Date : / /
Page No.:

INTRO TO OS&ESD:

Unit 1 : General purpose OS :

Unit 2 : Real time OS:

Unit 3 : Embeded System Disign:

UNIT 1: General purpose Os : means os used for laptops , like MacOS, Windows etc.

Chapter 1 : Introduction:

→ What is an Operating System?

A special piece of software that Abstracts and Arbitrates the use of computer system is called as Operating system.

Abstract^n means makes computer resource /hardware look to simpler for the user /user applicat^n.

Arbitrat^n means makes computer resources manageable.

→ Operating System:

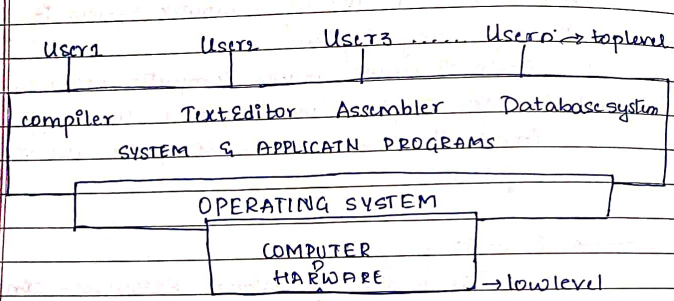
* Directs the operational Resources- control use of CPU, memory & peripherals.
* Enforces working policies - fair access of resources Limits to resource usange.

Date : / /
Page No.:

* Mitigates the complexity of difficult tasks.
* Abstracts hardware details.

☺ Operating system.....
"A program that acts as an intermediary between a user of a computer & the computer hardware."

* Operating System goals:
- Execute user programs and make solving user problems easier.
- Make the computer system convinient to use.
- Use the computer hardware in an efficient manner.



(intermediate)

↳ OS is the mediator between Sys & Applicatn programs (Compiler text Editor etc) and computer hardware.

Date : / /
Page No.:

→ User perspective OS defn:
Users want convenience, ease of use & good performance, don't care about resource utilizatn.

→ System perspective OS:
* OS is a resource allocator
→ manages all resources
→ Decides bt conflicting requests for efficient & fair resource use
* OS is a control program
→ controls execut^n of programs to prevent errors & improper use of the computer.

* The one program running @ all times on the computer is the "Kernel".

Date : / /
Page No.:

: Abstraction and Arbitration: (Resource Management)

* Why OS are needed?

→ OS is required for

- Hardware Abstractn : Turns hardware into something that applicatns can use
- Resource Management : Manage system's resources

Example : OS ABSTRACTION :

```
#include<stdio.h>
int main(){
    char str[] = "Hello world\n";
    printf("%s", str);
}
```

→ Str Hello world should be printed on monitor

processor memory —helloworld→ processor —Hello world + wordinates, color, depth, etc→ Graphics card → monitor.

→ String basically we write into memory (RAM) From RAM it has to be displayed onto monitor.

→ Before it gets displayed on monitor it has to undergo some stages

Date : / /
Page No.:

→ First strng has to be copied into processor. This happens by executing some instructns. (internal registers of)

→ From processor it has to copied into vedio buffer of graphics card. While copying along with string some attributes related to string also copied like color, x,y coordinates, depth etc

→ Graphics card in turn read from vedio buffer and displayed onto the monitor.

App

printf("%s", str);

sys call (write to stdout)

Operating System (OS) | device driver → Graphics Card → Monitor

→ Os provides easy way to program apps

→ " " Reusable functionality (Ex: printf written once we can use n times).

→ Os provides portable (If hardware is changed without any change we can use same program).

Date : / /
Page No.:

(Arbitration)

OS AS A RESOURCE MANAGER:

→ OS must manage CPU, memory, network, disk etc
→ Resource managment → allows multiple apps to share resource
  → protects apps from each other
  → Improves performance by efficient utilizatn of resources.

OS Types:

→ Applicatn Specific
  – Embedded OS
    • eg. Contiki OS, for extremely memory constraint environments
  — Mobile OS
    • Android, iOS, Ubuntu Touch, Windows Touch
  — RTOS
    • QNX, VxWorks, RTLinux
  — For Servers
    • Redhat, Ubuntu, Windows Server
  — Desktops
    • Mac OS, Windows, Ubuntu

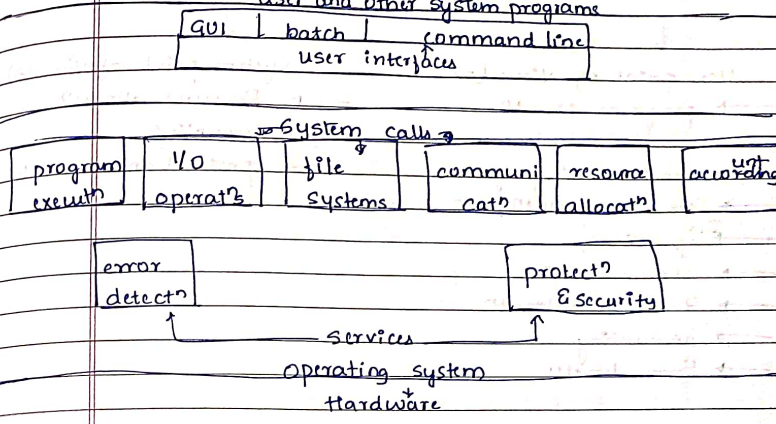
OS

Date : / /
Page No.:

* Services and Structures:

OS have
→ user interface
→ system calls
→ Services

SERVICES:



→ 3 types of user interfaces
① GUI : Graphical User Interface : usually find in Windows Environment
② batch Interface : Here a file will written
③ Command line interface : usually find in linux environment

Date : / /
Page No.:

→ Services provided by OS can be
In view of users &
In view of system point of View

Services

User point of View (5)
→ program execut^n
→ allow to execute, run, load terminate the program
→ File manupulat^n, I/O operations,
→ communicat^n
→ ~~Resource allocat^n~~
→ ~~accounting~~
→ Error detect^n

System point of View (3)
→ Resource allocat^n
→ accounting
→ protect^n and Security

* Services are implemented using System calls

* System calls: are programming interface to the services provided by the OS.
System calls are accessed by users by using API (Applicat^n programming interfaces).

