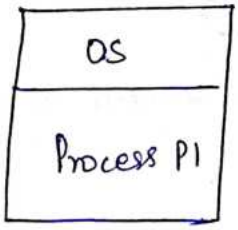
1. Uniprogramming OS

OS allows only one process to reside in the main memory.

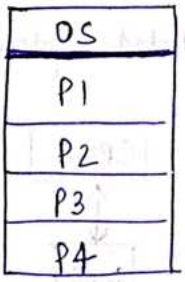


→ Single process cannot keep CPU & I/O devices busy simultaneously.

→ Not good at CPU utilization.

2. Multiprogramming OS

OS allows multiple processes to reside in main memory.



→ Better CPU utilization than uniprogramming.

Degree of multiprogramming :- No. of running programs (processes) in main memory.

★ As degree of multiprogramming increases, CPU utilization also increases. (but upto a certain limit)

↓ discussed later in virtual memory.

Types of multiprogramming OS

- **Preemptive**
  - Process can be forcefully taken out of CPU
- **Non preemptive**
  - Process runs in CPU till its wish
    - ↳ either till proces terminates
    - ↳ it requires I/O operation

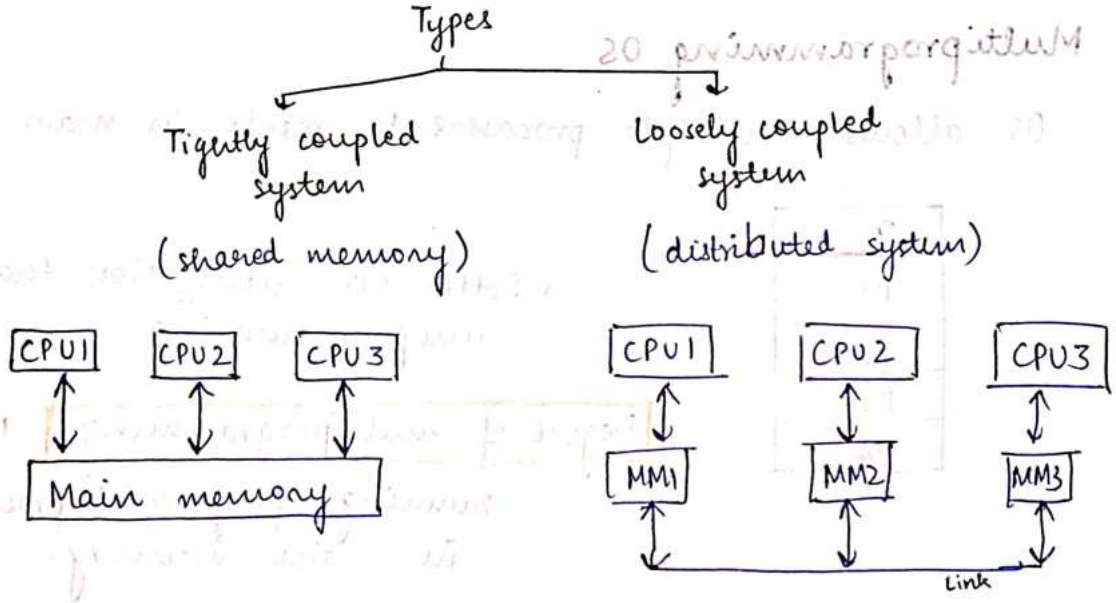
3. Multitasking OS

Extention of multiprogramming OS in which processes execute in round robin fashion.

also known as time sharing OS

4. Multiprocessing OS

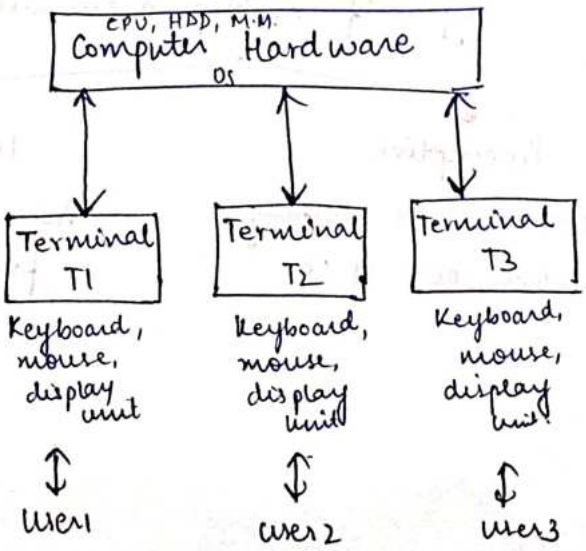
OS is used in systems with multiple CPUs



5. Multi user OS

→ OS allows multiple users to access single system simultaneously.

Every user has a different desktop & will work as if he were working alone.



users work parallely on different OS system.

Windows → not multiuser

Linux, Unix → multi user os.

6. Embedded OS

An OS for embedded computer systems.

Designed for specific purpose, to increase functionality & reliability for achieving a specific task.

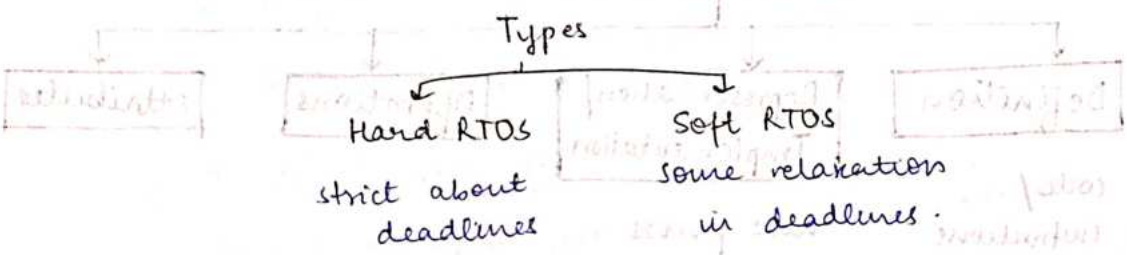
→ User interaction with OS is minimal.

7. Real time OS

Real Time Operating systems (RTOS) are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed within a short time or within certain deadlines.

Ex→ OS used for rocket launching

Every process has a deadline.



8. Hand held OS

OS used in hand-held devices.

Example → iOS, Android etc.

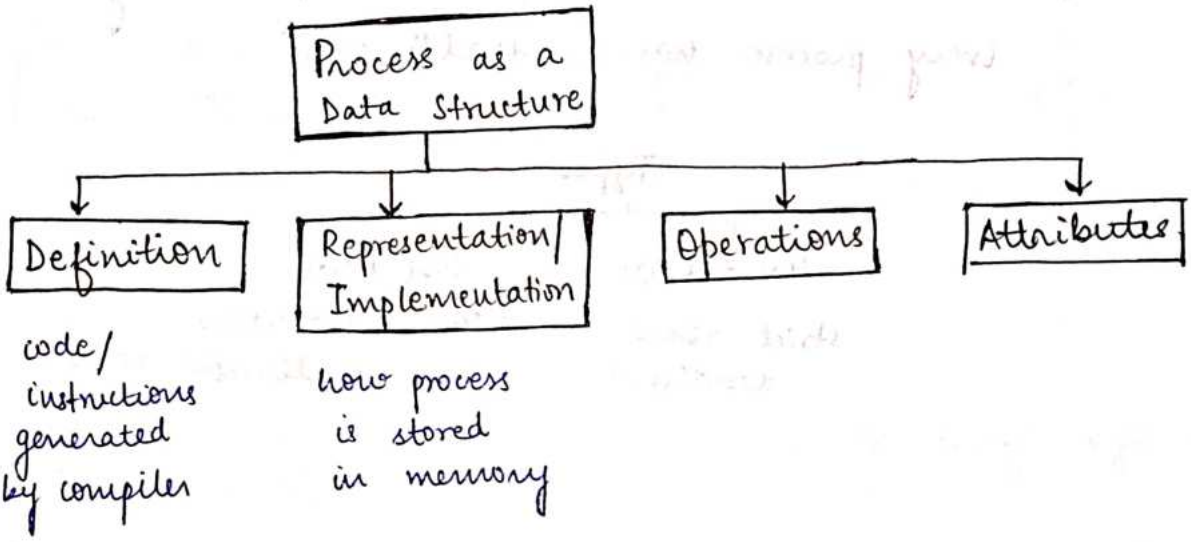
L3

PROCESS

↦ program under execution is called process.

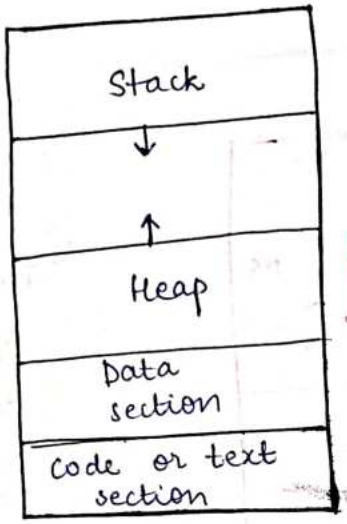
Process = Program (code) ⇓ instructions + Runtime activity ↳ operands & other information

Process
- ↦ Program under execution
- ↦ An instance of a program
- ↦ Schedulable / Dispatchable unit (CPU)
- ↦ Unit of execution (CPU)
- ↦ Locus of control (OS)



Representation of a Process

Stack grows downwards
Heap grows upwards



{ local variables, function parameters, return addresses

{ Dynamic memory allocation

Public/ static variables }

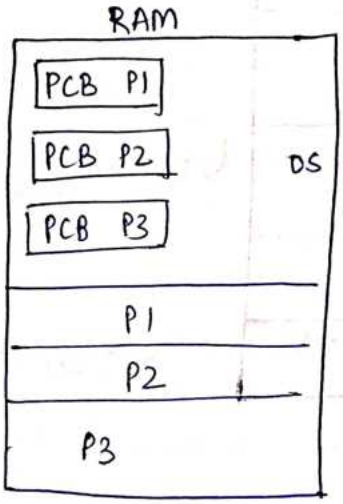
{ Instructions

Attributes of a process —

1. Process ID
2. Program counter
3. General Purpose registers
4. List of open files
5. List of open devices
6. Priority
7. State
8. Protection (memory limits)

} Process Control Block (data structure that contains details of the processes)

Process Control Block is also known as process descriptor.



"The content of PCB of a process are collectively known as Context" → of that process

— Mahatma Gandhi

'Context switch'

Context switch
stop a running process and run other process.
↳ context save
↳ context load

* context switch is done by Dispatcher.

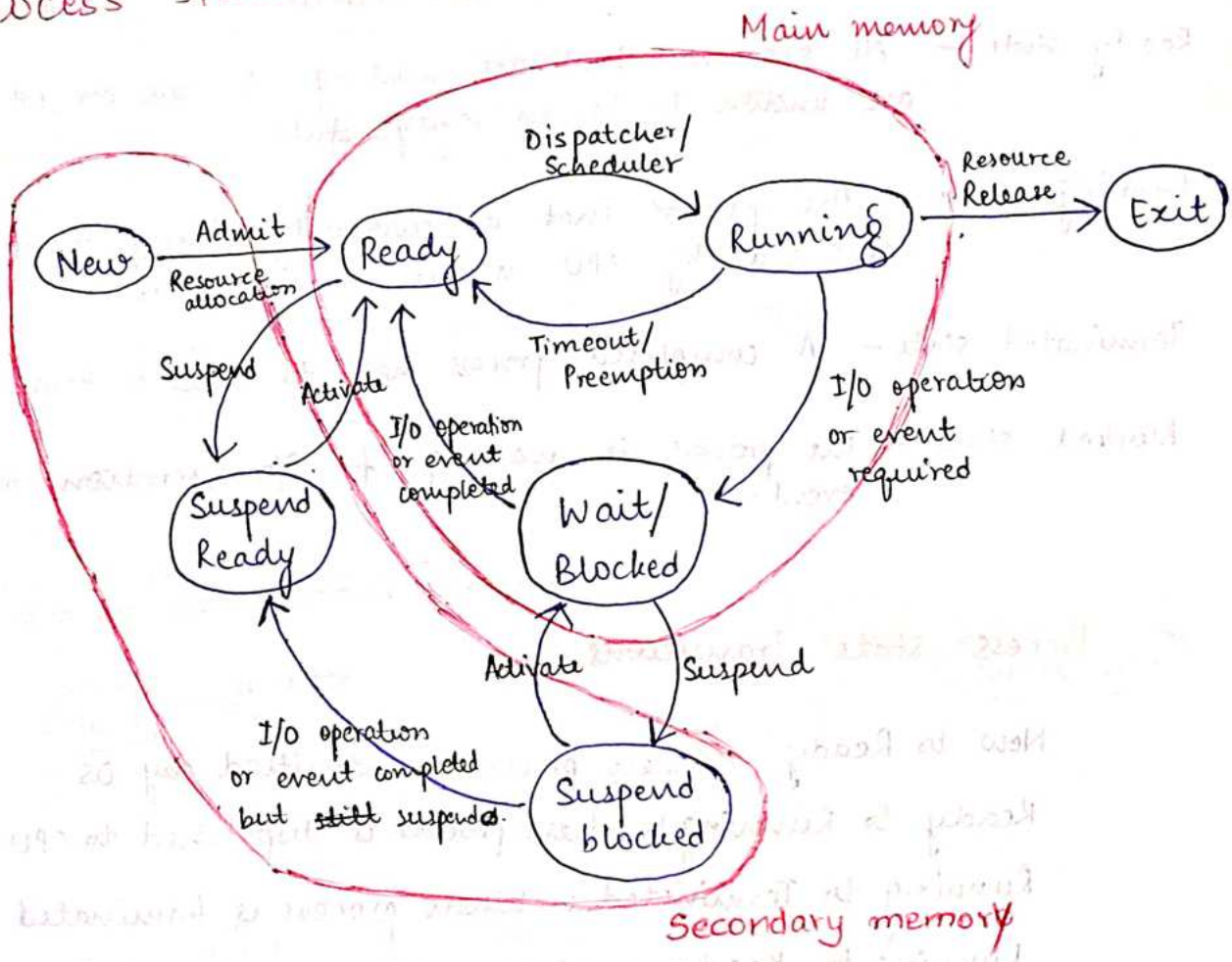
*** While running, a process can access its PCB from main memory?

False

because PCB resides in the OS section of mm
a process cannot access info of OS.

L-4

Process states



Transitions that a process can do according to its own will / wish —

- Running state → Terminate state
- Running state → Blocked/ Wait state.

All other transitions are done by operating system. Process has no control over other transitions.

New state - All installed processes residing in the secondary memory are known to be in new state.

Ready state - All processes that are waiting to run on CPU are known to be in ready state.

Running state - The process that is currently being executed by CPU is in running state.

Terminated state - A completed process has its state as terminated.

Blocked state - The process is waiting for I/O operation or an event.

Process state Transitions.

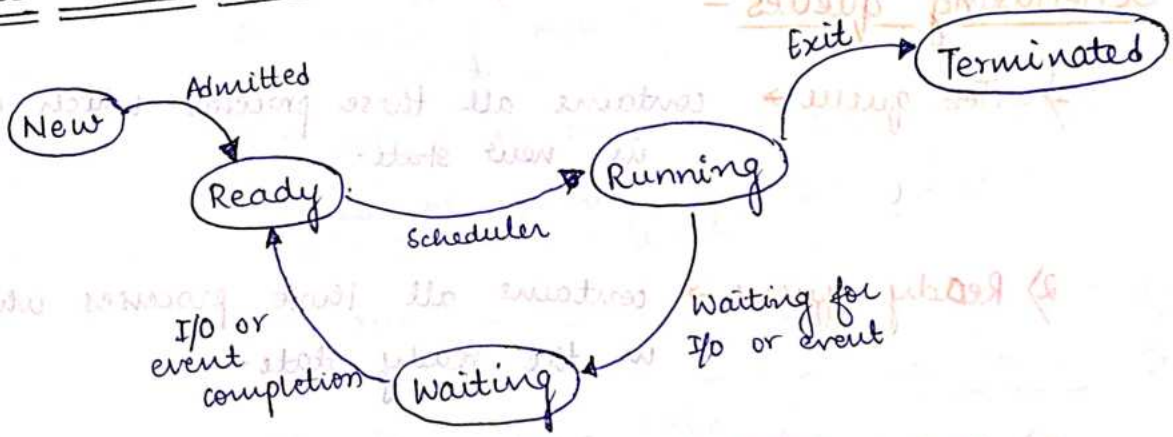
New to Ready :- When process is admitted by OS.
Ready to Running :- When process is dispatched to CPU.
Running to Terminated :- When process is terminated
Running to Ready :- When process is pre-empted
Running to Blocked :- When I/O operation or event
Blocked to Ready :- When process completes I/O or event

2 Transitions are voluntary -
↦ Running to Blocked.
↦ Running to Terminated

Ques :- n processes m same CPU's
Find max & min no of processes in following states -

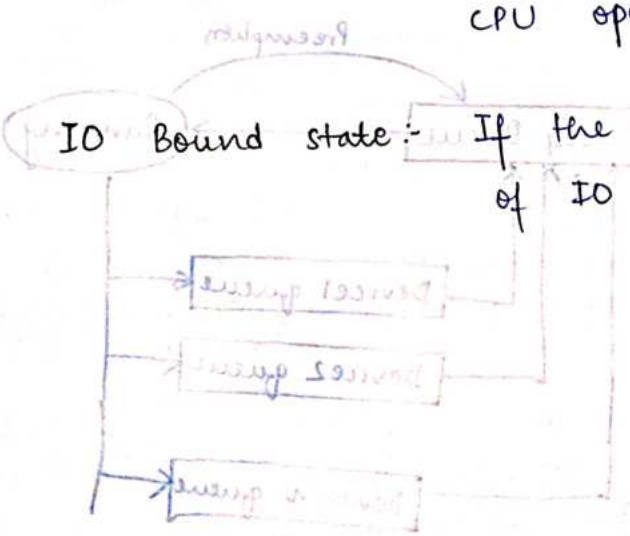
	min	max
Running	0	m
Ready	0	n
Blocked	0	n

Process States : Non pre-emptive -



CPU Bound state :- If the process is intensive in terms of CPU operations.

