit that is possible with a memory hierarchy

Answer : C

Q Which of the following is incorrect for virtual memory?

a) Large programs can be written

b) More I/O is required

c) More addressable memory available

d) Faster and easy swapping of process

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:

(A) multitasking

(B) virtual memory

(C) paging system

(D) none of the above

Q._____ is a scheme where only a portion of the job's address space is actually loaded into physical memory.

(A) Swapping

(B) Virtual memory

(C) Kernel Swap

(D) SPOOL

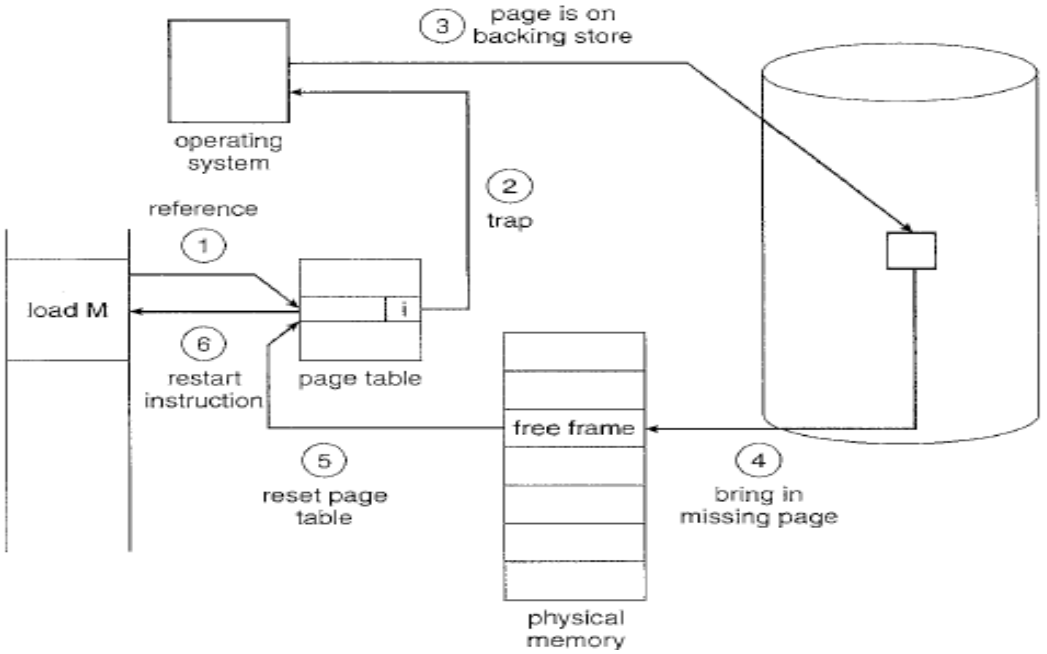
(E) None of these

Answer : B

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

- Page Fault Handling
- Allocating Frames
- Disk Operation
- Updating Tables
- Restarting the Process
- Instruction Restart Capability



Q A page fault

(A) is an error specific page.

(B) is an access to the page not currently in memory.

(C) occur when a page program occur in a page memory.

(D) page used in the previous page reference.

Q A page fault .....

(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

Q. While executing a program, if the program references a page which is not available in the main memory then it is known as? **[Asked in TCS NQT 2019]**

(A) Demand Paging

(B) Frame Fault

(C) page fault

(D) processor fault

Ans : C

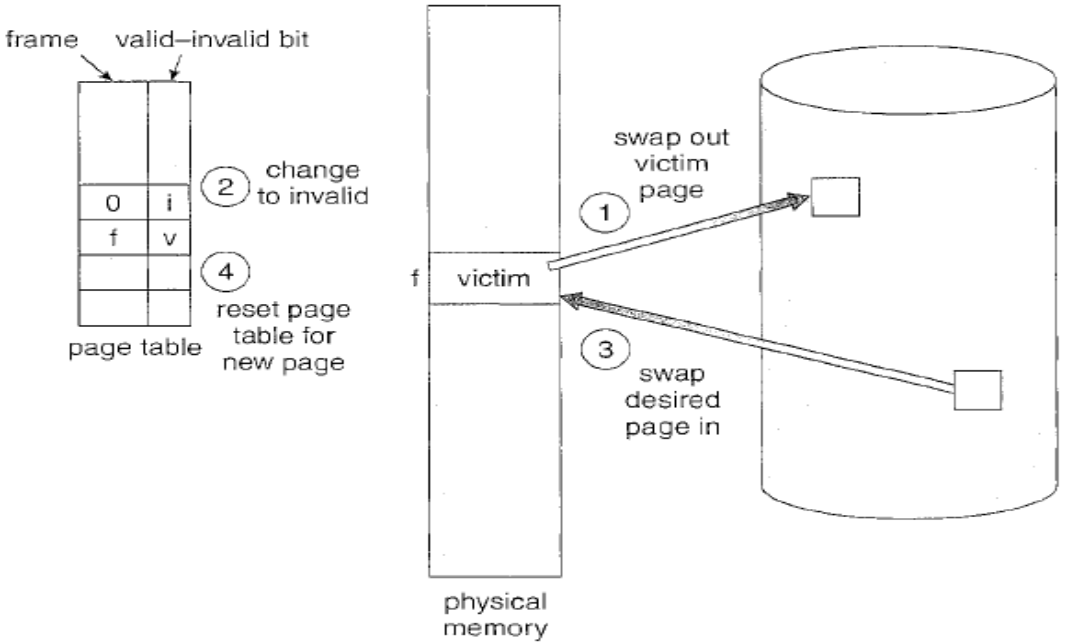
Q In a paged memory, the page hit ratio is 0.40. The time required to access a page in secondary memory is equal to 120 ns. The time required to access a page in primary memory is 15 ns. The average time required to access a page is _____.