Date : / /
Page No.:

Ex: a Win32 API for Windows
b POSIX API for POSIX - based systems (all versions of UNIX, Linux, & Mac OS X)
c. Java API for Java Virtual Machine (JVM)

* System call sequence to copy the contents of one file to another file.

source file → destinat^n file

example Sys call Sequence
Acquire input file name
Write prompt to screen
Accept i/p
Acquire o/p file name
Write prompto to screen
Accept i/p
Open the i/p file
if file doesn't exist, abort
create output file
if file exist, abort
loop
Read from i/p file
Write to o/p "
until read fails
close o/p file
write complet^n message to screen
Terminate normally

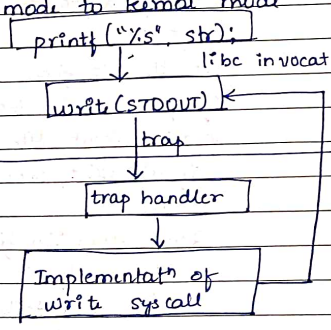
Date : / /
Page No.:

→ *. Operating Modes of of system / computer system

2 modes
1. User Mode
2. Kernel Mode

1 User Mode : User related applicatns are run in user mode
It is non-privilaged mode

2 Kernel mode : OS related applicatns are executed
It has got more privilages than user mode
Whenever system calls have to be executed system has to go for kernel mode
∴ Always there will continuous switching from user mode to kernel mode

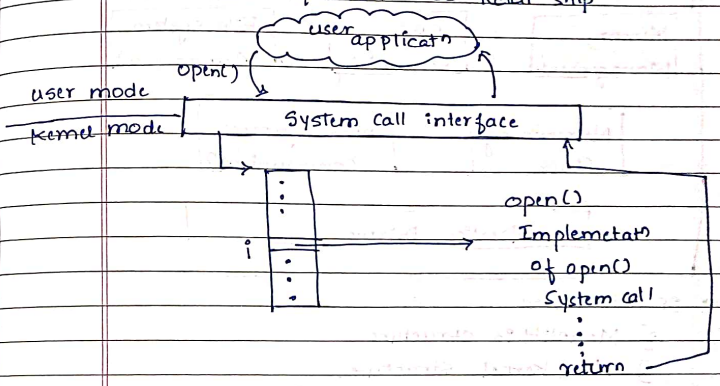


Date / /
Page No.:

* System Call Vs Procedure Call

System Call	Procedure Call
→ Uses a trap Instructn	→ Uses a CALL Instructn
→ System shifts from user space to kernel space	→ Stays in user space (or Kernel space)... no shift
→ TRAP always jumps to a fixed address	→ Re-locatable address

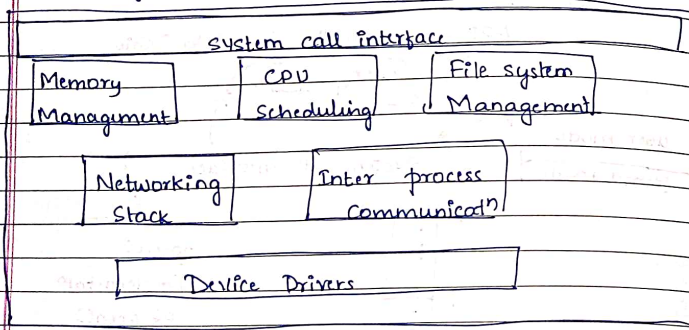
→ System Call interface :
It is intermidiate between the User Space API and system calls.
API - System call - OS Relatn ship



Date : / /
Page No.:

* Types of System calls
  - → Process control
  - → File management
  - → Device management
  - → Informatn Maintenance
  - → Communication
  - → Protectn

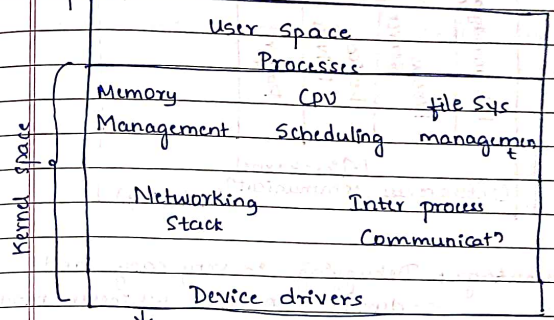
* OS STRUCTURES:
  What goes into an OS?



→ OS Structures:
  1] Monolithic structure
  2] MicroKernel Structure.

Date / /
Page No.:

1] Monolithic Structure: A single structure. All kernal, User space all are placed in same structure.
In user space we have only user space process remaining that all memory management, CPU resche-duling, File sys management all will be in kernel space.

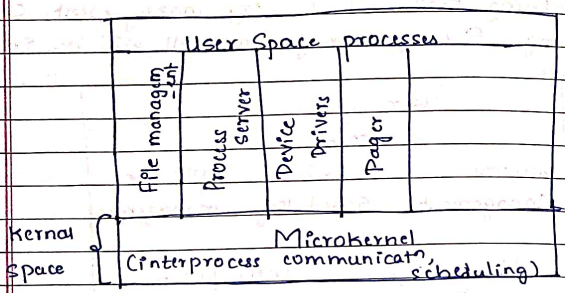


↓
Advantage: We don't need to have very well difined interfaces to deal with diff components of OS

But debugging may be the issue

Date : / /
Page No.:

2] Microkernel structure: In this core kernel related things are in the kernal space all the other modules will be in userspace.



Advantage: Debugging is very easy and during run time whichever module are recorded they can be loaded

Microkernel is
- highly modular
- Intteractions bt components strictly through well defined interfaces (no backdoors).
- Kernel has basic inter process communicatn & scheduling Everything else in user space.

* Program Optimization:

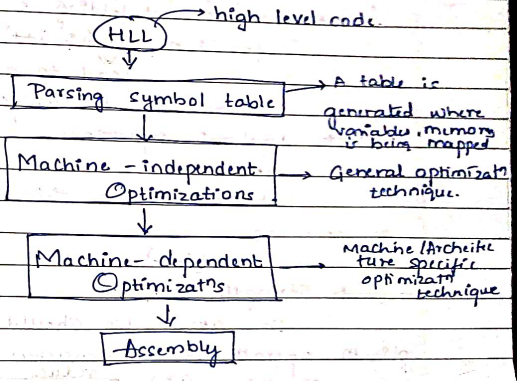
→ Constraints: Time, memory, power

COMPILATION:

W.r.t embedded system

→ compilatn - translatn + Optimization
↳ of high level language to Assembly level language

Compilatn flow:



Date : / /
Page No.:

* W of the following correspond to architecture specific optimizatn technique?
  Register reuse ✓
  Instructn select ✓
  Instructn sequencing ✓
  Expression Simplificatn ✗

OPTIMIZATION TECHNIQUES:

① * Arm specifixatn optimizatn technique : using condn codes

② * Expression Simplificatn optimizatn technique
  → W.r.t constant folding :
  for (i=0, i<8/1; i++)
  lead to an constant
  u should not be in for loop
  → Algebraic expression:
  a*b + a*c
  We can reduce this by taking a as common
  a(b+c)
  → Strength reductn
  In a*2 : we should not use multiplicatn operatn as it consumes more tym
  so use a*2 —reduce to→ a<<1

→ Deadcode executn:
  Deadcode means a part of code u is never executed but will be a part of your code
  Ex: #define DEBUG 0
  if (DEBUG) dbg(p1); → These kind will never executes. We should eliminate these deadcode.