IO Bound state :- If the process is intensive in terms of IO operations.



A process which has just terminated but has to relinquish its resources is called Zombie process

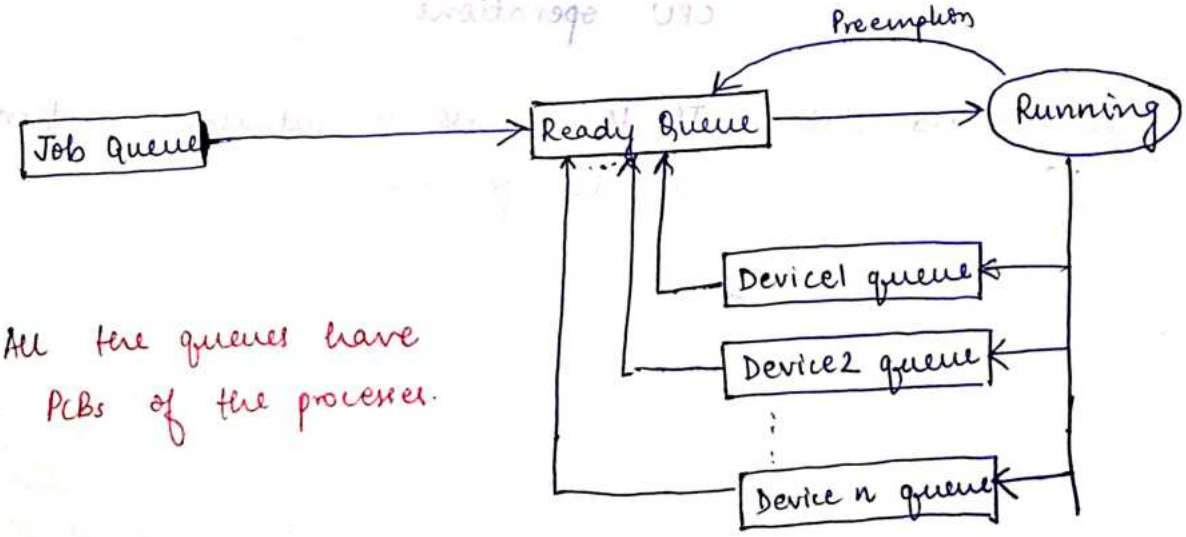
Process scheduling is needed
↳ for better ~~or~~ resource utilization.

Scheduling queues -

1) Job queue → contains all those processes which are in new state.

2) Ready queue → contains all those processes which are in the ready state.

3) Device queue → contains all those processes which are waiting for a specific device.
[every device has its own device queue]



All the queues have PCBs of the processes.

Names are given by frequency of usage.

Types of schedulers –

1) Long Term scheduler (Job)

[Resource allocation] Schedules the processes from new state to ready state [from job queue to ready queue]

2) Short Term scheduler (CPU)

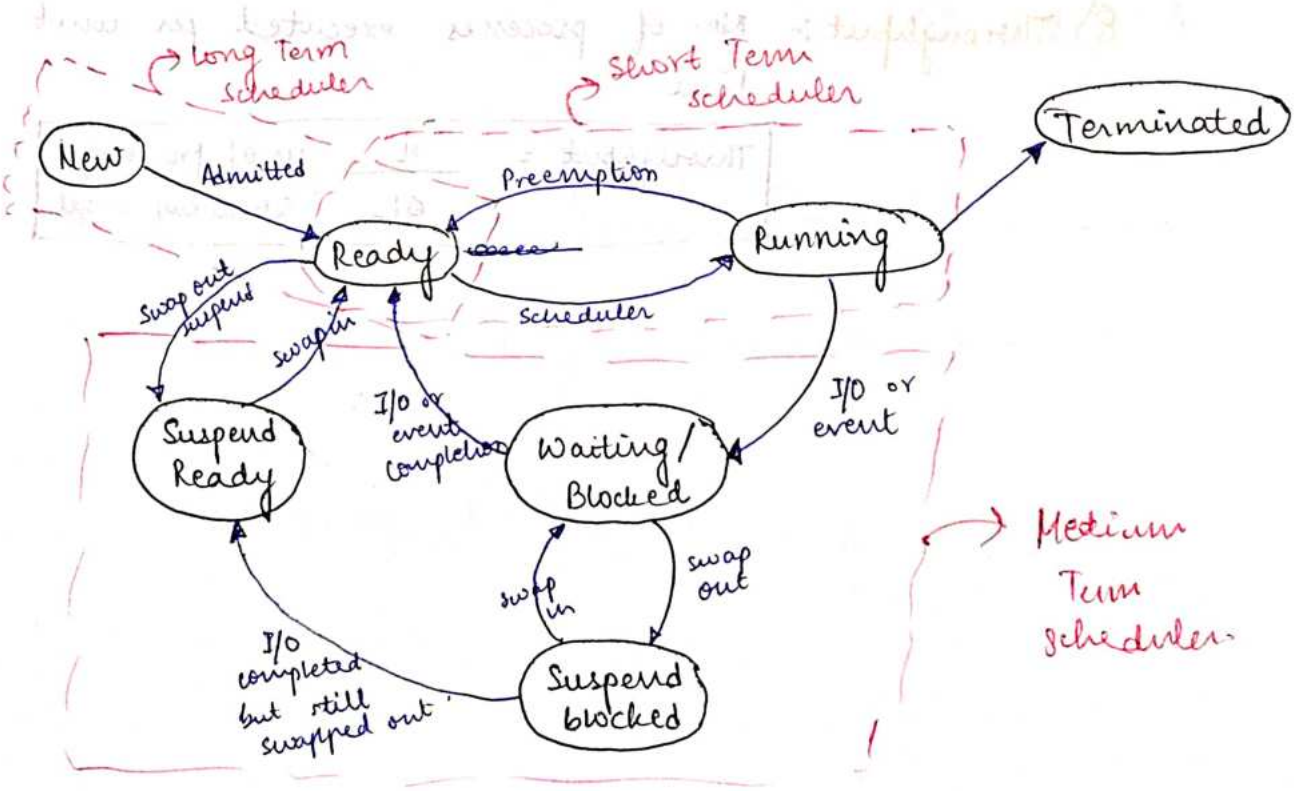
selects one of the ready processes to run on CPU.

3) Mid term scheduler

does swapping of processes from main memory to secondary memory to create space.

Swapping is ~~do~~ known as rolling when it is based on priority of the process.

The area in secondary memory where swapped process is kept is called swap space. It can be accessed only by OS.



L-6

Scheduling Times

1) Arrival Time :- time at which process arrives in the system

2) Burst Time :- amt. of time for which process runs on CPU.

3) Completion Time :- time at which process completes

4) Turn around time :- time from arrival to completion

$$TAT = CT - AT$$

5) Waiting Time :-

$$WT = TAT - BT$$

6) Response Time :- amt. of time after arrival at which process gets allocated CPU for the first time.

~~7) Deadline :-~~

7) Scheduling length (L) :- total duration of time for which short term scheduler was scheduling the processes

$$SL = \max(CT) - \min(AT)$$

8) Throughput :- No. of processes executed per unit time

$$\text{Throughput} = \frac{n}{SL} \quad \frac{\text{No. of processes}}{\text{Scheduling length}}$$

Scheduling algorithms —

1) First come First Serve (FCFS)

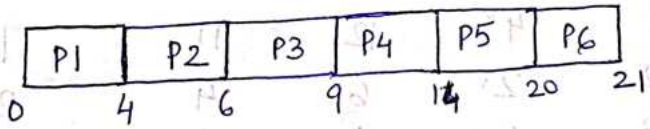
Criteria :- Arrival Time (AT)

Tie breaker :- Process with smaller process ID

Type :- Non preemptive

Process	AT	BT	CT	TAT	WT
P1	0	4	4	4	0
P2	1	2	6	5	3
P3	2	3	9	7	4
P4	3	5	14	11	6
P5	4	6	20	16	10
P6	5	1	21	16	15

Gantt chart :-



$$(TAT)_{avg} = \frac{4+5+7+11+16+16}{6} = \frac{59}{6} = 9.83$$

$$(WT)_{avg} = \frac{0+3+4+6+10+15}{6} = \frac{38}{6} = \frac{19}{3} = 6.33$$

Scheduling length, $L = \max(CT) - \min(AT)$

$= 21 - 0 = 21$

$$\text{Throughput} = \frac{n}{L} = \frac{6}{21} = \frac{2}{7} = 0.285$$

FCFS suffers from convoy effect

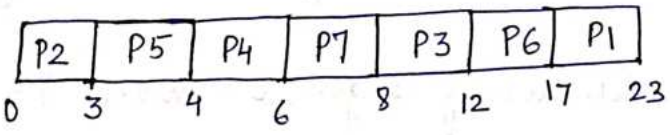
If a large process is scheduled first, then, it slows down the system's performance.

2) Shortest Job First (SJF)

Criteria :- Burst Time (smallest BT process scheduled first)
Tie Breaker :- FCFS
Type :- non-preemptive

Process	AT	BT	CT	TAT	WT
P1	0	6	23	23	17
P2	0	3 ✓	3	3	0
P3	1	4 ✓	12	11	7
P4	2	2 ✓	6	4	2
P5	3	1 ✓	4	1	0
P6	4	5 ✓	17	13	8
P7	6	2 ✓	8	2	0

Gantt chart -



$$(TAT)_{avg} = \frac{23+3+11+4+1+13+2}{7} = \frac{57}{7} = 8.14$$

$$(WT)_{avg} = \frac{17+0+7+2+0+8+0}{7} = \frac{34}{7} = 4.85$$

$$L = \max(CT) - \min(AT) = 23 - 0 = 23$$

$$\text{Throughput} = \frac{n}{L} = \frac{7}{23}$$

3) Shortest Remaining Time First (SRTF)

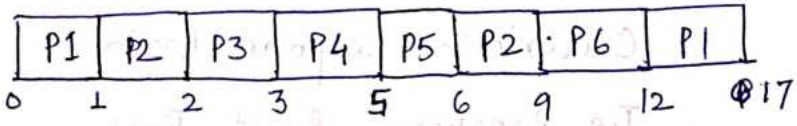
Criteria :- ~~Burst~~ Burst time

Tie Breaker :- FCFS

Type :- Pre-emptive.

Process	AT	BT	CT	TAT	WT
P1	0	6	17	17	11
P2	1	4	9	8	4
P3	2	1	3	1	0
P4	3	2	5	2	0
P5	4	1	6	2	1
P6	5	3	12	7	4

Gantt chart :-



$$(TAT)_{avg} = \frac{17+8+1+2+2+7}{6} = \frac{37}{6} = 6.16$$

$$(WT)_{avg} = \frac{11+4+0+0+1+4}{6} = \frac{20}{6} = \frac{10}{3} = 3.333$$

$$L = \max(CT) - \min(AT) = 17 - 0 = 17$$

$$\text{Throughput} = \frac{n}{L} = \frac{6}{17}$$

No. of context switches :- 7

Problems with SJF and SRTF —

1. Starvation (indefinite time)
2. No fairness
3. Practical implementation not possible.

4) Highest Response Ratio Next (HRRN)

Objective:- not only favours short jobs but also decreases waiting time of longer jobs.

Criteria:- Response Ratio = $\frac{WT + BT}{BT}$

Tie Breaker:- Burst Time

Type:- Non preemptive

5) Priority based scheduling

Criteria :- Priority

Tie Breaker :- FCFS

Type — Preemptive / Non preemptive } will be given in question.

Priority can be fixed or dynamic

↳ will be given in question.

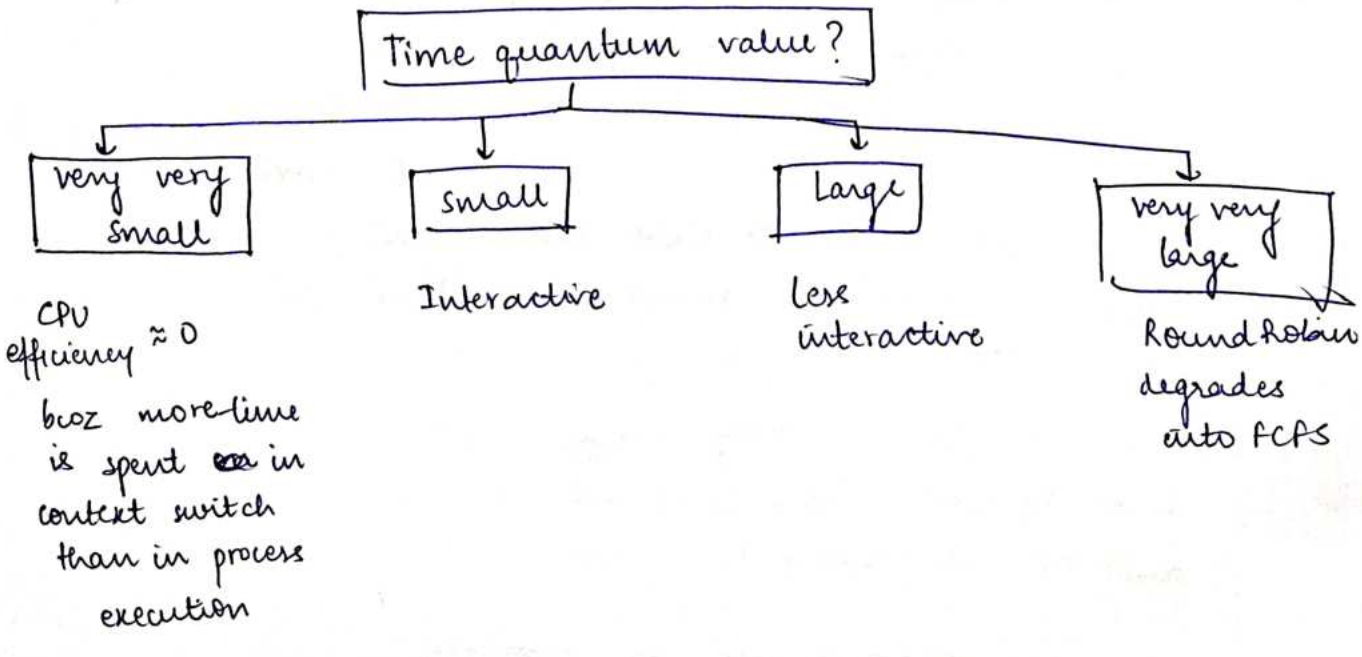
Priority based algorithm suffers from starvation
if higher priority processes keep arriving then, low priority processes may wait till indefinite time

Solution :- Aging
↳ after a fixed duration, increase priority of all waiting processes by 1
↳ applicable for dynamic priority

6) Round Robin

Criteria :- AT + Q Q = Time quantum
Tie Breaker :- Process ID
Type :- Preemptive

Objective :- provides interactiveness



Advantages of round robin —

1. fairness
2. interactiveness for time sharing system
3. Burst time of processes not required to be known beforehand.

If the waiting time (I/O time) for a process is p (fraction of time) and there are n processes in the memory, then, CPU utilization is

Total waiting time fraction = p^n

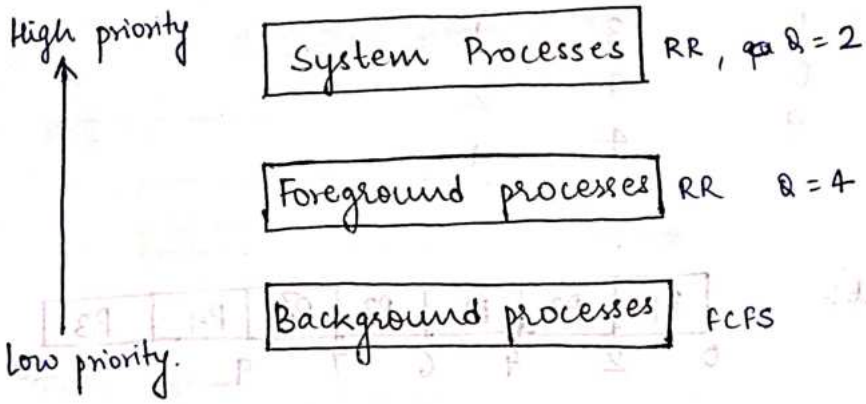
$\therefore$ CPU utilization = $1 - p^n$

L-11

Multilevel Queue (MLQ) Scheduling

Different types of processes need different scheduling algorithms.

Different types of processes are segregated and kept in different ready queues —



Priority is implemented on queues (not processes)

Scheduling among queues —

↳ Fixed priority preemptive scheduling method —

- Priority of system processes queue is highest.
- If background process is executing (when other 2 queues are empty) and a system process arrives, then, background process is preempted
- suffers from starvation.