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

Page Replacement

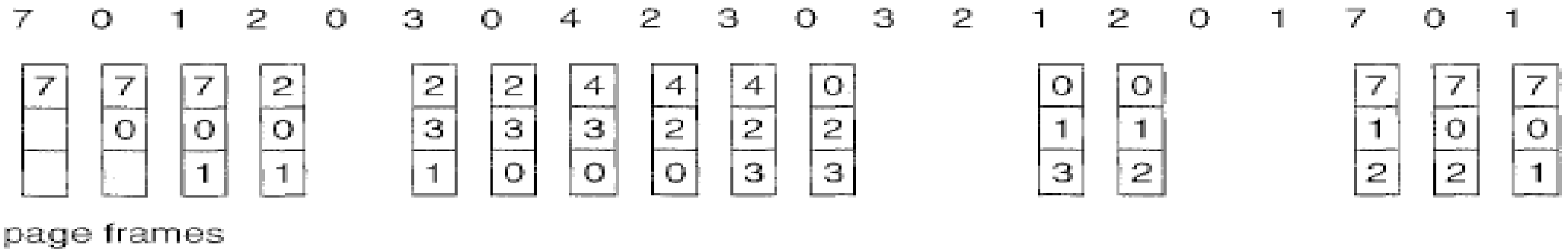
- With the increase in multiprogramming each process will get less amount of space in main memory so, the rate of page faults may rise. thus, to reduce the degree of multiprogramming the operating system swaps out processes from the memory freeing the frames and thus the process that requires to execute 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.



Page Replacement Algorithms

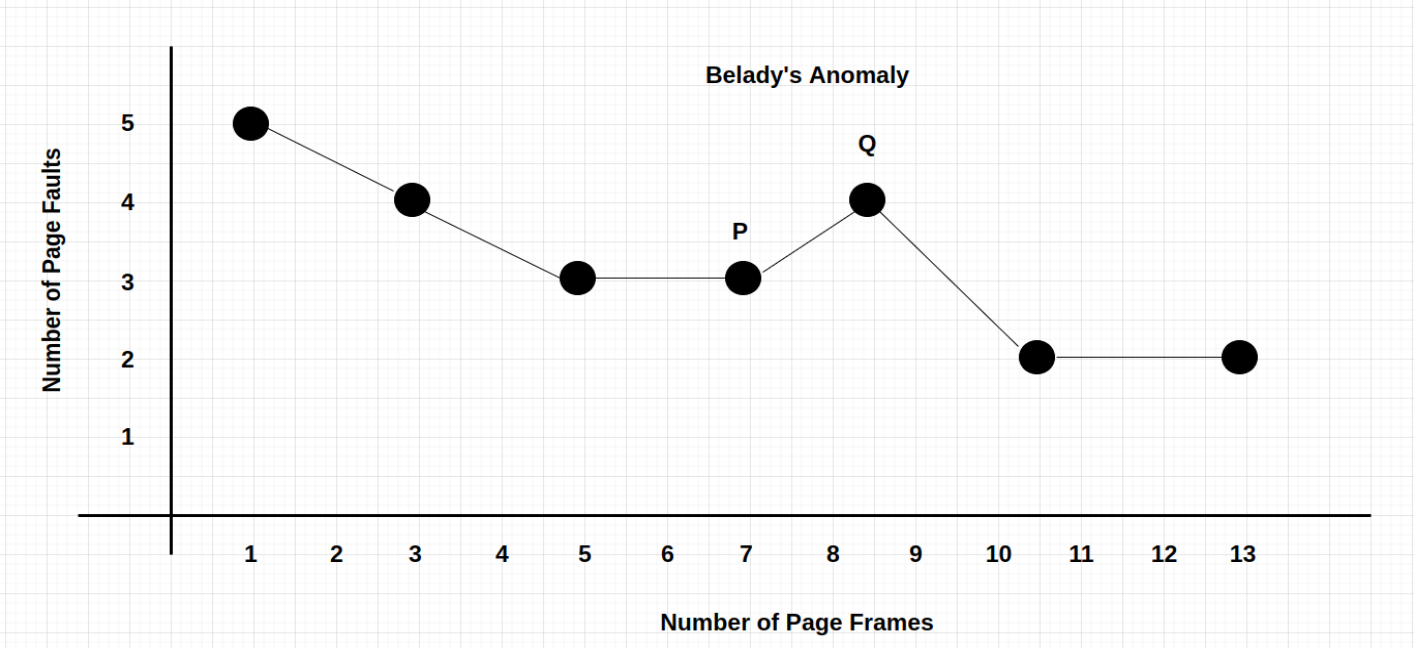
- **First In First Out Page Replacement Algorithm: -** A FIFO replacement algorithm associates with each page, the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. i.e. the first page that came into the memory will be replaced first.



- 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. 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?