③ Procedure inlining: Instead of writing a func write the statement as it is.
  This eliminates procedure linkage overhead

→
Procedures	Inline-procedure
• save memory	• Save time

Correct statements W.r.t code Optimizatn
* Using inline functions compared to functions saves time
* Using functions compared to inline functions saves memory.
* Dead code elimation saves memory.
* Moving constants out of loop saves time

④ Loop unrolling:

```
for(i=0; i<4; i++)
    a[i] = b[i] * c[i];
```

here for loop has to undergo 4 times i++ & less operation increment also should happen ∴ it consumes tym so we unroll the loop & write

```
a[0] = b[0] * c[0];
a[1] = b[1] * c[1];
a[2] = b[2] * c[2];
a[3] = b[3] * c[3];
```

we can save tym

But sometimes we don't prefer full unrolling

Ex: if we want to try unrolling only 2 tyms

```
for (i=0; i<2; i++)
{
  a[i*2]   = b[i*2]   + c[i*2];
  a[i*2+1] = b[i*2+1] + c[i*2+1];
}
```

↓

Saves no of iterations & types but costs memory.

⑤ Loop fusion and distribution:

↳ means combining of two loops into 1: distributing the loop.

CHAPTER 2

2. PROCESS MANAGEMENT

BASICS OF PROCESS:

PROCESS: Program in execution is called as process
↳ executing instance of the program
Process is also called as task, jobs.

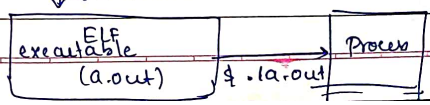
We have disk, CPU and main memory
→ The program/applications will be stored in the disk/ secondary memory. To convert them into process they have to be lauch to main memory (RAM).
So when program will be executed they will be part of main memory
So "process" are the executing instance of the application/program.
There may be multiple instances of the same program

both are of disk

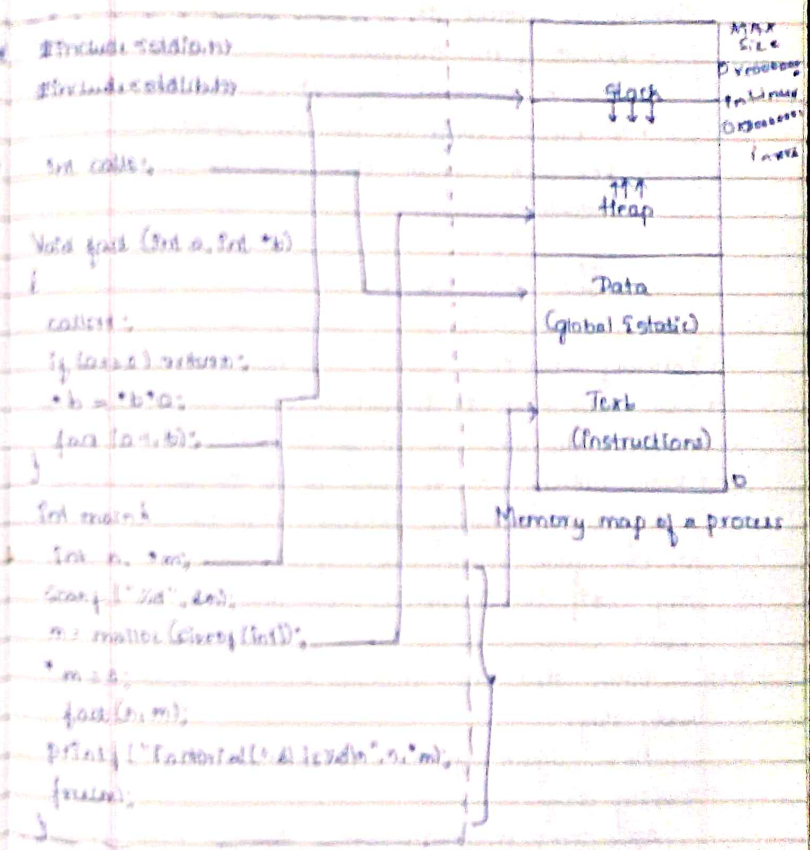
Ex:

```
#include<stdio.h>
int main(){
  char str[] = "Hello world\n";
  printf ("%s", str);
}
```

↓ $gcc hello.c



Process Memory Map in User Space



Date : / /
Page No.:

Process is related to
- Execution of the program
- has to be part of RAM (mainy memory)
- To execute progress we use commands
- For each program there may be multiple instance (processes)

STATE (stage) OF PROCESS

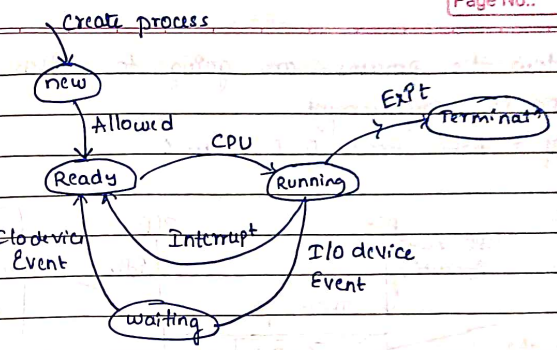
→ Initially when it is created → new process stage
→ new process when it is ready to execute → ready stage
→ From ready stage it can go to run^n stage
→ From running " it may go to it go to ready stage upon occuring Interrupts (high priority Interrupt) or else from running stage if it we go to waiting stage if it is waiting for any i/o device or event

When that event/i/o device occurs it again go to ready stage.

Now again getting excecess to CPU it goes to Running stage.

After completing all events through Exit it goes to Terminate stage.