↳ Time slicing

The entire time of CPU is divided into fractions and each queue is assigned a fraction of time in which it executes.

Ex → System process — 60% CPU time
foreground process — 30% CPU time
Background process — 10% CPU time

~~problem~~ no starvation in time slicing.

Question

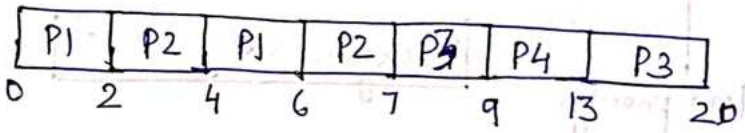
Fixed priority preemption.

high priority
Queue 1 : RR with Q = 2

Queue 2 : FCFS

Process	AT	BT	Queue
P1	0	4	1
P2	0	3	1
P3	0	9	2
P4	9	4	1

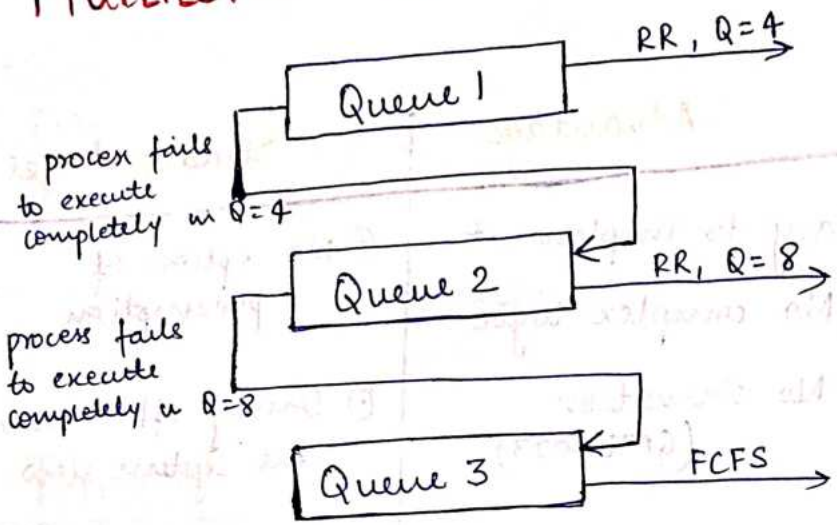
Gantt chart :-



Disadvantages of Multilevel Queue scheduling

- Starvation if high priority queues never become empty.
- inflexible in nature

Multilevel Feedback Queue (MLFQ) Scheduling



Priority of process is reduced dynamically by transferring it to lower priority queue.

Advantage –
↳ Flexible

Disadvantage –
↳ Starvation
(problem can be solved by aging
moving process to higher level queue after certain duration)

Scheduling algorithms : A comparison.

Algorithm	Advantages	Disadvantages
FCFS	⊙ Easy to implement ⊙ No complex logic ⊙ No starvation (GATE 2023)	⊙ No option of preemption ⊙ Convoy effect makes the system slow.
SJF	⊙ Minimum average waiting time among non preemptive algorithms ⊙ Better throughput	⊙ No practical implementation ⊙ No option of preemption ⊙ Longer processes suffer from starvation
SRTF	⊙ Minimum ^{avg} waiting time among all algorithms. ⊙ Better throughput	⊙ No practical implementation ⊙ Longer processes may suffer from starvation
Priority based.	⊙ Better response to real time ~~so~~ situations	⊙ Low priority processes may suffer from starvation.

Round Robin	⊙ No starvation ⊙ Better interactiveness ⊙ Burst time not required to be known	⊙ $(TAT)_{avg}$ & $(WT)_{avg}$ is more ⊙ Can degrade to FCFS

L-12

Thread

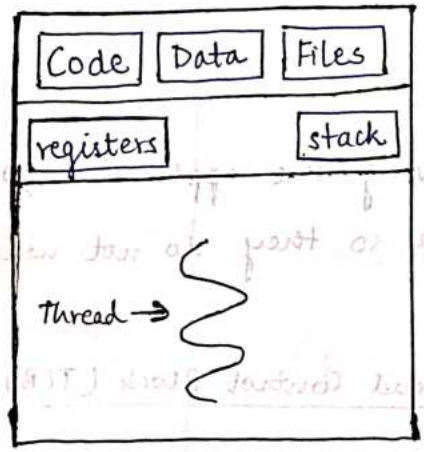
↳ component of a process
↳ Lightweight process

⊙ Threads run in parallel improving the application performance.
⊙ Threads can share common data so they do not need interprocess communication
⊙ Each thread has its own Thread Control Block (TCB)

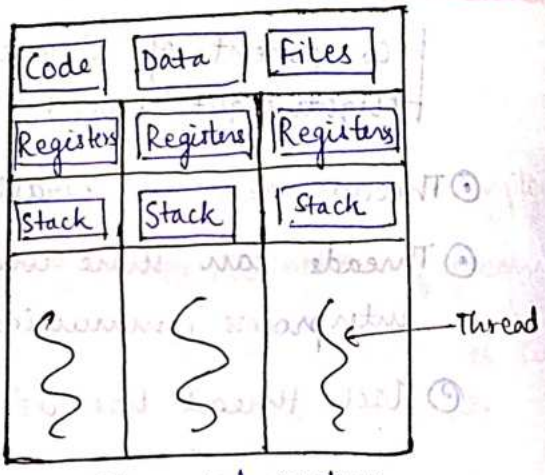
Process v/s Thread [GFG]

Process	Thread
1. Process means any program in execution.	segment of a process
2. more time in creation & termination	takes less time in termination & creation.
3. Process is isolated	Threads share same memory
4. If one process is blocked, it will not affect other processes	If user level threads is blocked, all other user level threads are blocked
5. Process has its own PCB, address space & stack	Thread has parent's PCB, its own TCB, address space & stack.
6. System call is involved in it	No system call is involved, it is created using API

Shared among all Threads	Unique for each thread
Code section	Thread ID
Data section	Register set
OS Resources	Stack
Open files & signals	Program Counter



Single Threaded system



Multithreaded system

Advantages of multithreading

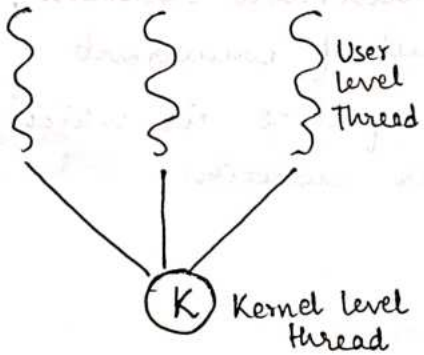
- Responsiveness
- Faster context switch
- Resource sharing
- Economy
- Communication

Types of threads

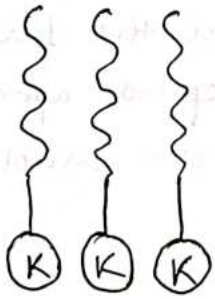
- User level thread (multithreading in user processes)
- Kernel level thread (multithreading in kernel processes)

User Threads	v/s	Kernel Thread
Multithreading in user process		Multithreading in kernel process
Created without kernel intervention		Kernel itself is multithreaded
Context switch is very fast (bcoz kernel isn't involved)		Context switch is slow
If one thread is blocked, OS blocks entire process		Individual threads may be blocked.
Generic & can run on any OS		Specific to OS
Faster to create & manage		Slower to create & manage

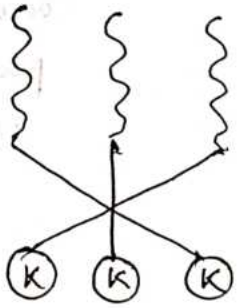
Multithreading models —



Many to one



One to one



Many to many

L-13

PROCESS SYNCHRONIZATION

2 types of processes

↳ Independent :-
no any communication with any other process.

↳ Cooperating / Coordinating / Communicating
can affect other process or can be affected by other process.

- Problems without synchronization —
  - ↳ Inconsistency
  - ↳ Loss of data
  - ↳ Deadlock

Critical section :-
code ~~sectio~~ segment where the ~~sec~~ shared variables / resources can be accessed.

Race Condition :-
A race condition is an undesirable situation, it occurs when the final result of concurrent processes depends upon the sequence in which the processes complete their execution.

Requirements of critical section problem solutions —

1. Mutual exclusion
2. Progress
3. Bounded waiting

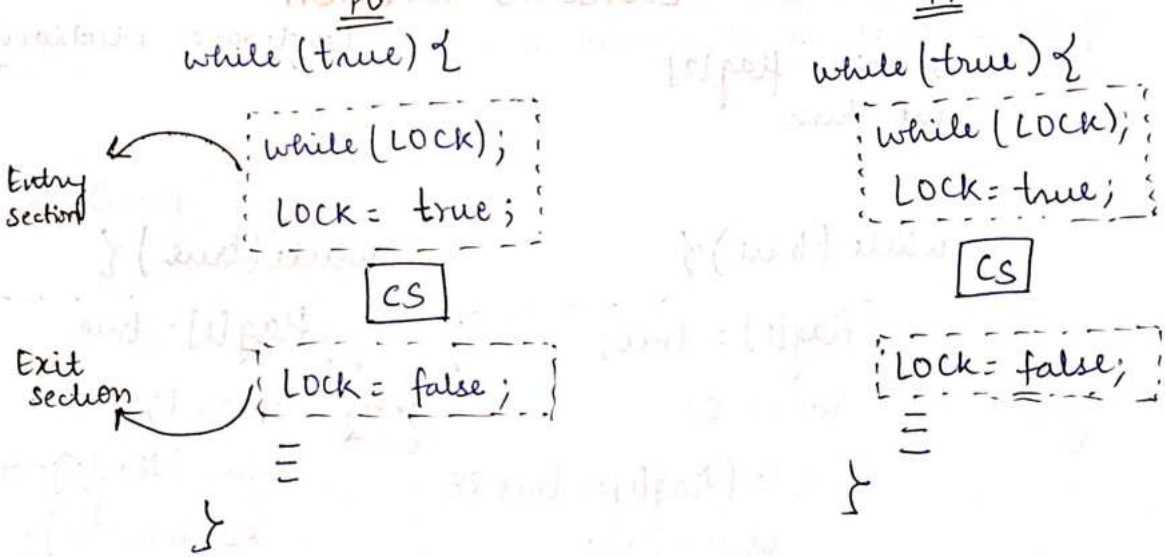
Mutual exclusion :- if one process is using CS, then other process cannot use that C.S.

Progress :- if no any process is there in C.S. & atleast one process wants to enter C.S. then, it should be allowed.

Bounded waiting :- waiting of process should be bounded.

Solution 1 : LOCK VARIABLE (software solution)

boolean LOCK = false



Mutual exclusion — X

Solution-2: Using TURN variable

(software solution)

```
int turn = 0;

P0                                  P1
while (true) {                      while (true) {
    while (turn != 0);  ---Entry section---  while (turn != 1);
    CS                              CS
    turn = 1;  ---Exit section---   turn = 0;
}                                   }
```

Mutual exclusion - ✓

Progress - X (bcoz of strict alteration)

Bounded waiting - ✓

Solution-3: Peterson's solution

(software solution)

```
boolean flag[2]
int turn

while (true) {                      while (true) {
    flag[0] = true;                     flag[1] = true;
    turn = 0;                           turn = 1;
    while (flag[1] == true &&           while (flag[0] == true
           turn == 0);                         && turn == 1);
    CS                                  CS
    flag[0] = false;                    flag[1] = false;
}                                   }
```

Entry section

Exit section

Mutual exclusion - ✓

Progress - ✓

Bounded waiting - ✓

L-15

SYNCHRONIZATION HARDWARE

instruction support by CPU which can be used to provide synchronization

↳ privilaged instructions ~~not~~ which execute in kernel mode.

1. TestAndSet( )

Returns the current value flag and sets it to true.

```
Boolean LOCK = false;

Boolean TestAndSet (Boolean *trg)
{
    Boolean rv = *trg;
    *trg = TRUE
    Return rv;
}
```

atomic operation

```
while (true) {
    while (TestSetLock (&Lock));
        CS
    LOCK = false;
}
```

Busy waiting solution

↳ A process executes on CPU but cannot go further because of CS solution, then, the process is known to be busy waiting

2. Swap( )

```
Boolean Key, Lock = false;
while (true) {
    Key = True;
    while (key == True)
        Swap(&key, &Lock);
    CS
    Lock = False;
}
```

preemption

No mutual exclusion

```
void Swap(Boolean *a,
Boolean *b) {
    Boolean temp = *a;
    *a = *b;
    *b = temp;
}
```

L-16

Synchronization Tools

- Semaphores
- Monitor.

Semaphore –

Synchronization tool that is an integer value which can be accessed using the following functions only-

atomic operations
- ⊙ wait() / P() / Degrade() — lower / down
- ⊙ signal() / V() / Upgrade() — upper / up

busy waiting

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

```
signal(s){
    s++;
}
```

wait :- decreases the value of semaphore variable by one if it is greater than or equal to 0.

signal :- increases the value of semaphore variable by one.

Types of semaphore —

- Binary semaphore
  - ⊙ only 2 values 0 or 1
  - ⊙ used to implement soln of CS problem for mutual exclusion.
- Counting semaphore
  - ⊙ unlimited domain of values (integer)
  - ⊙ used to control the accen of a resource that has multiple instances.

Critical Section solution –

```
S = 1

P0                              P1
while (true) {                  while (true) {
    wait(s)                         wait(s)
    CS                              CS
    signal(s)                       signal(s)
}                               }
```

Characteristics of semaphores –

- used to provide mutual exclusion
- used to control access of resources
- semaphore solution can lead to → Deadlock, Starvation
- Busy waiting solution
- may lead to priority inversion → low priority process को CS mil jaaye high priority process अटकी रह जाए!
- machine independent

Question

Consider a semaphore S initialized with value 10. What should be the value of S after executing 6 times P() and 8 times V() function on S –

Ans = $10 + 8*1 + 6*(-1) = \underline{12}$

Question

Consider a semaphore S, initialized with value 1. Consider 10 processes P1, P2 ... P10. All processes have same code but one process P10 has signal(s) in place of wait(s). If all processes to be executed only once, then, max. no. of processes which can be in CS together → while loop ignore.

```
while (true) {
    wait(s)
    CS
    signal(s)
}
```

Ans. 3

S = 1 0 1 0 1 0 1 0 1

P1 | P10 | P2 | P10 | P3 | P10 | P4 | P10 | P5 ....

∴ all 10 processes can be in CS.

L-18

Classical Problems on synchronization

1. Producer - Consumer problem / Bounded Buffer problem

- Producers must block if the buffer is full.
- Consumers must block if the buffer is empty.

- 3 variables
  - Mutex: Binary semaphore to take lock on buffer (Mutual exclusion)
  - Full: Counting semaphore to denote the no. of occupied slots in buffer
  - Empty: Counting semaphore to denote the no. of empty slots in buffer

- Initialization — (Initially buffer is empty)

  Mutex = 1
  Full = 0
  Empty = n

Producer

```
Producer( ){
    wait (Empty)
    // produce an item
    wait (mutex)
    // add item on buffer
    signal (mutex)
    signal (full)
}
```

Consumer

```
Consumer( ){
    wait( Full)
    wait ( mutex)
    // Remove item from buffer
    signal (mutex)
    // Consume the item
    signal ( Empty)
}
```

2. Reader - Writer Problem

- If a writer is writing a file, no other person should be allowed to read/write at the same time.
- If a reader is reading a file, other readers can read but no writer should be able to write.

- 3 variables –
  - Mutex: binary semaphore for mutual exclusion
  - wrt: binary semaphore to restrict readers & writers if writing is going on.
  - readcount: integer variable, number of active readers.

- Initialization –

```
mutex = 1
wrt = 1
readcount = 0
```

Writer

```
Writer( ){
    wait(wrt)
        //Perform
          writing
    signal(wrt)
}
```

Reader

```
Reader( ){
    wait(mutex)
    readcount++;
    if(readcount == 1){
        ~~signal(wrt)~~ wait(wrt)
    }
    signal(mutex)
        //Perform reading
    wait(mutex)
    readcount--
    if(readcount == 0)
    }   signal(wrt)
    signal(mutex)
}
```

3. Dining Philosopher Problem

```
1. Philosopher(i) {
2.     wait( chopstick[i] )
3.     wait( chopstick[(i+1) % K]
4.         //Eat
5.     signal(chopstick[i])
6.     signal(chopstick[(i+1) % k]
7. }
```

The above solution can lead to <u>deadlock</u> [if all the philosophers get preempted after line 2 i.e. they pick one chopstick]

Solution to deadlock –

↦ There should be atmost (k-1) philosophers on the table if k chopsticks are there.

↦ A philosopher should be allowed to pick their chopstick only if both are available at the same time

* ↦ One philosopher should pick right chopstick first & then left while others pick left first & then right.

<u>(k-1) philosophers</u>

```
wait ( chopstick[i])
wait( chopstick[(i+1)%.k])
    //Eat
signal(chopstick[i])
signal(chopstick[(i+1)%.k])
```

<u>1 philosopher</u>

```
wait ( chopstick[(i+1)%.k])
wait( chopstick[i]);
    //Eat
signal(chopstick[(i+1)%.k])
signal(chopstick[i]);
```

L-21

Deadlock

Allocation of resources to the process is done by operating system.

Operations on resources –

- Request – Process to OS → Please give me this resource plee(s)
- Use – Resource allocated, process uses it
- Release – release of resource by process so that it can be used by some other process.