(a) LRU (Least Recently Used

(b) OPT (Optimal Page Replacement)

(c) MRU (Most Recently Used)

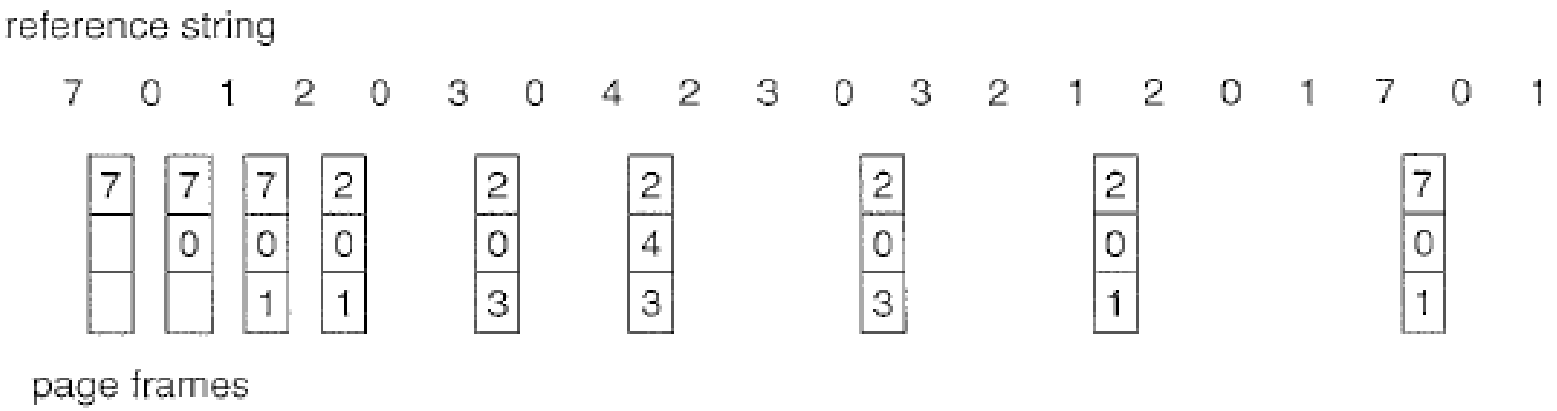
(d) FIFO (First In First Out)

Q In which one of the following page replacement policies, Belady's anomaly may occur?

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

Optimal Page Replacement Algorithm

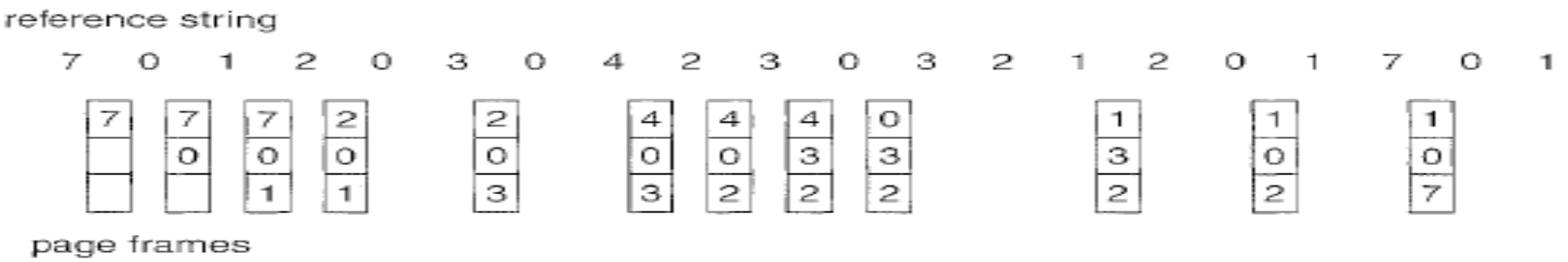
- Replace the page that will not be used for the longest period of time. It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.
- Unfortunately,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

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.