Date : / /
Page No.:



PROCESS CONTROL BLOCK (TASK CONTROL BLOCK) (PCB)

Process ID (number of each block)
PC (program control)
State
Memory list
I/O
Accounts
⋮

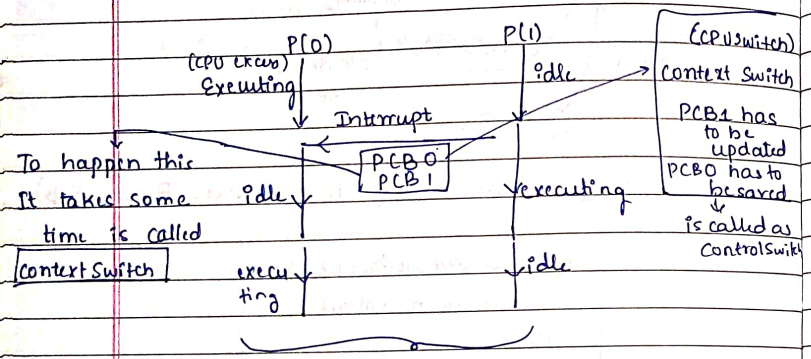
All the accepts related to process will be part of Process control block.

→ OS is going to represent the process in terms of PCB

Date : / /
Page No.:

* How the processes are going to switch depending on CPU requirement.

→ If I have Process 0 (P0) & Process 1 (P1)



Process Schedulers:

Date : / /
Page No.:

PROCESS SCHEDULERS:

* process Scheduling Queues:

↓

① Job queue: Initially when the processes are created they are placed in job queue.

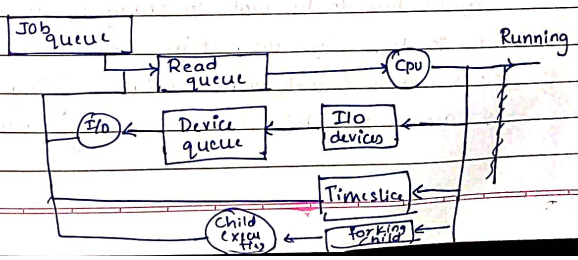
In job queue, processes will not be ready for execution.

② Ready queue: When processes are ready for execution & they are in need of CPU they are placed in Ready queue.

③ Device queue: Processes which r ready for execution & if they are in need of IIO devices, such processes are placed in device queue.

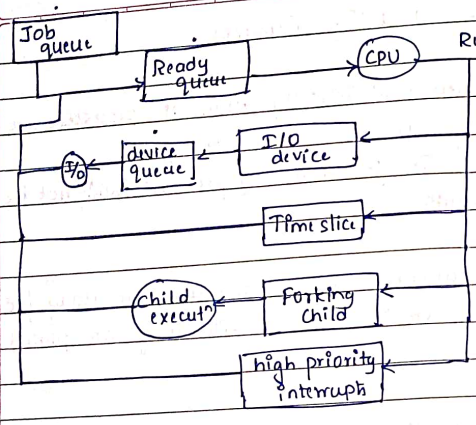
processes are going to switch between these 3 kinds of queues.

representation of process Scheduling



H/W * How are ios & Android are similar?
" " they different?

Date : / /
Page No.:



Forking child means a sub process in the same (main) process

SCHEDULERS:

3 types of schedulers that a process is going to deal with are:

1] Long term schedulers: The process w is in Job queue has to be sent to ready queue → This work is done by long term schedulers.

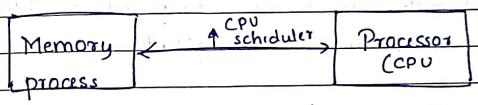
2] CPU Scheduler/Short term Scheduler: going to give the access of CPU to the ready queue process

When some processes are not required w are ready queue such processes can be sent back to job queue by medium term scheduler

3] Medium term Scheduler: Swap process bt ready queue and job queue. Jobqueue ⟺ Ready queue

* Job queue is imp in deciding "degree of multiprogramming", because job queue is going to place process into ready queue

—: CPU/Short term scheduling :—



→ CPU scheduling is basis for multiprogrammed OS.

Process Execut^n cycle:

→ CPU burst: pro When process is doing CPU relat^n oper^n tym spent is called CPU burst.

→ I/O burst (wait): Whenever process is doing Wordated operat^n tym spent is called I/O burst.

A process can do either I/O related operat^ns or CPU related Operat^ns.

Date : / /
Page No.:

CPU related operat^{n}s → logical operat^{n}s
→ dead store
→ Add store
→ Read from file
} CPU burst

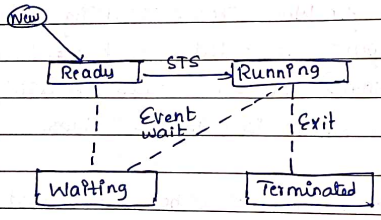
I/O related operat^{n} → Reading a sensor data
Wait for I/O → I/O burst

* Types of Processes :-

• CPU bound processes : This processes will have long durat^{n} CPU burst compared to I/O burst
Ex: Simulat^{n}, compilat^{n}.

• I/O bound processes : This process will have long duration I/o burst compared to CPU burst
Ex: Data base applicat^{n}

• CPU Scheduler (STS) (short term scheduler).



Date : / /
Page No.:

* Scheduling types :
1. Running to wait state
2. Running to ready "
3. Wait to ready state
4. Exit

1 and 4 scheduling cond^{n}s are Non preemptive or cooperative.

2, 3 scheduling condit^{n}s are preemptive scheduring

→ In preemptive Scheduling : the resource will be dealloca-ted. → Ex: Windows 95, Mac OS

→ In nonpreemptive scheduling : Resource will not be deallocated
↳ Ex: Windows 3

* DISPATCHER :
↳ A module that handover CPU to the process
↳ Dispatcher module gives control of CPU to process

func^{n}s of dispatcher are:
1] Context Switch : saving the ~~previous~~ current process (PCB) & updating new process (PCB)
2] switch to user mode
3] jump to locat^{n} to start the execut^{n} for current process

Dispatch latency : Time interval between exiting one process from running to ready state & making process w & in ready state to start its execut^{n} this is called dispatch latency

→ This dispatch latency has to be as minimum as possible.
→ The func^ns performed by dispatcher should be very fast.