OS be like – ja ji le apni zindagi

काम खतम तो resource छोड़ दि
~~पतली~~ process

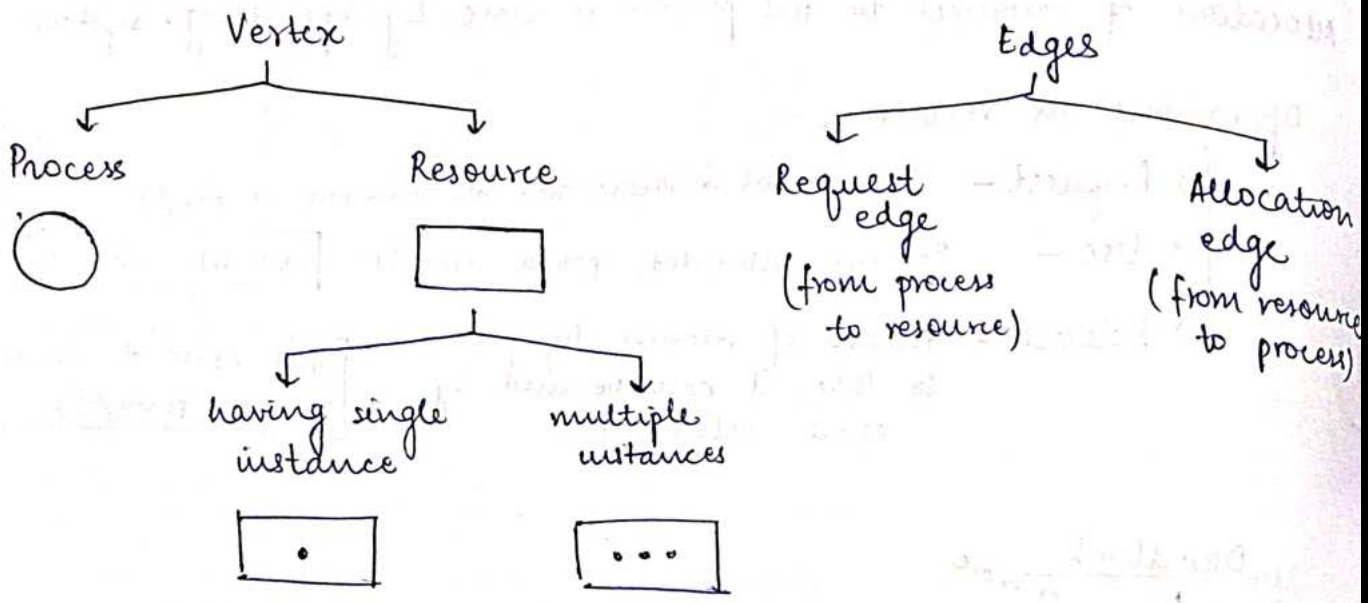
Deadlock

- If 2 or more processes are waiting for an event that is never going to happen, then, the situation is called deadlock.

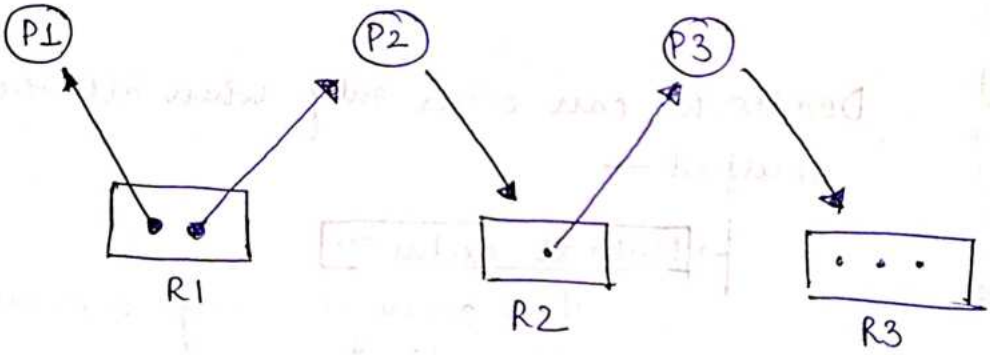
Deadlock can occur only when all the conditions are satisfied —

- **Mutual exclusion**
  - If a process is using a resource no other process can use it.
- **Hold & wait**
  - a process is holding a resource and is requesting (waiting) for other resource.
- **No preemption**
  - a ~~process~~ resource cannot be preempted (i.e. resource cannot be forcefully released from the process)
- **Circular wait**
  - all deadlocked processes should wait for each other in circular manner.

Resource Allocation graph –



Example –



1 instance of R1 is allocated to P1 & 1 instance is allocated to P2.

P2 is requesting resource R2

Resource R2 (instance) is allocated to P3

P3 is requesting resource R3

Recovery from Deadlock —

- Make sure that the deadlock never occurs (Prevent the system from deadlock or avoid deadlock)
- Allow deadlock, detect and recover.
- Pretend that there is no any deadlock (Ostritch algo).

L-22

Deadlock Handling Techniques

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery
- Deadlock Ignorance.

1. Deadlock Prevention —

Prevent any of the four conditions of deadlock from occuring.

1. **Preventing mutual exclusion —**

⊙ Have enough resources to provide simultaneous execution.

↓ required resources will be so many.

⊙ Make all processes independent.

↳ not practically possible.

∴ Mutual exclusion cannot be violated practically.

2. Preventing Hold & Wait

A process will either hold or wait but not together.

1. → If all resources are available then acquire the resources else it waits till all are available (process may suffer from starvation)

2. → New resources can be requested only after the currently allocated resources are released.

3. → Numbering of ~~resources can be done. Only~~ higher numbers

3. Preventing No Preemption

P1 requests for R1 & R1 is held by P2 which is also in wait for other process / resource.

OS may preempt R1 from P2 & give to P1 [bcoz R1 was not currently utilized by P2].

* Not every type of resource can be preempted.

4. Preventing circular wait

A sequence number is assigned to each resource. A process can request a resource whose sequence no. is greater than the sequence no. of the one it is currently holding.

L-23

2. Deadlock Avoidance

In deadlock avoidance, OS tries to keep system in safe state.

Safe state – deadlock will never occur
Unsafe state – deadlock may / may not occur.

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system does not cause deadlock.

Banker's algorithm:- Banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety of the system.

Example from Galvin

Process	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3 ✓
P_1	2	0	0	3	2	2	1	2	2 ✓
P_2	3	0	2	9	0	2	6	0	0 ✓
P_3	2	1	1	2	2	2	0	1	1 ✓
P_4	0	0	2	4	3	3	4	3	1

Available

A	B	C	
3	3	2	
+ 2	0	0	(P1 res)
5	3	2	
2	1	1	(P3 res)
7	4	3	
0	1	0	(P1 res)
7	5	3	
3	0	2	(P2 res)

$< P1 \rightarrow P3 \rightarrow P0 \rightarrow P2 \rightarrow P4 >$
safe sequence.

∴ The system is in safe state.

Banker's algorithm –

'n' numbers of processes 'm' number of resources

1: Allocation – n x m (matrix)
2: Max – n x m matrix
3: Need – n x m matrix
4: Available – m size array.

Banker's safety algorithm –

1. Let Work and Finish be vectors of length 'm' & 'n' resp.
   Initialize: Work = Available
   Finish[i] = false ; for i = 1, 2, 3 .... n

2. Find an i such that
   (a) Finish[i] = false
   (b) Need[i] <= Work
   if no such i exists, goto step 4.

3. Work = Work + Allocation[i]
   Finish[i] = true
   goto step 2

4. If Finish[i] = true for all i,
   then system is in a safe state.

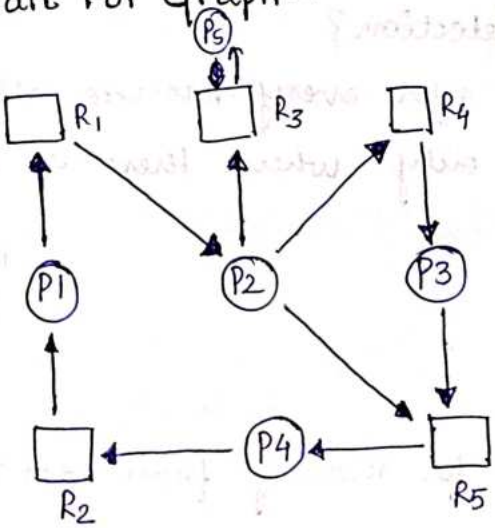
3. Deadlock Detection

↳ wait for graph

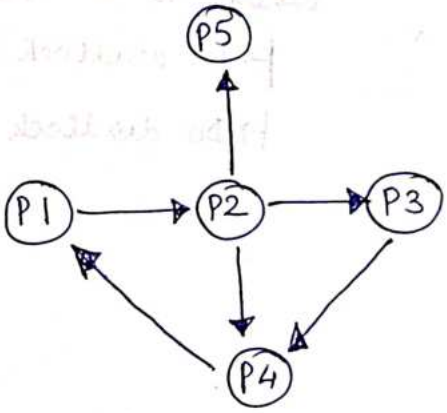
2 situations
- → when all resources have single instance
- → when ~~no~~ resources have multiple instances ↳ detection algo.

→ When all resources have single instance, deadlock detection is done using wait for graph.

Wait For Graph -



Resource allocation graph



Wait For Graph

Edge from P_i to P_j — P_i is waiting for resource that is held by P_j

If a cycle exists in the wait For Graph, then, deadlock exists

Processes deadlocked - P1, P2, P3, P4

P5 can completely execute.

If more than one instance is present for a resource, then, presence of cycle indicates ~~possibul~~ possibility of deadlock.

IT ✓✓✓✓✓ CS ✓✓✓✓

Deadlock detection in case of multiple instances, ~~dete~~ is done by finding safe sequence (same as banker's algorithm).

When to do deadlock detection?

↦ Do deadlock detection after every resource allocation

↦ Do deadlock detection only when there is some clue.

like some thresholds for CPU utilization.

3 basic approaches for recovery from deadlock —

↦ Inform the system operator and allow him/her to take manual intervention.

↦ Terminate one or more processes involved in the deadlock.

Terminate all the processes involved in the deadlock

Terminate processes one by one until deadlock is broken.

↦ Preempt resources.

Important issues to be raised

→ Selecting a victim

→ Rollback

→ Starvation

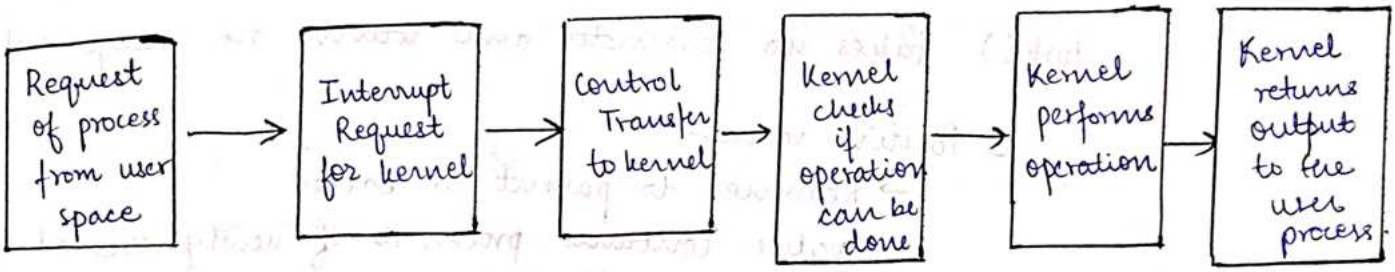
(process becomes victim again & again)

L-27

System call

Programmatic way in which a computer program requests a service from the kernel.

How system call works?



*some operations can only be done by kernel.

Services provided by System Calls —

1. Process Creation and management
2. Main memory management
3. File access, directory & file System Management
4. Device Handling (I/O)
5. Protection
6. Networking

Process Control

- create process
- terminate process
- load/ execute
- get/set process attribute
- allocate & free memory.

File Management

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

Communication

- create, delete communication channel.
- send/receive messages
- transfer status info
- attach/ detach remote devices.

Protection

- get/set file permissions.

fork( )

Fork system call is used for creating a new process, which is called child process which runs concurrently with the processes that makes the fork() call (parent process)

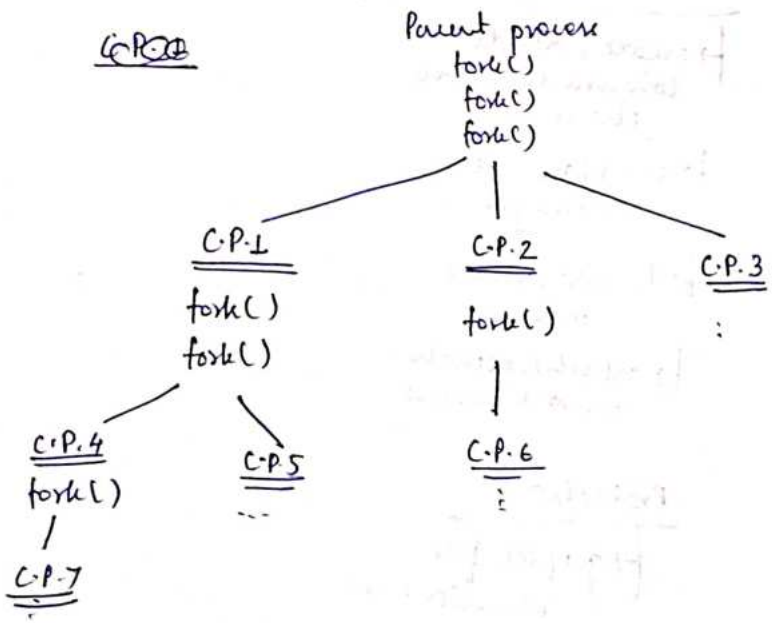
Parameters and Return value –

fork() takes no parameter and returns an integer value

- Positive value :-
  - → Returned to parent or caller.
  - → value contains process ID of newly created child process.
- Negative value :-
  - creation of child process was unsuccessful.
- Zero :-
  - returned to newly created child process.

Ques How many child processes are created ?

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



7 child processes

```
fork()
fork()
 :
fork()
```
} n statements

Total no. of child processes created $= 2^n - 1$

Total no. of processes $= 2^n$

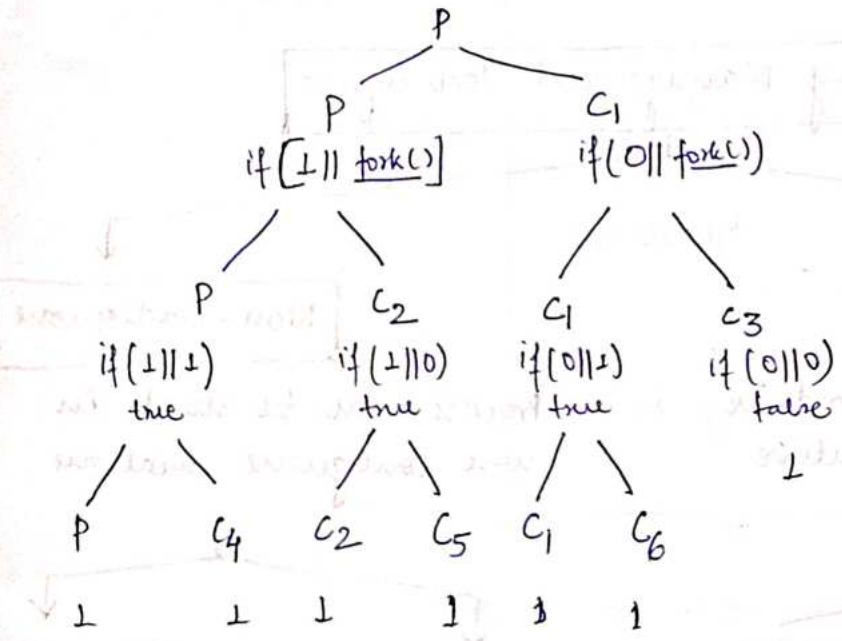
Ques O/p of the following code —

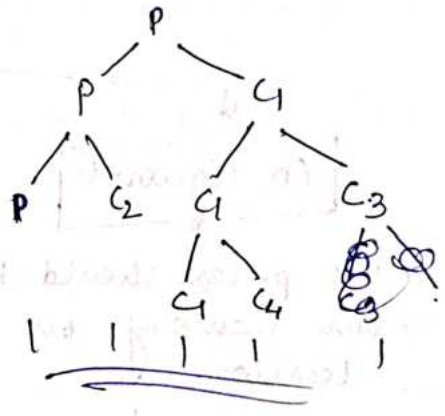
```
int main() {
    if (fork() || fork())
        fork();
    print("1");
    return 0;
}
```

(return true value; true value)

11

Some compilers do not check 2nd condition if 1st condition is true in logical OR. In that case 5 1's printed





No. of 1's printed = 7 (1111111)

