Page No.
Date / /

* Operating System

=> An operating system is software that manages the computer hardware, as well as providing an environment for application programs to run.

For eg. WINDOWS, LINUX/UNIX, MAC-OS etc.

* Purpose of an OS

=> To provide an environment for an user to execute programs on computer hardware in a convenient & efficient manner.

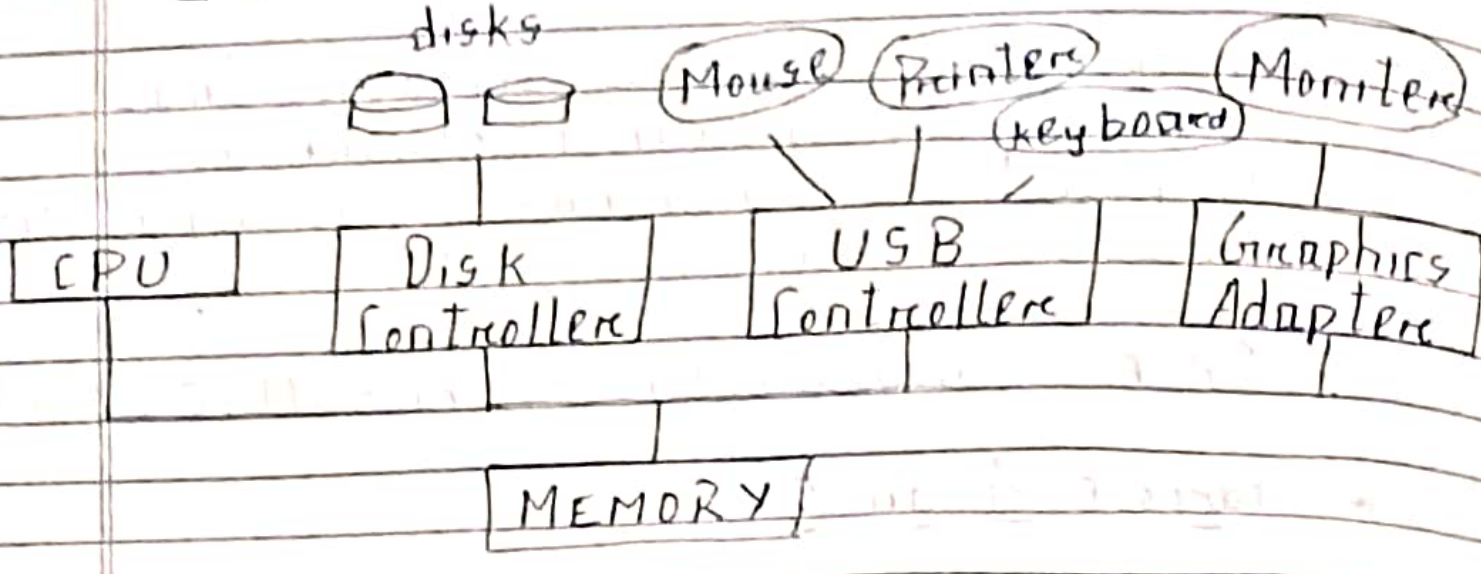
=> To allocate the separate resources of the computer as needed to perform the required tasks.

=> As a control program, it serves 2 major func's :-

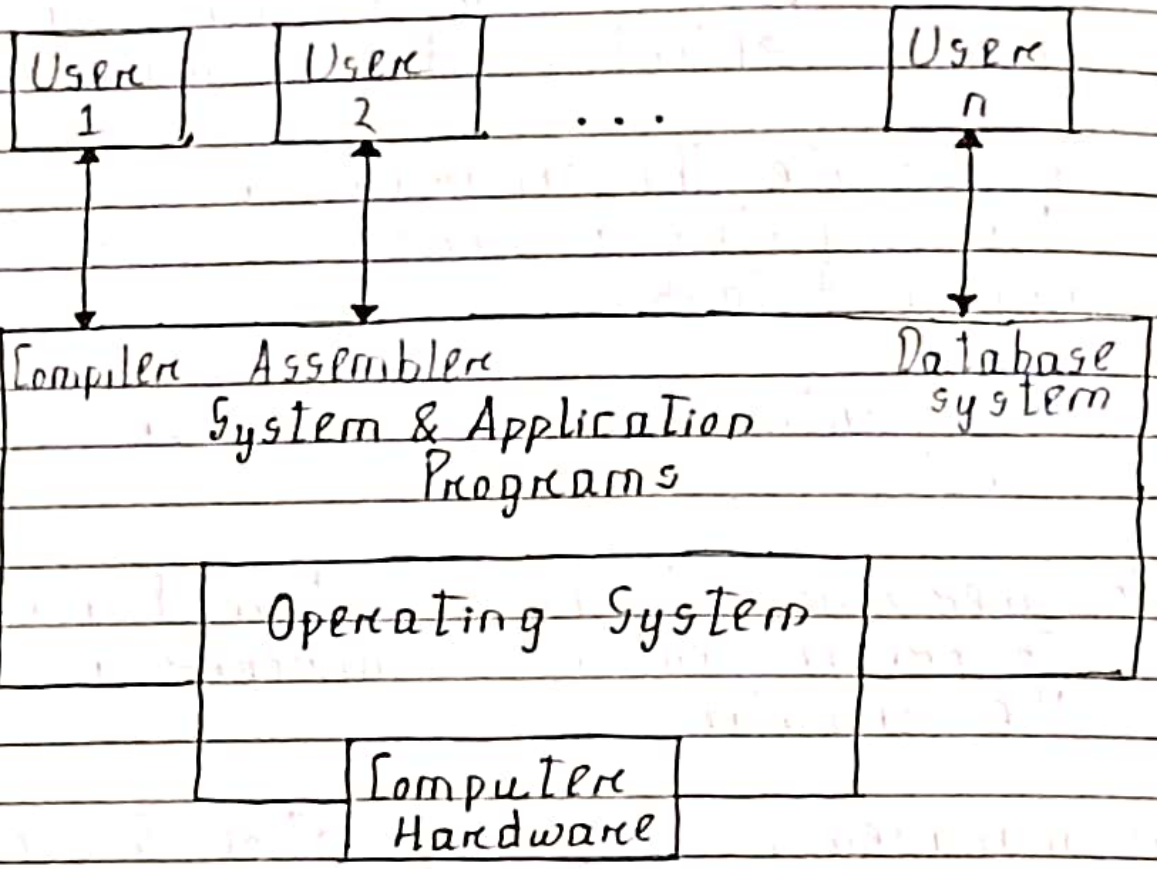
i) supervision of the execution of user program to prevent errors & improper use of the computer.

ii) management of the operation & control of I/O devices.

* Computer System Organization



* Computer System Structure



Page No.
Date / /

1) Hardware → provides basic computing resources (CPU, memory, I/O devices)

2) OS → controls & coordinates the use of the hardware among the various application programs for the various users

3) Application programs → Define the ways in which the system resources are used to solve the computing problems of the users.

4) Users → people, machines, other computers

* Computer System Operation

- I/O devices & the CPU can execute concurrently.
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers.
- I/O is from the device to local buffer of controller.

Page No.
Date / /

* Common Func^n's of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt
- An OS is interrupt driven

Interrupt Timeline

Page No.
Date: / /

* I/O Structure

- Synchronous

After I/O starts, control returns to user program only upon I/O completion.

Wait instruction idles the CPU until the next interrupt.

Wait loop (contention for memory access)

At most one I/O request is outstanding at a time, no simultaneous I/O processing.

- Asynchronous

After I/O starts, control returns to user program without waiting for I/O completion.

System call - request to the OS to allow user to wait for I/O completion.

Device-status table contains entry for each I/O device indicating its type, address, & state

Page No.
Date : / /

* Storage Device Hierarchy

Registers
↑ ↓
Cache
↑ ↓
Main Memory
↑ ↓
Solid-state disk
↑ ↓
Magnetic disk
↑ ↓
Optical disk
↑ ↓
Magnetic tapes

* Multiprogramming OS

This type of OS is used to execute more than one jobs simultaneously by a single processor. It increases CPU utilization by organizing jobs so that the CPU always has one job to execute.

The concept of multiprogramming is as follows :-

=> All the jobs that enter the system are stored in the job pool. The OS loads a set of jobs from job pool into main memory & begins to execute.

=> During execution, the job may have to wait for some task, such as an I/O operation, to complete. In a multiprogramming system, the OS simply switches to another job & executes.

=> When that job needs to wait, the CPU is switched to another job, & so on.

=> When the first job finishes waiting & it gets the CPU back.

=> As long as at least one job needs to execute the CPU is never idle.

* Multitasking OS

=> It is a logical extension of multiprogramming. It provides extra facilities such as -

- Faster switching bet^n multiple jobs to make processing faster.
- Allows multiple users to share computer system simultaneously.
- The users can interact with each job while it is running.

These systems use a concept of virtual memory for effective utilization of

memory space. Hence, in this OS, no jobs are discarded. Each one is executed using virtual memory concept.

* Multiprocessor OS

Also known as parallel OS or tightly coupled OS Have more than one processor in close communication that sharing the computer bus, the clock & sometimes memory & peripheral devices It executes multiple jobs at same time & makes the processing faster.

Advantages :-

=> By increasing the no. of processors, the system performs more work in less time. The speed-up ratio with N processors is less than N.

=> Can save more money than multiple single-processor systems, because they can share peripherals, mass storage & power supplies

=> If one processor fails to done its task, Then each of the remaining processors must pick up a share of the work of the failed processor. The failure of one processor will not halt the system,

only slow it down.

Multiprocessor OS are classified into 2 categories :-

1) Symmetric
2) Asymmetric

1) In symmetric multiprocessor OS, each processor runs an identical copy of the OS, & these copies communicate with one another as needed.

2) In asymmetric multiprocessor OS, a processor called master processor that controls other processors called slave processor. Thus, it establishes master-slave relationship.

Page No
Date : / /

* OS Services

1) Program Execution

The purpose of computer systems is to allow the user to execute programs. So the OS provides an environment where the user can conveniently run programs.

2) I/O Operations

Each program requires an input & produces output. This involves the use of I/O. So, the OS are providing I/O makes it convenient afor the users to run programs.

3) File System Manipulation

The output ~~program~~ of a program may need to be written into new files or input taken from some files. The OS provides this service.

4) Communications

The processes need to communicate with each other to exchange information during execution. It may be betⁿ processes running on the same computer or running on the diff computers. Communication can be occur in 2 ways :- i) shared memory
ii) message passing

5) Error Detection

An error in one part of the system may cause malfunctioning of the complete system. To avoid such a situation OS constantly monitors the system for detecting the errors.

* Functions of OS

1) Process Management

A process is a program in execution. A process needs certain resources, including CPU time, memory, files & I/O devices, to accomplish its task.

OS is responsible for the following activities in connection with process management.

- Process creation & deletion.
- Process suspension & resumption
- Provision of mechanisms for:
  - process synchronization
  - process communication

2) Main-Memory Management

Page No.
Date: / /

Memory is a large array of words or bytes each of its own address. It is a repository of quickly accessible data shared by the CPU & I/O devices.

Main memory is a volatile storage device. It loses its contents in the case of system failure.

OS is responsible for the following activities in connections with memory management:

File creation & deletion

- keep track of which parts of memory are currently being used & by whom.
- Decide which processes to load when memory space becomes available.
- Allocate & de-allocate memory space as needed.

3) File Management

A file is a collection of related info defined by its creator. Commonly, files represent programs & data.

OS is responsible for the following activities in connections with file management :-

- File creation & deletion

- Directory creation & deletion.
- Support of primitives for manipulating files & directories.
- Mapping files onto secondary storage.
- File backup on stable storage media.

4) I/O System Management

Consists of :-

- A buffer-caching system.
- A general device-driver interface
- Drivers for specific hardware devices

* Dual-Mode Operation

=> Sharing system resources requires OS to ensure that an incorrect program cannot cause other programs to execute incorrectly.

=> Provide hardware support to differentiate bet^n atleast 2 mode of operations :-

a) User mode → execution done on befalf of a user.
b) Monitor mode → execution done on behalf (kernel mode) of OS.

PROCESS

Page No :
Date : / /

* Process

=> A process is sequential program in execution

=> It defines the fundamental unit of computation for the computer

=> Components of process are :
1) Object Program
2) Data
3) Resources
4) Status of the process execution

=> Object program i.e. code to be executed. Data is used for executing the program. While executing the program, it may require some resources. Last component is used for verifying the status of the process execution. A process can run to completion only when all requested resources have been allocated to the process.

* Difference bet^n Process & Program

1) Both are same beast with diff name or when this beast is sleeping (not executing) it is called program & when it is executing becomes process.

2) Program is a static object whereas a process is a dynamic object.

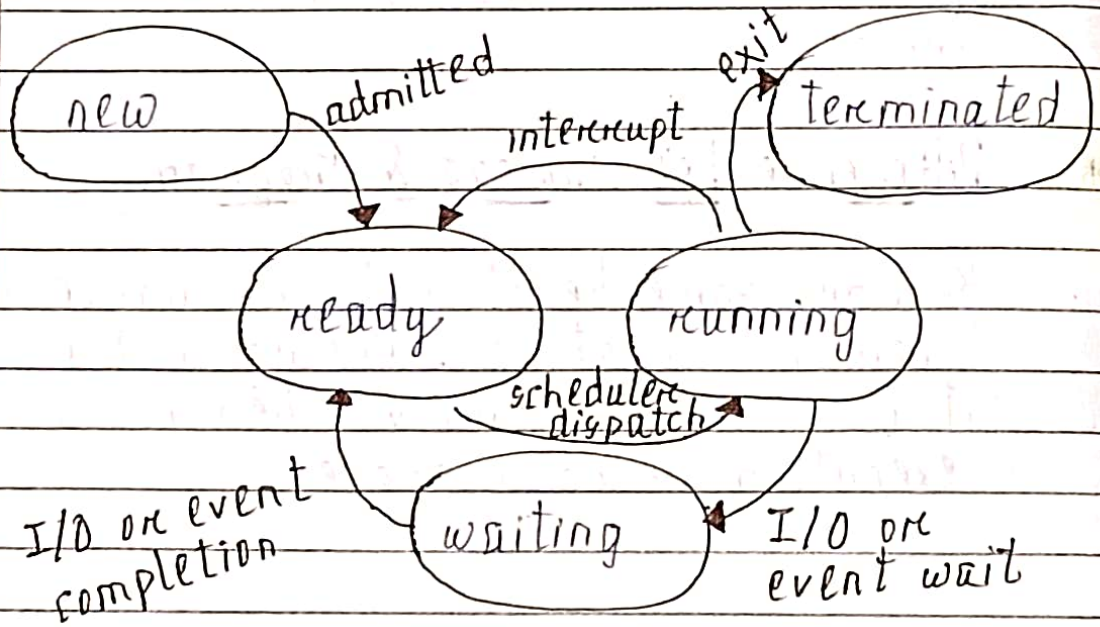
3) A process is an 'active' entity whereas a program is a 'passive' entity.