* SCHEDULING CRITERIA:

* Why Scheduling Criteria?
→ As we know we have many scheduling types (Algorithm) (nonpreemptive, preemptive and preemptive). These Algorithms performance is different for different processes so in order to understand how these Algorithms performance, we need to study scheduling criteria

→ Scheduling criterias:

① CPU utilization: OS has to ensure that CPU is allocated to any of process during interval of execut^n. CPU should not be idle.
② Throughput: OS will be executing diff processes.
↳ is a fn no. of process completed per unit time.
③ Turnaround Time: Total time spend by the processor in the ready state, in running state or execut^n or waiting for a I/O.
④ Waiting Time: Whenever process requests for I/O device it enters into waiting ~~time~~ state. The tym spent in wait-ing state is called waiting time.
⑤ Response time: time interval is the ist request sent by the process and the first response (by) of OS.

→ Optimized Criteria:
① CPU utilizat^n should be maximum
② Throughput should be maximum.
③ Turnaround time should be minimum.
④ Waiting time should be minimum
⑤ Response time should be minimum
} for better performance of Scheduling Algorithms.

BASIC DEFN:

① Arrival time (AT): The time interval @ w process enters ready queue.
② Burst time / Execut^n time (BT): Time spent by procss in executing CPU related operat^ns
③ Turnaround time (TAT):
TAT = BT + Wait time
④ Completion time (CT) time interval @ w process complete the task and exits the system

Date : / /
Page No.:

⑤ Wait time (WT) = time interval process waiting in ready queue and for I/O

WT = TAT - BT

⑥ Response time (RT) = time interval bt request sent by process and first response of the os.

RT = Time @ w process gets CPU - Arrival time

: SCHEDULING ALGORITHMS — ① FCFS :

① FCFS : First come First serve
- This is non preamptive scheduling algorithm
- follows FIFO queue.

Exercise 1

processes	AT	BT
P_1	0	5
P_2	1	3
P_3	3	2

AWT : Avg wait time : We have to find AWT to know the performance.

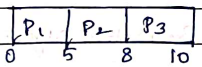
Soln : We have to plot a gantt chart

gantt chart : Time vs Process

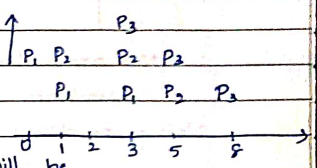
Gantt chart will help in determining execution time & wait time

Date : / /
Page No.:

At t=0, P_1 allocates with resource & at t=5 it completes execuith (task)



FIFO



At 5 P_1 completes and P_2 will be allocated and till 8 it completes the task.

At 8 P_3 allocates and completes task @ 10.

→ WT — P_1 = 0-0 = 0 = tym @ w the resource is allocated - Arrival time = WT

P_2 = 5 - 1 = 4

P_3 = 8 - 3 = 5

Avg Wait time AWT = $\frac{\text{Wait}(P_1+P_2+P_3)}{3} = 9/3 = 3$ //.

→ Advantages and disadvantages of FCFS:

Advantages : Easy to understand & easy to implement

No extra hardware required to implement this Algorithm.

Disadvantages : * Avg wait time is normally high compared to others algorithms.

* When the diff process (CPU bound & I/O bound processes), it lead to convoy effect.

Date : / /
Page No.:

② Scheduling Algorithm : Shortest job first Scheduling
→ SJF Algorithm

Non preemptive Algorithm | Preemptive Algorithm

Non preemptive: If all processes arrive @ same time interval then SJF Non preemptive Algorithm is applied.

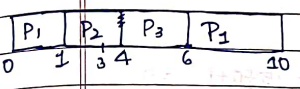
Preemptive : If the processes arrive @ different intervals of time then SJF Premptive algorithm is applied.
↳ also called as Shortest Remaining time first Algorithm.

Exercise 1. Preemptive version of SJF

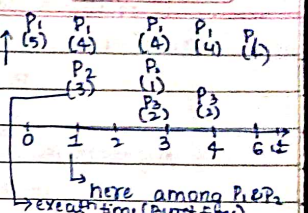
process	AT	BT
P_1	0	5
P_2	1	3
P_3	2	2

Here we need to calculate AWT - Avg Wait time

soln: We have to start with gantt chart
↓ Time Vis process.



Date : / /
Page No.:



scheduler should should select 1 P

As P_1 At time 1, as P_2 burst time is less (3) that P_1 burst time(4) acc to preemptive algorithm P_2 is allowed to get access of CPU by preempting P_1.

CPU is allocated to P_2, hence P_1 will stop its execution and will be waiting in readyqueue and P_2 is allocated with CPU resource @ t=1.

P_2 will continue to execute till t=3

@ t=3, we have P_1, P_2, P_3, CPU schedulers has to select one process by employing preemptive algorithm. So @ t=3, P_2 is selected and hence P_2 will continue to execute till 4 & stop @ 4.

@ t=4, as P_3 has less burst time, it will start execution till t=6,

@ t=6, we have only P_1 that will execute and completes @ t= 10.

In this case, we have ignored context switch time

↳ time @ w resource is allocated - Arrival time

Wait time: $P_1 = 0-0+6-1 = 5$

$P_2 = 1-1 = 0$

$P_3 = 4-3 = 1$

Avg wait time = $\frac{P_1+P_2+P_3}{3} = \frac{5+0+1}{3} = 2$

Avg wait time (preemptive STF) $\leq$ Avg wait time (FCFS)
2 < 3

∴ preemptive STF is much better

→ Advantages and disadvantages of ~~preemptive~~ STF

Advantages: 1] less / Minimum AWT compared to FCFS
2] Non premptive STF is suitable for Batch processing OS.
3] Easy to implement since we know execution time
4]

Disadvantages: 1] difficult to find remaining time of every process.
↳ To overcome this we can use approximate time method.

↳ here also we have premptive & non premptive

③ Scheduling Algorithm: Priority based Scheduling Algorithm
↳ In this each and process will be assigned with priority number.

Based on priority, the CPU scheduler will selects the process & assign resource.

priority no. starts with 0 to 7/255/or 4095 (depends on system).

0 - highest priority
7/255/4095 - lowest priority
} might be reversed based on system.

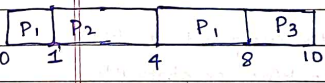
Whenever 2 processes (of same priority) arrives @ same interval will be executed in FCFS manner.

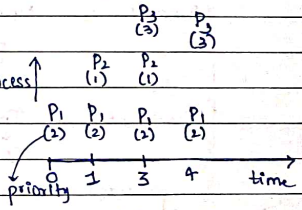
Exercise:

process	AT	BT	priority
P_1	0	5	2
P_2	1	3	1
P_3	3	2	3

↳ preemptive Scheduling

Gantt chart





@ t=1, As P_2 has high priority P_1 will preempted with P_2 & P_2 will be allocated with resource

@ t=3, since p_2 has highest peiority it will continue its execut^n till t = 4

@ t=4, P_2 is completed & it will exit,
Now P_1 & P_3 are there as P_1 has more priority it will start execut^n. & completes @ t=8

@ t=8, P_3 will be allocated & executed & completes @ 10.