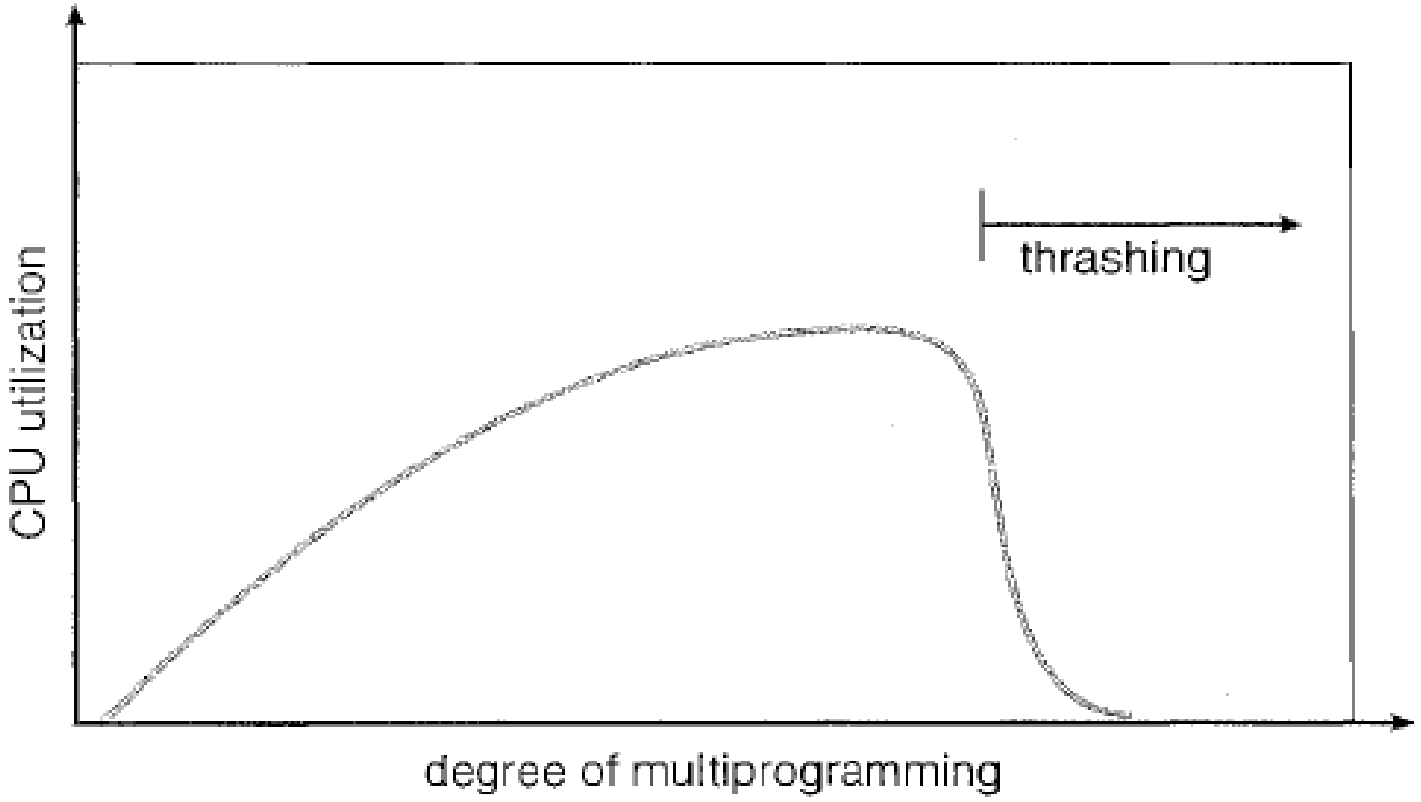
- LRU is much better than FIFO replacement. The LRU policy is often used as a page-replacement algorithm and is considered to be good. LRU also does not suffer from Belady's Anomaly.

Q Consider a demand paging system with four page frames (initially empty) and LRU page replacement policy. For the following page reference string, the number of page fault are?

7, 2, 7, 3, 2, 5, 3, 4, 6, 7, 7, 1, 5, 6,1

Thrashing

- A process is thrashing if it is spending more time paging than executing. High paging activity is called Thrashing.



Causes of Thrashing

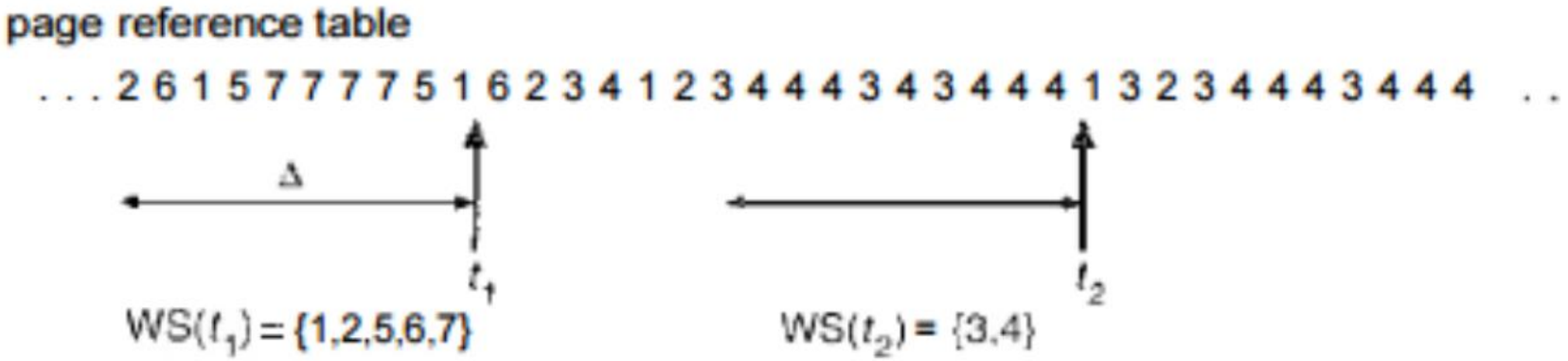
- When CPU utilization is low, an increase in the number of processes (multiprogramming) may lead to more page faults as processes compete for memory.

- This excessive paging causes a queue at the paging device, emptying the ready queue and further dropping CPU utilization. In response, the CPU scheduler may increase multiprogramming, worsening the situation.

- This cycle, known as thrashing, significantly decreases system throughput as processes spend most of their time managing memory rather than executing, leading to a sharp drop in overall performance.