4) A program resides in secondary storage whereas a process resides in main memory.

5) The span time of a program is unlimited but the span time of a process is limited.

6) A program is an algorithm expressed in programming language whereas a process is expressed in assembly or machine language.

* Process State



Page No.
Date: / /

As a process executes, it changes state.

- New state : The process is being created.
- Running state : A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant.
- Blocked (or waiting) state : A process is said to be in blocked state if it is waiting for some event to happen such that as an I/O completion before it can proceed. A process is unable to run until some external event happens.
- Ready state : A process is said to be ready if it needs a CPU to execute. A ready state process is runnable but temporarily stopped running to let another process run.
- Terminated state : The process has finished execution.

* Process Control Block (PCB)

=> Each process contains the process control block (PCB). PCB is the DS used by the OS. OS groups all info that needs about particular process.

Pointer	Process State
Process Number	
Program Counter	
CPU Registers	
Memory allocation	
Event Information	
List of open files	

1) Pointer : Pointer points to another PCB. It is used for maintaining the scheduling list.

2) Process State : May be new, ready, running, waiting & so on.

3) Program Counter : Indicates the address of the next instruction to be executed for this process.

4) Event Information : For a process in the blocked state this field contains info concerning the event for which the process is waiting.

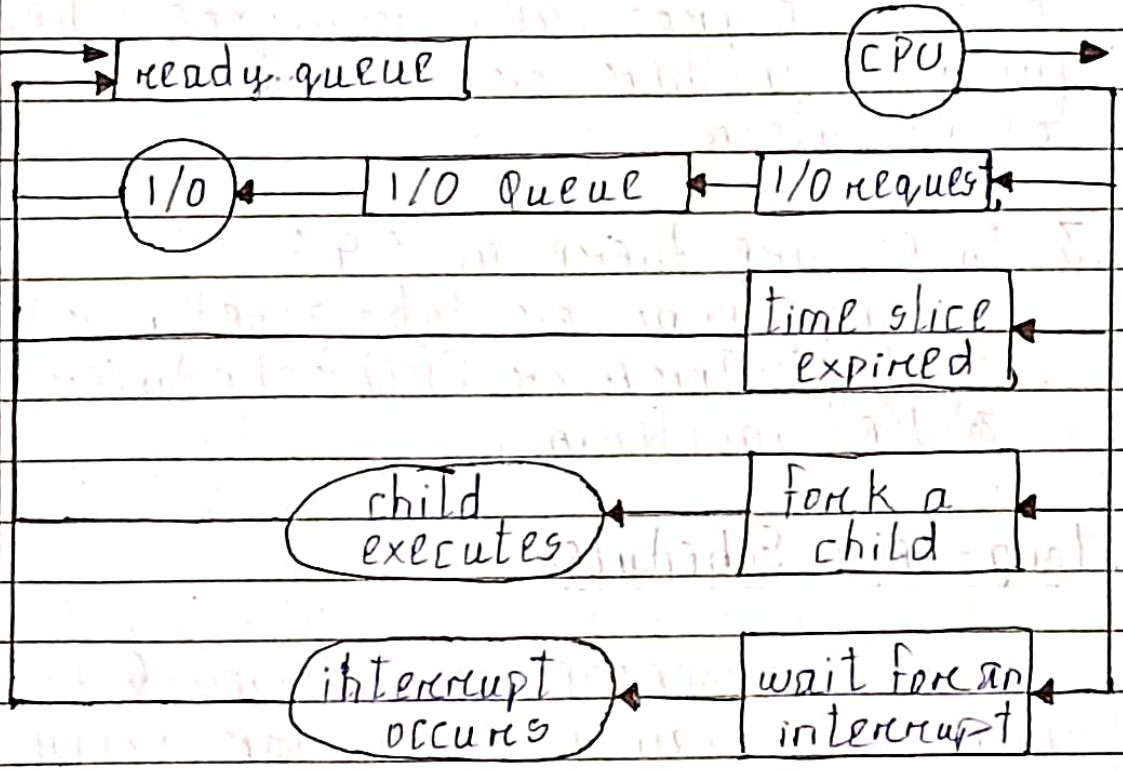
5) CPU Register : Indicates general purpose register, stack pointers, index registers & accumulators etc. number of register & type of register totally depends upon the computer architecture.

6) Memory Management Information :- May include the value of base & limit register Useful for deallocating the memory when the process terminates.

7) Accounting Information :- Includes the amount of CPU & real time used, time limits, job or process no.s, account no.s etc.

=> PCB also includes the info about CPU scheduling, I/O resource management, file management info, priority & so on. It simply serves as the repository for any info that may vary from process to process.

* Process Scheduling



Process Scheduling Queues

- **Job Queue** : Consists of all processes in the system. These processes are entered to the system as new processes.

- **Ready Queue** : This queue consists of the processes that are residing in main memory & are ready & waiting to execute by CPU. A ready queue header contains pointers to the first & final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

* Schedulers

A scheduler is a decision maker that selects the processes from one scheduling queue to another or allocates CPU. for execution.

3 types are there in OS :-
1) Long-term or Job scheduler
2) Short-term or CPU scheduler
3) Medium-term

1) Long-Term Scheduler

=> It selects processes from discs & loads them into main memory for execution.

=> It controls the degree of multiprogrammin

=> Because of the longer interval bet' executions, the long-term scheduler can afford to take more time to select a process for execution.

2) Short Term Scheduler

=> It selects a process from among the processes that are ready to execute & allocates the CPU.

=> It must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request.

3) Middle Term Scheduler

=> It schedules the processes as intermediate level of scheduling.

Context Switch

=> The context of a process is represented in the PCB of the process.

=> It includes the values in CPU register process state, memory management info, open files for the process & so on

Page No.
Date / /

=> When the CPU switches from one process to another, it is said that a context switch occurs.

=> During a context switch, the system must save the context of the old process & load the save context for the new process.

=> Context-switch time is pure overhead; the system does no ~~usual~~ useful work while switching.

Dispatcher

=> It is a small program that switches the processor from one process to another.

=> Dispatcher module gives control of the CPU to the process selected by the short-term scheduler.

=> Time taken by the dispatcher to stop one process & start another running is called the dispatch latency.

Functions :-

- get the new process from the scheduler.

- switch out the context of the current process.
- give CPU control to the new process.
- jump to the proper location in the new program to restart that program

Types of Scheduling

1) Pre-emptive

=> When a process is forcefully removed from CPU.

i) When a process switches from running to ready state → Pre-emptive.
ii) When a process switches from waiting to ready state → Pre-emptive

2) Non-pre-emptive

=> Process is not removed until they complete the execution.

i) When a process switches from running to waiting state → Pre-emptive
ii) When a process switches from running to terminated state → Pre-emptive

Page No
Date / /

* CPU Scheduling Algorithms

Scheduling Criteria

1) CPU Utilization :- Objective is to keep CPU as busy as possible. Its range from 0 to 100%. In practice, 40-90%.

2) Throughput :- Rate at which no of processes completed per unit time.

3) Waiting Time :- Amount of time a process has been waiting in the ready queue.

Waiting Time = Turn Around Time − Burst Time

4) Turn Around Time (TAT) :- The total time interval bet^n the time of process submission & time of its completion.

TAT = Completion Time − Arrival Time
= Waiting Time + Burst Time

5) Response Time :- Amount of time after which a process gets the CPU for the first time after entering ready queue.

Response Time = Time at which process first gets the CPU − Arrival Time

1) First-Come, First-Served (FCFS)

=> As the name suggests, the process which arrives first, gets executed first

=> Runs the processes in order they arrive at the short-term scheduler.

=> Removes a process from the processor only if it blocks (i.e. goes into the wait state) or terminates.

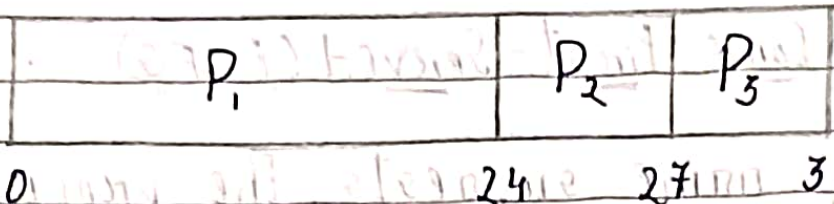
=> The FCFS scheduling algorithm is non-preemptive.

→ Once the CPU has been allocated to a process, that process keeps the CPU until it releases it either by terminating or by requesting I/O.

→ It is a troublesome algorithm for time-sharing systems.

Example:

Process	Arrival Time	Burst Time
P_1	0	24
P_2	0	3
P_3	0	3



Waiting Time (W.T.)

$P_1 \longrightarrow 0$
$P_2 \longrightarrow 24$
$P_3 \longrightarrow 27$

Avg. W.T. $= \frac{0+24+27}{3} = 17$

Avg. TAT $= \frac{24+27+30}{3} = 27$

- Convoy Effect

=> When using FCFS scheduling, if I/O bound (short CPU burst) processes are scheduled after CPU bound (long CPU burst) processes, the avg. waiting time increases.

Q-1 Consider the following set of processes, the processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5 at time 0.

P_1	10
P_2	1
P_3	2
P_4	1
P_5	5

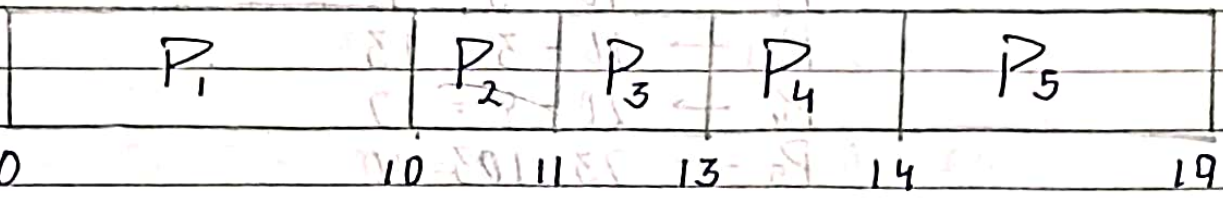
Page No.
Date : / /

Draw Gantt Chart that illustrate the execution of these processes using FCFS algorithm.

i) Calculate the waiting time of each process & avg. WT of processes.

ii) Calculate TAT of each process & avg. TAT of processes.

A-



W.T $P_1 \rightarrow 0$
$P_2 \rightarrow 10$
$P_3 \rightarrow 11$
$P_4 \rightarrow 13$
$P_5 \rightarrow 14$

$$\text{Avg. W.T.} = \frac{10+11+13+14}{5} = 9.6$$

$$\text{Avg. TAT} = \frac{10+11+13+14+19}{5} = 13.4$$

Q-2

Process	A.T.	B.T
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Page No.
Date / /

Calculate avg. WT & avg. TAT using FCFS algorithm.

A-

P_1	P_2	P_3	P_4	P_5

0 10 16 20 23 28

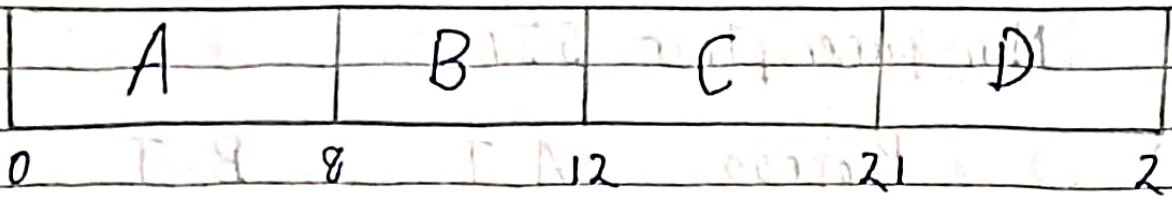
W.T
$P_1 \rightarrow 0$
$P_2 \rightarrow 10 - 3 = 7$
$P_3 \rightarrow 16 - 3 = 13$
$P_4 \rightarrow 20 - 8 = 12$
$P_5 \rightarrow 23 - 10 = 10$

$$\text{Avg. WT} = \frac{10 + 7 + 12 + 13}{5} = 13.8$$

$$\text{Avg. TAT} = \frac{10 + 16 + 20 + 23 + 28}{5} = 19.4$$

Q3 Suppose the scheduler is given 4 tasks, A, B, C, D. The tasks are inserted in order A, B, C, D. Find the avg. WT & avg. TAT of all processes.

Task	Units
A	8
B	4
C	9
D	5



WT
A → 0
B → 8
C → 12
D → 21

Avg. WT = $\frac{8+12+21}{4}$ = 10.25

Avg. TAT = $\frac{8+12+21+26}{4}$ = 16.75

2) Shortest-Job-First (SJF) / Shortest-Job Next Scheduling

=> SJF scheduling assigns the process with shortest CPU burst, to the CPU as soon as CPU is available.

=> The process with the shortest expected processing time is selected for execution, among the available processes in the ready queue.

=> SJF can be ~~pre-~~emptive or non-pre-emptive.

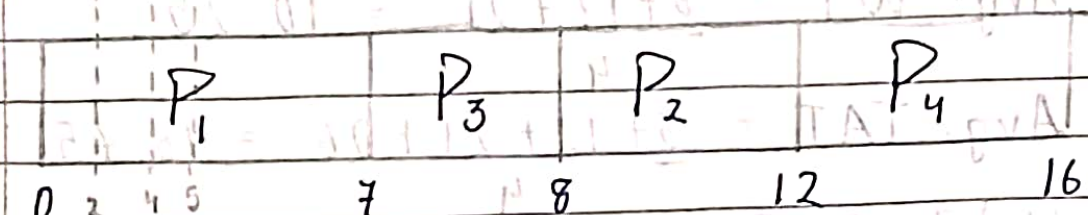
Note Pre-emptive version is called Shortest-Remaining-Time-First (SRTF)

- **Non-preemptive SJF**

1)

Process	A.T.	B.T
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

P_2 P_3 P_5



W.T. $P_1 \rightarrow 0$
$P_2 \rightarrow 8-2=6$
$P_3 \rightarrow 7-4=3$
$P_4 \rightarrow 12-5=7$

Avg. W.T $= \frac{0+6+3+7}{4} = 4$

2)

Process	A.T.	B.T
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Page No.
Date : / /

P_1	P_4	P_3	P_5	P_2

0 10 13 17 22 28

W.T
$P_1 \rightarrow 0$
$P_2 \rightarrow 22 - 3 = 19$
$P_3 \rightarrow 13 - 3 = 10$
$P_4 \rightarrow 10 - 8 = 2$
$P_5 \rightarrow 17 - 13 = 4$

$$\text{Avg. W.T} = \frac{19 + 10 + 2 + 4}{5} = 7$$

3)

Process	A.T	B.T
P_1	0	6
P_2	0	8
P_3	0	7
P_4	0	3

P_1	P_4	P_3	P_2

0 6 9 16 24

W.T
$P_1 \rightarrow 0$
$P_2 \rightarrow 16$
$P_3 \rightarrow 9$
$P_4 \rightarrow 6$

$$\text{Avg. WT} = \frac{16 + 9 + 6}{4} = 7.75$$

Page No.
Date / /

Disadvantages

⇒ Impossible to know the length of the next CPU burst.