Wait time : $P_1 = 0-0+4-1 = 3$
$P_2 = 1-1 = 0$
$P_3 = 8-3 = 5$

Avg wait time = $\frac{P_1+P_2+P_3}{3} = \frac{5+0+3}{3} = 8/3 = 2.67$

AWT (priority based) < AWT (FCFS)

→ Advantages & disadvantages of priority based
Advantages : Non-preemptive are suitable for batch process OS
AWT is better compared to FCFS
Disadvantage : Indefinite blocking or starvat^n occurs where we have more no. of processes & occurs @ diff time
high priority P makes low prority P to Indefinite blocking

Indefinite blocking occured in IBM 7094
↳ Soln for this problem is Aging
Aging : is basically OS over a period of execut^n time it will try to ↑ the priority of low priority processes

④ Scheduling Algorithm : Round Robin Scheduling

Round robin scheduling is designed for Time sharing system
Each and every process will be executed for a certain fixed period of time
Irrespective of arrival time, priority, burst time, every process will be executed for a certain period (interval) of time.
This fixed time is called as "Quantam time".
Once the execution completes, process will be preemptive and made to stay in ready queue (FIFO queue).

Example: Give Quantam time = 3, calculate AWT

Process	AT	BT
P_1	0	5
P_2	1	3
P_3	3	2

Date : / /
Page No.:

@ t=3, since P_2 has highest priority it will continue its execution till t = 4

@ t=4, P_2 is completed & it will exit,
Now P_1 & P_3 are there as P_1 has more priority it will start execution & completes @ t=8

@ t=8, P_3 will be allocated & executed & completes @ 10.

Wait time : $P_1 = 0-0+4-1 = 3$

$P_2 = 1-1 = 0$

$P_3 = 8-3 = 5$

(AWT) Avg wait time $= \frac{P_1+P_2+P_3}{3} = \frac{5+0+3}{3} = 8/3 = 2.67$

AWT(STF) < AWT (priority based) < AWT (FCFS)

→ Advantages & disadvantages of priority based

Advantages : Non-preemptive are suitable for batch process OS
AWT is better compared to FCFS

Disadvantage : Indefinite blocking or starvation
↓
Occurs where we have more no. of processes & occurs @ diff time
high priority P makes low prority P to wait indefinite time called as Indefinite blocking

Date : / /
Page No.:

Indefinite blocking occured in IBM 7904
↳ soln for this problem is Aging

Aging : is basically OS over a period of execution time it will try to ↑ the priority of low priority processes

(4) Scheduling Algorithms : Round Robin Scheduling

Round robin scheduling is designed for Time sharing system

Each and every process will be executed for a certain period of time

fixed

Irrespective of arrival time, priority, burst time, every process will be executed for a certain period (interval) of time.

This fixed time is called as "Quantam time"

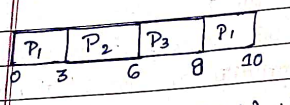
Once the execution completes, process will be preemptive and made to stay in ready queue. (FIFO queue).

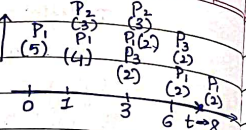
Example: Give Quantam time = 3, calculate AWT

Process	AT	BT
P_1	0	5
P_2	1	3
P_3	3	2

Date : / /
Page No.:

Gantt chart





Avg weight Time (AWT) : $P_1 = 0-0+8-3 = 5$

$P_2 = 3-1-0 = 2$

$P_3 = 6-3 = 3$

$AWT = \frac{P_1 + P_2 + P_3}{3} = \frac{5+2+3}{3} = \frac{10}{3} = 3.33$

AWT = 3.33 msec.

AWT(RS) > AWT (FCFS)

→ Convoy effect can be reduced by this algorithm

→ We have to select Quatam time correctly.

Date : / /
Page No.:

* INTER PROCESS COMMUNICATION :- (IPC)

Defination : It is nothing but the process of communi-cation of one process with another process

→ How one process communicate with other process.

(Use) Advantages of IPC :

→ Used in information sharing

→ used in modularity

→ " " computational speed

→ Used convenience.

→ Used in info sharing : It can share files, I/O devices like printer & scanner from one process to another.

→ Modularity : If big processes are u can be divided (broken) into sub process as modules u will be able to perform processes is called modularity.

→ Computational speed : extension of modularity, to increase computational speed.

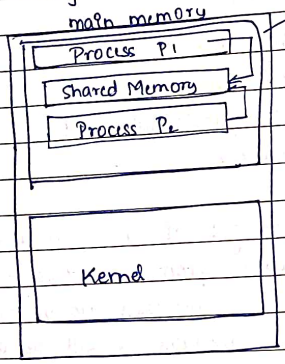
→ Convience to use.

Date : / /
Page No.:

Mechanisms used in IPC:

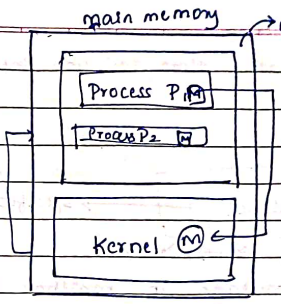
1] Shared Memory
2] Message Passing

1] Shared Memory: If @ all two processes say P_1 want to communicate with P_2, P_1 (writes) sends the data to Shared memory & P_2 (reads) receives it from shared memory.



2] Message Passing:
P_1 writes the data (message) to the kernel and P_2 reads from the kernel in this method.

Date : / /
Page No.:



TYPES OF MESSAGE PASSING TECHNIQUE

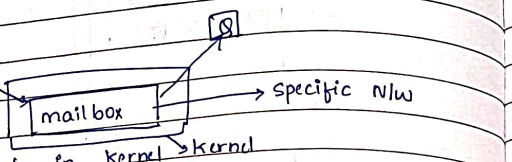
→ Features of message passing
1] Naming
2] Synchronisatn
3] Buffering

1] Naming : Whenever two process communicate they have to identify each other ∴ is done by naming.
Naming can be done in 2 ways.
1] Direct communicatn
2] Indirect communicatn

1] Direct communicatn:
Send [P, message]
Recieve [Q, message]
P, Q are name of process.

