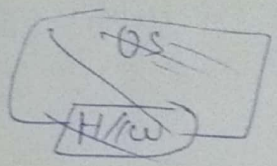
Operating System :

Interface between user of computer and hardware

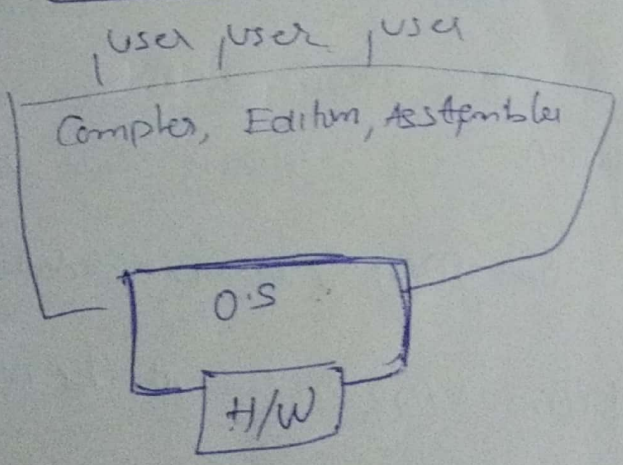
→ It provides an environment in which user can execute the program conveniently and efficiently

→ O.S creates an abstract machine that provides an easy to use interface for developing and executing the applications.

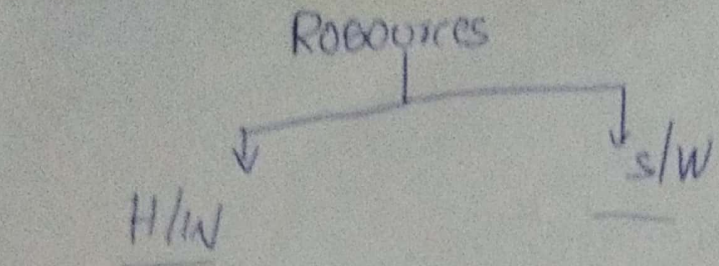


→ O.S resides top of the hardware

Abstract view of O.S



→ operating system is a resource allocator



cpu, mem, I/Od

semapho, monitor, fifo, pipe, file

Component of O.S

1) Control Unit:

Two operations

1) Generation of Timming & control signals
2) execution of micro-operations

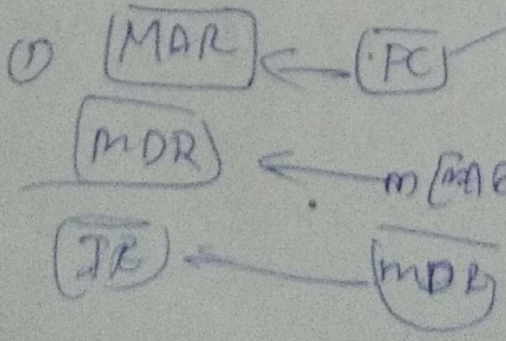
→ Interrupts are obtained in fetch and Execution cycle

Fetch Mprogram do's:

→ Take address from ~~Counter~~ PC and keeping into MAR

→ goes to location and getting ~~data~~ data and keeping on MDR (memory data register)

→ finally from MDR it will be placed on IR



→ program:

A finite set of instructions.

→ Each instruction has to do fundamental actions

1) fetch cycle
2) execute cycle
3) Interupt cycle

→ Instruction cycle

→ Each instruction in program can divided into set of micro-programs

ex: Fetch M prgm

set of collection of M operation constitute a M program.

→ These M operations are implemented by "CONTROL SIGNALS".

fetch - M-program

(1) MAR ← PC

(2) MDR ← M(MAR)

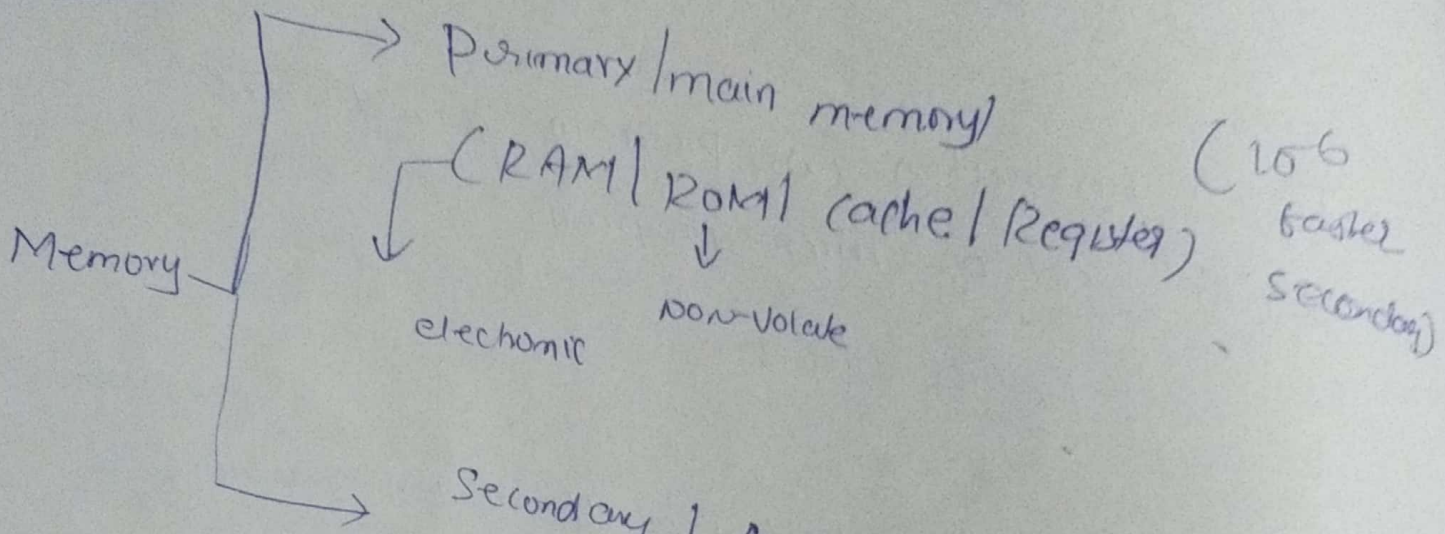
(3) IR ← MDR

Control signals

① PCout , MARin
② MARout , MDRin
③ MDRout , IRin

} These codes min 5 signal

Storage



1ms = 10^{-3} sec
1μs = 10^{-6} sec
1ns = 10^{-9} sec
1PC = 10^{-12}

Memory capacity	Accesstime	Bandwidth (MB)	managed by
Register	0.5 pc to JPC	20,000 to 100,000	compiler
cache	2ns to 10ns	10,000 to 20,000	H/w
main memory (RAM)	100 to 200	5000 to 1000	O.S
Disk	5ms to 10ms	20 to 110	O.S

RAM

Static RAM (flipflops)

Cache is builded

Dynamic Ram

(capacitors)

Refreshment required

ALU :

→ It performs data manipulation instructions i.e

1) Arithmatic instructions
2) Logical instructions
3) Shifting instructions

→ Program cannot Execute with CPU which is loaded in secondary memory.

→ according to von neuman architecture program must follow Stored program Concept.

→ Sequential flow of ~~exction~~ execution

→ Stored program concept

A program need to be execute that must be reside in main-main memory In prior to the execution

→ According to von neum. arteture sequence of execution

fetch, decod } fetch cycle
exe, store } Execute cycle

interrupt } interrupt cycle

Architecture

Basic

1. Von-neuman

other

6. superscalar
7. distributed
3. Harvard Architecture
4. Data flow
5. N/w architecture

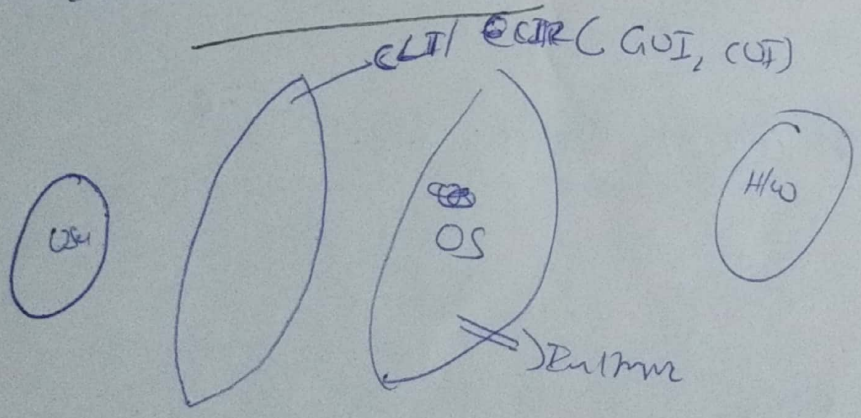
Design principles of OS & Architecture

→ User program cannot interact directly with O.S

i.e it need 1 level of interface is called

User interface (or) OS interface (or) common interface

(or) Commandline interface



Two types of interface

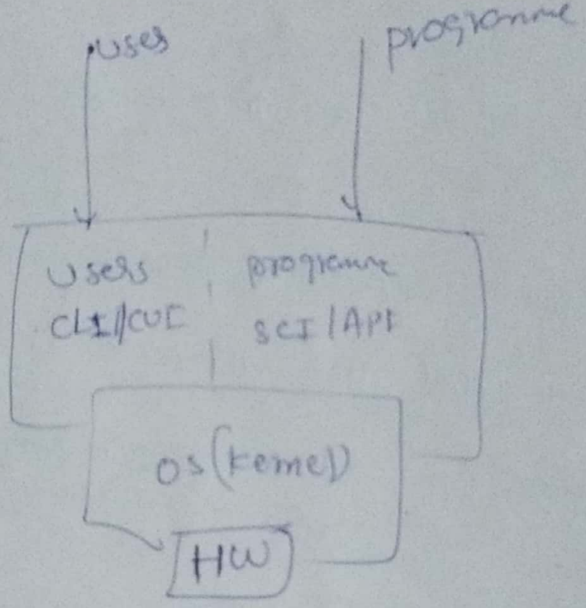
1. User & OS ⇒ UI / CLI [common interface / Command Line interface]
2. User and H/w = OS