⇒ SJF scheduling can't be implemented practically, It is because there is no specific method to predict the length of the upcoming CPU burst.

▪ How to predict next CPU burst?

Let t_n be the length of the nth CPU burst & τ_{n+1} be the prediction of the next CPU burst.

$$\boxed{\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n}$$

where α, $0 < \alpha < 1$, $\alpha = 1/2$

If $\alpha = 0 \Rightarrow \tau_{n+1} = \tau_n$ (no effect)
$\alpha = 1 \Rightarrow$ only last burst matters
$\alpha = 1/2 \Rightarrow$ the actual burst & predict values are imp.

3) Priority Scheduling

⇒ Each process is assigned a priority.

⇒ The CPU is allocated to the process with the highest priority.

=> Can be pre-emptive or non-preemptive

Process	B.T.	Priority
A	10	2
B	5	4
C	7	1
D	6	3

C	A	D	B
0	7	17	23 28

$$\text{Avg. WT} = \frac{0+7+17+23}{4} = 11.75$$

$$\text{Avg. TAT} = \frac{7+17+23+28}{4} = 18.75$$

4) Round Robin

=> Each process gets a small unit of CPU time (time quantum). After this time has elapsed, the process is preempted & added to the end (tail) of the ready queue.

=> Newly-arriving processes (& processes that complete their I/O bursts) are added to the end of the ready queue.

=> If there are n processes in the ready queue & the time quantum is q, then

no process waits more than (n-1)q time units.

=> When the CPU is free, the scheduler picks the first & lets it run for one time quantum.

=> If that process uses CPU for less than one time quantum, it is moved to the tail of the list.

=> RR = FCFS + Preemption

=> If time quantum is too large, RR reduces FCFS.

=> If time quantum is too small, RR becomes processor sharing.

=> Shorter time quantum means context switches.

Process	A.T.	B.T.
P_1	0	10
P_2	3	6
P_3	3	4
P_4	8	3
P_5	13	5

Time quantum for each process = 4

Page No.
Date : / /

A-

P_1	P_2	P_3	P_1	P_4	P_2	P_5	P_1	P_5

0 4 8 12 16 19 21 25 27 2

$$\text{Avg. WT} = \frac{17+12+5+8+10}{5} = 10.14$$

Process	B.T.
P_1	24
P_2	3
P_3	3

Time quantum $q = 4$

P_1	P_2	P_3	P_1

0 4 7 10 30

$$\text{Avg. WT} = \frac{4+7+10}{3} = 7$$

Process	B.T.
P_1	53
P_2	17
P_3	68
P_4	24

Time quantum $q = 20$

P_1	P_2	P_3	P_4	P_1

0 20 37 57 77 97

QUESTIONS

1) What is the CPU - I/O burst cycle?

A- => The CPU-I/O burst cycle refers to the alternating phases of CPU & I/O operations in the execution of a process.

=> This cycle consists of 2 primary parts :-

i) CPU burst : The period during which the process is executing instructions & utilizing the CPU. In this phase, the CPU is actively working on computations or processing data.

ii) I/O burst : The period during which the process waits for input/output operations to complete, such as reading from disk, network or other devices. During this phase the CPU is usually idle or working on another process.

2) What is the func^n of short term scheduler?

A- The short term scheduler is a critical component in an OS to select which of the ready processes should be executed next by CPU.

Page No.
Date / /

Func's

→ It selects from the pool of ready processes & decides which one gets the CPU next.

→ It ensures that all processes get a fair share of the CPU time, balancing the load among them.

→ It keeps the CPU as busy as possible by minimizing idle time & maximizing throughput.

3) What is a non-preemptive scheduling schema?

A- A non-preemptive scheduling schema is a type of CPU scheduling where once a process starts its execution, it runs to completion without being interrupted by other processes.

Characteristics

→ Processes are not interrupted once they start executing.

→ Context switches happen only when a process finishes or voluntarily yields the CPU.

4) What is a preemptive scheduling schema?

A- A preemptive scheduling schema is a type of CPU scheduling in which the OS can interrupt or preempt a currently running process to assign CPU time to another process.

Characteristics

→ Higher priority tasks can interrupt & take over from lower priority tasks. If a higher-priority process becomes available, the CPU is allocated to it immediately.

→ By allowing higher priority & time-critical processes to execute sooner, preemptive scheduling improves CPU utilization & ensures that no process monopolizes the CPU.

5) What are the basic 5 states of a process?

A- 5 basic states :- i) new state
ii) ready state
iii) running state
iv) waiting state
v) terminated state

Page No
Date : / /

Process Creation

=> Parent process creates children processes, which in turn create other processes, forming a tree of processes

=> Process identified & managed via a process identifier (pid)

Resource Sharing

- Parent & children share all resources
- Children share subset of parent's resources.

Execution

- Parent & children execute concurrently.
- Parent waits until children terminate.

Types

fork :- create a new process
exec :- overwrites the process address
wait :- space with new program
exit :- waits for the child to terminate
used to terminate the process

Page No.
Date : / /

1) Fork()

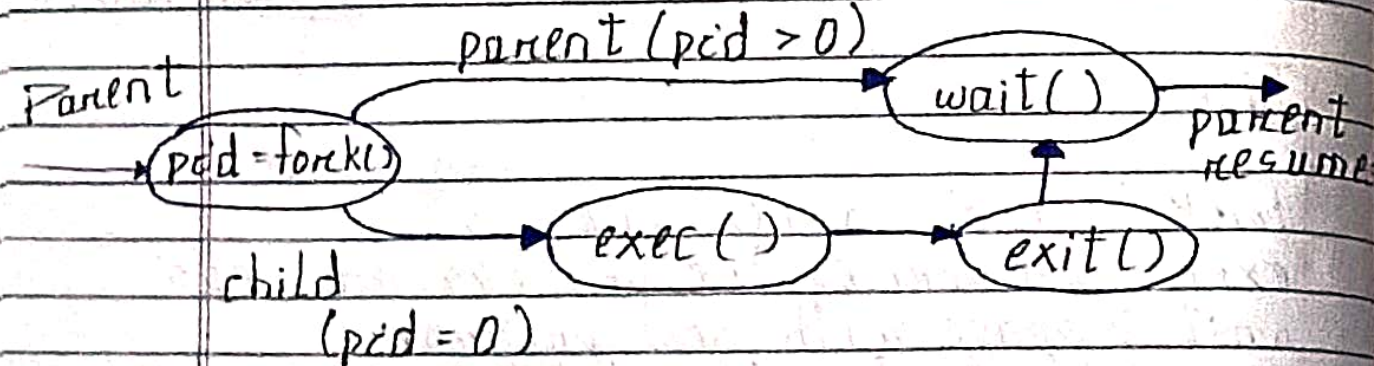
→ Create a new process by duplicating the calling process.

→ Child (new) process is an exact duplicate of the parent process.

→ To the child, return 0.

→ To the parent, return the pid of the child process.

Note: fork() is called once in parent process
Return twice
- → once in parent process
- → once in child process



2) Exec()

→ The exec family of system calls (execl, execle, execlp, execv, execve, execvp) replaces the program

executed by a process.

→ When a process calls exec(), all code & data in the process is lost & replaced with the executable of the new program

3) Exit()

→ A process terminates at the end of the program execution by using the exit() system call

→ A process may terminate in 2 ways:-

a) Voluntarily : exit() system call
b) Involuntarily :
   i) Aborted by user (ctrl + c) or other process (kill)
   ii) divide by zero, segment violation

Why parent kill a child process

A parent process might terminate a child process because :-

- The child process used more resources than it is allowed
- it is no longer needed.
- The parent process is terminating.

Page No
Date / /

Zombie & Orphan

=> The init process is the parent of all process in UNIX / LINUX, identified by the PID of 1. It is executed by the kernel during booting of a system & is responsible for creating all other process.

=> When a parent process dies before a child process, the kernel knows that it's not going to get a wait() call, so instead it makes these processes "orphans" & puts them under the care of init.

=> If the child process finished, the kernel turns the child process into a zombie process.

=> If the parent process calls the wait() system call, the zombie will disappear. This is known as "reaping".

=> If the parent doesn't perform a wait call, init will adopt the zombie & automatically perform wait & remove the zombie.

Page No.
Date: / /

Thread

=> A Thread is a single sequential flow of execution of tasks of a process so it is known as Thread of execution or Thread of control.

=> Often referred to as a light weight process.

Need of Thread

It takes far less time to create a new Thread in an existing process than to create a new process.

Context switching is faster when working with threads.

It takes less time to terminate a Thread than a process.

Types of Threads

In the OS, there are 2 types of Threads:
1) User level
2) kernel level

1) User level Thread

=> The OS does not recognise the user level Thread.

Page No.
Date /

⇒ User Threads can be easily implemented & it is implemented by the user.

⇒ If a user performs a user level Thread blocking operation, the whole process is blocked.

⇒ The kernel level thread does not know nothing about the user level thread.

Eg. Java Thread, POSIX Threads etc.

Advantages

→ Can be easily implemented than the kernel Thread.

→ Faster & efficient.

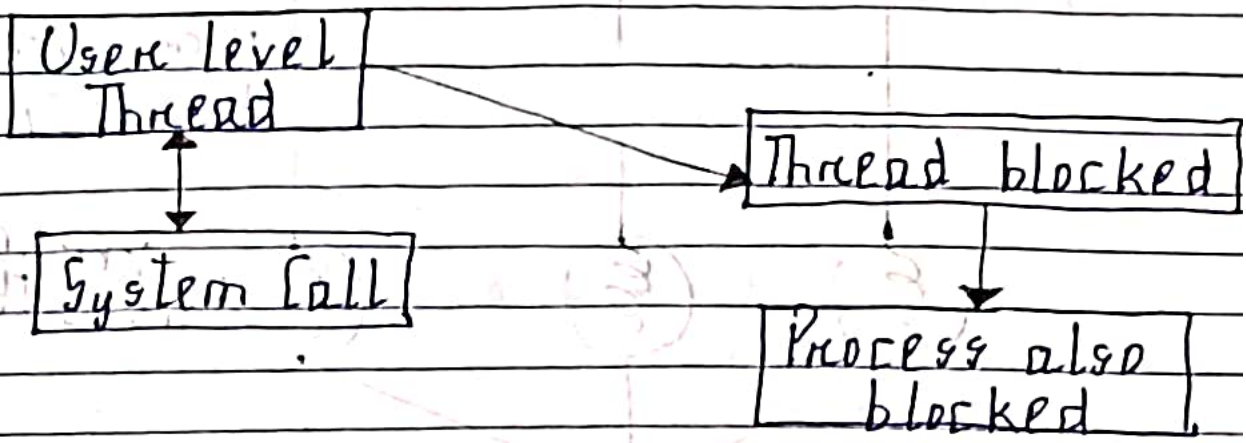
→ Context switch time is shorter than the kernel level Threads.

→ Simple to create, switch & synchronize Threads without the intervention of the ~~Thread~~ process.

Disadvantages

→ Lack coordination bet^n the Thread & the kernel.

→ If a thread causes a page fault, the entire process is blocked.



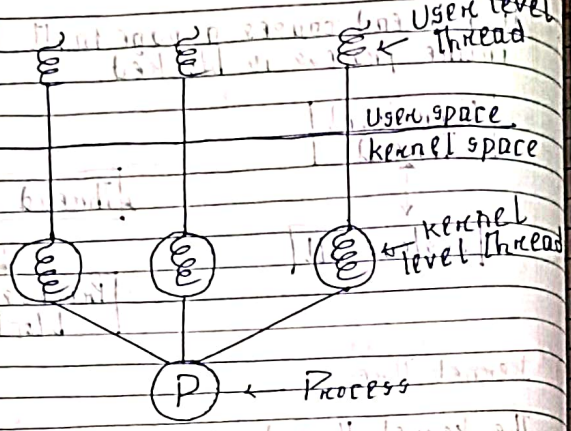
2) Kernel Threads

=> The kernel thread recognizes the OS.

=> There is a thread control block & process control block in the system for each thread & process in the kernel-level thread.

=> The kernel knows about all the threads & manages threads.

=> The kernel level thread offers a system call to create & manage the threads from user-space.



Advantages

→ Fully aware of all Threads.

→ Good for those applications that block the frequency.

→ The scheduler may decide to spend more CPU time in the process of Threads being large numerical.

Disadvantages

→ Implementation is very difficult.

→ Slower than user level Thread.

Difference bet^n Process & Thread.

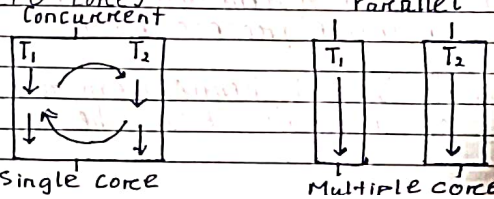
	PROCESS	THREAD
1	Cannot share the same memory.	Can share memory & files.
2	Takes more time to create process.	Takes less time to create a Thread.
3	Execution is slow.	Execution is fast.
4	System calls are required to communicate with each other.	System calls are not required.

Utility of Threads

Threads are a lightweight mechanism for concurrency & parallelism:

Concurrency : do many tasks at the same time (e.g., communicate with clients)

Parallelism : take advantage of multiple CPU cores



Inter-Process Communication And Synchronization

Inter process synchronization is essential to ensure that multiple processes can work together efficiently & safely without interfering with each other.

Sol^n of Process Synchronization

1) Software Approach
2) Hardware Approach
3) Operating System
4) Compiler or programming language

On the basis of synchronisation, processes are categorised as :-

Independent Process

The execution of one process does not affect the execution of other processes.

Cooperative Process

A process that can affect or be affected by other processes executing in the system.