L-28

Memory management

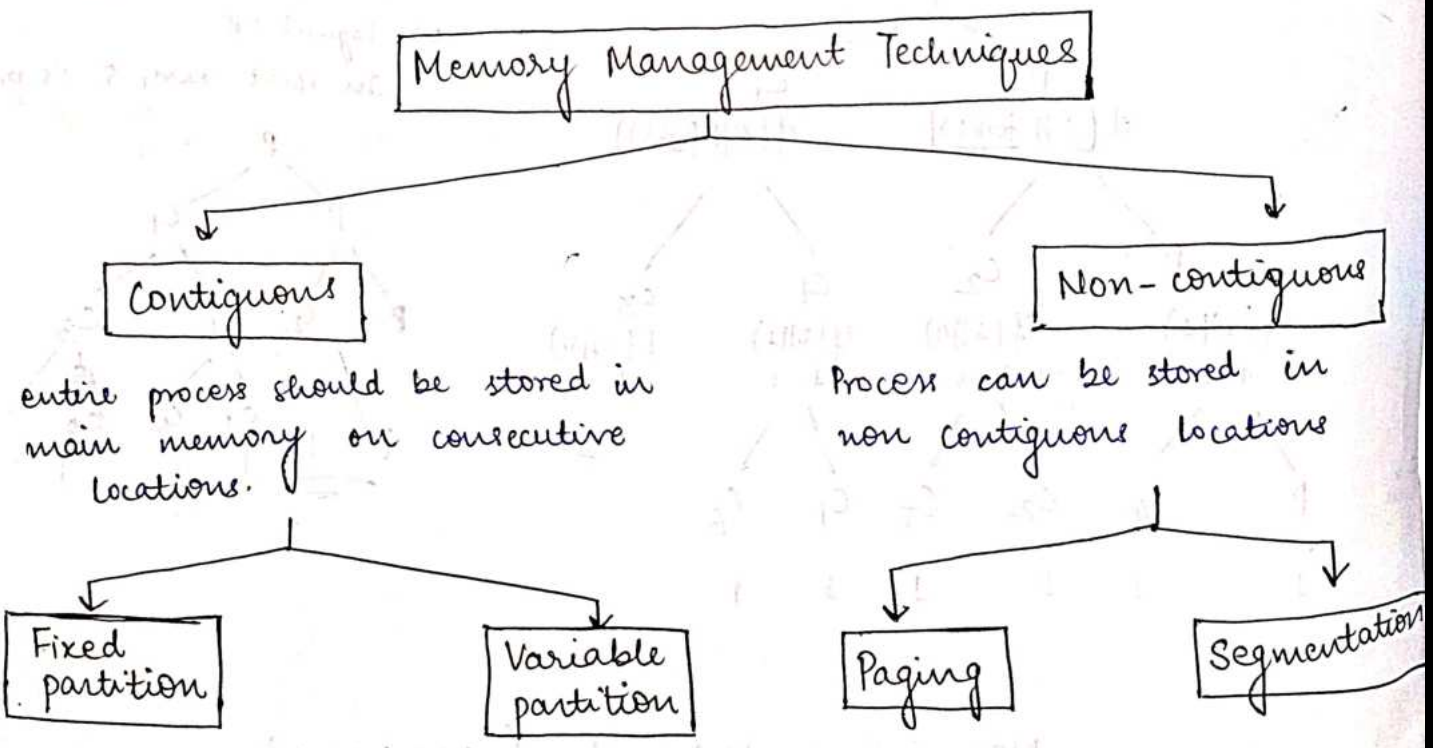
Module of OS.

Functions

- Memory allocation (e.g. new → ready state process state transition)
- Memory deallocation (removing process from M.M after completion)
- Memory protection (a process P1 should not be able to access memory alloted to P2)

Goals of memory management —

- Maximum utilization of space. → minimum memory fragmentation
- Ability to run larger programs with limited space. → using virtual memory concept.



Contiguous Memory Management

Entire process should be stored on consecutive memory locations

1. Fixed Partition Contiguous MMT

The m.m. is divided into fixed no. of partitions and each partition can be used to accomodate maximum one process.

The size of partitions can be equal or different.

Partition Allocation Policy

New process should be accomodated in which partition of main memory.

Partition Allocation Policy	Allocated partition
First fit	First partition of size greater than the process size.
Best fit	Smallest partition of size greater than process size
Worst fit	Largest partition which can accomodate process.
Next fit	First fit but search starts from last allocated partition

* Fixed partition technique suffers from internal fragmentation
  - ↳ wastage of extra space in a partition

* Max degree of multiprogramming is limited by number of partitions.

2. Variable Partition Contiguous MMT

- Main memory is divided into partitions initially.
- When a new process arrives, a new process is created of same size as process size & process is allocated that partition.
- Hence no internal fragmentation.

Variable partition technique suffers from external fragmentation

↳ if enough space is available in memory to store a process but space is not contiguous, then, wastage of space is known as external fragmentation.

Solution to external fragmentation is compaction

costly process

↳ collect all the allocated processes to one side of memory so that other side of memory can have entire free space contiguously.

Non Contiguous Memory Management

⊙ Process is scattered in memory, not allocated at one area

⊙ 2 Techniques

- Paging :- scattered in same size of memory area
- Segmentation :- scattered in variable size of memory area.

1. Paging

⊙ Process is divided into equal size of pages.

⊙ Physical memory is divided into frames (of same size as page).

⊙ Pages are scattered in frames.

Page table is used to map a page to the frame where it is stored.

No. of entries in page table = no. of pages in process

~~No~~ of Size of each entry = frame no. bits + extra bits.

Size of page table =

No of entries in P.T. * 1 entry size

= No. of pages in process * 1 entry size

= No of pages in process * (frame no. + extra bits)

★→ Page table is stored in the main memory itself in the form of pages.

When CPU requests a specific content ~~from~~ —

- ↳ Find out content is present in which page.
- ↳ Search page table & get frame no.
- ↳ Goto frame no. & extract content.

∴ 2 times ~~main~~ physical memory is accessed

- ↳ one for page table
- ↳ one for actual content.

Logical address space:-

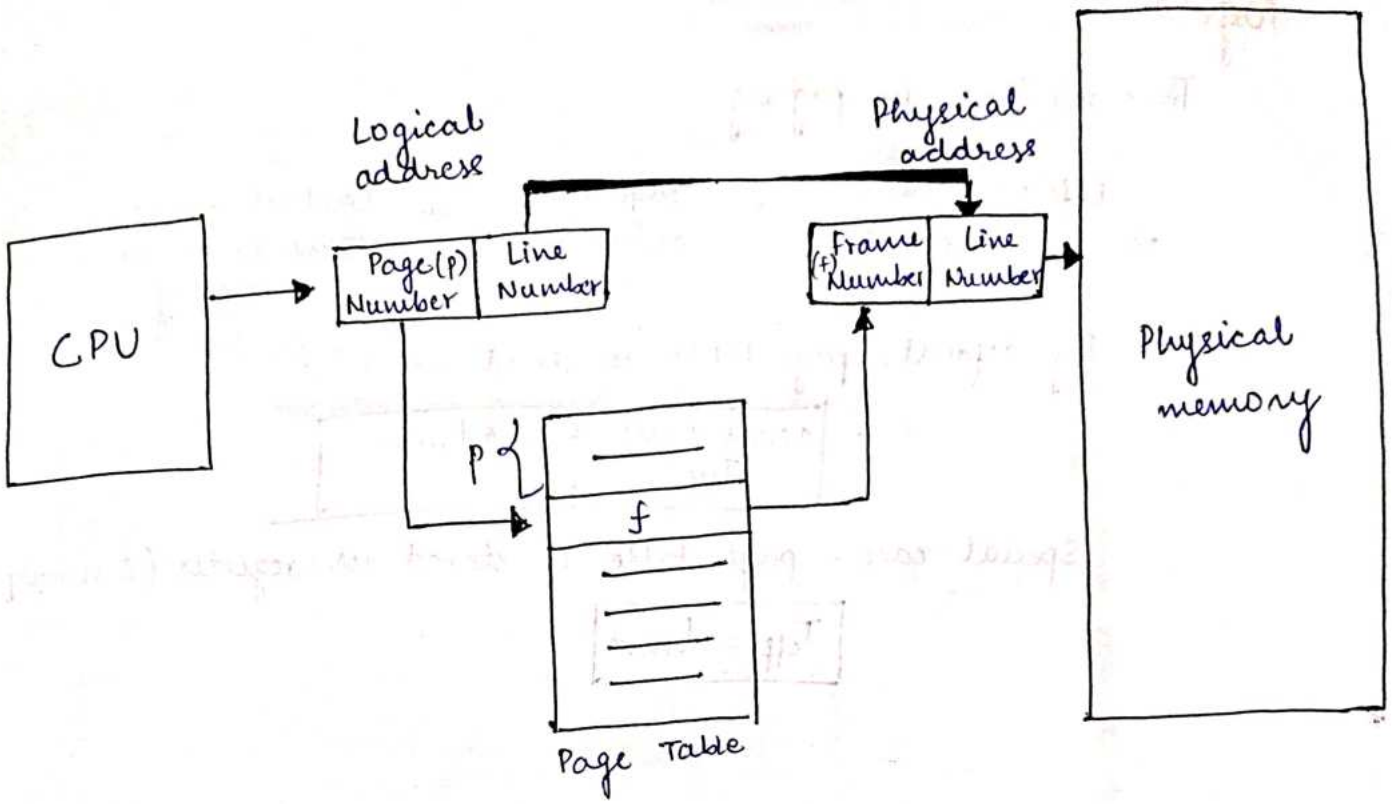
collection of logical addresses.

L.A.S. is same as process size.

Physical address space:-

collection of all physical addresses (physical memory size)

Byte no. = Line no. = Displacement = Page offset.



Page Table Base Register (PTBR)

↳ special purpose register that stores the starting address of the location/page where page table is stored in memory.

L-32

Paging

Time required in paging —

Effective main memory access time = Page table access time + Content access time from main memory

By default, page table is stored in m.m.

$\therefore$ access time $T_{eff} = 2 * t_{mm}$

Special case:- page table is stored in register (& is very small)

$T_{eff} = t_{mm}$

To improve performance of paging,
TLB is used
(similar to working of cache)

TLB

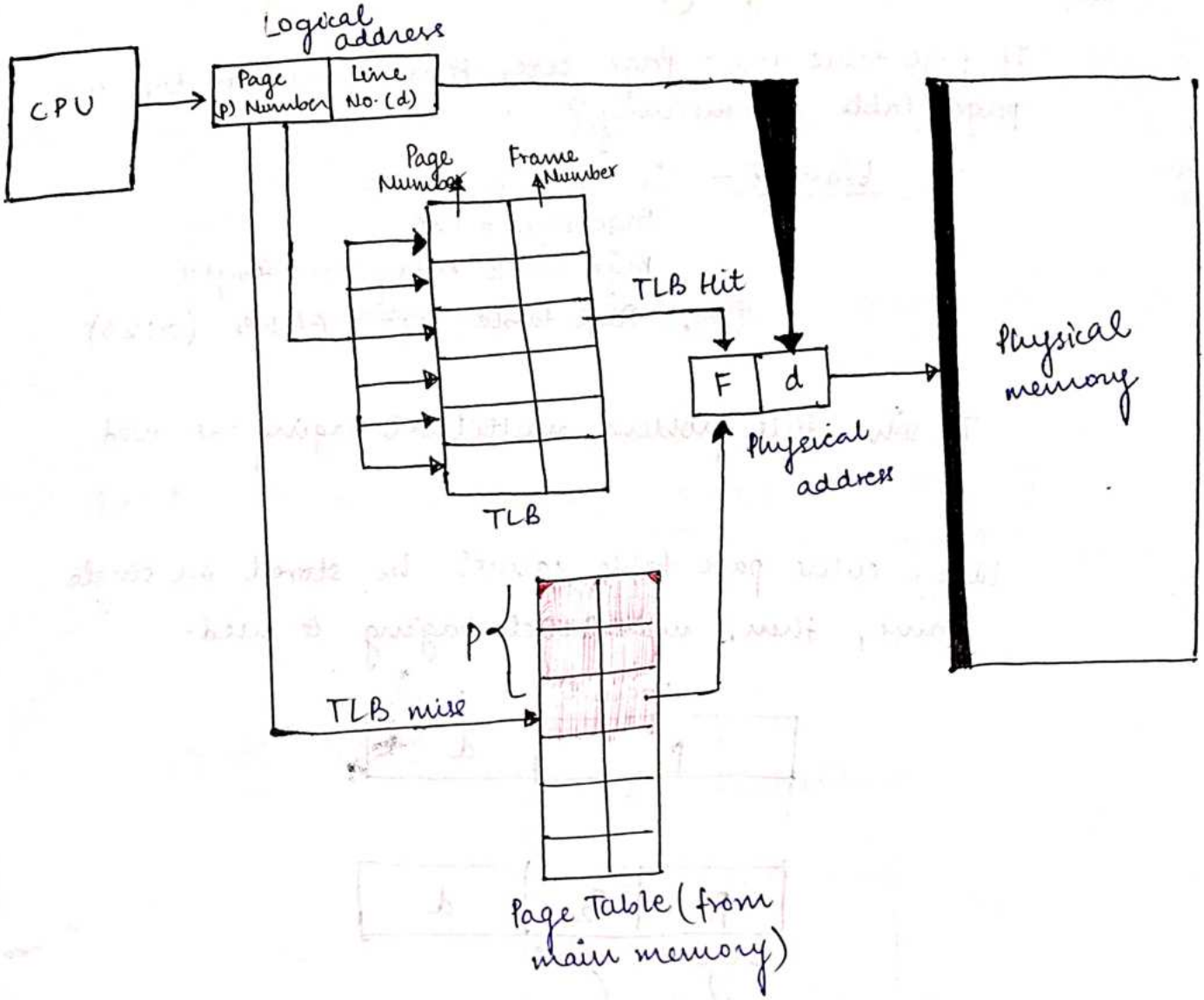
↳ Translation Lookaside Buffer.

↳ access time of TLB is very small.

↳ frequently used page tables is to are kept in TLB.

A Translation Lookaside Buffer is a memory hardware that is used to reduce the time taken to access a user memory location.

It stores few frequently used page table entries so that the CPU can get physical address without accessing the main memory.



$$T_{effective} = \underbrace{H*(t_{TLB} + t_{MM})}_{\text{TLB Hit}} + \underbrace{(1-H)(t_{TLB} + 2*t_{MM})}_{\text{TLB miss}}$$

$$= H*t_{TLB} + H*t_{MM} + t_{TLB} + 2*t_{MM} - t_{TLB}*H - 2*t_{MM}*H$$

$$= H\,t_{MM} + t_{TLB} + t_{MM} + t_{MM} - H\,t_{MM} - H\,t_{MM}$$

$$\boxed{t_{effective} = t_{TLB} + t_{MM} + (1-H)\,t_{MM}}$$

L-35

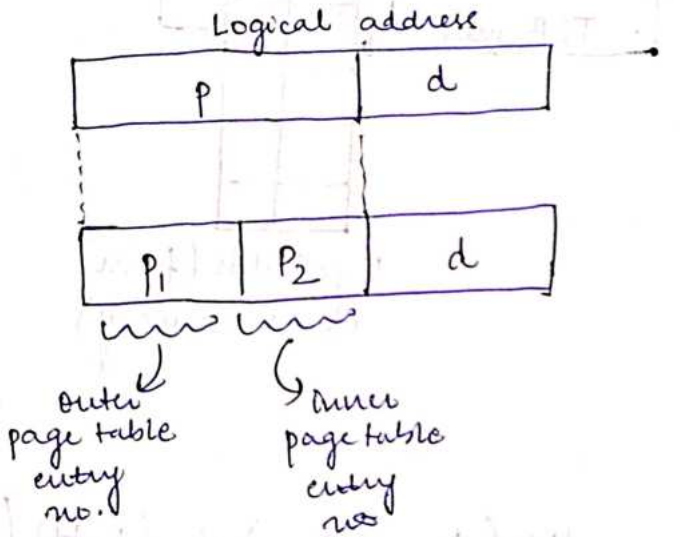
Multilevel Paging –

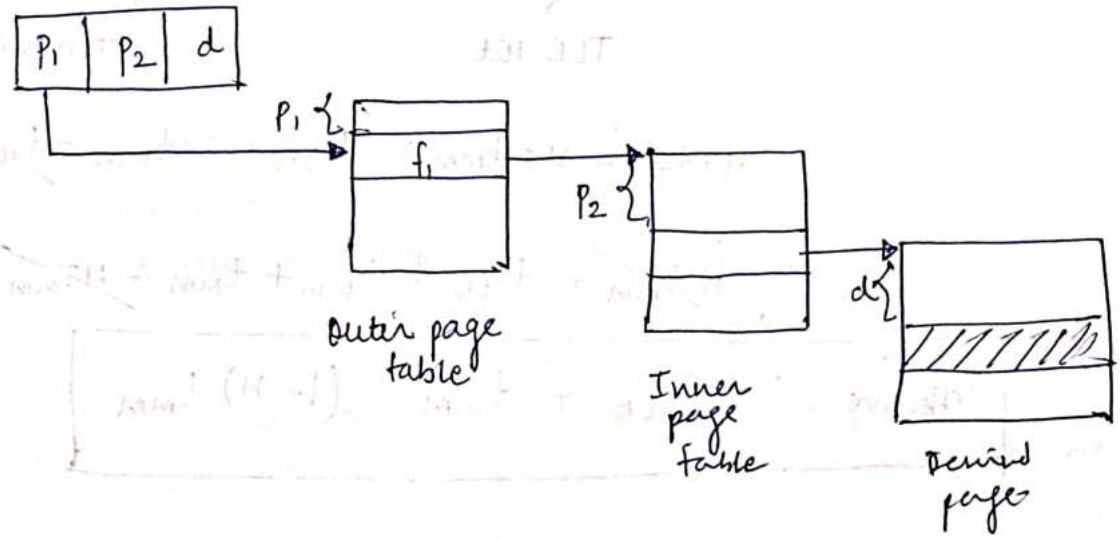
If page table size > Page size, then, how to store the page table in memory?