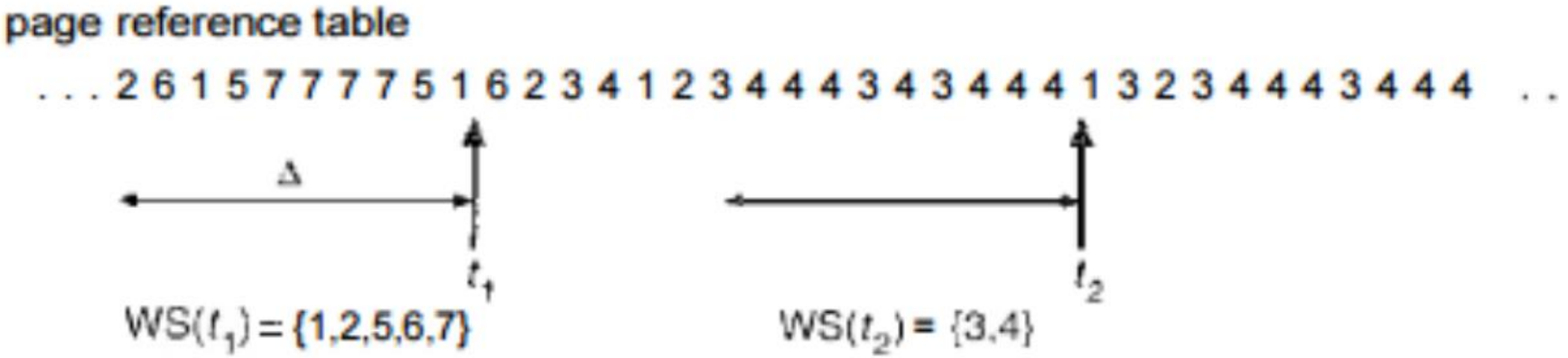
Solution (The Working Set Strategy)

- 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.
- 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 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.



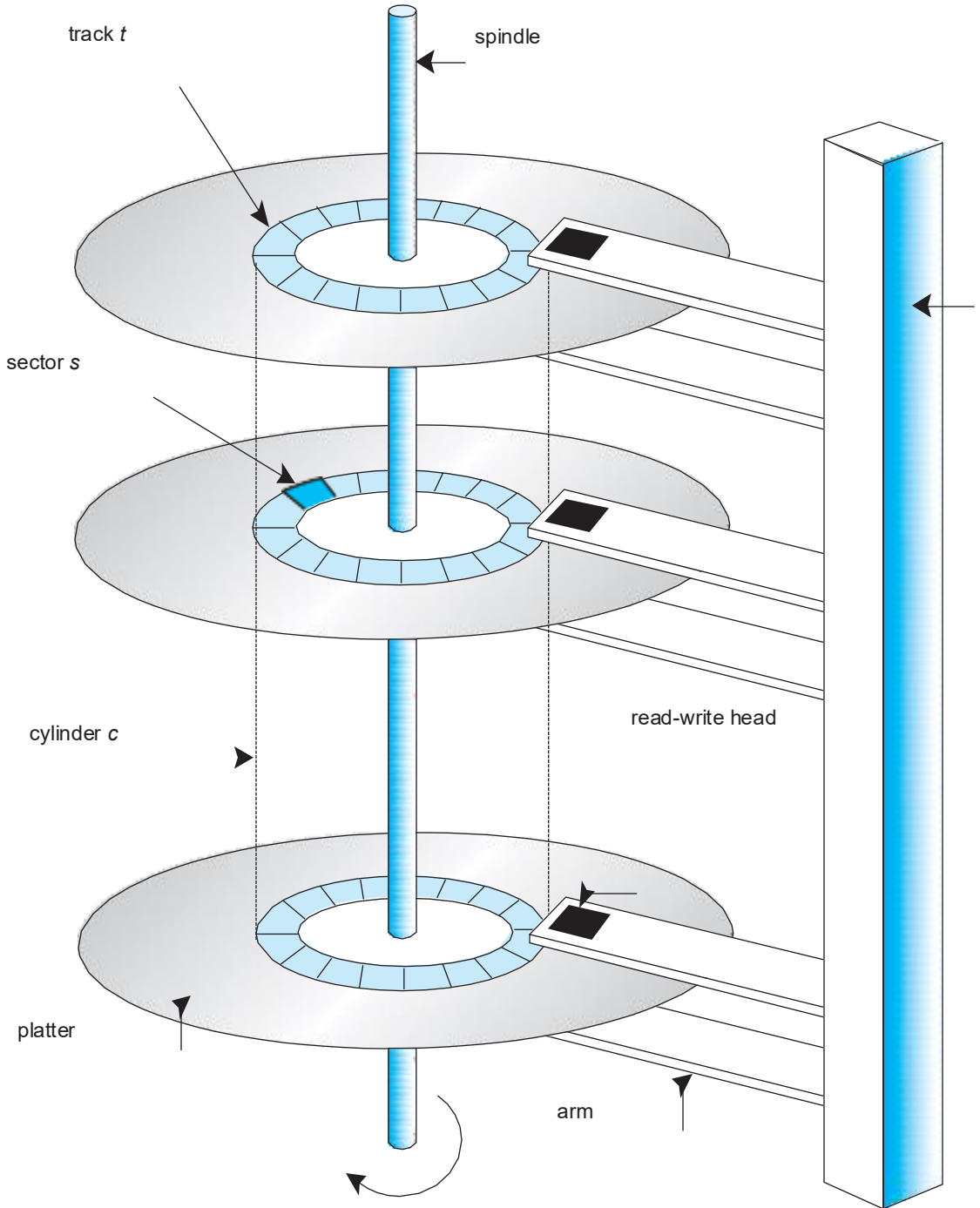
Q Working set model is used in memory management to implement the concept of