→ programmer having own interface

System call interface (or) API interface (Application program interface)

(SCI) (API)

* Abstract view of interfacing diagram *



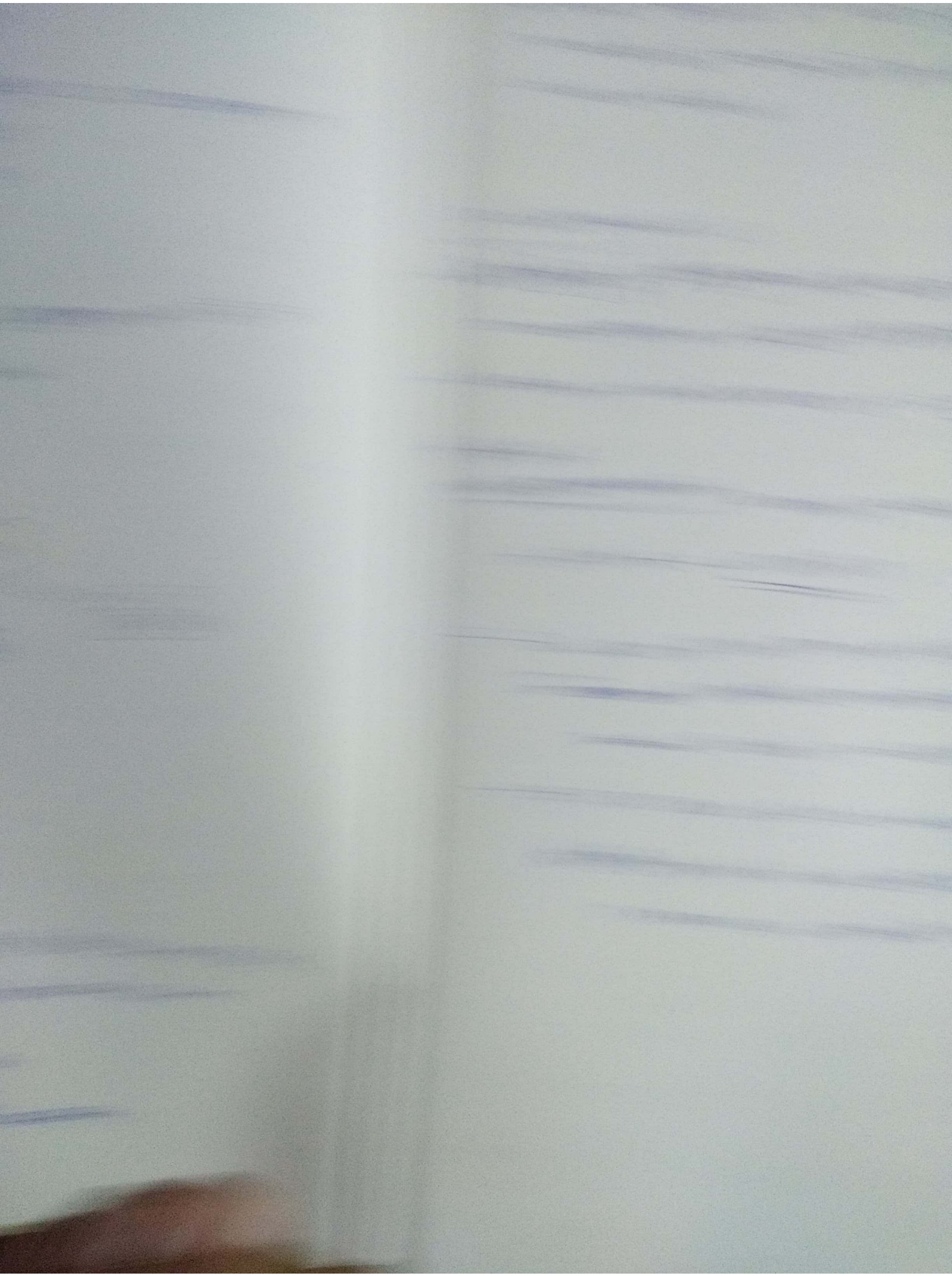
function (services) & Goals of an OS

Function

1) Execution of program
2) I/O operation
3) file system
4) Communication with different processes
5) protection to the user application

Goals

1) Convinient (user friendly)
2) Efficiency (execution time should be less)
3) Reliability [Correct result, error free,]
4) Scalability [ability to evolve in different version]
   → construct a system such a way as it permits effective development, testing, introduction of new functionalities with out affecting the existing one's
   eg: windows xp, → 7, → 8, → 10
5) Portability -
   easily movable across platforms



→ while executing if any I/O interrupt occurs process has to execute. In this situation CPU again will be in idle state

Multiprogrammed OS:

→ Capability of loading more than one program into main memory called multi-programming

→ keeping several processes in to mm

* **
multiprogramming = Running multiple programs on CPU at a time → wrong x

→ we can load only multiple programs. ~~In case~~ whenever mainmemory has many programs ~~then~~ is then it is multiprogramming

→ The CPU will switch between multiple programs ~~fastly~~ so that the user ~~gets~~ feels has if ~~he~~ is executing ~~the~~ programs parallely

multiprogrammed O.S is again classified in

1) NON- preemptive OS
2) preemptive OS

NON-preemptive: voluntary (whenever it wish) releasing of process from CPU

→ process can release CPU

+ when its execution completes
+ when I/O required
+ when system calls are occured

Preemption:-

→ forceble deallocation of the running process from the CPU

→ process can realse (i) not only whenever completes execution but is also when time slice expires. ii) When high priority process is enter in to the system.

multiprogrammed O.S can be (i) single user (ii) multi users

Single user :

→ more than one process can be submitted to the system by only one user

eg: windows 95, 98, 2000, XP, NT, etc

Multiple user.

→ Multiple user can submit multiple process to the system.

eg. client server OS like Linux, Windows server

* Example for NON preemptive multiprogrammed OS

* windows 3.0, window 3.11,

→ In these type of O.S other process are eagerly waiting for the one process to ~~get~~ leave CPU. → Technical term is called starvation.

* Examples for Preemptive multiprogrammed O.S

* windows 95, 98, 2000, XP, Vista, Linux, Unix etc

NOTE:

→ UNDER preemptive Based OS user process (Program) can preempt at any point of time and 'n' no of times.

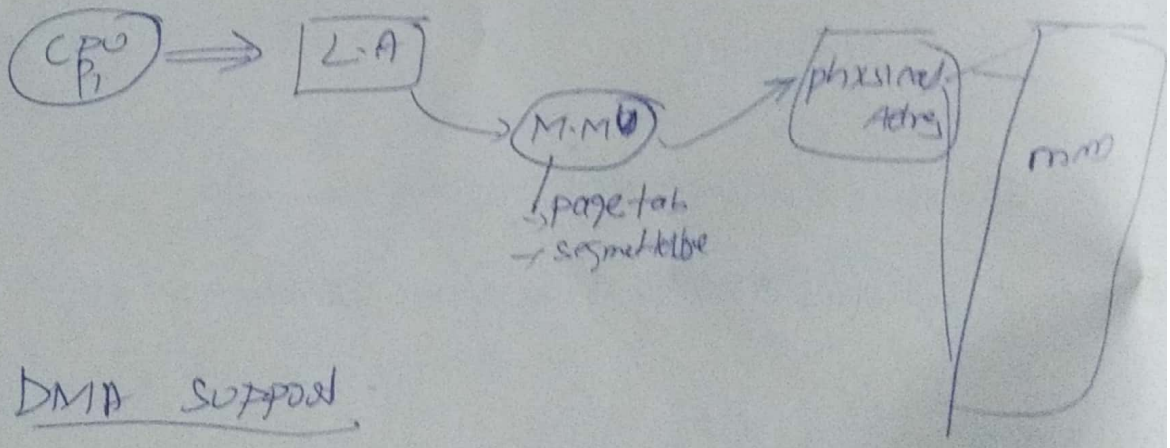
Timesharing OS:

It is also like multiprogramming OS except In Timesharing OS all process are allocated & assigned with some particular time.

** In order to design multiprogramed OS the H/W support is required are:

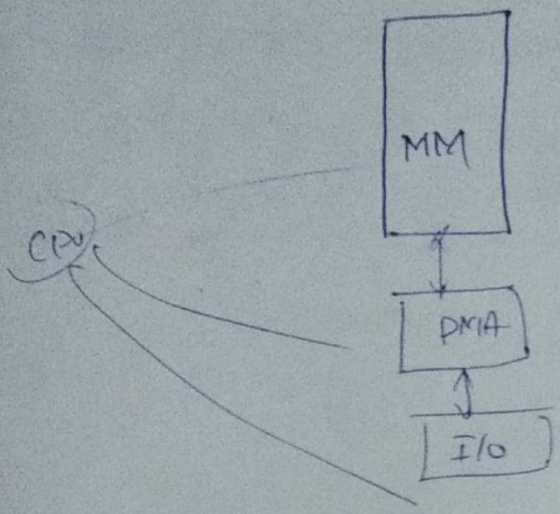
1. Address Translation (L.A to PA) → memory
2. DMA operation → I/O operation
3. At least two modes of CPU operation.