Process Synchronization problem arises in the case of Cooperative process because resources are shared in Cooperative process

Race Cond^n

A race cond^n is an undesirable situation that occurs when a device or system attempts to perform 2 or more operations at the same time.

Here several processes access & manipulate shared data concurrently. The final value of the data depends on which processes finishes last.

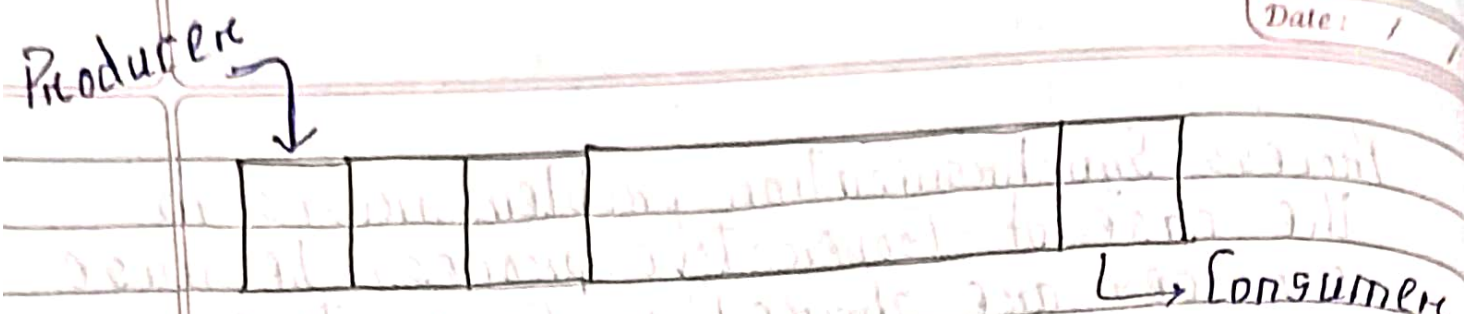
BOUNDED BUFFER PROBLEM

Also known as Producer Consumer problem.

One of the classical problems of synchronization.

Problem Statement :-

There is a buffer of n slots & each slot is capable of storing one unit of data. There are 2 processes running namely, producer & consumer, which are operating on buffer.



Buffer of n slots

Problem:-
A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. Those 2 processes won't produce the expected output if they are being executed concurrently.

Solution to the Buffer Problem

Producer code:-

```
int count = 0;
void producer (void)
{
  int itemP;
  while (1)
  {
    Produce_item (itemP)
    while (count == n);
    buffer [in] = itemP;
    in = (in+1) mod n
    count = count + 1;
  }
}
```

Page No.
Date: / /

Consumer code :-

```
int count;
void consumer(void)
{
  int itemC;
  while (1)
  {
    while (count == 0);
    itemC = buffer[out];
    out = (out + 1) mod n;
    count = count - 1;
  }
}
```

"in" used in producer code represent the next empty buffer.

"out" used in consumer code represent first filled buffer.

count keeps the count no. of elements in the buffer.

1) Software Based Sol^n

Critical Section Problem

⇒ A critical section is a code segment that can be accessed by only one process at a time.

⇒ It means designing a way for cooperative processes to access shared resources without creating data inconsistencies.

Mutual Exclusion

If a process is executing in its critical section, then no other process is allowed to execute in the critical section.

Progress

If no process is executing in the critical section & other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, & the selection cannot be postponed indefinitely.

Bounded Waiting

A bound must exist on the no. of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section & before that request is granted.

→ Software based sol^n is the one that requires no special support from OS or hardware.

```
do
{
  entry section
    [CRITICAL SECTION]
  remainder section
} while (true);
```

→ 2 processes P_i, P_j.
Shared variable turn.
initial turn = i

Algorithm for P_i	Algorithm for P_j

while (true)
{
while (turn == j);
[CRITICAL SECTION]
turn = j;
}
	while (true)
	{
	while (turn == i);
	[CRITICAL SECTION]
	turn = i;
	}

1) MUTUAL EXCLUSION ✓
2) PROGRESS ✗
3) BOUNDED WAIT ✓

Peterson's Soln

In Peterson's soln, we have shared 2 variables :-

int turn :- The turn variable indicates whose turn it is to enter the critical section next.

boolean flag [i] :- Indicate if a process is ready to enter its critical section.

Algorithm Pi

```
while (true)
{
  flag [i] = true;
  turn = j;
  while (flag [j] && turn == j);

  [CRITICAL SECTION]

  flag [i] = FALSE;
}
```

Algorithm Pj

```
while (true)
{
  flag [j] = true;
  turn = i;
  while (flag [i] && turn = i);

  [CRITICAL SECTION]

  flag [j] = false;
}
```

1) Mutual Exclusion ✓
2) Progress ✓
3) Bounded Wait ✓
} Good Soln

Busy Waiting

The process waits & continuously keeps on checking for the cond^n to be satisfied.

The process who is executing the remainder section is interfering in the decision making who will enter next in the critical section.

2) Hardware Based Sol^n

The modern computer system provide special hardware instructions to solve critical ~~pre~~ section problem.

Hence, certain hardware based sol^n to synchronization are proposed.

i) Test And Set ( ) Instructions

Uses a boolean variable lock. The initial value of the lock is false.
If the value of lock is false, this means that no process is in its critical section.

Definition of Test And Set() Instruction

```
boolean
Test And Set (boolean *lock)
{
boolean initial = *lock;
*lock = TRUE;
return initial;
}
```

Implementation of Test And Set() Instruction

```
do
{
  while (Test And Set (&lock));
      [CRITICAL SECTION]
    lock = FALSE;
      [REMAINDER SECTION]
}
while (TRUE);
```

ii) Swap() Instruction

Uses 2 boolean variables lock & key.

Definition of swap() instruction

```
void swap (boolean *a, boolean *b)
{
    boolean t = *a;
    *a = *b;
    *b = *t;
}
```

Implementation of swap() instruction

```
do
{
  key = TRUE;
  while (key == TRUE)
      swap (&lock, key);

  lock = FALSE;

}
while (TRUE);
```

1) MUTUAL EXCLUSION ✓
2) PROGRESS ✓
3) BOUNDED WAIT X

Page No.
Date : / /

Advantages

Easy to implement & improves the efficiency of the system.

Supports any no. of processes may it be on the single or multiple processor system.

Disadvantages

Processes waiting for entering their critical section consumes a lot of processor's time which increases busy waiting.

Mutex Locks

=> It is a mutual exclusion object that synchronizes access to a resource.

=> The process must acquire the lock before entering critical section. It releases the lock when it exits the critical section.

```
while (TRUE)
{
    Acquire lock

    Critical Section
```

release lock

Remainder section

}

A mutex lock has a boolean variable "available" whose value indicates whether the lock is available or not.

If the lock is available call to 'acquire' func success.

```
acquire()
{
while(!available);

   AVAILABLE = FALSE;
}
```

```
release()
{
   AVAILABLE = TRUE;
}
```

Spin Lock :- It is a synchronisation mechanism where a thread repeatedly checks a cond" to acquire a lock, without giving up the control of CPU.

Page No.
Date: /

Difference bet^n Spinlock & Busy waiting

	SPIN LOCK	BUSY WAITING
1	Specific to acquire locks.	General concept for waiting on a cond^n.
2	Synchronizes access to shared resources.	Waits for any cond^n to change
3	Used in critical sections.	Used in broader scenarios.

3) Semaphores

=> This is an OS based sol^n.

=> It uses 2 atomic operations :-
i) wait ()
ii) signal ()

=> It is an integer value, which can be accessed only through the above 2 operations.

wait () → P (Proberen, "to test")
signal () → V (Verhogen, "to increment

wait (S)	signal (S)
{	{
while (S<=0);	S++;
S--;	}
}	

When one process modify the semaphore value no other process can simultaneously modify the same value.

There are 2 types of semaphores :-
i) Binary
ii) Counting

i) Binary Semaphores

They can only be either 0 or 1.

They are also known as mutex locks, as the locks can provide mutual exclusion.

0 : Resource is unavailable
1 : Resource is available.

Typically initialized to 1.
Cannot count beyond 1.
Ensures exclusive access to a resource.

Page No.
Date: /

iii) Counting Semaphores

=> A semaphore that can take non-negative integer values.

=> Used to control access to a finite no. of resources.

=> Can count from 0 to any +ve integer.

=> Tracks the no. of available resources.

Advantages

- Machine-independent.
- Do not allow multiple processes to enter the critical section.
- Allows more than one thread to access the critical section.

Disadvantages

- Priority inversion
- Semaphore programming is a complicated.

Page No.
Date: / /

Q- S = 10
6P → operations
4V → operations

What is the value of S?

A- 10 - 6 + 4 = 8

Q- S = 0
(5P 7V 10P 13V 17P)

What is the value of S?

A- 0 - 5 + 7 - 10 + 13 - 17
= -12

Q- Semaphore S = 7, operations performed 20 P (wait) & 15 V (signal) operations then how many process get blocked at the end?

A- 7 - 20 + 15 =

Q- Let S be a semaphore with initial value S = 0. The following signal (V) & wait (P) operations are performed.

4P 6V 9P 13V 14P

At the end of operations, how many process get blocked?

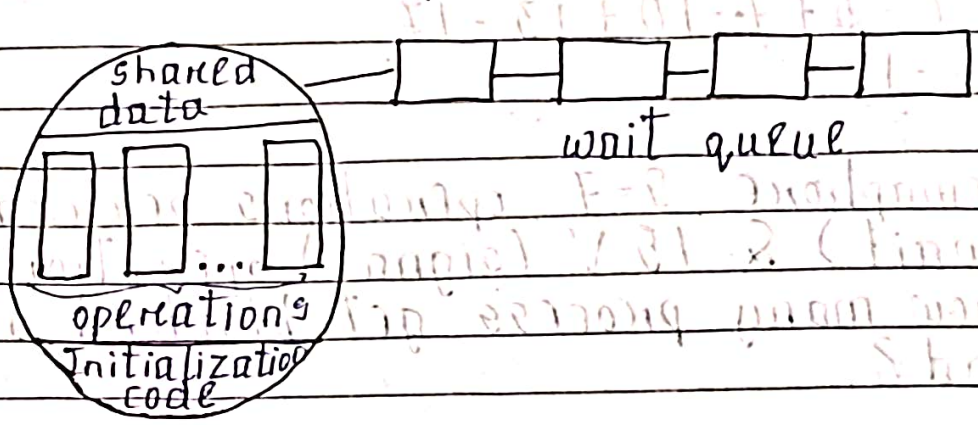
A- 0 - 4 + 6 - 9 + 13 - 14
= 8

Monitors

=> Monitor is an abstract datatype (ADT).

=> It encapsulates data with a set of func^n's to operate on that data.

=> It is a high level synchronisation mechanism used to solve ordering problems in semaphores.



=> Contains a lock & a set of cond^n variables.

=> The lock is used to enforce mutual exclusion.

=> The cond^n variables are used as a wait queue so that other process or threads can sleep while lock is not available.

Page No.
Date: / /

Sol^n of Producer Consumer Problem using Semaphores

m(mutex), a binary semaphore which is used to acquire & release the lock.

empty a counting semaphore whose initial value is the no. of slots in the buffer, since, initially all slots are empty.

full, a counting semaphore whose initial value is 0.

```
Producer : do
           {
             wait (empty);
acquire lock // wait (mutex); //add data to buffer
             signal (mutex); //release lock
             signal (full);
           }
           while (TRUE)

Consumer :- do
           {
             wait (Full);
             wait (mutex); // acquire lock
           //remove data from buffer
             signal (mutex); //release lock
             signal (empty);
           }
           while (TRUE)
```

Page No.
Date: / /

The Readers - Writers Problem

A data set is shared among a no. of concurrent process :

Readers :- only read the data set ; they do not perform any updates.
Writers :- can both read & write.

Problem :-

Allow multiple readers to read at the same time.

Only one single writer can access the shared data at the same time.

Shared data

Semaphore rw_mutex initialized to 1.
Semaphore mutex initialized to 1.
Integer read_count initialized to 0.

Reader Process :-

```
while (true)
{
    wait (mutex) ;
    read_count ++ ;
    // first reader
```

```
if (read_count == 1)
    wait (rw_mutex);
signal (mutex);
...
//reading is performed
...
wait (mutex);
read_count--;
//last reader
if (read_count == 0)
    signal (rw_mutex);
signal (mutex);
}
```

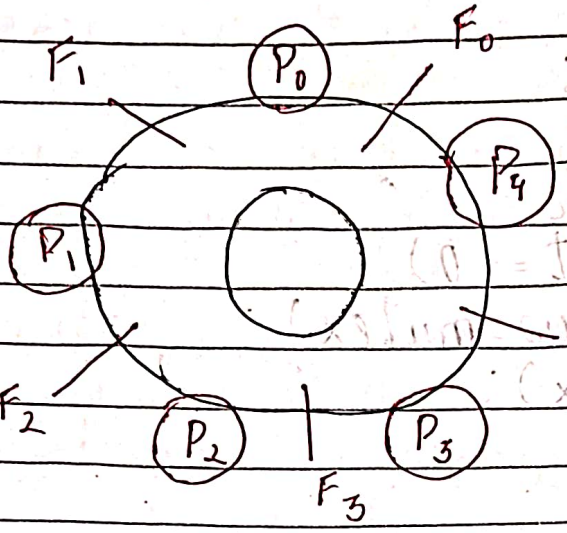
Writer Process:

```
while (true)
{
  wait (rw_mutex);
  ...
  //writing is performed
  ...
  signal (rw_mutex);
}
```

The Dining Philosopher's Problem

It states that k philosophers are seated around a circular table with one chopstick bet^n each pair of philosophers.

There is one chopstick bet^n each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him.



Sol^n using Semaphores

S[i] → array of semaphores initially to 1.

```
void philosopher (void)
{
 while (true)
 {
 //THINK
 wait (chopstick [i]);
 wait (chopstick [(i+1) % 5];
 // EAT
 signal (chopstick [i]);
 signal (chopstick [(i+1) % k]);
 }
}
```

Page No.
Date: / /

This sol^{n} ensures that no 2 neighbours eat simultaneously, it can lead to deadlock if :-

All n philosophers grab their left chopstick & waits for their right chopstick.

To ~~remove~~ avoid deadlock, some of the sol^{n}s are as follows :-

i) Max^{m} no. of philosophers on the table should not be more than four.

ii) A philosopher at an even pos^{n} should pick the right chopstick & then the left chopstick while a philosopher at an odd pos^{n} should pick the left chopstick & then the right chopstick.

iii) Only in case if both the chopsticks are available at the same time, only then a philosopher should be allowed to pick their chopsticks.

iv) All the 4 starting philosophers should pick the left chopstick & then the right chopstick, whereas the last philosopher should pick the right chopstick & then the left chopstick.