(A) Swapping

(B) Principal of Locality

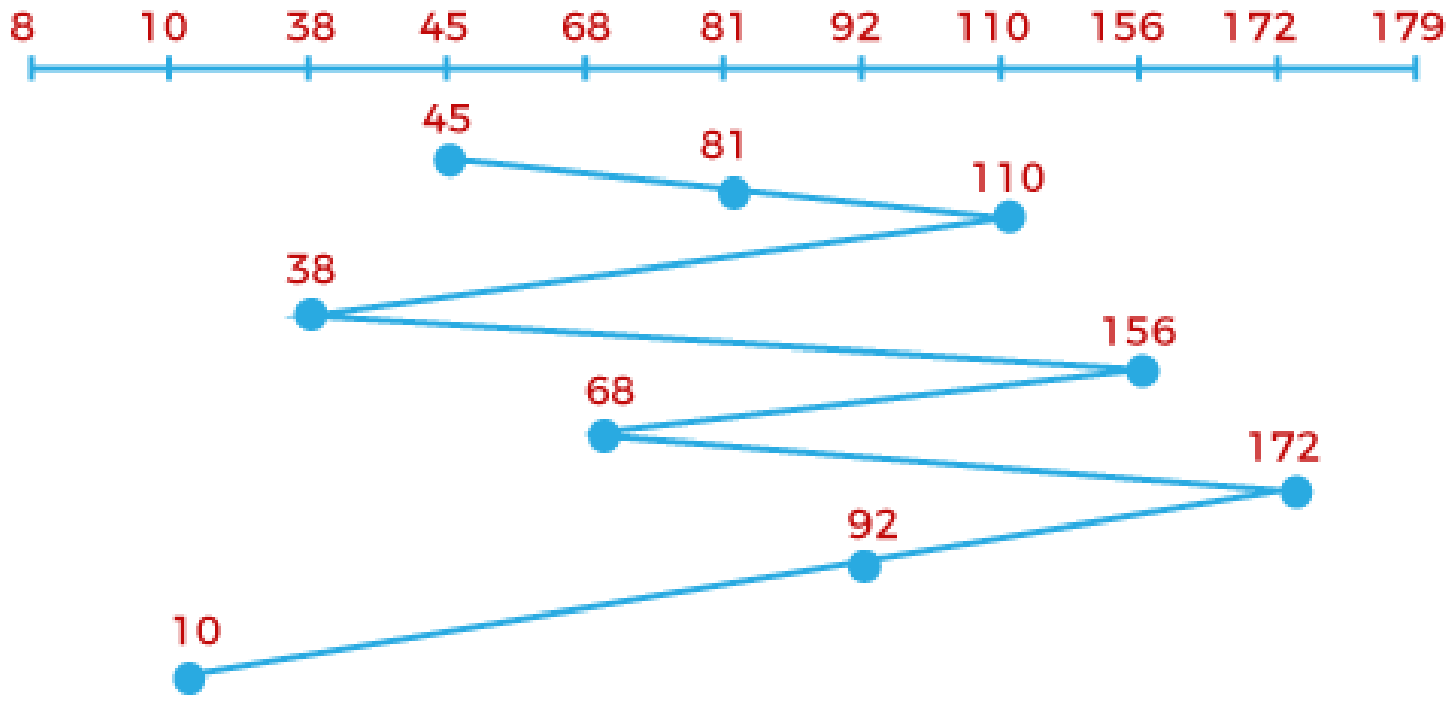
(C) Segmentation

(D) Thrashing


track *t*
spindle
sector *s*
cylinder *c*
read-write head
platter
arm

Disk scheduling

- When one i/o 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 Disk Scheduling Algorithm

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 the fastest service.

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 possible service

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.

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.

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

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: -

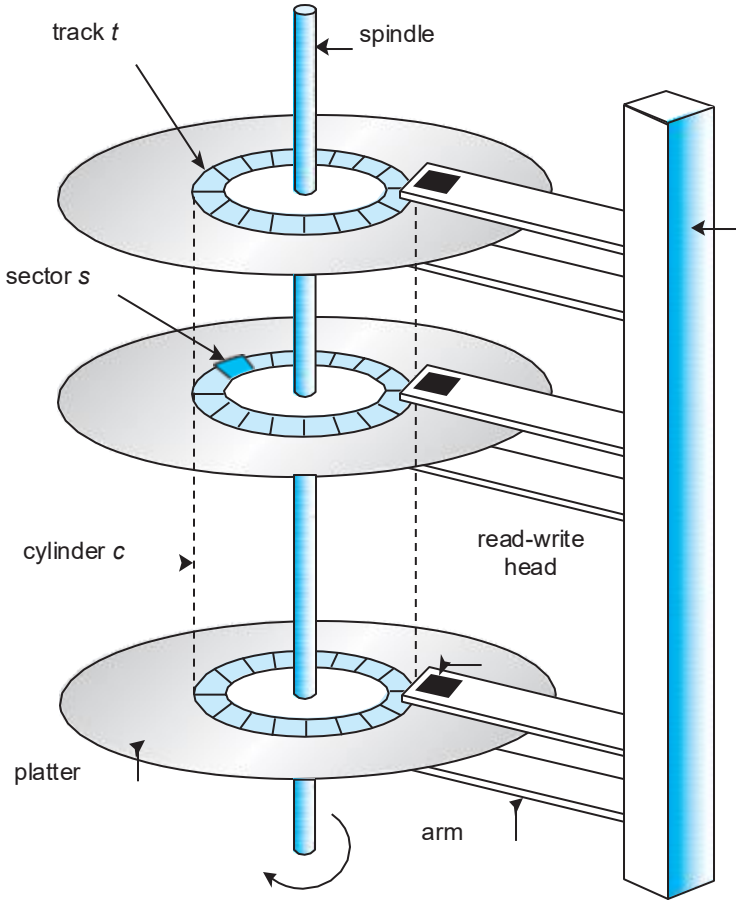
- Overhead to find the last request and go to initial position is more
- Should not be used in case of more load.