① Converting Logical Address to physical Address



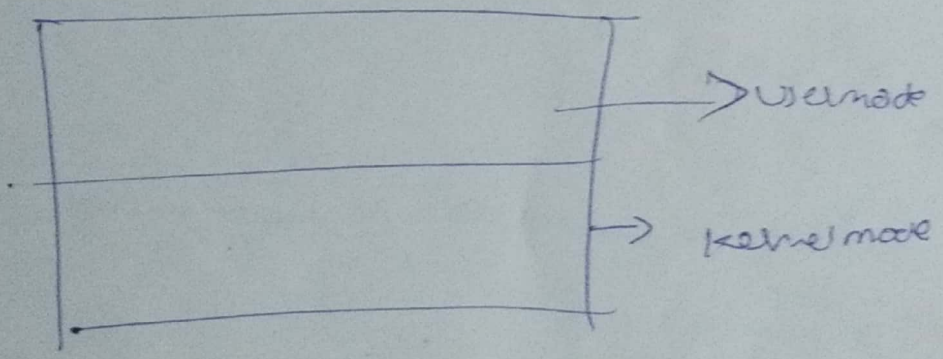
② DMA support

To speed up memory operation DMA controller is used to transfer bulk amount of data from memory to Disk to memory and vice-versa without involvement of the Processor



→ At least Two modes of CPU operation Should be there

→ Kernel instruction will run in kernel mode with extra atomic mode (with out preemption)

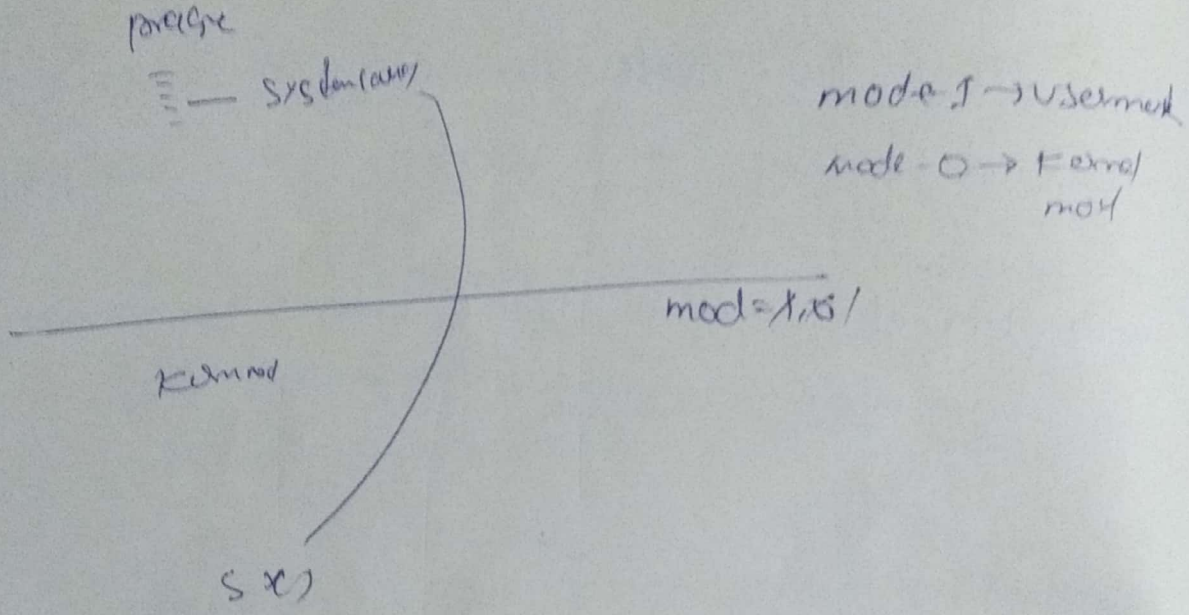


→ All user application program are being executed in user mode with preemption (or) Interruptability

→ And kernel programs/interrupt instruction are will be executed atomical (Non preemptive mode)

atomic nature → without preemption

→ when user program need service from OS via system calls. Transmission must be made from user to kernel mod to get user request satisfied



The mode bit is changed i.e. De user mode changes to kernel vice versa

Mode switching:

1) all library calls / predefined calls are replace by "Branch save address" (BSA)

2) System calls are replaced by an privilege instruction → SVC (supervisory call)

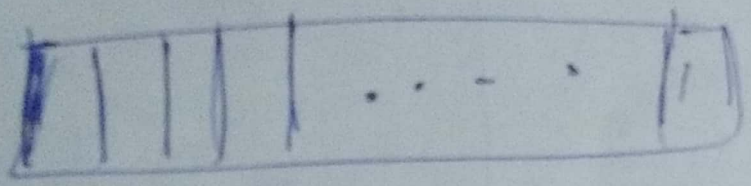
→ When SVC (supervisor call) is encountered, a

S/w interrupt is generated

→ for S/w interrupt, ISR will be executed

→ ISR will check mode bit through

PSW (Program Status Word)



→ The interrupt produced is software interrupt

```
#
=
main()
{
=
concat()
.
pid
.
fork() // SVC (System call replaced with SVC)
.
while()
}
```

when fork() is executed, it is replaced with SVC whenever SVC exception software interrupt will be occured and it will change

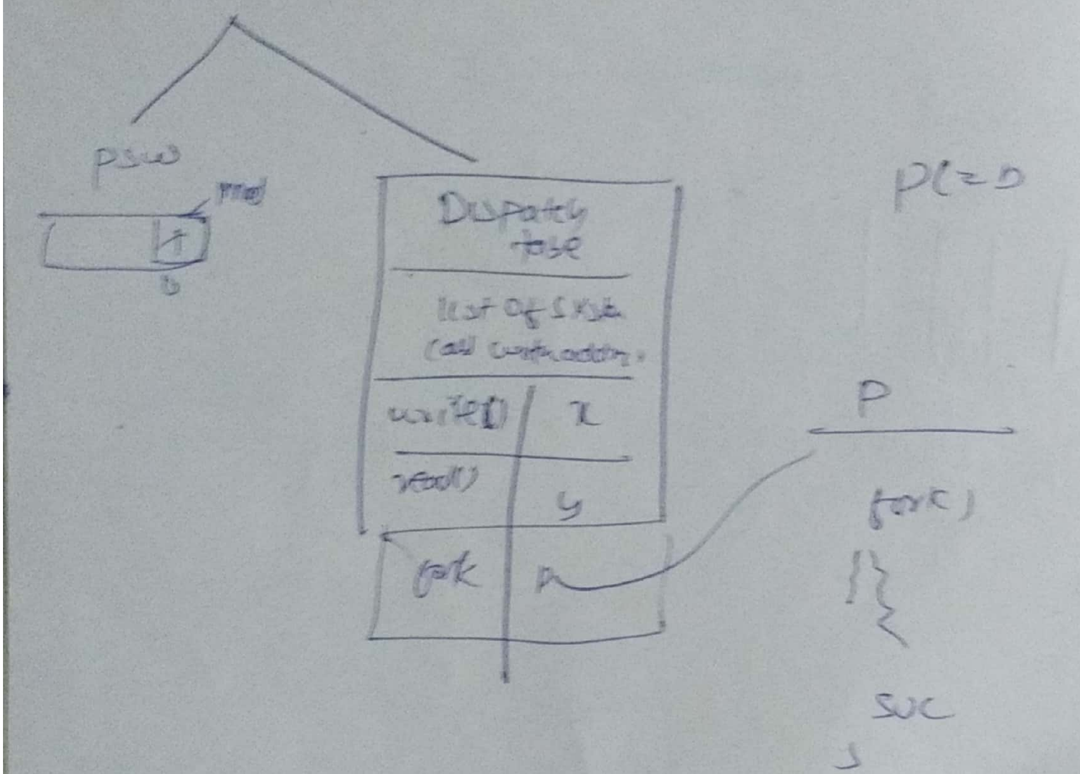
the mode bit and runs in kernel mode to

execute system call..

System call implementation

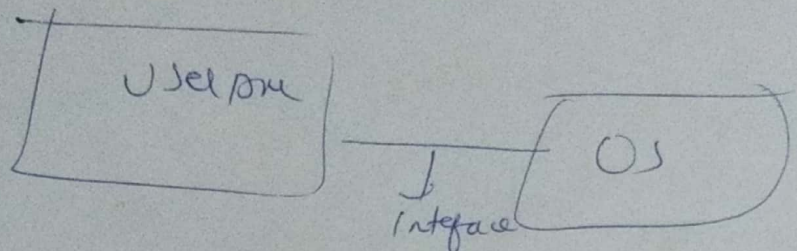
→ SVC → supervision call generates software interrupt

→ ISR



By

System call are extended information which provides information between user program and operating system



System calls are varies from os to os.

Classification of system calls

1) Program process control system call
2) file manage system call
3) communication system call
4) Device management system call
5) Information maintance system call

1)
* Allocate(), create(), Terminate(), free()

2)

*) create(), open(), close(), delete() read(), write(), modify()...

3)

establish connection()

send(), Recieve()

Device manag

setDeviceAttribute()
getDeviceAttribute()
releaseDevice()

(5) Information.

Time, Date, process Attributes

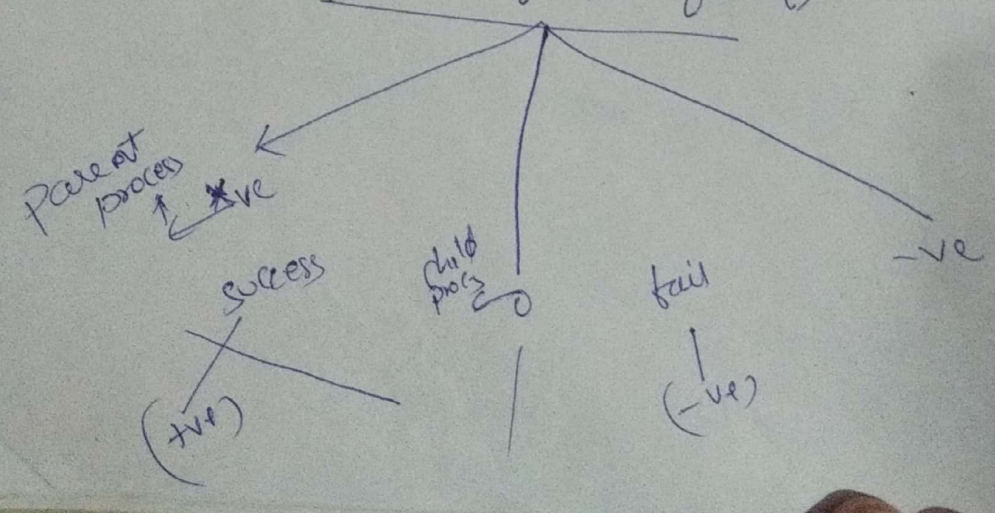
Case Study:

fork():- (Unix/Linux System call) equivalent to Create() in windows

→ fork used to create a new process
→ It creates a exactly duplicate of the orginal process including all descriptors, registers set.