Date : / /
Page No.:

2] Indirect communicatn: here we use intermediator called as mail box



• Mailbox will be in kernel
P writes message in mailbox & Q receives it from mailbox.

send (mailbox, message) } here we don't
receive (mailbox, message) } use names

3] Synchronizatn can be done in 2 ways:
① ↳ Blocked sender / blocked reciever (synchronous)
② ↳ unblocked sender / unblocked receiver (Asynchronous)

① In this if message is send then it is blocked until it is received by another process suppose another process is busy it has to wait.
② This is Asynchronous ; deast bother about the process sending & receiving.

3] Buffering : 3 types:
→ zero buffering (Capacity)
→ finite capacity
→ Infinite capacity.

Date / /
Page No.:

* Zero Capacity: As there is no buffering, sender has to wait until the message is received by reciever.
* finite capacity: have limited buffering
* Infinite " : no limitatn for no. of buffers.

Date : / /
Page No.:

CHAPTER 3

MEMORY MANAGEMENT

INTRODUCTION:

Computer sys: CPU + memory

How speed is the CPU and how large is the memory. These 2 factors decides what all the computer system can do.

→ Memory Criterion:

1. Size : we except to be large
2. Access time : time taken by the system to write or read the memory

It should be very less

3. per unit cost : We except this should be very less

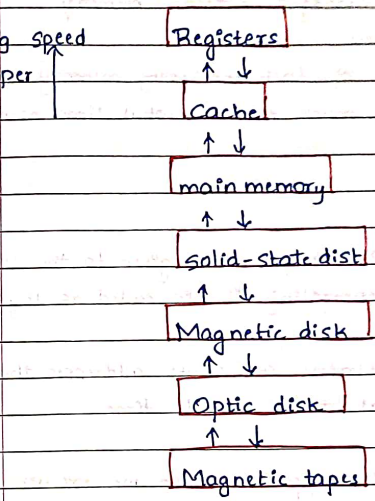
Satisfying all 3 criterion in single is not possible.

∵ As size ↑, access time ↑.

Locality of reference: If a program is in secondary mem & if it has to be executed it has bring to main mem & is nearer to CPU & execute in quicker way is called locality of reference & help us to attain all the memory criterion.

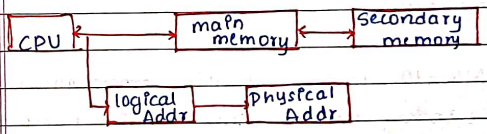
Date : / /
Page No.:

Memory Hierarchy



→ To transfer from the secondary memory to main memory

* It has allocate memory
* Address translatn



Page No.:

* What is Memory Management ?

→ Handles or manages primary memory
→ moves processes
→ keeps track of each and every memory locatn
→ Checks how much and @ what time memory to be allocated

* Basics:

Logical Address: If address is given to the executable code on compiling it is called as logical address.
→ logical Addresses are those addresses that are given by CPU at the compile time.

Logical Process Address space: When we execute there will be mapping of datas of code in memory map and this layout is called process Address space and the address
Physical address: Actual memory locatn in the main memory.

→ Memory Allocatn : means allocating process a memory from secondary memory into main memory.

Date : / /
Page No.:

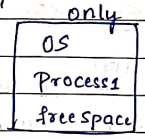
* Memory Allocatn Techniques:

Types

1) single - partition allocation
2) Multiple - partition allocation

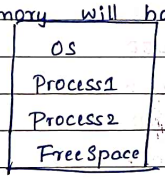
* Single Partition Allocation:

here Memory is 2 types:
low memory and high memory.
In low memory we will have OS
In high " only 1 Process will be there



* Multiple Partition Allocation:

here we have
low memory have OS
high memory will have n no of processes



Date : / /
Page No.:

* What is Memory Management ?

→ Handles or manages primary memory.
→ moves processes
→ Keeps track of each and every memory locatn
→ Checks how much and @ what time memory to be allocated

* BASICS:

Logical Address: IP address which is given to the executable code on compiling it is called as logical address.
→ logical Addresses are thate addresses that are given by CPU at the compile time.

Logical /process Address space: When we execute there will be mapping of datas of code in memory map and this layout is called process Address space. and the address

Physical address: Actual memory locatn in the main memory.

→ Memory Allocatn : means allocating process a memory from secondary memory into main memory.

Date / /
Page No.:

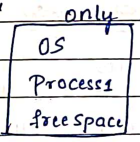
* Memory Allocatn Techniques:

Types

1) single - partition allocation
2) Multiple - partition allocation

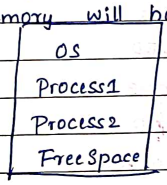
* Single Partition Allocation:

here Memory is 2 types:
low memory and high memory.
In low memory we will have OS
In high " only 1 Process will be there



* Multiple Partition Allocation:

here we have
low memory have OS
high memory will have n no of processes



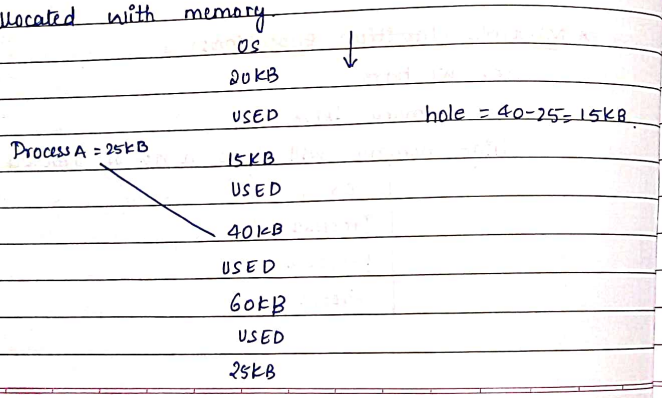
2 types:
Fixed Partitn & Variable Partitn.

→ Fixed Partition: In this, memory is being fixed partitioned slot. Each slots can be equal or unequal.

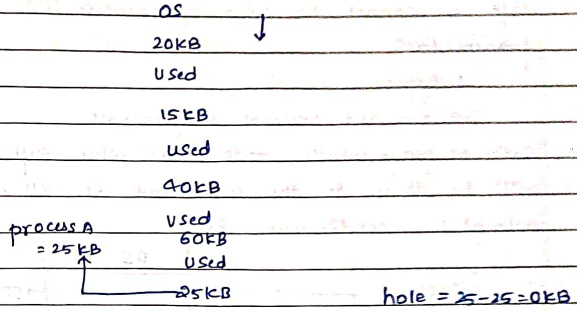
→ Variable Partitn: Whenever process requires it will be allocated. also called as dynamic partitn.