DEADLOCK

A deadlock is a situation where a set of processes are unable to proceed because each process is waiting for a resource that another process holds.

Cond's for Deadlock

1) Mutual Exclusion : only one process at a time can use a resource.

2) No Preemption : a resource can be released only voluntarily by the process holding it, after that process has completed its task.

3) Hold & wait : a process holding at least one resource is waiting to acquire additional resources held by other processes.

4) Circular Wait : A set of processes are waiting on each other in a circular chain with each process waiting for a resource held by the next process in the chain.

Resoure Allocation Graph

It is a directed graph used to represent the allocation of resources to processes in a system.

Page No.
Date: / /

Components of RAG

1) Process

2) Resource Type with 4 instances

3) P_i requests instance of R_j

4) P_i is holding an instance of R_j

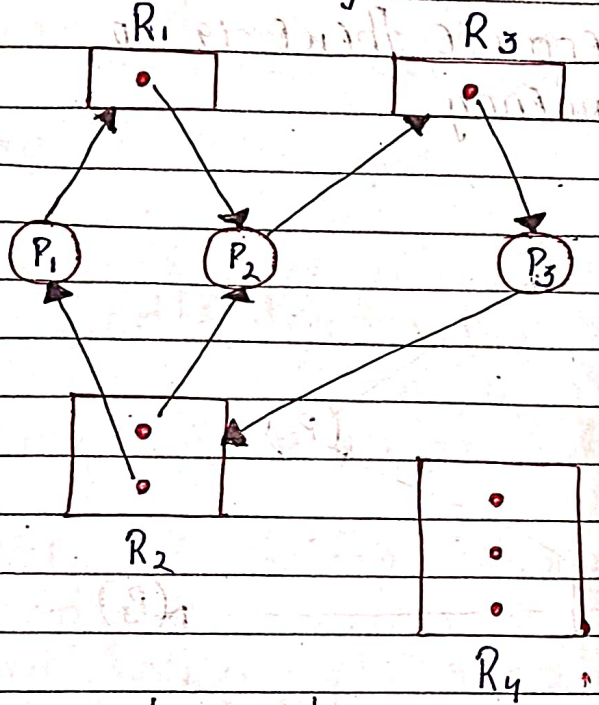


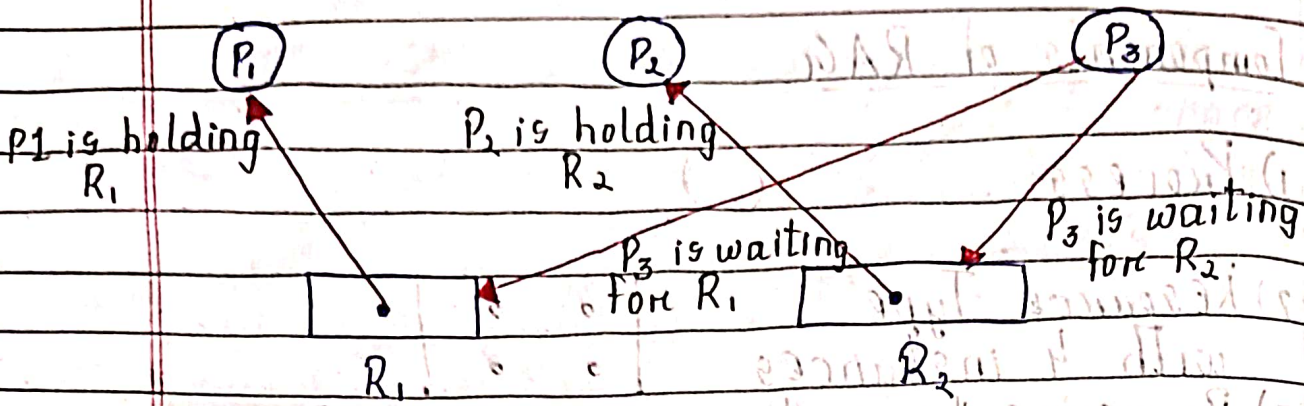
Fig. RAG with Deadlock

If graph contains no cycles => no deadlock

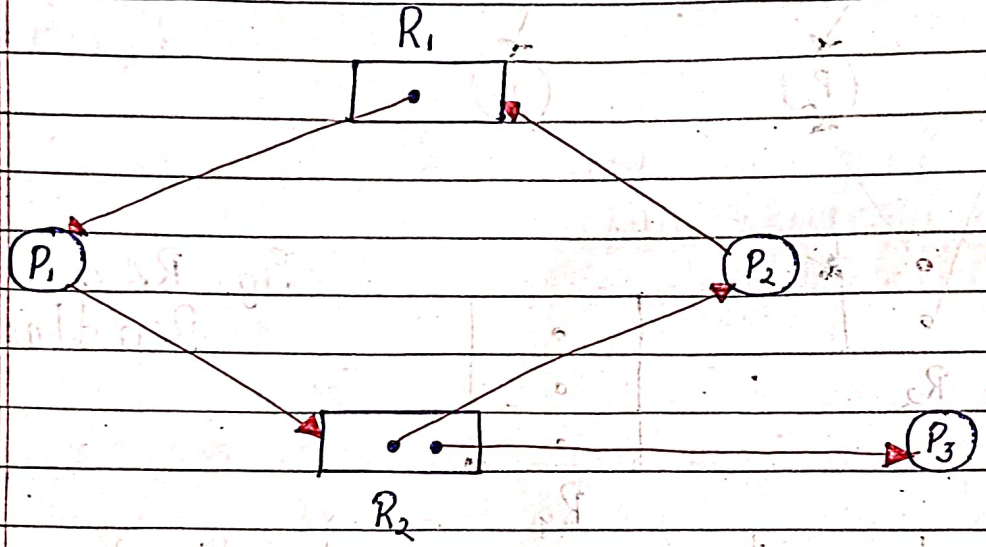
If a graph contains a cycle:

i) If only one instance per resource type, then deadlock.

ii) If several instance per resource type, the possibility of deadlock.



Processes P_1 & P_2 acquiring resources R_1 & R_2 while process P_3 is waiting to acquire both resources. Here there is no deadlock because there is no circular dependency.



	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Availability (0, 0)
P_3 ×
Available (0, 1)
P_1 ×
Available (1, 0)
P_2 ×
New available (1, 2)

∴ So, there is no deadlock in this RAG.

Various Methods to handle Deadlock

1) Deadlock Ignorance

This is one of the widely used methods to handle deadlock.

Also known as Ostrich Algorithm.

Here the system deliberately ignores the deadlock, similar to how an ostrich is said to bury its head in the sand when faced with danger.

Here UNIX & WINDOW restart or reboot the system to ignore deadlock.

2) Deadlock Prevention

It is a strategy to ensure that atleast one of the necessary cond's for a deadlock to occur is never allowed.

i) Mutual Exclusion :- more than one process can have access to a single resource at the same time.

ii) Hold & wait :- if a process is holding a resource & wants to have additional resources, then it must free the acquired resources.

iii) Pre-emption :- If a process holding some resources cannot acquire additional ones, preempt the resource it holds.

iv) Circular Wait :- impose a strict ordering on resource allocation & processes can only request resources in ascending order.

3) Deadlock Avoidance

It is a strategy to ensure that the system never enters a state that could lead to a deadlock.

Page No.
Date: / /

Safe State

A state is safe if there exists a sequence of processes such that each process can complete its execution using currently available resources & those held by previously completed processes.

Unsafe State

An unsafe state does not guarantee that deadlock will occur but indicates the possibility.

If a system is in safe state ⇒ no deadlock
If a system is in unsafe state ⇒ possibility of deadlock

* Banker's Algorithm

⇒ It enables the OS to share resources among all the processes without creating deadlock or any critical situation.

⇒ The algorithm is named so because it mimics the process through which a bank determines whether or not it can safely make loan approvals.

Page No.
Date: / /

Let n = no. of processes
m = no. of resources types

Max : n x m
Allocation : n x m
Need [i, j] = Max [i, j] - Allocation [i, j]

Q- There are 5 processes & 3 diff. types of resources. Apply banker's algorithm & find out whether the system is in safe state or not.

Process	Allocation			Max^m		
	A	B	C	A	B	C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

Available = [3, 3, 2]

A- Need = Alloc - Max^m

	A	B	C
P_0 →	7	4	3
P_1 →	1	2	2
P_2 →	6	0	0
P_3 →	0	1	1
P_4 →	4	3	1

Page No.
Date: / /

Need <= Available
New available = Available + Allocation

For P_1 1, 2, 2 <= 3, 3, 2
TRUE
New available = (3, 3, 2) + (2, 0, 0)
= (5, 3, 2)

For P_3 0, 1, 1 <= 5, 3, 2
TRUE
New available = (5, 3, 2) + (2, 1, 1)
= (7, 4, 3)

For P_4 4, 3, 1 <= 7, 4, 3
TRUE
New available = (7, 4, 3) + (0, 0, 2)
= (7, 4, 5)

For P_2 6, 0, 0 <= 7, 4, 5
TRUE
New available = (7, 4, 5) + (3, 0, 0)
= (10, 4, 5)

For P_0 7, 4, 3 <= 10, 4, 5
TRUE
New available = (10, 4, 5) + (0, 1, 0)
= (10, 5, 5)

∴ Safe sequence = [P_1, P_3, P_4, P_2, P_0]

Page No.
Date: / /

Q- There are 3 resources type with $R_1 = 9$, $R_2 = 3$, $R_3 = 6$ & have 4 processes. Find out is there any safe sequence or not? (Available (1, 1, 2))

Process	Allocation			Maxm		
	A	B	C	A	B	C
P_1	1	0	0	3	2	2
P_2	5	1	1	6	1	3
P_3	2	1	1	3	1	4
P_4	0	0	2	4	2	2

A- Need

	A	B	C
$P_1 \rightarrow$	2	2	2
$P_2 \rightarrow$	1	0	2
$P_3 \rightarrow$	1	0	3
$P_4 \rightarrow$	4	2	0

∴ Safe sequence $[P_2, P_3, P_4, P_1]$.

For P_2 1, 0, 2 <= 1, 1, 2
TRUE
Available = (6, 2, 3)

For P_3 1, 0, 3 <= 6, 2, 3
TRUE
Available = (8, 9, 4)

For P_4 4, 2, 0 <= 8, 9, 4
TRUE
Available = (8, 9, 6)

MEMORY MANAGEMENT

Page No.
Date: / /

Memory Management is a method in the OS to manage operations bet" main memory & disk during process execution.

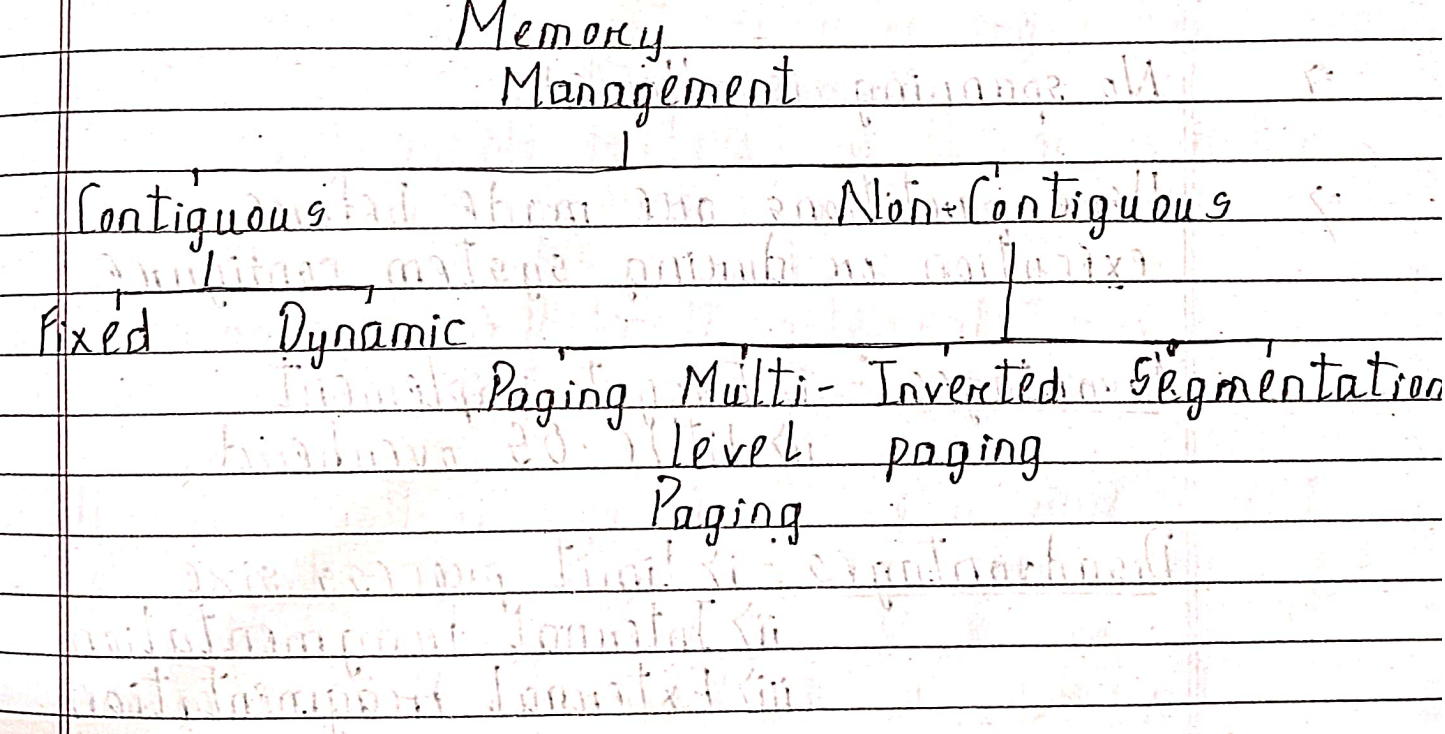
The task of subdividing the memory among diff processes is called memory management

Why Memory Management is Required?

⇒ Allocate & deallocate memory before & after process execution.

⇒ To keep track of used memory space by processes.

⇒ To minimize fragmentation issues.



Page No.
Date: / /

Contiguous

⇒ Each process is contained in a single contiguous block of memory.

⇒ Executing process must be loaded entirely in main memory.

⇒ It can be divided into :-
- i) Fixed
- ii) Dynamic

i) Fixed Partitioning

⇒ No. of partitions in RAM are fixed but size of each partition may or may not be same.

⇒ No spanning is allowed.