→ fork system call returns '0' for the child process and child parent id is return to parent process

→ Return value of the fork()

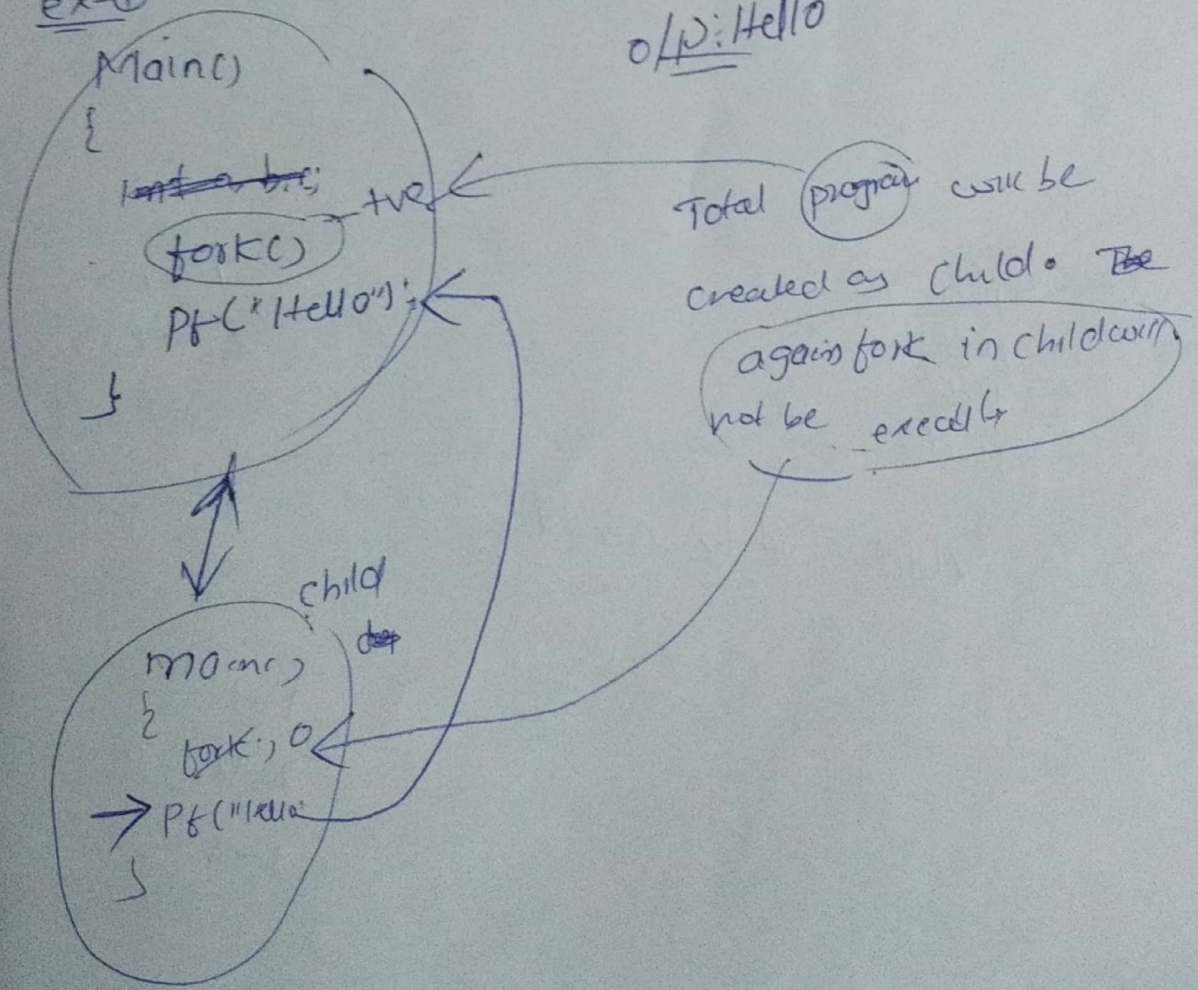


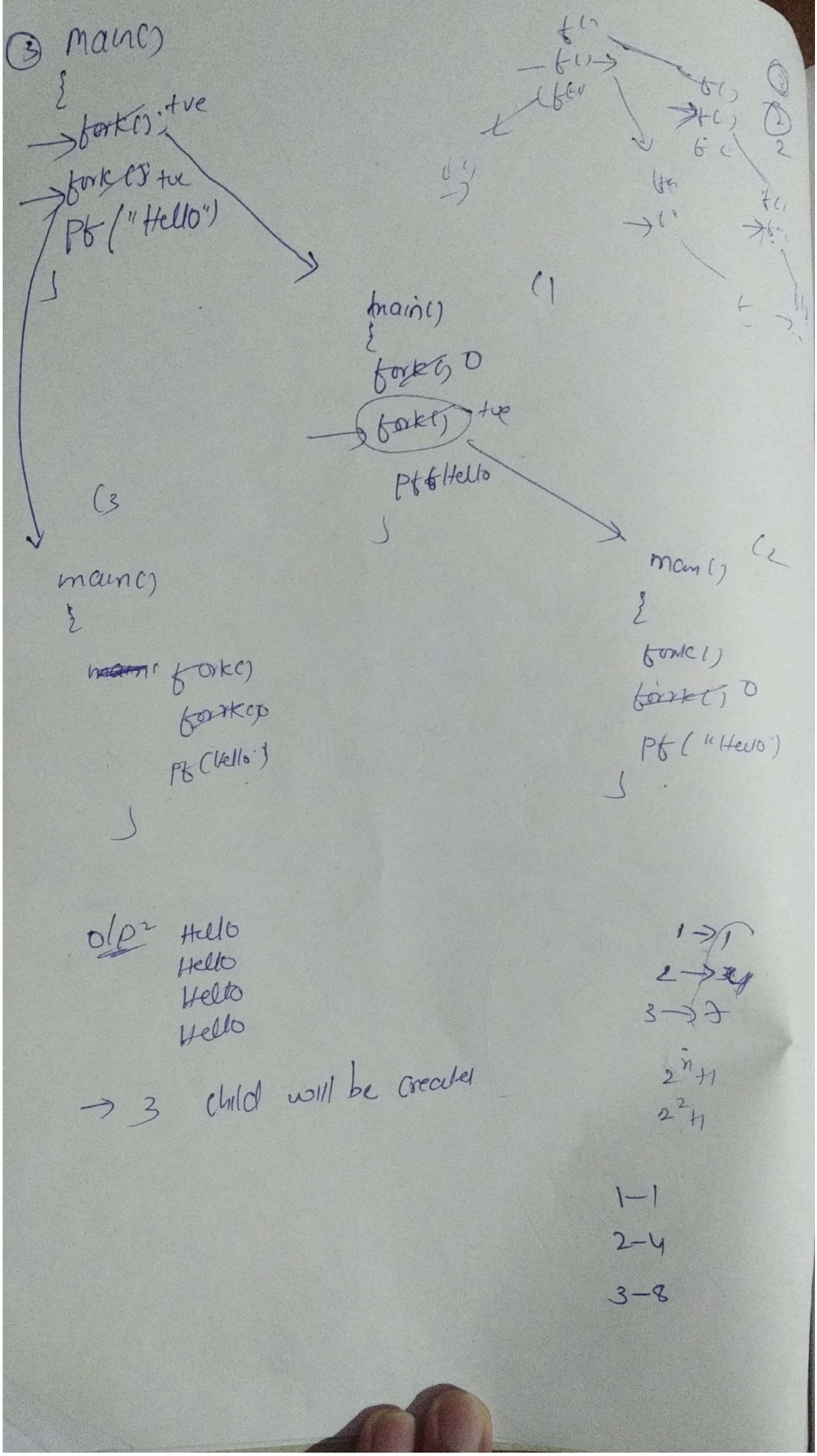
```
main()
{
   int id;

   id = fork();

   if (id==0)
   {
     // child
   }
   else
   {
     // parent
   }
}  → child/parent
```