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.


track t
spindle
sector s
cylinder c
read-write
head
platter
arm

- **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.

Total Transfer Time = Seek Time + Rotational Latency + Transfer Time

- Total time will be = (File Size/Track Size) *time taken to complete one revolution.

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 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

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.

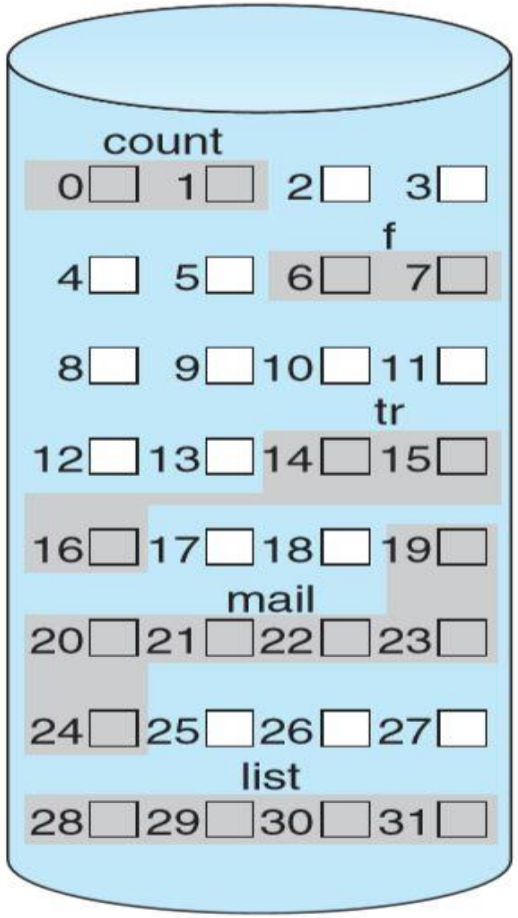
Q. A file is a sequence of? **[Asked in E-Litmus 2018]**

(A) Bits
(B) Bytes
(C) Lines
(D) All of the above

Ans : D

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

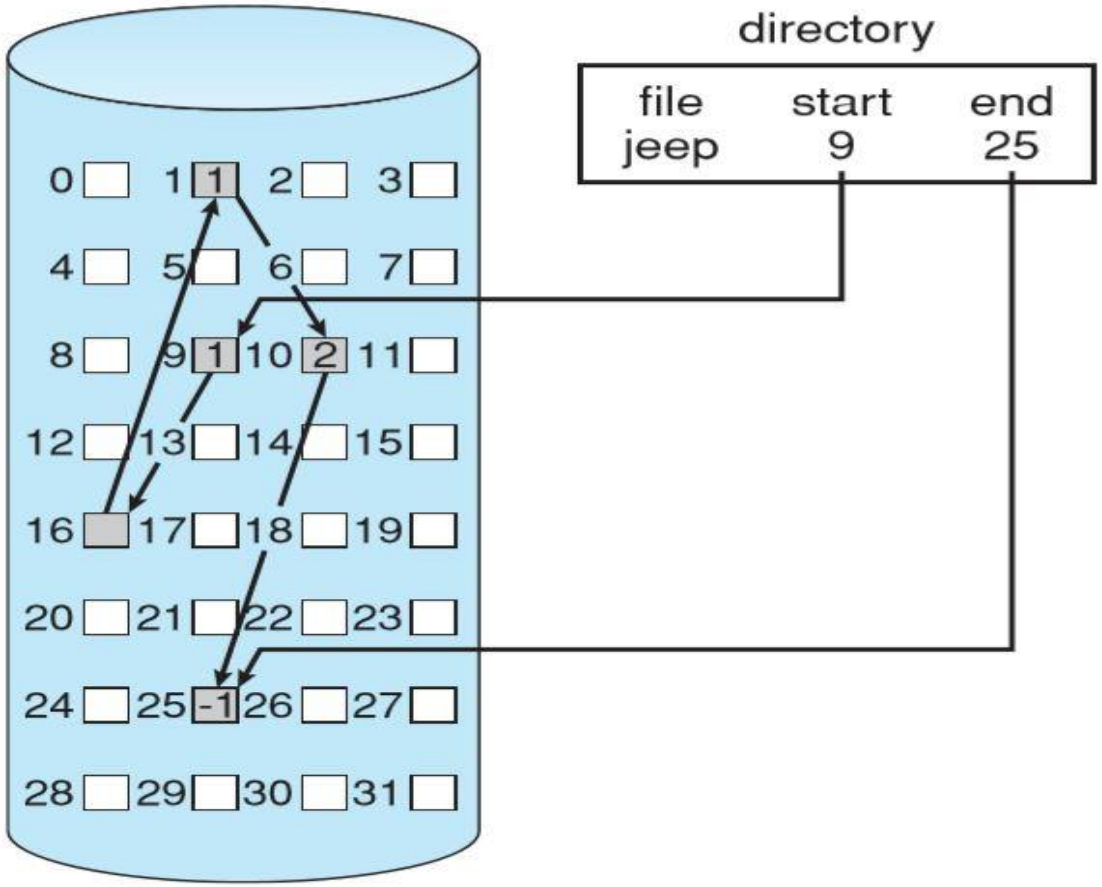
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- **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

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.