Example – Process size = 16MB
Page size = 1KB
Page table entry size = 4 bytes
∴ Page table size = 64kB (>1kB)

To solve this problem, multilevel paging is used.

When entire page table cannot be stored on single frame, then, multilevel paging is used.





Ques

No. of pages = 2^{15}

Page size = 2KB

Page Table entry = 4B

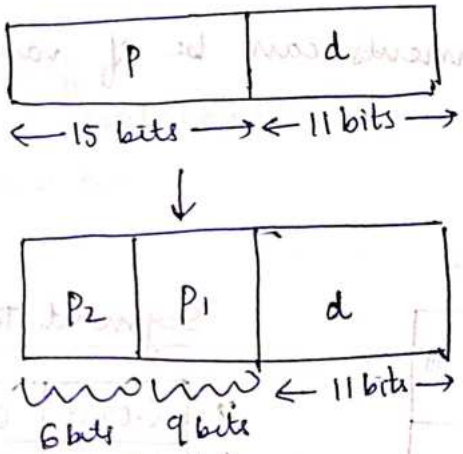
Page table size = $2^{15} * 4B = 2^{17} B = 128kB$ (> page size)

No. of page table entries in 1 page = $\frac{2KB}{4B} = \frac{2^{11} B}{2^2 B} = 2^9$

No. of pages required to store entire page table

$= \frac{128kB}{2kB} = 64$

Logical address =



↳ 9 bits because max. no. of entries in a page = 2^9

Ques

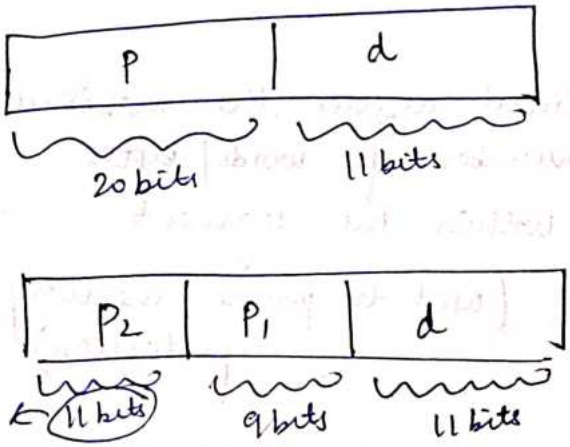
No. of pages = 2^{20}

Page size = 2KB

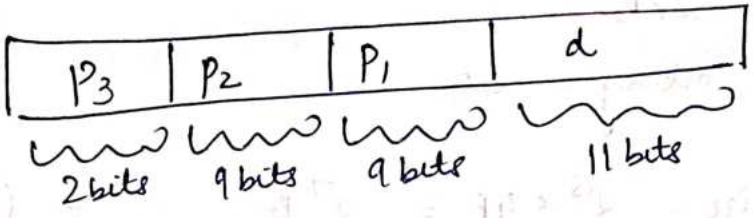
Entry size = 4B

How will the logical address look like?

No. of entries in 1 page = $\frac{2^{11}}{2^2} = 2^9$



∴ 3 level paging is done.



L-36

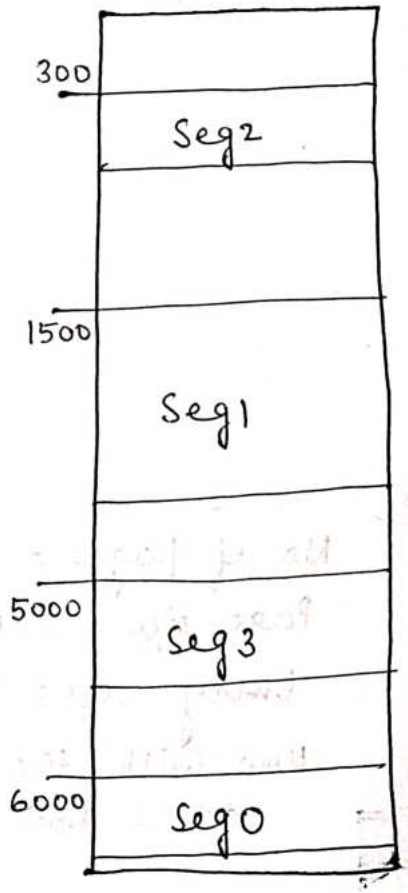
2. Segmentation –

- Divide a process into logically related partitions (segments)
- Segments are scattered in physical memory.
- Segments can be of variable sizes

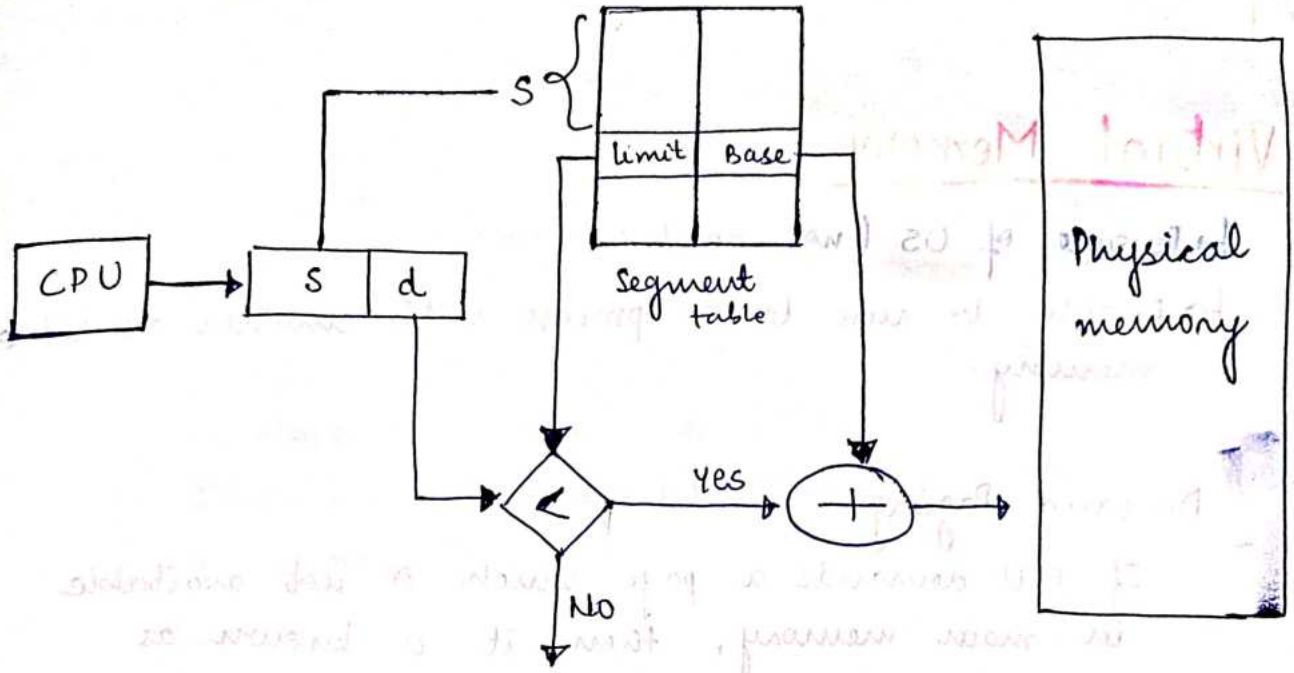
Process

Segment Table

	Base	Limit
0	6000	600
1	1500	1000
2	300	300
3	5000	450



- Limit defines the maximum number of words / bytes within the segment (used to provide memory protection)



* Maximum segment size is decided by the OS.
  ↳ used to define the no. of bits to be used in displacement/offset of logical address.

Ques

Maximum segment size = 2KB
No. of segments in process = 2^{20}
Logical address = ____ bits.

No. of bits in displacement = $\log 2^{11} = 11$
No. of bits for segment no. = 20
Logical address = 20 + 11 = 31 bits

L-37

Virtual Memory

↳ Feature of OS (not as hardware)

↳ Enables to run larger process with smaller available memory.

Demand Paging –

If CPU demands a page which is not available in main memory, then it is known as page fault.

Page fault service.
- In case of page fault, OS brings faulted page from secondary memory to m.m. by replacing a page (if req.) and updates page table

DMA might be required

CPU has an internal interrupt known as page fault.

service of internal interrupt is done by OS program
↳ system call

Demand Paging

Pure demand paging

all the pages are kept in ~~main~~ secondary memory. Pages are brought in main memory as & when required.

Demand paging

When a process needs to execute, some of the virtual pages are kept in main memory.

L-38

Page Replacement Policies

1. First In First Out (FIFO)
2. Optimal Policy
3. Least Recently Used (LRU)
4. Least Frequently Used (LFU)
5. Most Frequently Used (MFU)
6. Last In First Out (LIFO)
7. Second Chance

} most imp.

~~Cout~~ Counting Algorithms

Counting Algorithms look at the number of occurances of a particular page and use this as the criterion for replacement.

- Least Frequently Used (LFU)
- Most Frequently Used (MFU)

} replace page which has been used min/max. no. of times.

L-40

Frame Allocation

2 questions
1. What is the minimum number of frames that a process needs?
2. Is page replacement global or local?

P1 = 8 KB
P2 = 4 KB
No. of frames = 6

FRAME ALLOCATION

EQUAL ALLOCATION

equal no. of frames are given to each process irrespective of size.

Frames for P1 = 3
" " P2 = 3

PROPORTIONAL ALLOATION

frames are given to processes in ratio of their sizes.

Frames for P1 = $\frac{8}{8+4} \times 6 = \frac{8}{12} \times 6 = 4$

Frames for P2 = $\frac{4}{12} \times 6 = 2$

Frame allocation

Local Allocation

- page being replaced be in a frame belonging to the same process.
- Number of frames belonging to the process will not change.
- allows processes to control their own page fault rates.

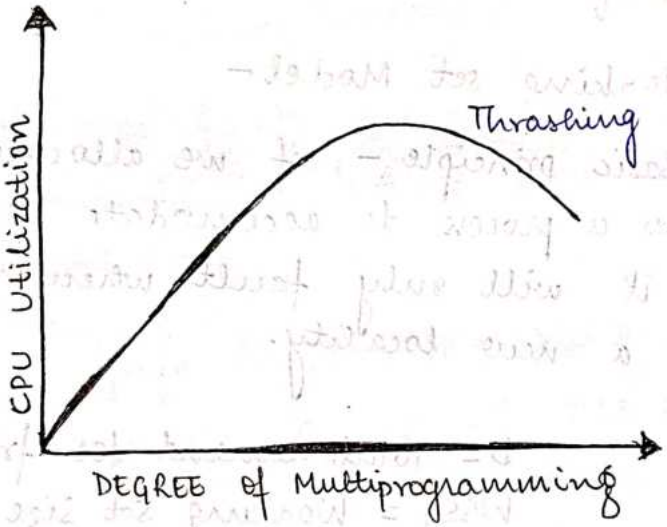
Global Allocation

- process can replace a page from a set that includes all the frames allocated to user processes.
- High priority processes can increase their allocation at the expense of lower priority processes.
- better throughput more efficient use of frames.

Thrashing

Thrashing is the condition or situation when the process is spending a major portion of its time servicing the page faults, but the actual processing done is negligible.

The situation can continue indefinetly until the user closes some running applications or the active processes free up additional virtual memory resources.



Causes of Thrashing –

- High Degree of Multiprogramming
- Lack of frames
- Page Replacement Policy

Handling Thrashing —

Locality Model —

A locality is a set of pages that are actively used together. The locality model states that as the process executes, it moves from one locality to another.

2 Techniques to Handle —

↳ Working set Model —

Basic principle — if we allocate enough frames to a process to accomodate its current locality, it will only fault whenever it moves to a new locality.

D = Total demand for frames.

WSS_i = Working Set Size for process i

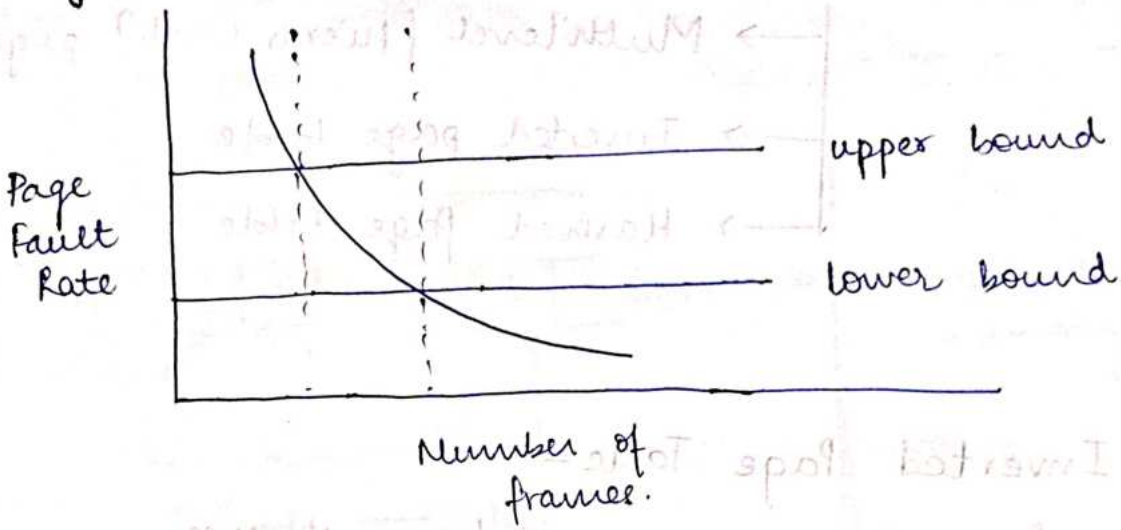
m = No. of frames available in memor

$$D = \Sigma WSS_i$$

If D > m, then, thrashing will occur as some processes would not get enough frames.

if D <= m, there would be no thrashing.

↳ Page Fault Frequency



if page fault rate > upper bound, increase number of frames allocated to that process.

If page fault rate < lower bound, decrease number of frames allocated to that process.

In other words, the graphical state of the system should be kept limited to the rectangular region formed in the diagram.

Types of page table

- Multilevel (Hierarchial) page ~~tage~~ table
- Inverted page table
- Hashed page table.

- **Inverted Page Table –**

One entry of each page frame of M.M.

Max. no. of entries = No. of frames in main memory

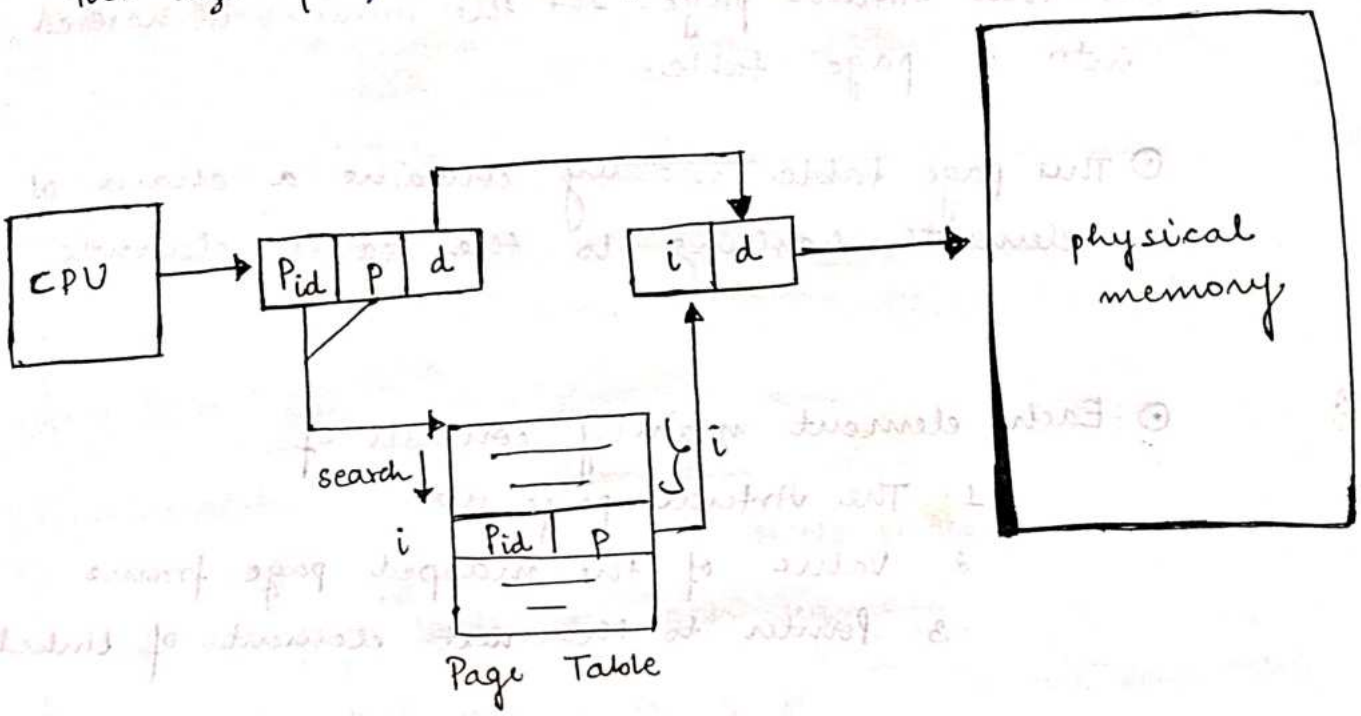
* Every process has its own page table but, there is only one inverted page table.