---

ex-①

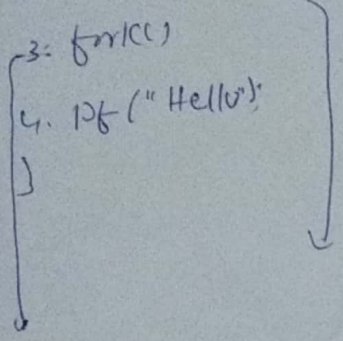
o/p: Hello

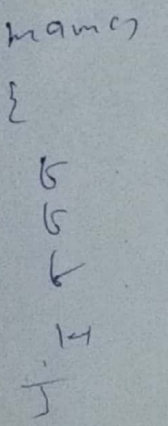


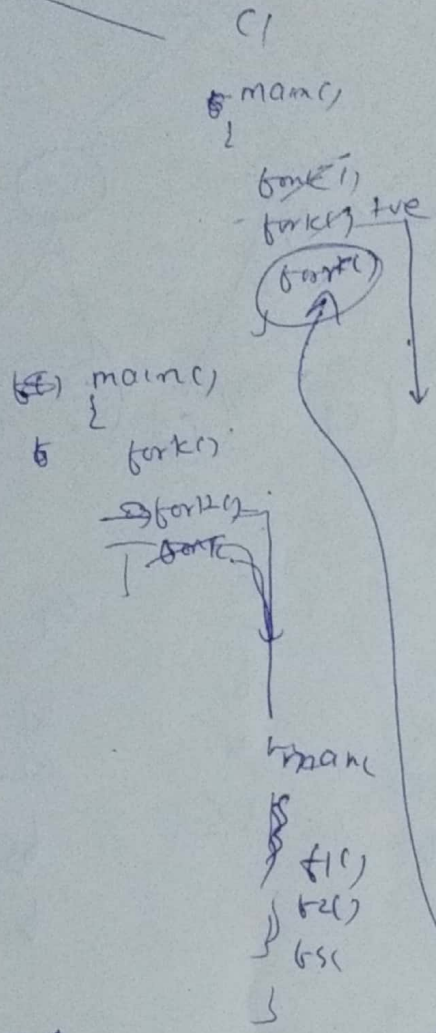


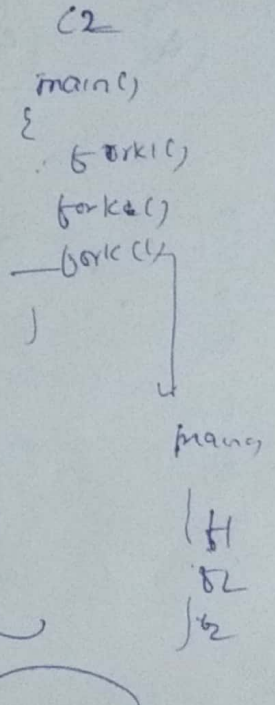
(4)

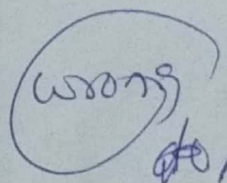
```
main()
{
1. fork();
2. fork();
3. fork()
4. Pf("Hello");
}
```











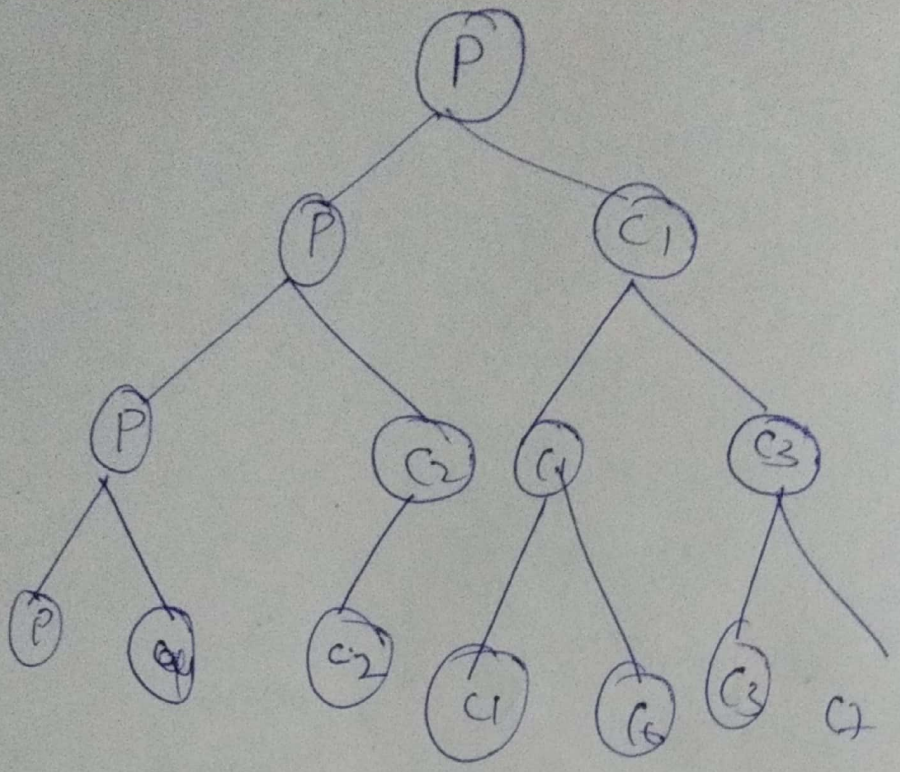
→ ~~Hello prints → 8~~

→ New childs → 6

→ Hello prints → (7)

wrong

not did



C1

b1
b2
b3
b4

C2

b1
b2
b3
P

C3

b1
b2
b3

C4

b1
b2
b3
b

C5

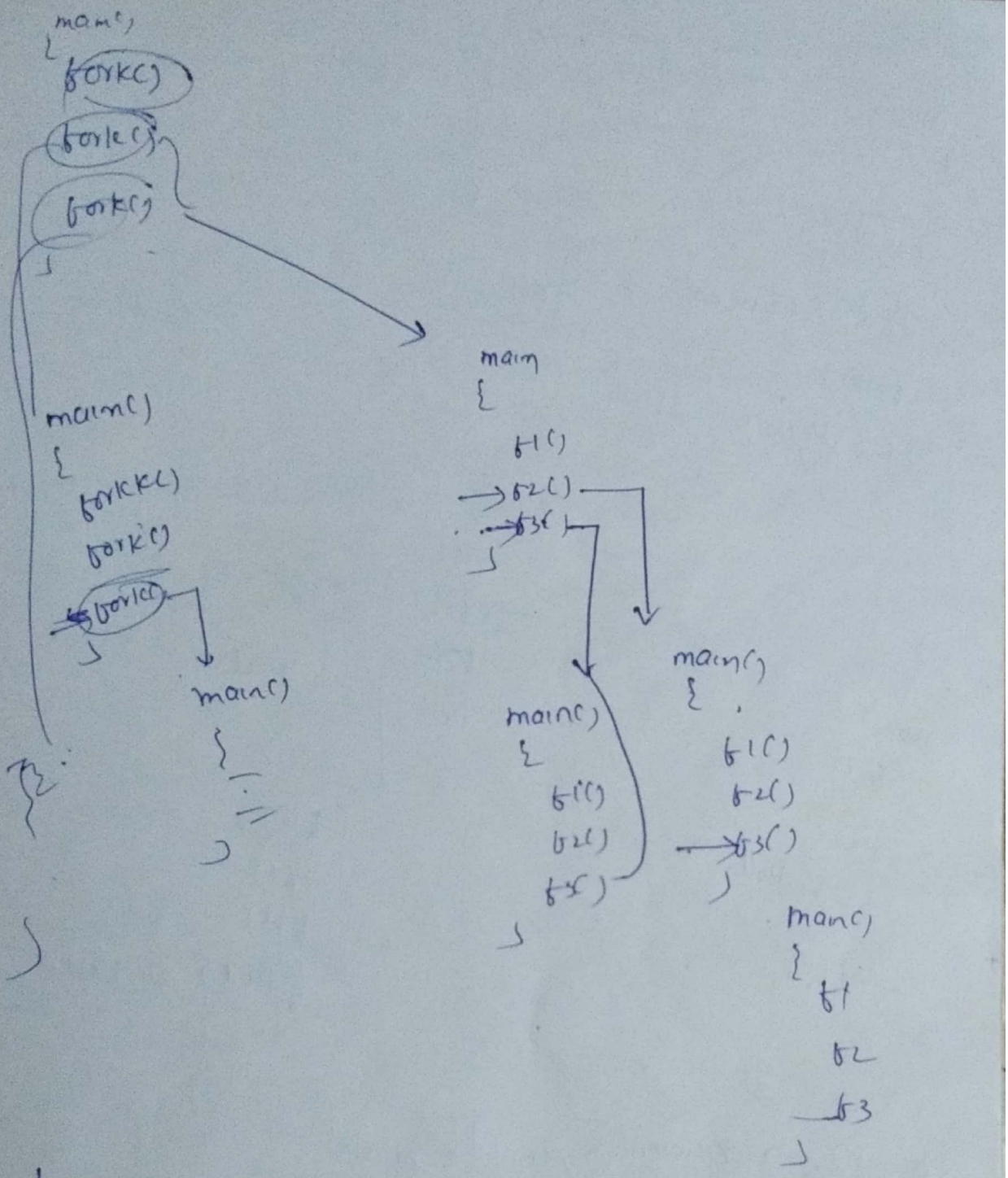
b1
b2
b3
P

C6

b1
b2
b3
b

C2

b1
b0
P



NOTE:

n → forks

new childs → $2^n - 1$

No of process (with parent) = 2^n

(or)

n2o

} Applicable when fork are in series

(5)

```
main()
{
    fork() <
    Pf("Between") ->
    fork()
    Pf("Hello")
}
```

Betwe

Hello

```
{
    f1()
    Pf()
    f2()
    Pf
}
```

Betwe
Hello

```
m
{
    f1()
    Pf
    f2()
    Pf(11)
}
```

Hello

```
f1()
Pf()
f2()
Pf()
```

Hello

a 2 → Between
4 → Hellos
> will be printed

2 n-

(6)

```
main()
{
  int i;
  for i ← 1 to n
      fork();
```

How many childs will be created

a) n² b) 2n c) 2^n - 1 d) 2^n

How many process will be created

a) 2n

→ New child = 5
print - 6 hrs

Homework → take || operator

(7)

```
main()
{
  int i;
  for i ← 1 to 6
  if(i%2==0)
      fork()
```

How many New child will be created

a) 1023 b) 30 c) 31 d) 32
$2^5 - 1 = 32 - 1$

(8)

```
main()
{
1. fork();
2. fork() & (fork())
3 Pb("Hello")
}
```

Process Management:

→ process Concepts
→ process states
→ proces schedulig
} = part-1

→ Interprocess communicaton
→ Synchronisaton Mechanism
⟹ process part-2

I. process Concepts

program v/s process

Program.