⇒ Here partitions are made before execution or during system configure.

Advantages :- i) Easy to implement
ii) Little OS overhead

Disadvantages :- i) Limit process size
ii) Internal fragmentation
iii) External fragmentation

ii) Variable Partitioning

=> Used to alleviate the problem faced by Fixed partitioning.

=> It is a memory allocation technique that allows memory partitions to be created & resized dynamically as needed.

=> The OS maintains a table of free memory blocks or holes, each of which represents a potential partition. When a process requests memory, the OS searches the table for a suitable hole that can accommodate the requested amount of memory.

Advantages :- i) No internal fragmentation
ii) No restriction on degree of multiprogramming
iii) No limitation on the size of the process.

Disadvantages :- i) Difficult implementation
ii) External fragmentation.

How to satisfy a request of size n from a list of free holes?

1) First Fit

=> It scans the linked list & whenever it finds the first big enough hole to store a process, it stops scanning & load the process into that hole.

2) Next Fit

=> It is a slight modification of the first fit algorithm.

=> Instead of scanning from the beginning of the memory, it resumes the search from where it left off during the last allocation.

3) Best Fit

=> It selects the smallest free memory block that is large enough to accommodate the process, thus minimizing wasted space.

4) Worst Fit

=> It selects the largest memory block available to satisfy a process's memory request. The idea is to leave the largest possible remaining fragments for future allocation.

Page No.
Date: / /

Fragmentation

=> It refers to the inefficient utilization of memory caused by the allocation & deallocation of memory blocks.

=> It occurs when memory is divided into small, unusuable sections that cannot be allocated to processes, even though enough total memory might be available.

=> It is of 2 types :- i) External
ii) Internal

i) External

Occurs in the free memory space.

Happens when free memory is divided into small, non-contiguous blocks, making it impossible to allocate to processes that need contiguous memory.

Sol^n :-

Compaction :- Rearrange memory blocks to combine fragmented free spaces into a single contiguous block.

Paging & Segmentation.

ii) Internal

Occurs within allocated memory blocks.

Happens when allocated memory is slightly larger than required, leaving unused space inside the block.

Soln :-

Use smaller allocation units to minimize wasted space.

Non-Contiguous

⇒ It is a technique where a process's memory is divided into multiple blocks that can be located in diff. parts of the physical memory.

⇒ This approach eliminates the requirement for a process, to occupy a single, continuous block of memory, enabling better utilization of available memory & reducing external fragmentation.

Page No.
Date: / /

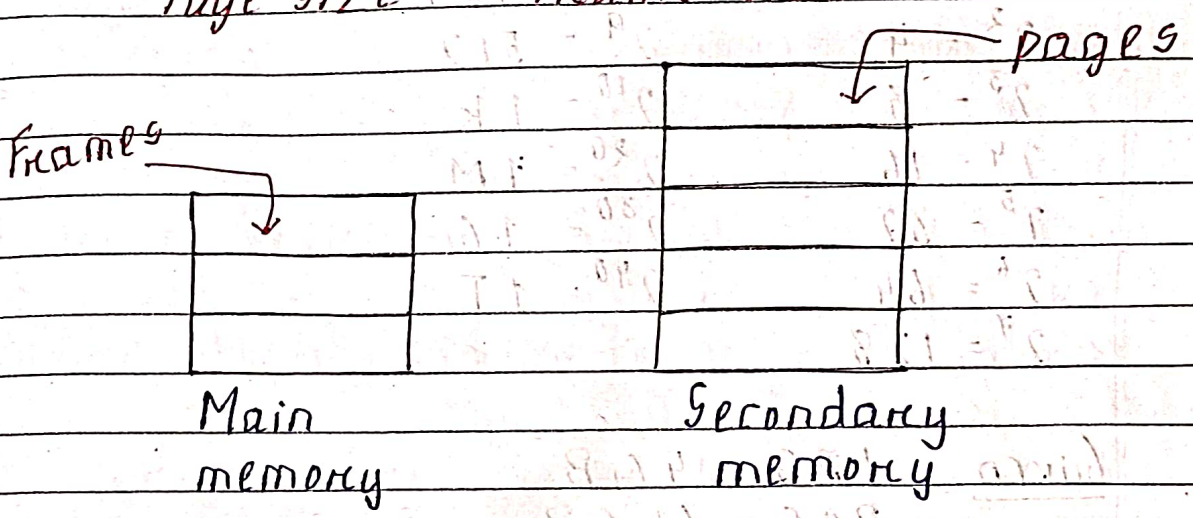
Paging

⇒ The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging.

⇒ Secondary & main memory are divided into equal fixed size partitions.

⇒ Partitions of secondary memory → pages
Partitions of main memory → frames.

Page size = Frame size



⇒ Address generated by CPU is divided into :

Page no. (p) : No. of bits required to represent the pages in logical address space.

Page offset (d) : No. of bits required to represent particular word in a page of logical address space.

⇒ Physical address is divided into :-

Frame No. (f) :- No. of bits required to represent the frame of physical address space.

Frame offset (d) :- No. of bits required to represent particular word in a frame of physical address space.

* No. of pages = $\frac{\text{Logical Address Space}}{\text{Page size}}$

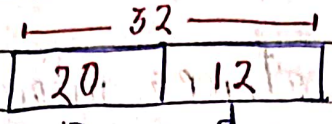
* $2^1 = 2$	$2^8 = 256$
$2^2 = 4$	$2^9 = 512$
$2^3 = 8$	$2^{10} = 1K$
$2^4 = 16$	$2^{20} = 1M$
$2^5 = 32$	$2^{30} = 1G$
$2^6 = 64$	$2^{40} = 1T$
$2^7 = 128$	

Q- Given :- LAS = 4 GB
PAS = 64 GB
Page size = 4 kB

Find :- i) No. of pages
ii) No. of frames
iii) No. of entries in page table
iv) Size of page table

Page No.
Date: / /

LAS = $2^2 \times 2^{30} = 2^{32}$

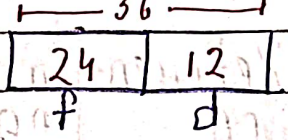


4K P

Page size = $2^2 \times 2^{10} = 2^{12}$

∴ No. of pages = 2^{20}

PAS = $2^6 \times 2^{30} = 2^{36}$



Page size = frame size

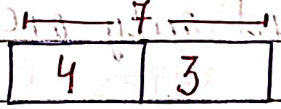
∴ No. of frames = 2^{24}

No. of entries = 2^{20}

Size of page table = $2^{20} \times 24$

Q- Consider a system which has LA = 7 bits, PA = 6 bits, Page size = 8 words. Then calculate no. of pages & no. of frames.

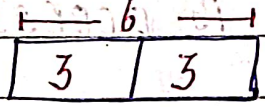
A- LA = 7 bits



Page size = 8 words = 2^3

∴ No. of pages = $2^4 = 16$

PA = 6 bits



∴ No. of frames = $2^3 = 8$

Page No.
Date: / /

Page Table Entry

=> It is an entry that stores information about a particular page of memory.

=> It keep track of the mapping bet^n virtual addresses used by a process & the corresponding physical addresses in the system's memory.

Frame No.	Present/ Absent	Protect- ion	Referen- ce	Cachi- ng	Dirty

i) Frame No :- It gives the frame no. in which the current page is present.

ii) Present / Absent Bit :- It says whether a particular page looking is present or not. In case, it is not present, that is called page fault.

iii) Protection :- It represents the protection level which is applied on the page. It can be read only or read & write or execute.

iv) Reference :- Says whether this page has been referred to in the last clock cycle or not. It is set to 1 by hardware when the page is accessed.

vi) Caching : CPU accesses cache which can be inaccurate in some of the cases, thus OS can disable the cache for the required pages. The bit is set to 1 if the cache is disabled.

vii) Modified : This bit will be set if the page has been modified otherwise it remains 0.

Segmentation

=> It is a technique in which each job is divided into several segments of diff. sizes, one for each module that contains pieces that perform related func^n.

=> Each segment is actually a diff. logical address space of the program.

=> When a process is to executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of memory.

=> The OS maintains a segment map table for every process & a list of free memory blocks along with segment no.s, their size & corresponding memory locations in main memory.

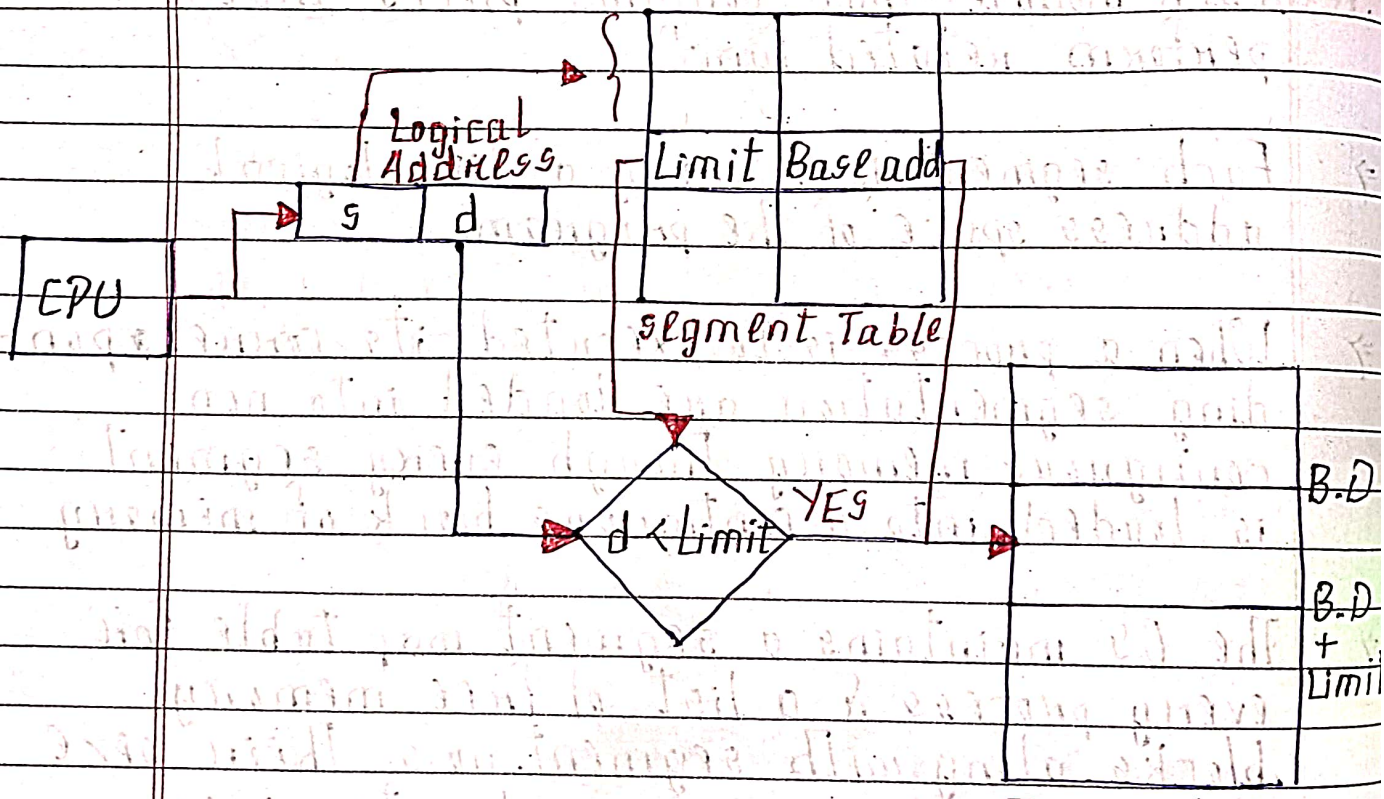
=> For each segment, the table stores the starting address & length of the segment.

Segment Table

=> It maps 2-D logical address into 1-D physical address.

=> It's each table entry has :-

Base address :- Starting physical address where the segments reside in memory.
Limit :- Specifies the length of the segment.



Page No.
Date: / /

Difference bet^n Paging & Segmentation

	PAGING	SEGMENTATION
1	The program is divided into fixed size pages.	The program is divided into variable size pages.
2	Fast	Slow
3	OS is accountable.	Compiler is accountable.
4	Invisible to user.	Visible to user.
5	May cause internal fragmentation.	May cause external fragmentation.

Virtual Memory

=> It is a storage scheme that provides user an illusion of having a very big main memory.

=> This is done by treating a part of secondary memory as main memory.

=> By doing this, the degree of multiprogramming will be increased. Thus CPU utilization will increase.

How virtual memory works?

In this scheme, whenever some pages needs to be loaded in the main memory for the execution, & the memory is not available for those many pages, then in that case, instead the OS search for the RAM that are not referenced & copy that into the secondary memory to make the space for the new pages in the main memory.

Demand Paging

⇒ In demand paging, the pages of a process which are least used, get stored in the secondary memory.

⇒ A page is copied to the main memory when its demand is made or page fault occurs.

⇒ Reduces memory allocation costs but may delay program execution due to initial page load.