∴ process ID is also ~~used to~~ stored to determine ~~a~~ page stored at a particular location belongs to which process.

Each entry in the page table contains the following field –

- Page Number
- Process ID
- Control bits [Modified bit, Replacement bit, etc.]
- Chained pointer [used if entries are needed to be connected as in a chain]

* Size of inverted page table is much smaller than the size of general page table.



Advantages of inverted page table –

↳ Reduced memory space.

Disadvantages of inverted page table –

↳ Longer lookup time

↳ Difficult shared memory implementation.

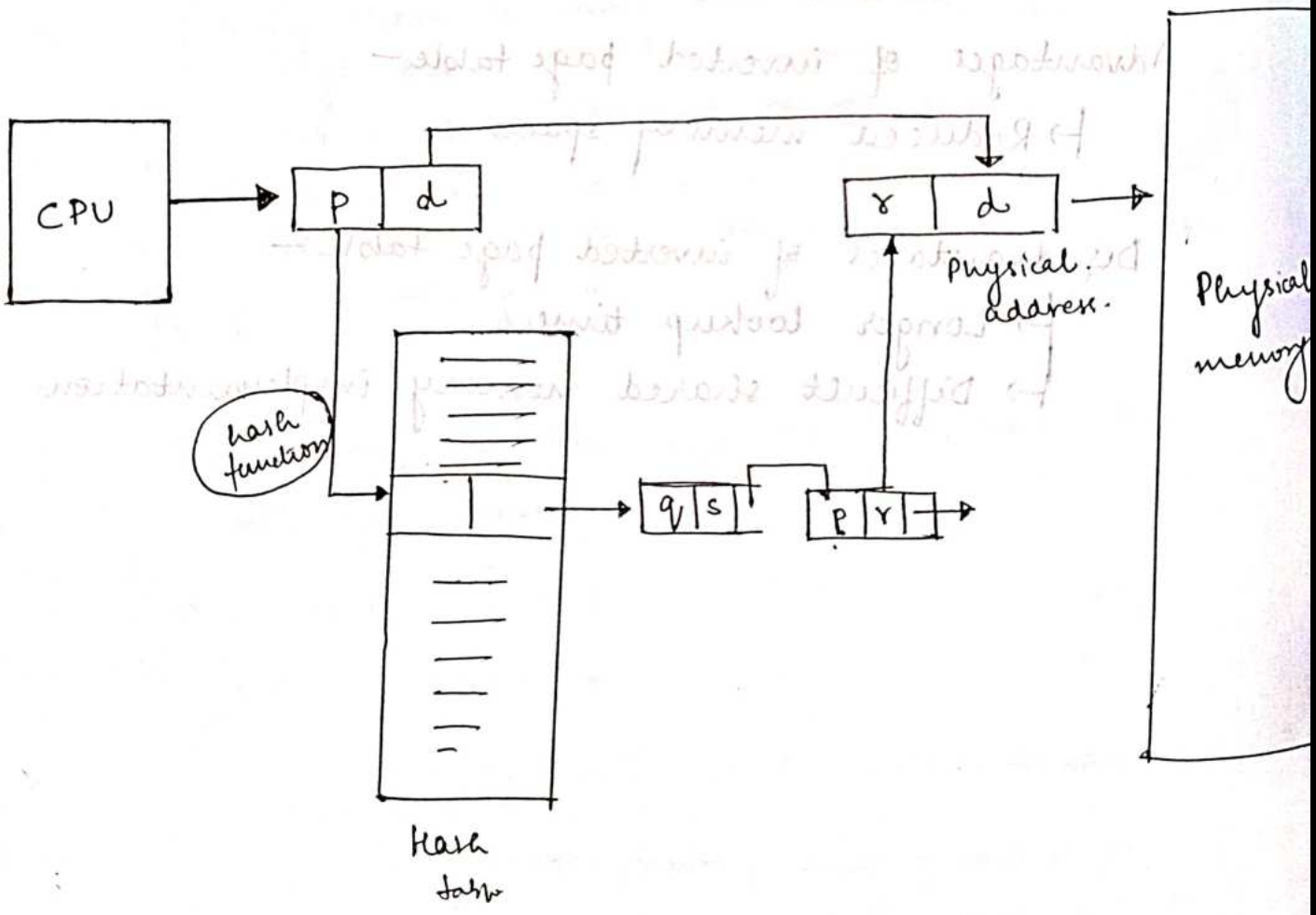
• Hashed Page Table

⊙ In this virtual page, the number is hashed into a page table.

⊙ This page table mainly contains a chain of elements hashing to the same elements.

⊙ Each element mainly consists of

1. The virtual page no.
2. Value of the mapped page frame
3. Pointer to the next element of linked list.



Advantages —

↦ Lookup time is reduced

↦ size of page table is reduced

L-42

Segmentation Paging

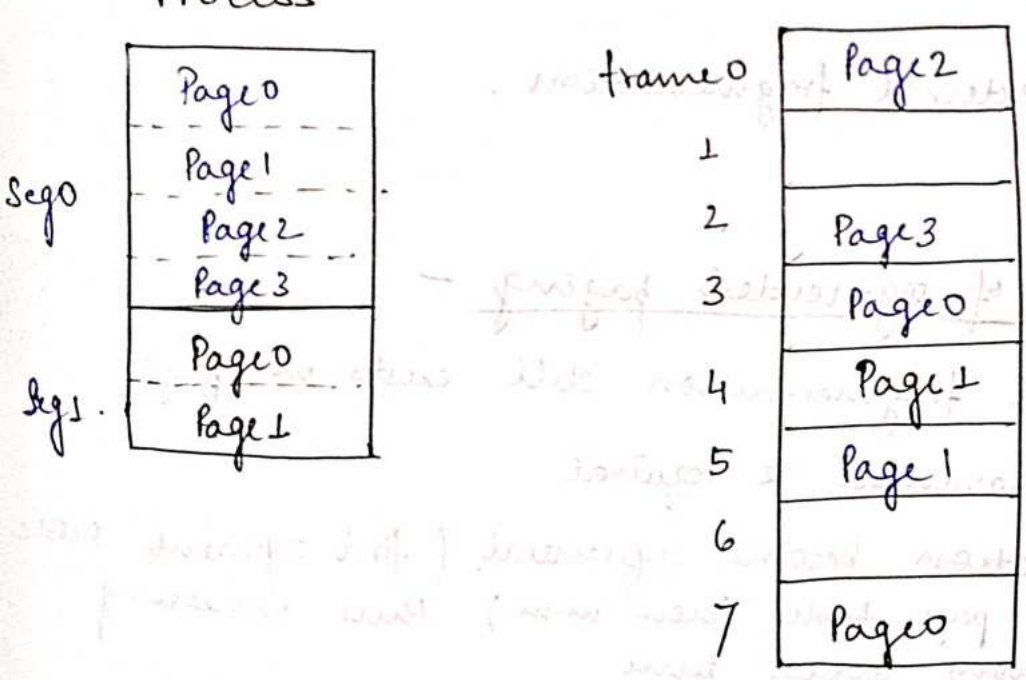
↦ combination of segmentation and paging.

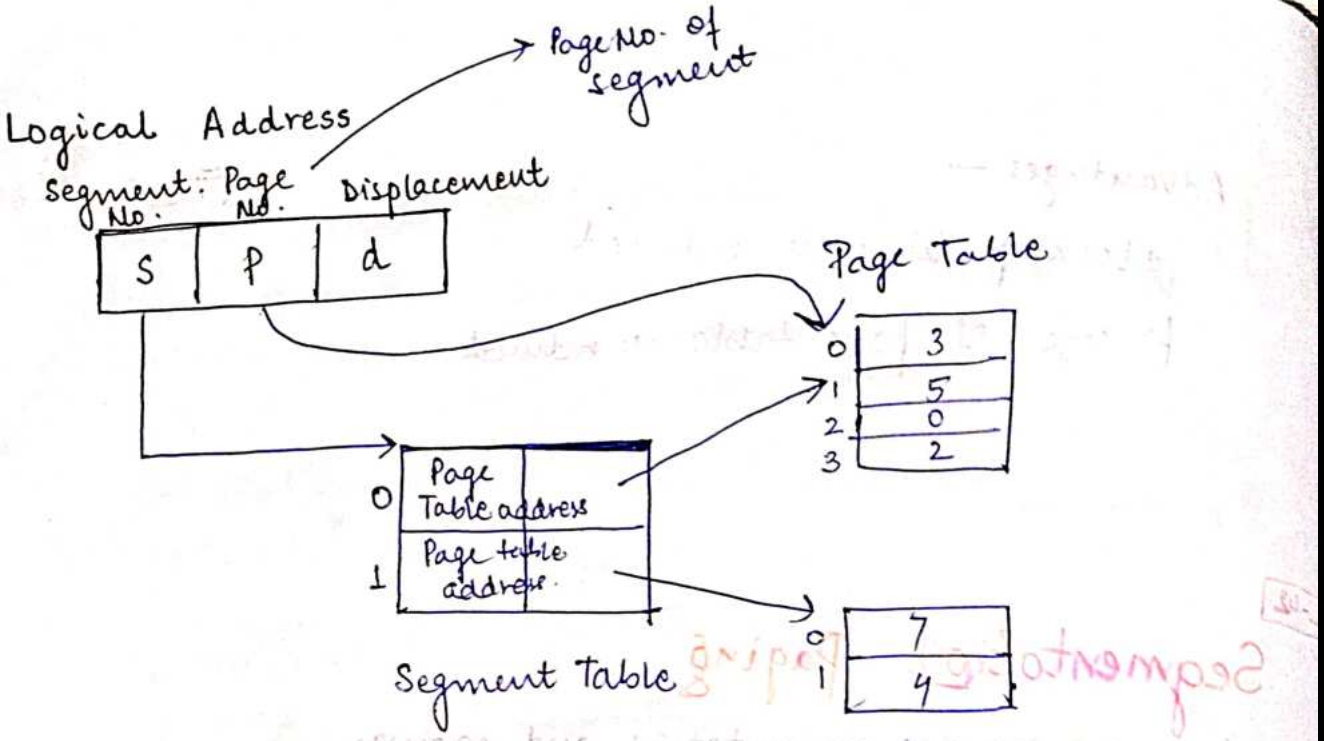
↦ divide each segment into multiple pages

↦ keep one page table for each segment.

↦ Segment table ~~entries~~ entry points to the page table of the segment.

Main memory has frames of size equal to page sizes and each page can be kept on any frame.





Advantages of segmented paging –

↦ Page Table size is reduced as pages are present only for data of segments (currently req. in m.m.) hence reducing the memory requirements.

↦ Gives programmers view along with the advantages of paging.

↦ Reduces external fragmentation.

Disadvantages of segmented paging –

↦ Internal fragmentation still exists in pages.

↦ Extra hardware is required.

↦ Translation becomes sequential (first segment table then page table then m.m.) thus increasing memory access time.

OPTIMAL PAGE SIZE $= \sqrt{2 * LAS * E}$

LAS = Logical Address space / Process size.

E = Page Table entry size

Multilevel paging & TLB

Effective memory access time $= H * (t_{TLB} + t_{mm}) + (1-H)(t_{TLB} + [illegible] + (n+1)t_{mm})$

File

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

File attributes.

- Name
- Extention
- Size
- Date
- Author
- Created, Modified, Accessed
- Attributes: Read only / Hidden
- Default Program
- Security Details.

Types of File system (provides beautiful view of files)

- FAT32 (File Allocation Table) } windows OS
- NTFS (New Technology File System) } windows OS
- HFS (Hierarchial File System) } macOS
- Ext2 | Ext3 | Ext4 } Linux
- Swap

FILE DIRECTORY STRUCTURE

1) Single Level directory —

- ⊙ simplest directory structure
- ⊙ all the files are contained in the same directory which makes it easy to support & understand.

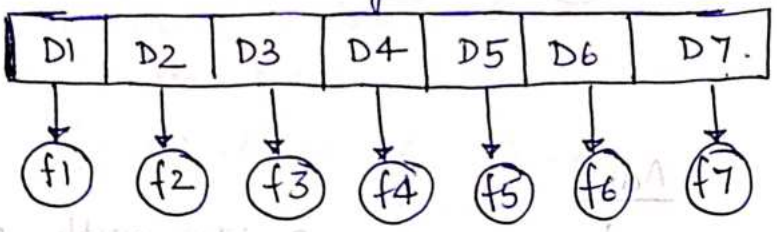
⟶ all the files should be of unique name.

Adv:-

- ↦ easy to implement
- ↦ searching is faster

Disadv :-

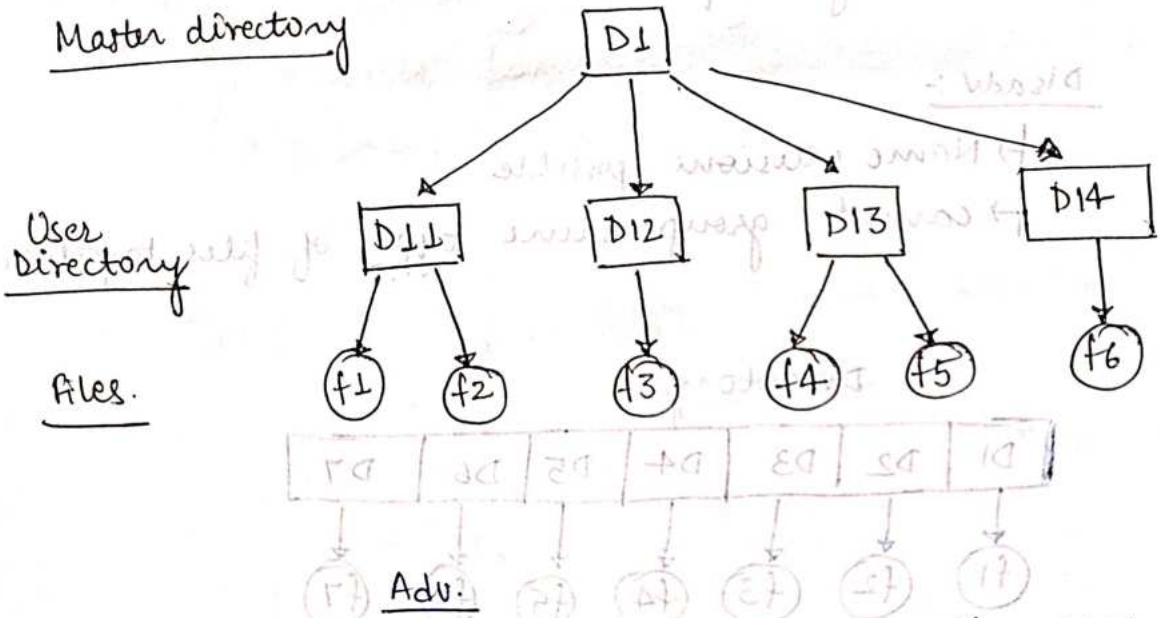
- ↦ Name collisions possible
- ↦ cannot group same type of files together.



2) Two Level Directory

- ⊙ Separate directory is created for each user
- ⊙ In a 2 level directory structure, each user has their own User File Directory (UFD).

  UFDs have similar structures but each list only files of a single user.
- ⊙ System's Master Directory (SMD) is searched whenever a new user ID is created.



Adv.

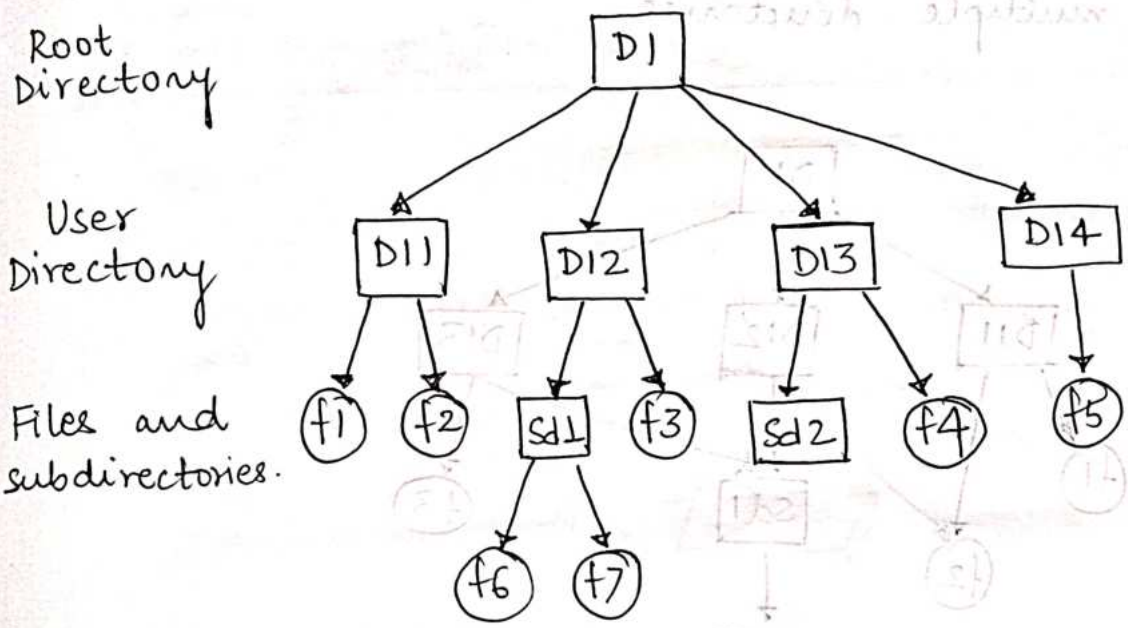
- ↦ more than 2 files with same name
- ↦ multiple users can work together.
- ↦ searching of the files is very easy.