→ program is a passive entity it cannot perform any actun itself
→ program stored in main memory and secondary memory
→ program is a like which stored on disk and doesnot consumes any resource
→ It need to be translated into process
→ program is a like a class in oops concept

Process

* process is a dispatchable (or) schedule unit
* process is a unit of cpu allocation
* program under execution called process.
* When program is loaded into main memory that becomes process
* process is an active entity, it can perform a task specified in the program.
* process will consume some resources. ex main memory, cpu, files, I/O devices and cpu registers
* process always resides in main memory
* process code is more than program code. it includes: * text section
  * program counter (PC)

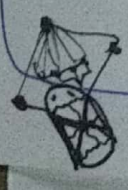
which specified current activity of process, also includes

* data section:

data section could hold temporary data includes function Parameters to be passed

process stack:

* Return address
* local variables which are stored in process stack.



It also includes
* Dynamic Memory allocator : (HEAP)

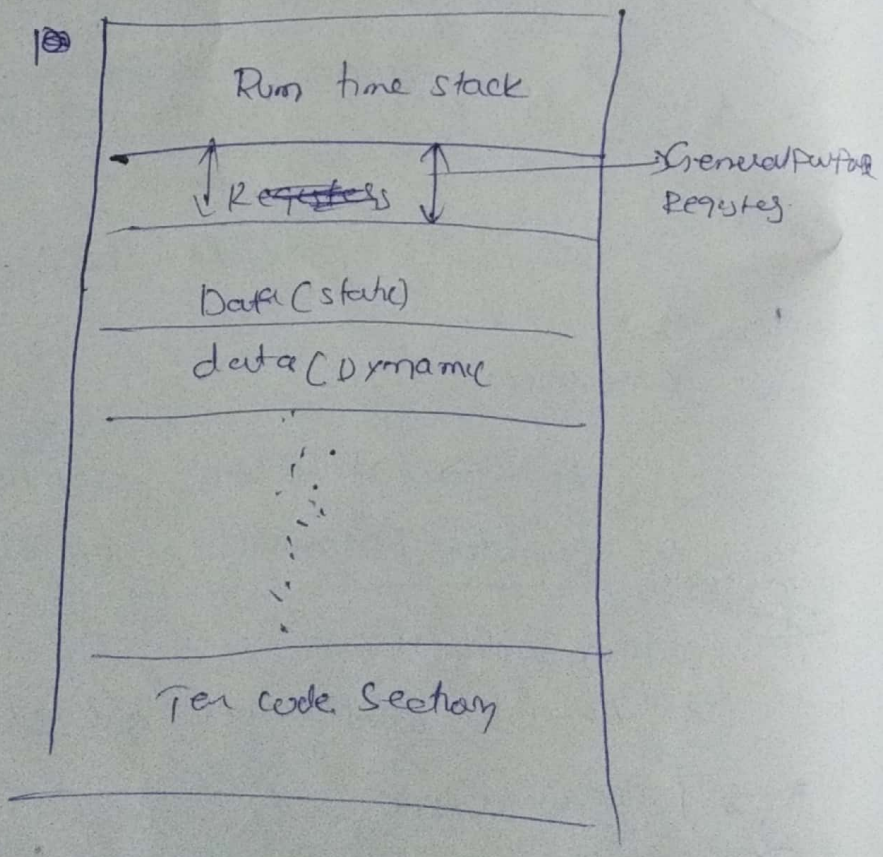
→ All Resarce allocator is done at load time
→ process also includes set of general purpose registers.

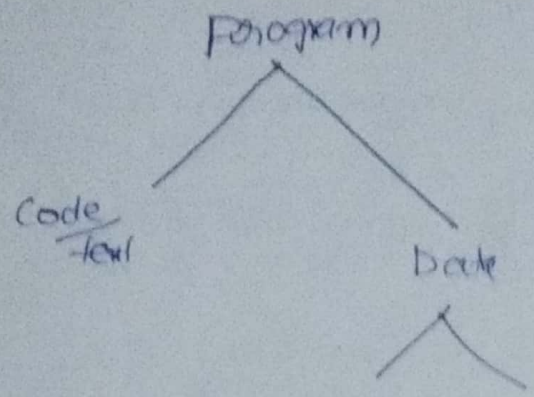
formal definition (or technical definition -

→ process is data structure (Developer point of view)
→ process ①
② Representation | structure | implementation
③ Operation
④ Attributes

Representation / structure

process Image





State

Size of variable
Data Structure
are known
load the

Dynamic

size of variable, Data
Structure are not known
at load the

Operations on process

(1) Create():

All resources are allocated at the time of creation of process

→ only partial resources allocated during execution dep remaining allocate

(2) Schedule():

(3) Run():

(4) Block() /wait()

(5) Suspend()

(6) Resume()

(7) Terminate()

Scheduler:

Take a process from ~~main memory~~ m.m to CPU for execution. it is also referred as

Dispatch
~~Run()~~
~~Dump~~ :

execution of program that is take address from program counter and get instruction from memory register

Block ():

Running process require I/O operation (or) an event to satisfy so that the process will move from Running state to Blocked state

suspend():

→ Suspend the Running process (or) ready process (or) blocked process, is to secondary memory bus for some time

Resume():

→ suspended process loaded into main memory

Termination

→ end of program.

→ All "Resource deallocation" takes place

Process Attributes

(1) Identification

1) Process ID
2) Process group id, 2) parent process id.

(2) CPU related Attributes

1) Program counter value
2) General purpose register (how
3) priority value
4) stack pointer value

(3) memory Related attribute.

→ "Size" of state memory allocation
→ MMU (memory managemed unit)
→ Limit registers (starting and ending address)
→ Values of TLB (Translation Look aside Buffers)
→

(4) file Related

list of open files

(5) Device Related

List of devices to be use

→ The above all attributes of a process are kept

in "PCB" [process control block]

PCB → Kernel stack data

I. Bharath

PCB:

→ It also reffered as Task control block

→ PCB is used to Represent a process in OS

→ PCB is a kernel datastructure

→ Every process has its own PCB

→ PCB contain many pieces of information about process

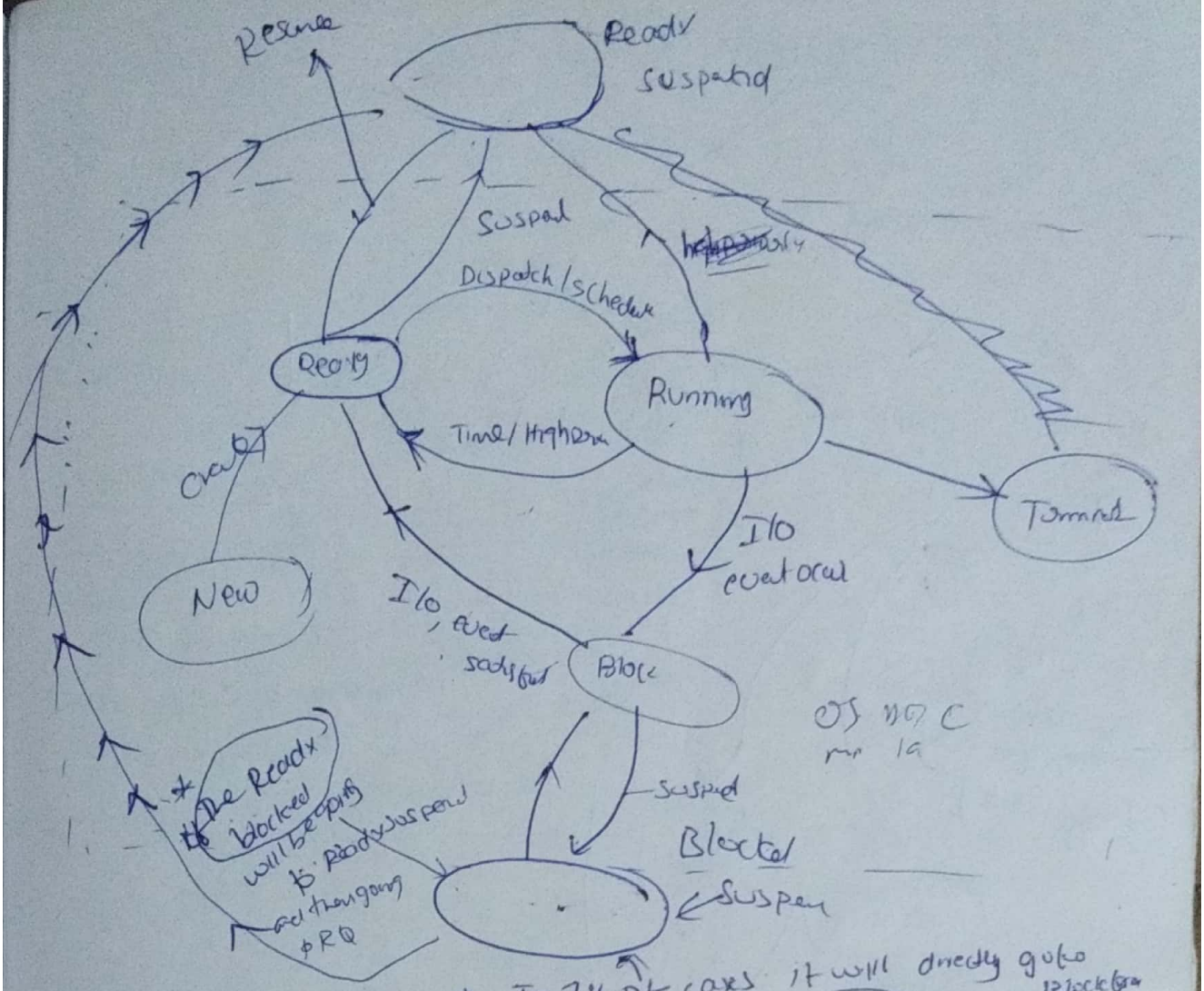
→ PCB is also refered "process object"