Memory Allocation Stratergies:

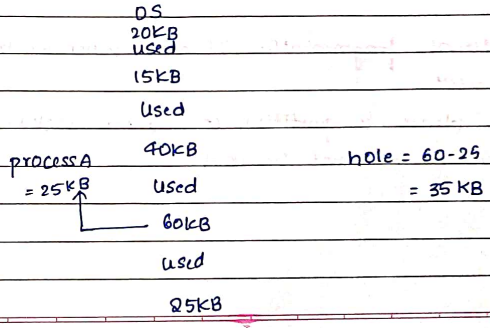
① First fit Stratergy: Here, (OS) it will start from first and then wherever sufficient space is there, process will be allocated with memory.



② Best fit Stratergy: OS looks for whole list of holes and allocate memory w is best suitable.



③ Worst fit Stratergy: It is opposite to Best fit Stratergy. (OS) It will look whole list of holes and allocate the largest memory size available



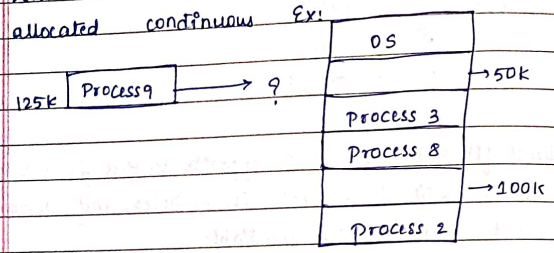
Date : / /
Page No.:

Fragmentation: As the processes are loaded and reboot from the memory there will be very little memory/space left w cannot be used by process. This is called as fragmentation.

2 types:

Internal and external fragmentation:

→ External fragmentation: This occurs when sufficient memory will be there in the table but it will n cannot be allocated continuous Ex:



→ Internal fragmentation: It occurs when we have fixed partition.

Where in partition so memory will be left. Whole memory will not be used.

Date : / /
Page No.:

MEMORY MANAGEMENT TECHNIQUE :

1] SEGMENTATION :

→ Why Segmentation?

We have seen there is a fragmentation problem in previous sect^n

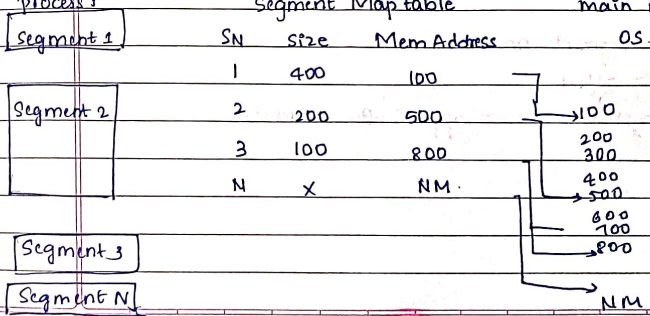
To overcome this we need a non contigous techniques. These techniques are Segmentation and paging.

Segmentation is in w

→ Memory is divided into variable size parts

→ & each part is known as segment.

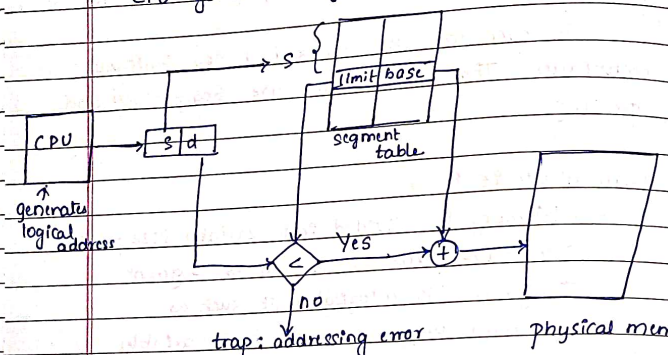
→ Segment is a logical unit such as main program, func., local variables



Date : / /
Page No.:

Segment
→ Mem map table is used to convert logical address to physical address.

→ logical to physical address translatn
CPU generates logical address.



s = Segment no.
d = offset
physical address = offset + Base addres

Date / /
Page No.:

2] PAGING:

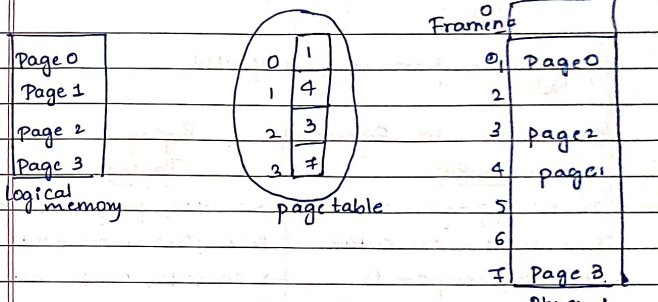
→ Why paging ?

To overcome fragmentatn problem.

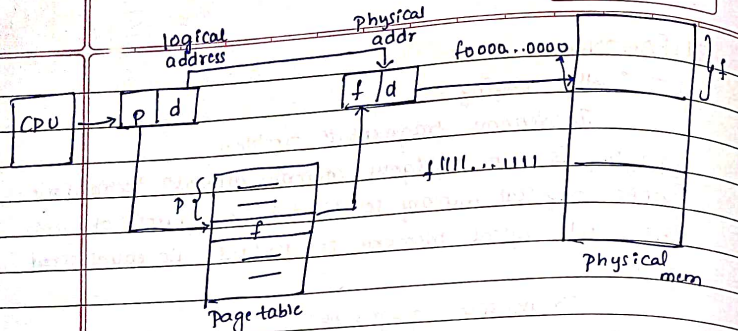
paging is non contigous memory allocatn technique where physical memory is divided into frames of same size and logical memory is divided into equal sized pages

Frame size = page size

To map frames & pages we have a page map table



Page translatn / logical to physical addr translatn



Frames = pages

CPU = page no (p) + offset (d)

physical addr = frame no (f) + offset (d)

Comparision bt Segmentatn & Paging

Segmentatn	Paging
→ In this, program is divided into variable size sections	→ In this, program is divided into fixed or mounted size pages
→ For this, compiler is accountable	→ In this, OS is accountable.
→ Here, sectn size is given by user	→ Page size is determined by hardware.
→ Segmentatn is slow	→ faster than Segmentatn
→ Segmentatn could result in external fragmentatn.	→ paging could result in internal fragmentation
→ Segmentation is visible to the user.	→ Paging is invisible to the user.