Translation Look Aside Buffer (TLB)

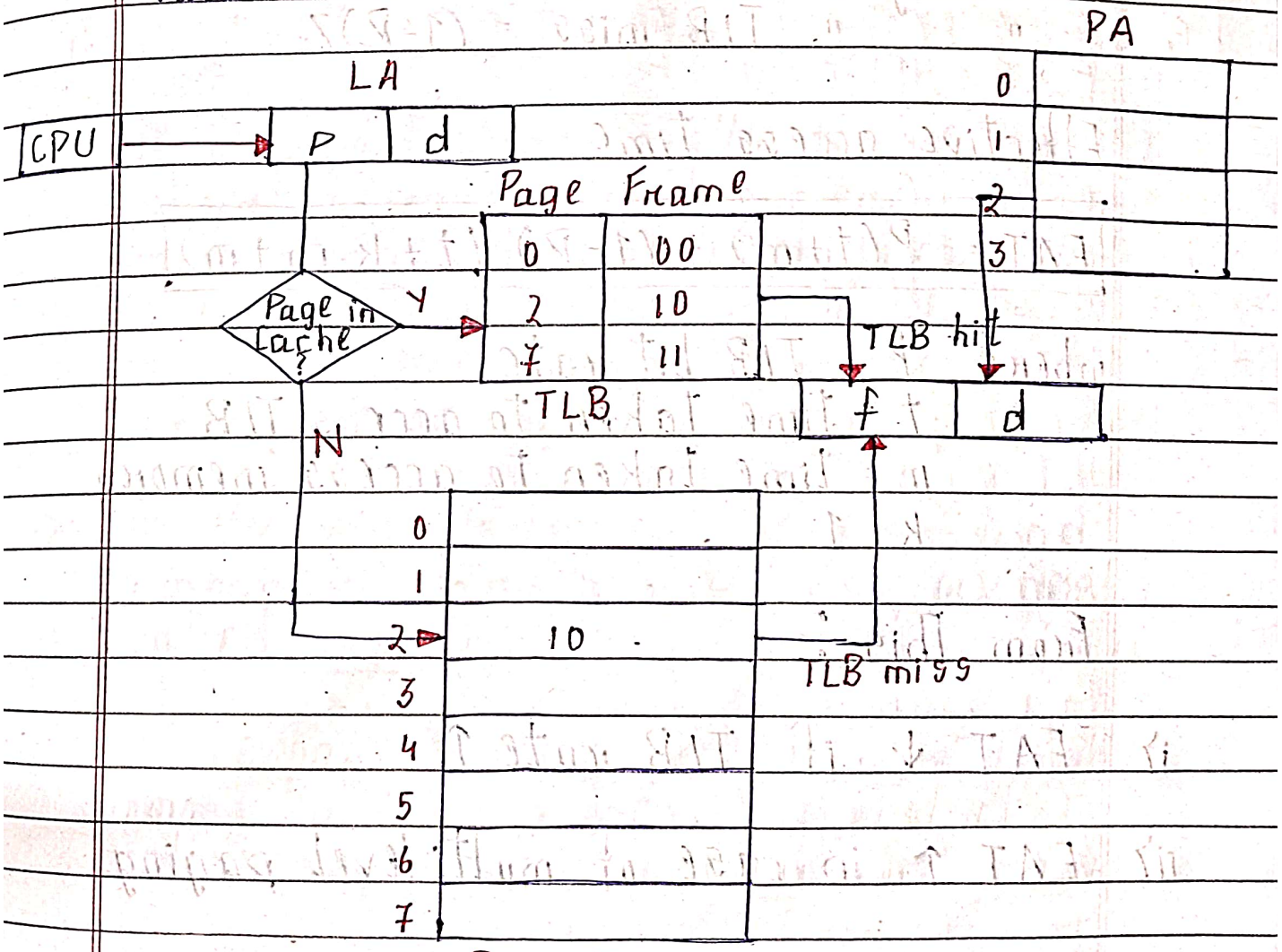
⇒ It is a memory cache which can be used to reduce the time taken to access the page table again & again.

Page No.
Date: / /

=> TLB is faster & smaller than the main memory but cheaper & bigger than the register.

=> TLB follows the concept of locality of reference which means that it contains only the entries of those many pages that are frequently accessed by the CPU.

=> Each entry in TLB consists of : a tag & a value.



Page No.
Date: / /

TLB hit is a cond^n where the desired entry is found in TLB. If this happens then the CPU simply access the actual location in the main memory.

If the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory & then access the actual frame in the main memory.

Probability of TLB hit = P %.
" " TLB miss = (1-P) %.

Effective access time

$$\boxed{EAT = P(t+m) + (1-P)(t+k \cdot m + m)}$$

where P = TLB hit rate
t = time taken to access TLB
m = time taken to access memory
k = 1

From this :-

i) EAT ↓ if TLB rate ↑

ii) EAT ↑ in case of multilevel paging.

Page No.
Date: / /

Page Replacement Algorithm

=> These are techniques used in OS to manage memory efficiently when the virtual memory is full.

=> When a new page needs to be loaded into physical memory & there is no free space, these algorithms determine which existing page to replace.

1) First in First Out (FIFO)

=> It removes the page in the frame which is allocated long back. This means the useless page which is in the frame for a longer time is removed & the new page which is in ready queue & is ready to occupy the frame.

=> If the page to be searched is found among the frames then, this process is called page hit.

=> If the page to be searched is not found among the frames then, this process is called page fault.

Page No.
Date: / /

Q- Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with 3 frames & calculate no. of page faults by using FIFO page replacement algorithm.

A- 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

F_3				2	2	2	4	4	4	2	2	2
F_2		1	1	1	1	3	3	3	0	0	0	0
F_1	6	6	6	6	0	0	0	6	6	6	1	1
	F	F	H	F	F	F	F	F	F	F	F	H

2	2	2	2	2	2	2	0
0	0	0	3	3	3	3	3
1	1	1	1	1	1	1	1
H	H	H	F	H	H	H	F

No. of page hits = 8
No. of page faults = 12
Hit ratio = $\frac{8}{12} = \frac{2}{3} = 2:3$
Page hit % = $\frac{8}{20} \times 100 = 40\%$
Page fault % = $\frac{12}{20} \times 100 = 60\%$

Belady's Anomaly

It is a phenomenon where increasing the no. of pages frames in memory can lead to an increase in the no. of page faults.

Page No.
Date: / /

2) Optimal Page Replacement Algorithm

It works on a principle that says:
Replace the page which is not used in the longest dimension of the time in future.

The principle means that after all the frames are filled then, see the future pages which are to occupy the frames. Go on checking for the pages which are already available & choose the page which is at last.

Q- Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0 for a memory with 3 frames & calculate no. of page faults using OPTIMAL page replacement algorithm.

A- ~~6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1,~~
6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0

				2	~~2~~	~~3~~	4	4	4	4
F_2		1	1	X	0	0	0	0	0	0
F_1	6	6	6	6	6	6	6	6	6	2
	F	F	H	F	F	F	F	H	H	F

1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	3	3	3	4	4
2	2	2	2	2	2	2	2	2	0
F	H	H	H	H	F	H	H	F	F

Page No.
Date: / /

Least Recently Used (LRU)

It works on a principle that says:- Replace the page with the page which is less dimension or time recently used page in the past.

Q- Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with 3 frames & calculate the no. of page faults using LRU page replacement algorithm.

A- 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

F_3				2	2	2	4	4	4	2
F_2		1	1	1	1	3	3	3	0	0
F_1	6	6	6	6	0	0	0	6	6	6
	F	F	H	F	F	F	F	F	F	F

2	2	2	2	2	2	2	2	2	2
0	0	0	0	0	0	0	1	1	1
1	1	1	1	1	3	3	3	3	0
F	H	H	H	H	F	H	F	H	F

No. of page hit = 8
page fault = 12
Hit ratio = 8 : 12 = 2 : 3
Page hit % = $\frac{8}{20} \times 100 = 35\%$
Page fault % = $\frac{12}{20} \times 100 = 65\%$

Page No.
Date: / /

Q- A system uses 3 page frames for storing process pages in main memory. It uses FIFO page replacement policy. What will be the total no. of page faults that will occure while processing the reference string below :
4, 7, 6, 1, 7, 6, 1, 2, 7, 2.

A-

F_3			6	6	6	6	6	6	7	7
F_2		7	7	7	7	7	7	2	2	2
F_1	4	4	4	1	1	1	1	1	1	1
	F	F	F	F	H	H	H	F	F	H

No. of page fault = 6

Q- Assume we have 3 frames & consider the reference string below :
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
Show the content of memory after each memory reference if FIFO, LRU & optimal page replacement algorithm is used. Find the no. of page fault in each case.

A- FIFO

F_3			1	1	1	4	0	0	0	3	3
F_2		0	0	0	0	3	3	3	2	2	2
F_1	7	7	7	2	2	2	2	4	4	4	0
	F	F	F	F	H	F	F	F	F	F	F

3	3	3	3	3	2	2	2	2	2	1
2	2	2	2	1	1	1	1	4	0	0
0	0	0	0	0	0	0	0	7	7	7
H	H	H	H	F	F	H	H	F	F	F

∴ No. of page fault = 15

Optimal

F_3			1	1	1	3	3	3	3	3	3
F_2		0	0	0	0	0	0	4	4	4	0
F_1	7	7	7	2	2	2	2	2	2	2	2
	F	F	F	F	H	F	H	F	H	H	F

3	3	3	3	1	1	1	1	1	1	
0	0	0	0	0	0	0	0	0	0	
2	2	2	2	2	2	2	2	7	7	7
H	H	H	H	F	H	H	H	F	H	H

∴ No. of page fault = 9

LRU

F_3			1	1	1	3	3	3	3	3	3
F_2		0	0	0	0	0	0	0	0	0	0
F_1	7	7	7	2	2	2	2	4	2	2	2
	F	F	F	F	H	F	H	F	F	H	H

3	3	3	1	1	1	1	1	1	1	
0	0	0	0	0	0	0	0	2	0	0
2	2	2	2	2	2	2	7	7	7	7
H	H	F	F	H	H	H	F	F	F	H

∴ No. of page fault = 12

FILE SYSTEMS

Page No.
Date: / /

A file is a named collection of related information that is recorded on secondary storage.

It is a sequence of bits, bytes, lines or records whose meaning is defined by the user.

Attributes of a File

Name : Only information which is in human-readable form.
Identifier : File is identified by a unique tag (no.) within file system.
Type : Needed for systems that support diff. type of files.
Location : Pointer to file location on device.
Size : Current size of the file.
Protection : Controls & assigns the power of reading, writing & executing.

File Operations

Creating a file
- allocate space & make entry in a directory.

Writing a file
- requires name of the file & the info to be written
- search the directory to find file's location.
- keep a write-pointer to the location in file.
- update the pointer after each write

Page No.
Date: / /

Reading a file
Same as writing a file

Repositioning within a file
Change the value of the file position pointer

Deleting a file
Deallocate the space & remove the entry

Truncating a file
Change the allocated space to zero, & deallocate its space.

File Type	Extension	Func
Executable	exe, com, bin	read to run machine language program
object	obj, o	compiled, machine language
source code	c, cc, java	source code
batch	bat, sh	commands to the command interpreter
text	txt, doc	textul data
word processor	wp, tex, doc	word-processor
library	lib, mpeg, dll	libraries of routines
print or view	zip, pdf, gif	ASCII or binary file

Page No.
Date: / /

File Access Method

=> It refers to the manner in which the records of a file may be accessed.

=> It can be accessed by diff ways :
- i) Sequential
- ii) Direct
- iii) Indexed sequential

i) Sequential Access

=> It is the most simplest access method.

=> In this, the information of the file is processed in order. i.e. one after another.

=> A process can read all the data in a file in order starting from beginning but can't skip & read arbitrarily from any location.

ii) Direct Access

=> There is no order of storage of files.

=> It allows arbitrary blocks to be read or write.

=> Very useful for immediate access to large amount of information.

Indexed Access

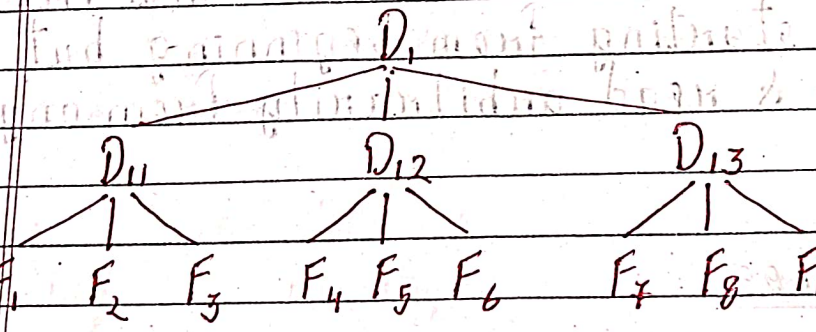
=> In this method, an index is created which contains a key field & pointers to the various blocks.

=> To find a record, we first search the index & then by using this index reaches to the desired record.

Directory Structure

=> It is a container that is used to contain folders & file.

=> It organises files & folders into a hierarchial manner.



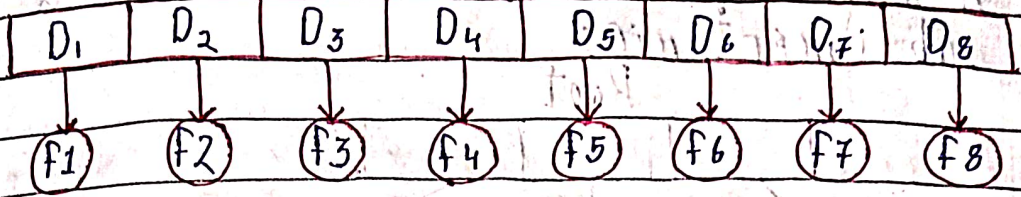
1) Single level

=> It is a simplest directory structure.

=> In it all files are contained in same directory which make it easy to support & understand.

Page No.
Date: / /

⇒ Since all the files are in the same directory, they must have the unique name.

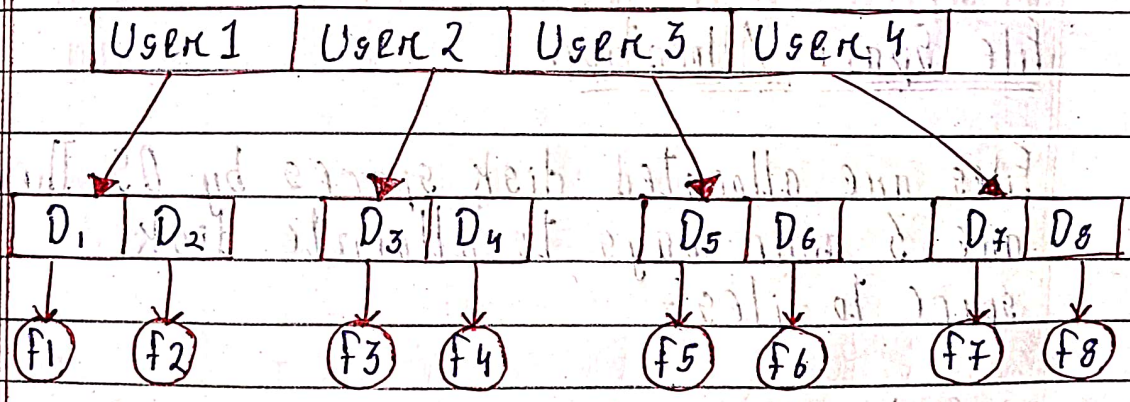


2) Two-level

⇒ Each user has their own user files directory (UFD).

⇒ The UFDs has similar structures, but each lists only the files of a single user.

⇒ When user login, the system's master file directory (MFD) is searched.