→ PCB destroyed after process is completed

Process states and state transistion diagram

* It depicts current activity of process The process states are

1) New, Ready, Running Block, Terminate



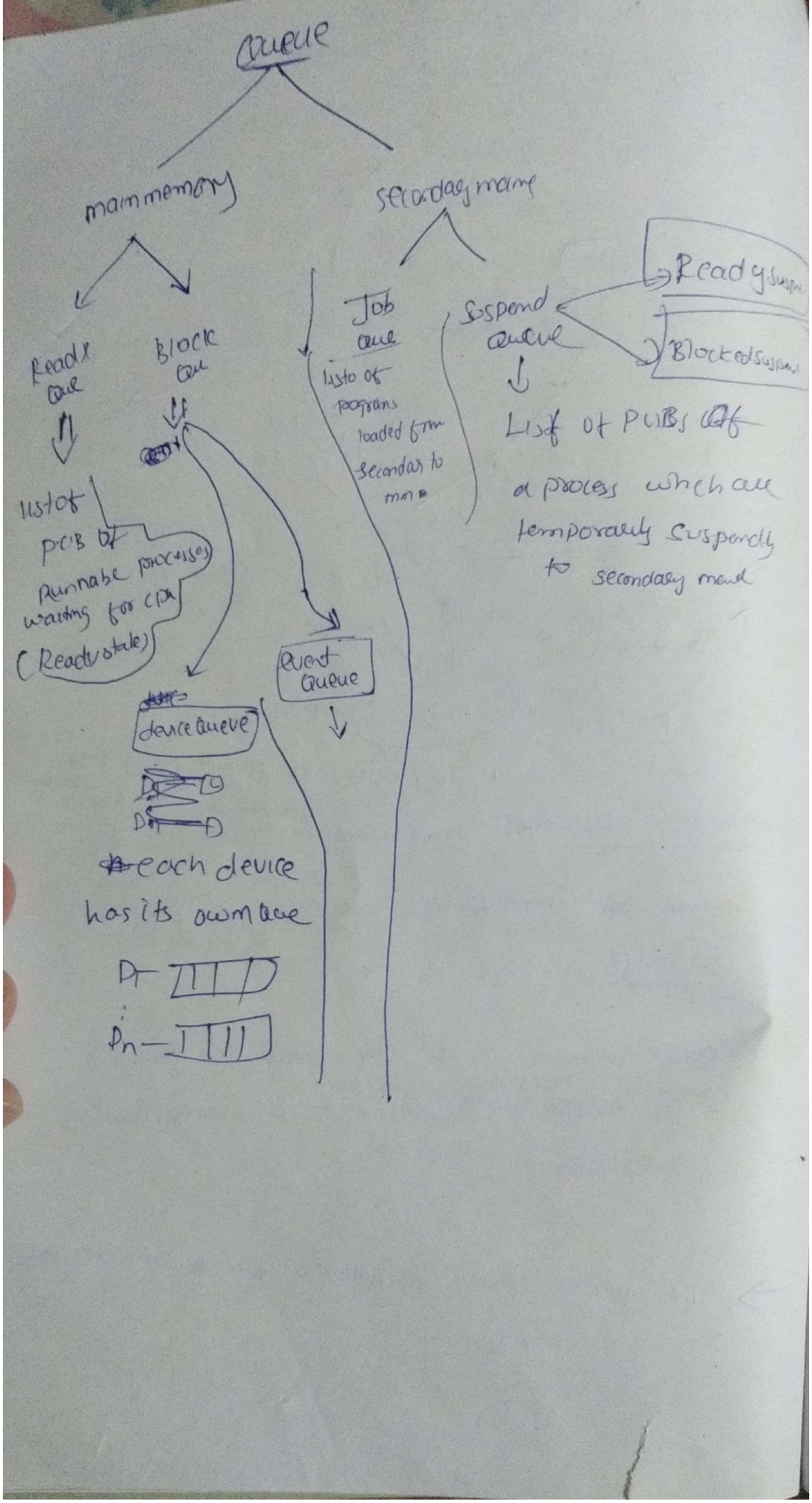
OS NOTE

* In 1% of cases it will directly goto Blocked or Ready/Block

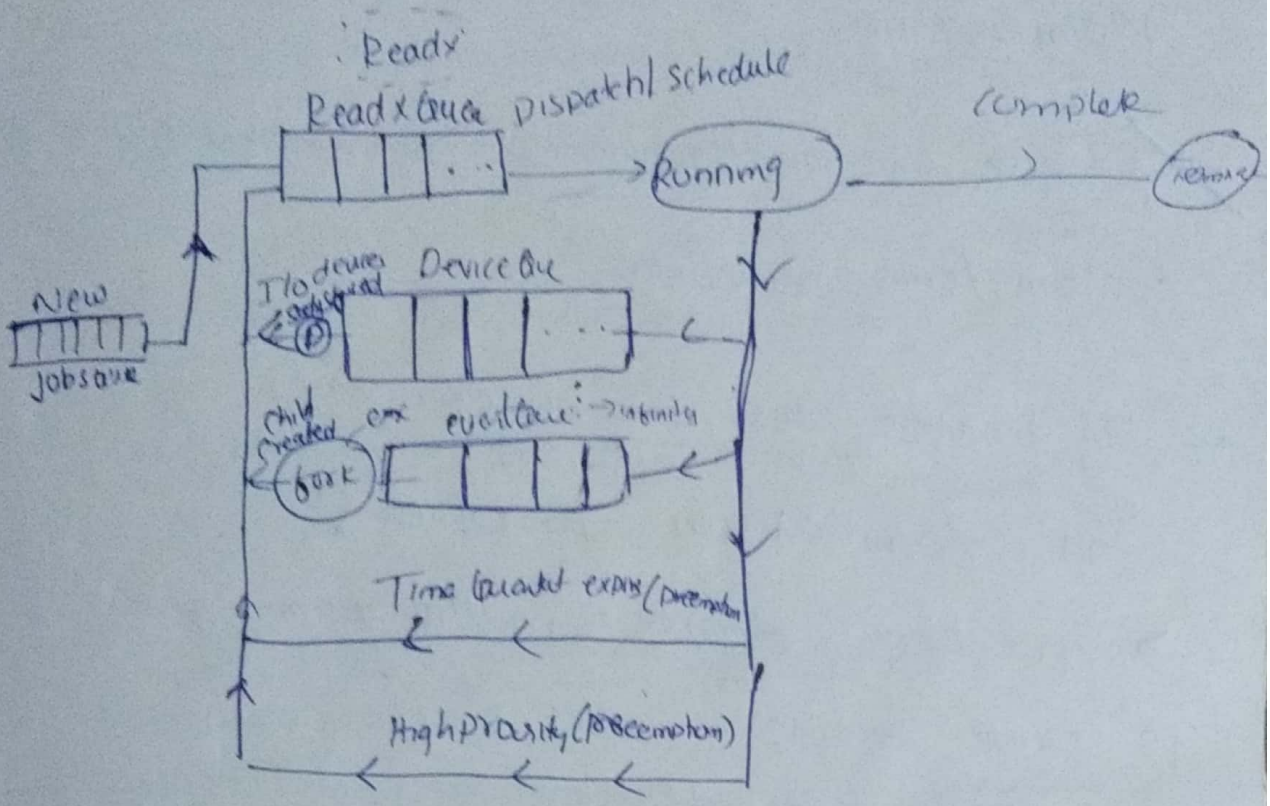
1) Blocked suspends ~~and Ready suspend~~ is secondary memory ~~of~~ where process are waiting for event to satisfy

2) Ready suspend is a secondary memory in which Process are waiting for Loading in to main memory for execution

→ The Block state contains ~~the~~ Don't know



Process Queues AND Queuing Diagram



scheduler and dispatcher:

Scheduler is responsible for selecting a processes from a queue and assign to resource

1) Long term scheduler
2) short term scheduler
3) medium term scheduler

long term scheduler

→ will load process from disk to Main memory.

→ Degree of multiprogramming is controlled by long term scheduler

→ It invokes less no of times in comparison to others

→ To keep degree of multiprogramming is stable average number process creation equal to average departure rate of processes

short term.

→ It selects process from among the Processes in Ready Que for allocation of the cpu.

→ Its frequency of execution is more in comparison to others

→ It invoked each time whenever a cpu resume new process for execution

It

Long term schedular need to select good mix of CPU bound and I/O bound process

to avoid low utilisation of CPU and I/O devices

→ CPU bound → The process which more utilises CPU I/O bound → The process which utilises I/O device

medium term schedular :-

→ It is used to suspend process from main memory to secondary memory

→ It decreases load of the CPU

Dispather :-

→ It is a component (or) module of the CPU schedular

→ schedular will dispather will do content switching

→ Content switching : (Between RQ and Running state)

→ process of switching a CPU from one process to another process i.e saving the P.C.B of old process and loading the PCB of a new process.

2 Types of conter switch

1) partical content switch :- only load operation is performed

ex Non-preemptive

full context switching:

* save and load

CST / CSL: Content scatch time / Contents switch latency

The amount time taken for content surce

is content switch time / conten switch Latency

→ CST depends on hardware, PCB size, operating system.

→ cst value vary from process to process

Process Scorfching :-

→ "Deleting" the P.C.B of old process and loading P.C.B of the new process is called processes switching.

* Process scheduling ~~Algorithm~~ *

→ When there are more than process in RQ, TO execute with the CPU a selection decision need to be made to pick a process among the ready process from the RQ for execution is called "Process scheduling" (or) "CPU scheduling"

* CPU scheduling is a fundamental function of the O.S i.e is every resource is scheduled before use.

* The objective of CPU scheduling :-

1) fairness
2) Maximize CPU utilization [how much is busy]
   ↳ only possible when RQ must have atleast one process
3) Maximize Throughput

Throughput :- It measures performance of system i.e is among of work per unit time.
(1 sec, millisec...)