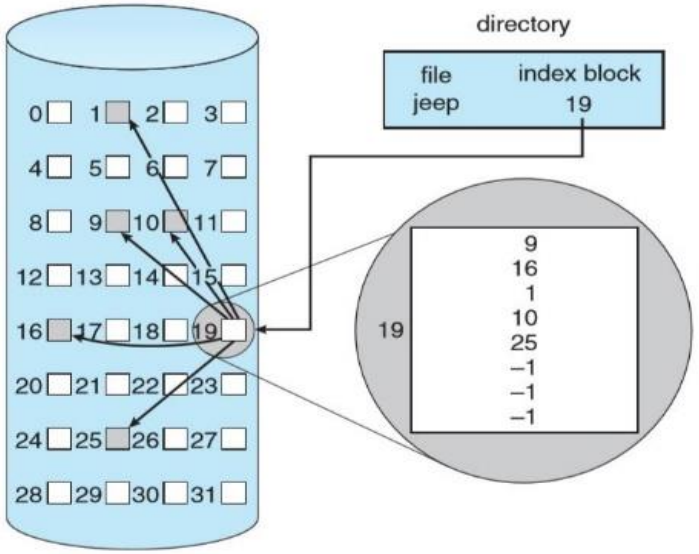
- **Disadvantage: -**
  - To find the i^{th} block of a file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block. Each access to a pointer requires a disk read.

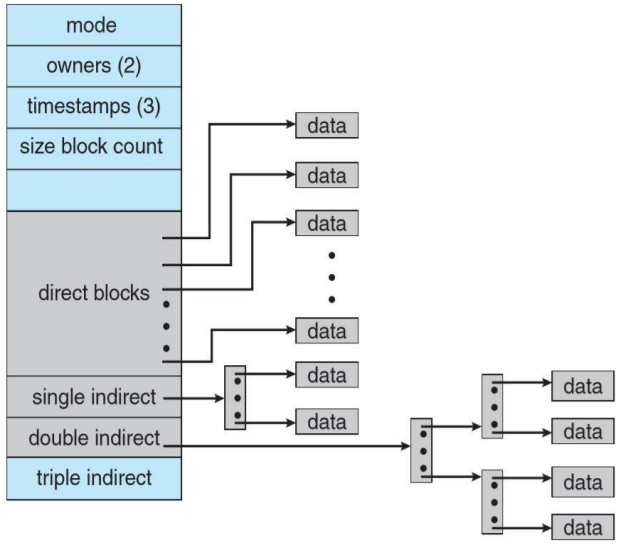
  - Another disadvantage is the space required for the pointers, so each file requires slightly more space than it would otherwise.

  - 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 i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} 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 Space Allocation, Which of the following ways are correct to allocate disk space to files? **[Asked in IBM 2021]**

(A) Contiguous Allocation

(B) Linked Allocation

(C) Indexed Allocation

(D) All of the above

Ans : C

Q. A file system with 300 GB uses a file descriptor with 8 direct block address. 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 **[Asked in L&T InfoTech]**

(A) 3 Kbytes

(B) 35 Kbytes

(C) 280 Bytes

(D) 290 Bytes

Answer (B)

Q._____ is a sequence of bytes organized into blocks that are understandable by the machine. **[Asked in IBM 2021]**

(A) object file

(B) source file

(C) text file

(D) None of the above

Ans : A

Q. What is true about Ordinary files? **[Asked in Tech Mahindra 2021]**

(A) These are the files that contain user information.

(B) These files contain list of file names and other information related to these files

(C) These files represent physical device like disks, terminals, printers, networks, tape drive etc.

(D) All of the above

Ans : A

Q. What is true about Directory files? **[Asked in TCS NQT 2019]**

(A) These files represent physical device like disks, terminals, printers, networks, tape drive etc.

(B) These may have text, databases or executable program.

(C) These files contain list of file names and other information related to these files.

(D) All of the above

Ans : C

Q. The user can load and execute a program but cannot copy it. This process is?
[Asked in Cognizant 2020]

(A) Execution

(B) Appending

(C) Reading

(D) Updating

Answer: A

Q. When access is granted to append or update a file to more than one user, the OS or file management system must enforce discipline. This is _____ **[Asked in Hexaware 2021]**

(A) Simultaneous access

(B) Compaction

(C) External Fragmentation

(D) Division

Answer: a

Q. Which of the following is not a part of the usage information? **[Asked in E-Litmus 2021]**

(A) data created

(B) identity of creator

(C) Owner

(D) last date modified

Answer: c

Q. Which of the following is not an appropriate criterion for file organisation? **[Asked in IBM 2020]**

(A) Larger access time

(B) ease of update

(C) simple maintenance

(D) economy of storage

Answer: a