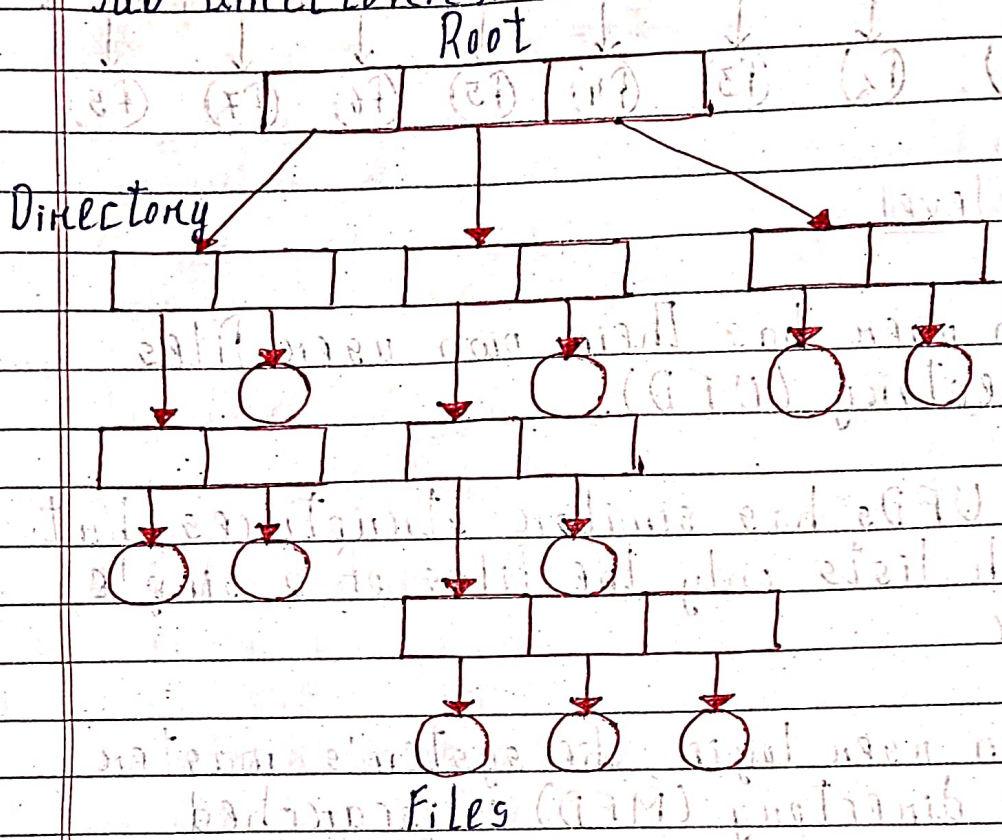


3) Tree-Structure

⇒ Allows user to create their own sub-directories & organise their files accordingly.

=> The tree has a root directory.

=> A directory contains a set of files & sub-directories.



File Space Allocation

Files are allocated disk spaces by OS. There are 3 main ways to allocate disk space to files:-

i) Contiguous
ii) Linked
iii) Indexed

i) Contiguous Allocation

=> Each file occupies a contiguous set of blocks on the disk.
For eg. if a file requires n blocks & is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2, ... b+n-1.

=> This means that given the starting block address & the length of the file, we can determine the blocks occupied by the file.

=> The directory entry for a file with contiguous allocation contains:
a) Address of starting block
b) Length of the allocated portion.

ii) Linked Allocation

=> Each file is a linked list of disk blocks which need not to be contiguous.

=> The disk blocks can be scattered anywhere on the disk.

=> The directory entry contains a pointer to the starting & the ending file block.

Page No.
Date: / /

=> Each block contains a pointer to the next block occupied by the file.

iii) Indexed Allocation

=> A special block known as the index block contains the pointers to all the blocks occupied by a file. Each file has its own index block.

=> The ith entry in the index block contains the disk address of the ith file block.

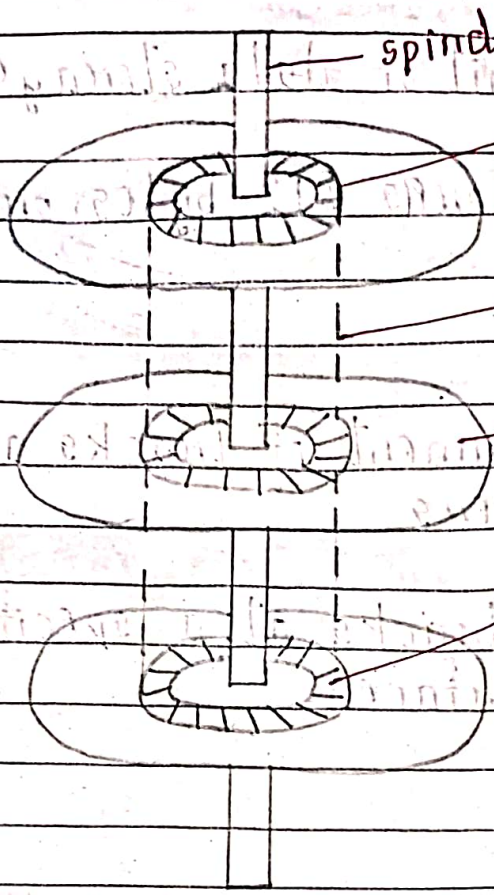
=> Directory contains the addresses of index blocks of files.

=> Each file has its own index block which stores the addresses of disk space occupied by the file.

Disks

Page No.
Date: / /

Physical Data Structure



1) Disk Platter

The actual storage medium, usually a flat, circular disk made up of metal or glass.

The surface is coated with a magnetic material where data is stored.

2) Tracks

Concentric circles on the disk platter where data is written.

Each track is divided into smaller sections called sectors.

Page No.
Date: / /

3) Sectors

The smallest unit of data storage on a disk.

Each sector stores 512 bytes or more.

4) Cylinders

A vertical alignment of tracks across multiple platters.

Represents all tracks at a specific radius across all surfaces.

5) Spindle

The central axis around which the platters rotate.

Maintain a const. rotational speed.

Disk Scheduling Algorithms

It determine the order in which disk I/O requests are processed to minimize seek time, improve efficiency & reduce latency.

Page No.
Date: / /

Seek Time : It is the time taken by the disk arm to locate the desired track.

Rotational Latency : The time taken by a desired sector of the disk to rotate itself to the pos^n where it can access the Read / Write heads is called rotational latency.

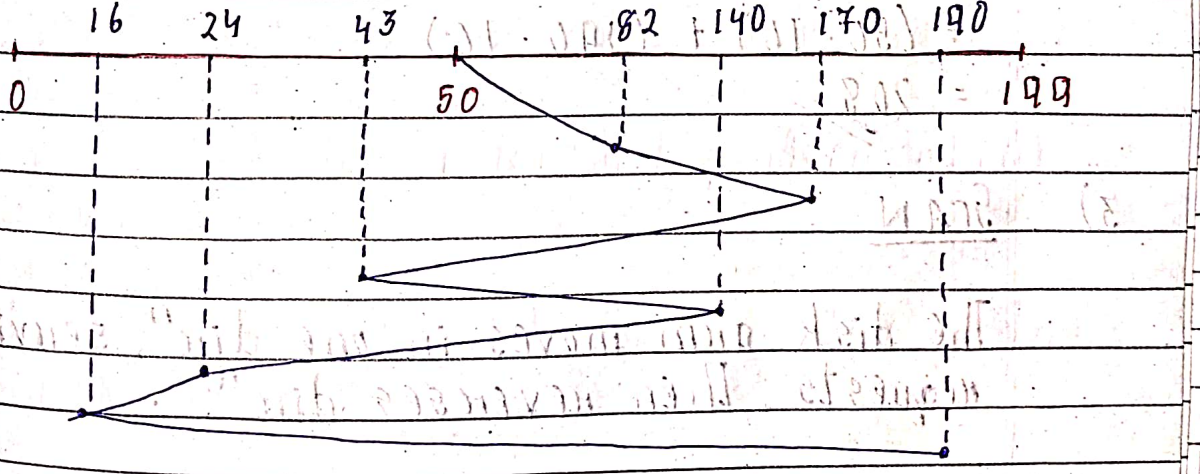
Transfer Time : It is the time taken to transfer the data requested by the processes.

1) FCFS Disk Algorithm

Requests are processed in the order they arrive in the queue.

Q. A disk contains 200 tracks (0-199) request queue having track no. 82, 170, 43, 140, 24, 16, 190 respectively.
Current pos^n of R/W head = 50.
Calculate total no. of tracks movement by R/W head.

A.



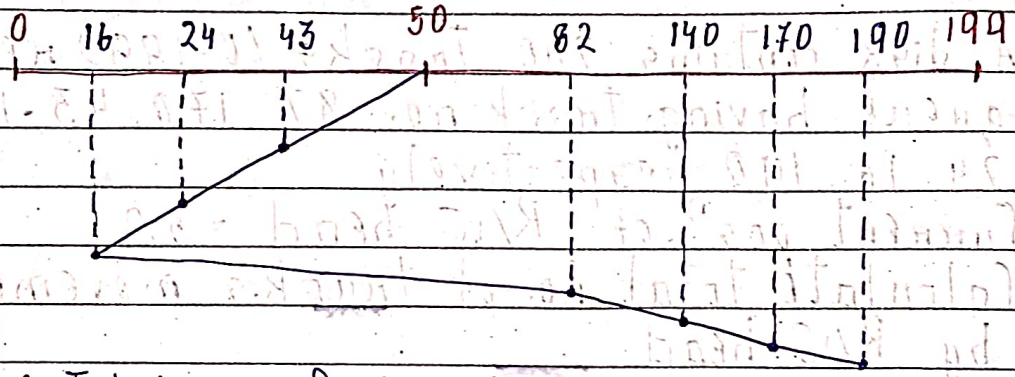
∴ Total no. of tracks
= (170-50) + (170-43) + (140-43) + (140-16) + (190-16)
= 642

2) Shortest Seek Time First (SSTF)

Selects the request closest to the current head pos^n.

Q- A disk contains 200 tracks request queue having track no. 82, 170, 43, 140, 24, 16, 190 respectively.
Current pos^n of R/W head = 50
Calculate the total no. of track movement by R/W. head.

A-



∴ Total no. of tracks
= (50-16) + (190-16)
= 208

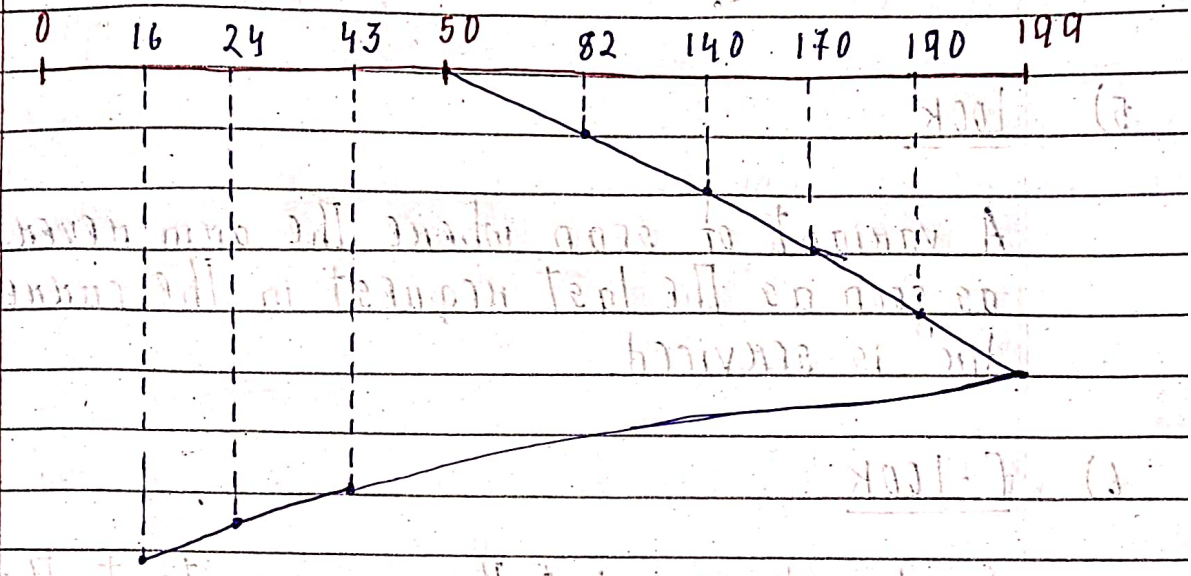
3) SCAN

The disk arm moves in one dir^n, servicing requests, then reverses dir^n.

Page No.
Date: / /

Q- A disk contains 200 tracks request queue having track no. 82, 170, 43, 140, 24, 16, 90 respectively.
Current pos^n of R/W head = 50
Calculate the total no. of track movement by R/W head.

A-



∴ Total no. of tracks
= (199 - 50) + (199 - 16)
= 332

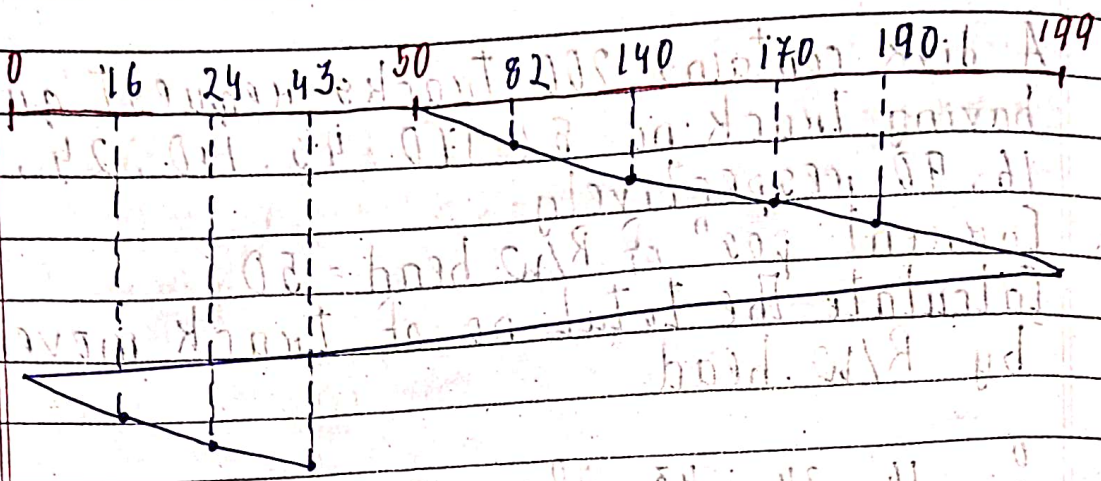
4) C-SCAN

Similar to scan but the arm moves to the end, resets & starts from the beginning.

Q- A disk contains 200 tracks request queue having track no. 82, 170, 43, 140, 24, 16, 90 respectively.
Current pos^n of R/W head = 50
Calculate the total no. of track movement.

Page No.
Date: / /

A-



5) LOOK

A variant of scan where the arm reverses as soon as the last request in the current dirⁿ is serviced.

6) C-LOOK

Similar to LOOK but the arm resets to the beginning instead of reversing dirⁿ.