Disadv.

- ↦ user cannot share the file with other users.
- ↦ users cannot create subdirectories.

3) Tree structure / Hierarchial structure

users can create files as well as subdirectories.



Adv:

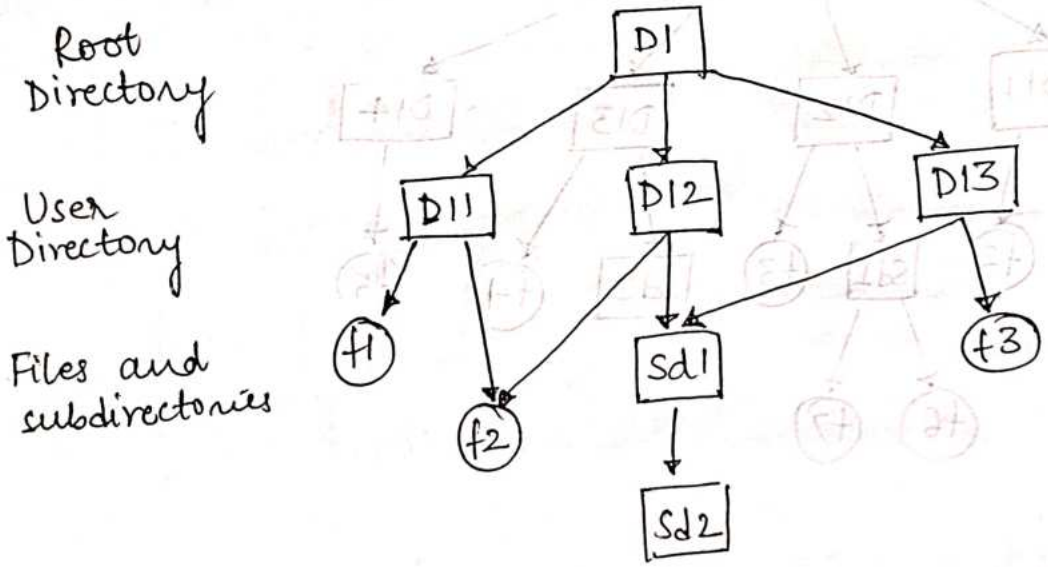
- ↦ subdirectories can be created inside user directories.
- ↦ more scalable than other 2 directory structures.
- ↦ searching is easier.

Disadv:

- ↦ file sharing b/w users not possible.
- ↦ if no. of subdirectories increases, searching can become complicated.
- ↦ users cannot modify root directory data

4) Acyclic Graph Structure

file in one directory can be accessed ~~through the directory it was present in~~ from multiple directories.



Adv.

↳ sharing of files and subdirectories is allowed between multiple users.

Disadv.

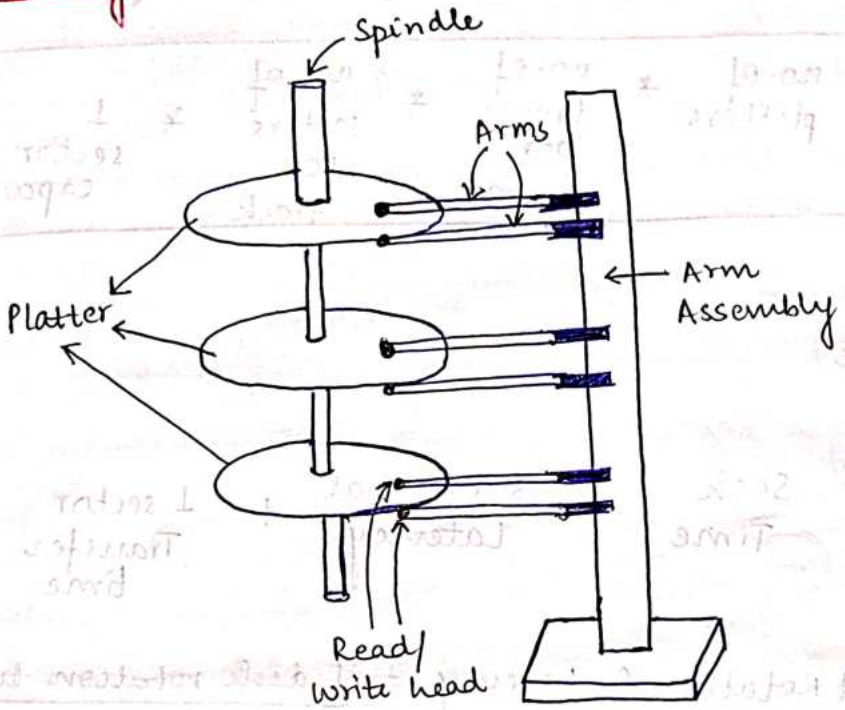
↳ difficult to implement due to complex structure.

↳ user must be very cautious to edit or delete a file as file is accessed by multiple users.

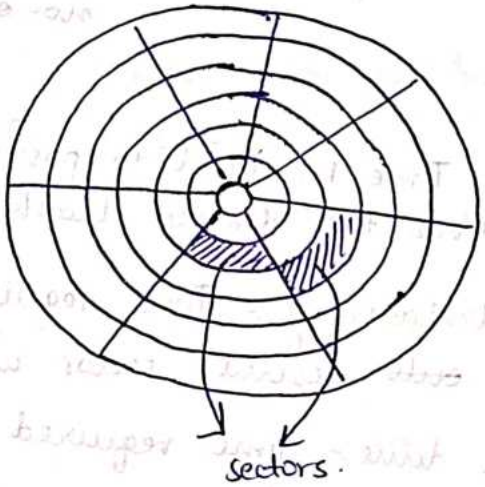
↳ if we need to delete a file, we need to delete all the references of the file inorder to delete it permanently.

L-45

Magnetic Disk



Top view of magnetic disk



Number of sectors (n_s)

2 * no. of platters * no. of tracks per surface * no. of sectors per track.

Track :- coincentric circles

Sector :- subdivision of track
smallest unit of unit of disk which can be read/ written at once.

Sector is addresable unit of disk.

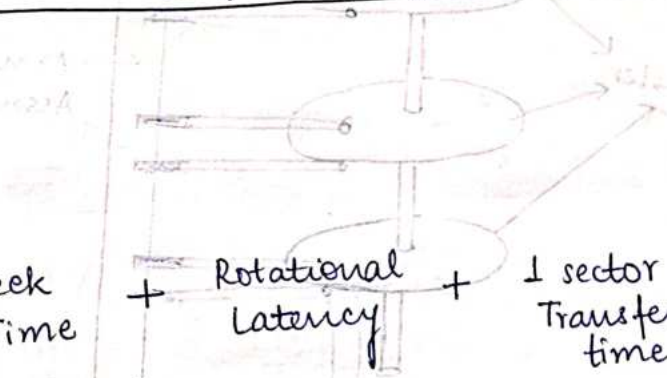
address size = $\log_2(n_s)$ bits.

Collection of consequtive sectors is called <u>cluster</u>.

Sector Capacity

Each sector has constant storage capacity.

Sector capacity	=	2 *	no. of platters	*	no. of tracks per surface	*	no. of sectors per track	*	1 sector capacity



Disk Access Time

Disk Access Time = Seek Time + Rotational Latency + 1 sector Transfer time.

Average ~~lot~~ Rotational Latency = $\frac{1 \text{ disk rotation time}}{2}$

1 sector transfer time = $\frac{1 \text{ disk rotation time}}{\text{no. of sectors per track}}$

Seek Time:- Time required to position the arm over the desired track.

Rotational Latency :- Time required to rotate the ~~tttb~~ desired sector under read/write head.

Transfer time :- Time required to read/write 1 sector.

DISK PARTITIONING / FORMATTING

Low level partitioning (physical)

creating tracks and sectors

done by manufacturers.

High level partitioning (logical)

creating C:, E: ... drives.

done by user using OS. (logical formatting)

2 types of partitions

Primary

used to store
- ↳ OS
- ↳ user file

Extended

used to store
- ↳ user file.

Only one OS can be installed in one partition

Disk Blocks

disk block is the logical representation of the smallest unit of disk.

1 block = 1 to 2 disk sectors

A file smaller than block size should occupy entire block. Hence, there will be internal fragmentation.

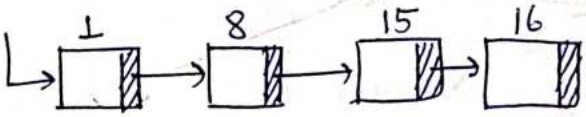
Free space Management –

- File system manager manages the free blocks within the disk.

2 ways

Free list

linked list of all the free blocks is maintained.



no searching needs to be done to find free block.

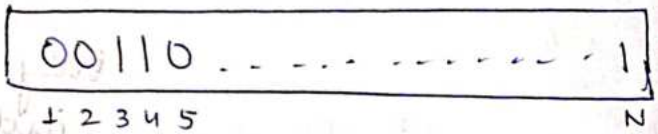
⊙ size is variable.

BitMap Method

n blocks in disk } ⇒ n bit information

ith bit represents whether ith block is free or not.

if bit i = 0 , it is free
i = 1 , it is occupied.



searching needs to be done to find first free block.

⊙ bitmap size is constant.

File Allocation Methods

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

1. Contiguous Allocation

File should be stored on contiguous blocks.

OS maintains File Allocation Table.

FILE NAME	Start block no	No. of blocks
abc.doc	4	3

i.e. 4,5,6 blocks are used to store the file.

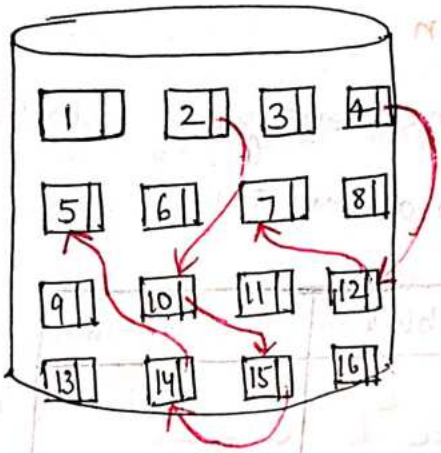
if file size increases (by adding more contents) then, contiguous blocks may not be available. Thus this file allocation strategy is not flexible.

Performance:

- Fragmentation :- Internal , External
- Increase in file size : Inflexible
- Type of access : sequential ; Random/direct

2. Linked Allocation

few bytes of each block are used to store the link of the next block where the file content is present.



File allocation Table.

File Name	start block no.	Last block
abc.doc	4	7
xyz.pdf	2	5

Performance:

- Fragmentation : Internal
- Increase in file size : Flexible
- Type of access : sequential.
  - big problem. Imagine watching 1hr of a movie from starting just to watch a 5 min climax scene

limitation of File Allocation Table (FAT)

more the no. of blocks, more will be the size of FAT.

Since FAT needs to be cached, therefore it is impossible to have as many space in cache.

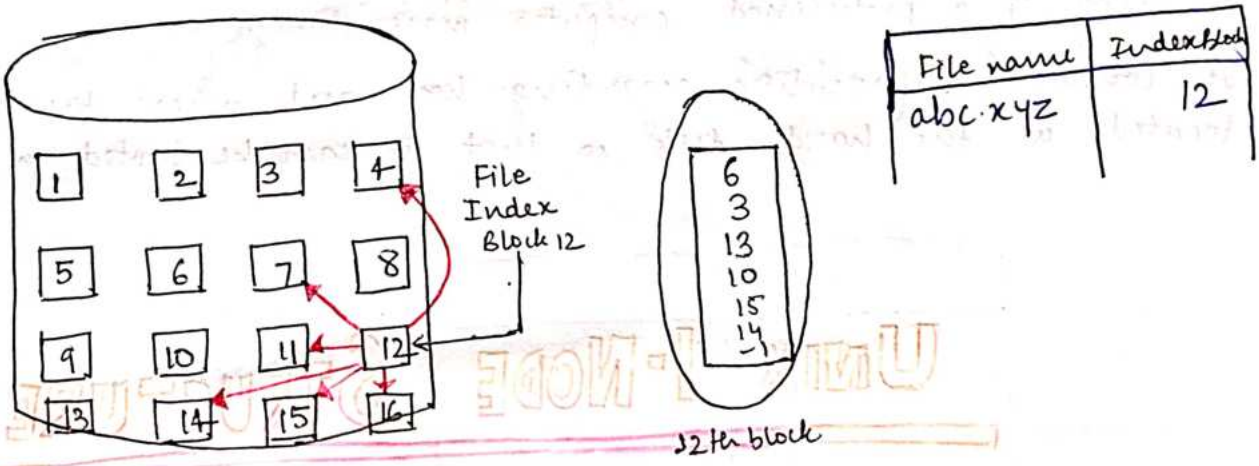
∴ Indexed file allocation method was proposed

3. Indexed Allocation

This scheme stores all the disk pointers in one of the blocks called indexed block.

each file needs index block

Indexed block doesn't hold file data, but it holds pointers to all the disk blocks containing data.



Performance:

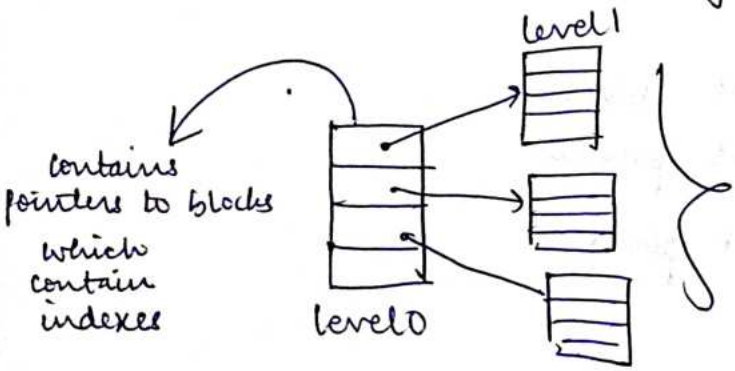
Fragmentation :- Internal

Increase in file size :- Flexible

Type of access :- Random / direct, Sequential

★ If the file is very large, then, all the block pointers may not be stored in single block.

∴ Multilevel indexing is done.



Ques Disk Block address = 16 bits
Disk Block size = 1KB
Index Block = 1KB
Maximum file size?

No. of entries in a block = $\frac{1\,kB}{2B} = 2^9 = 512$

∴ Max. file size = 512 * 1kB = 512 kB

[single level indexing]

Master Boot Record.

A Master Boot Record is a special type of boot sector at the very beginning of a partitioned computer mass storage device.
It contains information regarding how and where the OS is located in the hard disk so that it can be booted in the RAM.

UNIX I-NODE STRUCTURE

The i-node (index node) is a data structure in UNIX-style file system that describes a file system object such as a file or a directory.

- Each inode stores the attributes and disk block locations of the object's data.
- The number of i-node limits the total number of files/directories that can be stored in the file system.

I node contains following information —

- Administrative information (permissions, timestamps etc)
- 12 direct blocks (~~address~~ pointer which point to the first 12 blocks of files.
- 1 single indirect pointer
- 1 double indirect pointer
- 1 triple indirect pointer

Disk scheduling=

1. FCFS (First Come First Serve)

Adv
- Every request gets fair chance
- No indefinite postponement.

Disadv
- Does not try to optimize seek time.
- May not provide the best possible service.

2. SSTF (Shortest Seek Time First)

Adv
- Average response time decreases
- Throughput increases

Disadv
- Overhead to calculate seek time in advance.
- can cause starvation

3. Scan (Elevator)

[The arm should move towards larger value]

!! Last तक जाएगा !!

Adv
- High Throughput
- Low variance of response time
- Average response time good.

Disadv — while returning, no request served

4. C-Scan

similar to scan but after reaching corner, reset to 0 and then service the request. Request servicing is done in 1 direction only.

Adv — Provides uniform waiting time compared to Scan.

5. Look

arm doesn't move to the corner.
changes direction whenever reaches the last request to be serviced.

6. C-Look

request servicing is done in 1 direction only.

Magnetic disk questions

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

- Seek Time → will be given in question
- Rotational Latency → $\frac{1 \text{ rotation time}}{2}$
- Transfer Time → $\frac{1 \text{ rotation time}}{\text{no. of sectors per track.}}$

* Each sector is given unique address

∴ No. of address bits = $\lceil \log_2(\text{No. of sectors}) \rceil$

Address

$< c, h, s >$

- c → cylinder no.
- h → surface no.
- s → sector no.

$c = \lfloor \text{sector no.} / n_c \rfloor$

$h = \lfloor (\text{sector no.} \,\%\, n_c) / n_t \rfloor$

$s = \lfloor (\text{sector no.} \,\%\, n_c) \,\%\, n_t \rfloor$

Sector no = $c * n_c + h * n_t + s$

- n_c = no. of sectors per cylinder
- n_t = no of sectors per track.

$n_c = n_t * \text{No. of surfaces}$