4) minimize the TAT [Turnaround time] CT
5) minimize the WT [waiting time]
6) minimize the Response time

Turnaround time:

The time interval between completion and arrival time.

Waiting time:

processes waits in a Readyqueue for allocating of CPU

Response time:

→ The time at which first request is submitted to the system, to the time at which its response it generated

→ In a response system Response time should be less

Process Time for a process

(1) Arrival Time: The time at which process enter in to R.Q

(2) Submission time: The time at which process is Scheduled (or) Submitted to CPU for entering running

(3) Burst time: [service time] [CPU time]

The

amount of time process is spending in CPU

(or) I/O

2 Types → CPU Burst time
→ I/O Burst time

(5) Completion time:

The time at which process has completed its execution

(6) Waiting time: The amount of time process waiting in RQ

(7) schedule length

The total time required to complete total process

(8) Deadline time:

~~Notations~~: Notation time

If n- process of

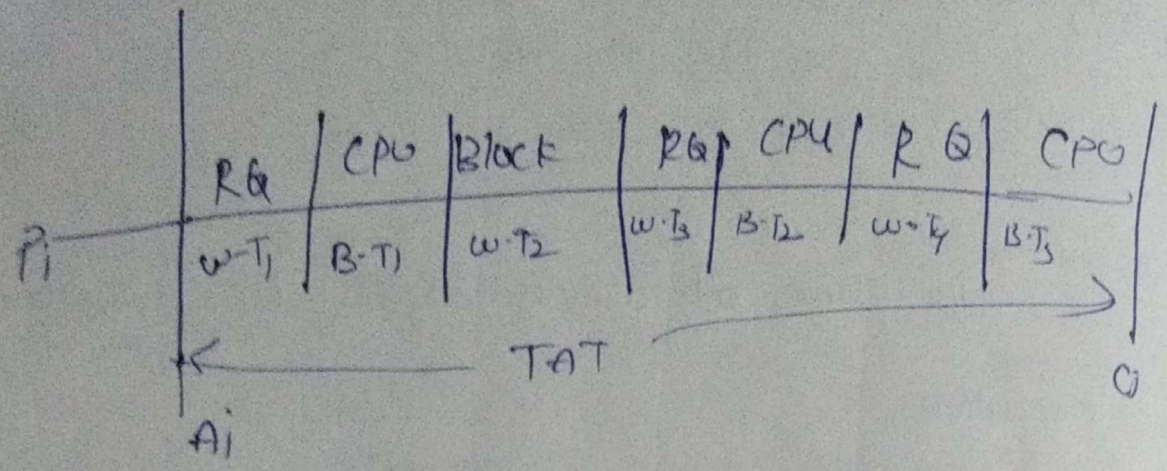
P_i - process

① AT (P_i) = A_i

② ST (P_i) = S_i

③ BT (P_i) = X_i

④ CT (P_i) = C_i



(5) $w.T(P_i) = w_i = TAT_i - x_i$

(6) $TAT(P_i) = TAT_i \; (C_i - A_i)$

(7) Avg w.t = $\frac{\sum_{n=} TAT_i - x_i}{n}$

(8) avg TAT = $\frac{1}{n} \sum_{i=1}^{n} C_i - A_i$

(9) Schedule length (L).

Let n ⇒ process

No of ways schedule = $n!$

If n=3

P1	P1	P2	P2	P3	P3
P2	P3	P1	P3	P2	P1
P3	P2	P3	P1	P1	P2

— 3! = 6 way

$L = Max(C_i) - min(A_i)$

* Throughput (μ)

for n-process

$$\mu = \frac{n}{L}$$

(2) Consider n process, m CPU, where (n > m)

find out lower bound and upper bound of processes which are present in Ready, running and block state.

Sol In the above question asked values are possibility

	Lowerbound	upperbound
Ready	0	n
Running	0	m
Block	1	n

Algorithms

(1) FCFS [first come first serve]

Criteria: Based on the A.T of process

mode: Non preemption

Gantchart:

CPU | Process |

→ Time

PId	AT	BT	CT	TAT	WT	ST
1	0	4	4	4	0	0
2	1	3	7	6	3	4
3	2	2	9	7	5	7
4	3	5	14	11	6	9
5	4	3	17	13	10	14
			51/5	41/5	24/5	34/5

CT-AT

TAT-BT

Submission

P	P1	P2	P3	P4	P5
0	4	7	9	14	17

(1) Schedule length = 17

(2) avg WT = $24/5 = 4.8$

(3) avg TAT = $41/5 = 8.2$

(4) % (Through put) = $5/17 \times 100\% =$

(5) % of idleness = 0

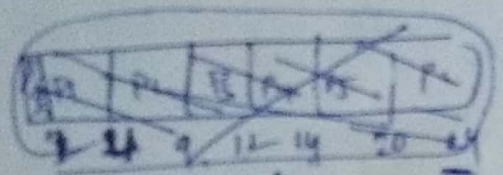
(6) % CPU utilization = 100%

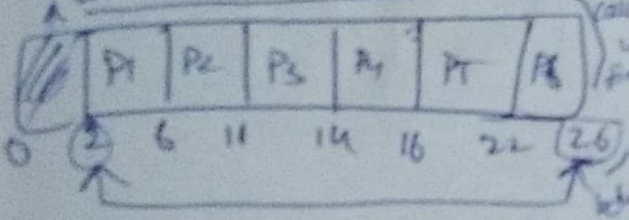
↓ formula

(100% − % of idleness)

8)

Pid	AT	BT	CT	TAT	WT
1	2	4	4	2	
2	4	5	9	5	
3	6	3	12	6	
4	8	2	14	6	
5	10	6	20	10	
6	12	4	26	12	





CT	TAT	WT	ST
6	4	0	2
11	7	2	6
14	8	5	11
16	8	6	14
22	12	6	16
26	14	10	22

→ L = 26-2 = 24

→ CT = 15.6

→ TAT = 9.8

→ WT = 4.8

→

* Important Points *

① avg. wt and avg TAT are not minimal [16 not minimal performance reduced]

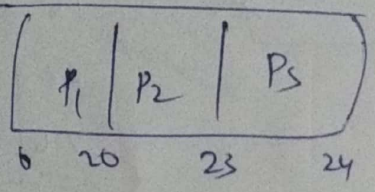
② In Dynamic situation, it suffers from "convoy effect" problem

Convoy effect: All other process are eagerly waiting for one big process to get off from the CPU

③ If A.T of process are same, PID is used to break the tie

Pid	A.T	B.T
1	0	20
2	0	3
3	0	1

CPU



avg WT = $\frac{43}{4}$ = $\frac{0+20+23}{3}$

If not use P1, P2, P3

avg WT = 5$\frac{1}{3}$

process scheduling (CPU scheduling)

② Shortest Job first (SJF)

(or)

Shortest process next

Criteria: Based on the Burst time of a process

Mode: Non preemptive

Concept: ① Allocate the CPU to small Burst Time

② If Two process have same CPU Burst time. Then FCFS is used to break Tie

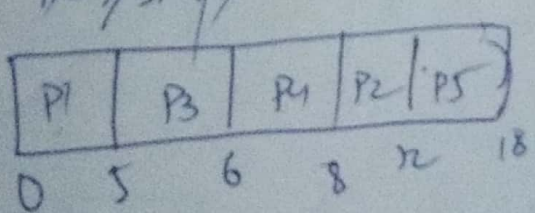
eg:-

PID	AT	BT
1	0	5
2	1	4
3	2	1
4	3	2
5	4	6

NOTE:

* if process submission time and arrival time is same waiting time = 0

P2, P3, P4, P5

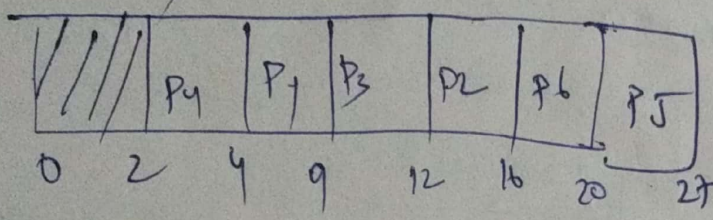


ST	TAT	CT	WT	TT
0		5		
5		12		
6		6		
8		8		
12		18		

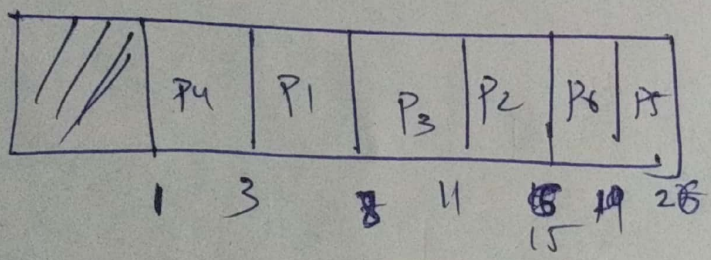
PId	AT	BT
1	2	5 ✗
2	5	4
3	8	3
4	1	2 ✗
5	5	7
6	6	4

→ look on arrival time

P4, P4, P2, P3, P5, P6



(X) because at P4 second P4 come



schedule length = 26 - 1 = 25

ST	CT	TAT

(3) Shortest Remaining Time first (SRTF)

* Criteria: Based on the B.T of a process
* Mode: Preemptive
* Concept:

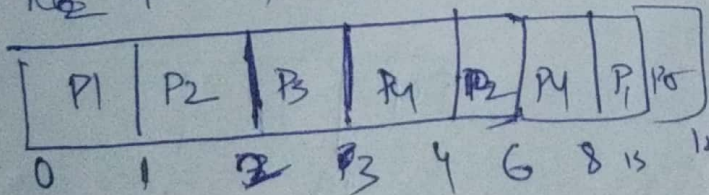
preempt Current executing process when new arrival process CPU burst time is shorter than current running process. Preempted process is placed in Ready Que.

Ex1

Pid	AT	BT	CT	TAT	WT
1	0	5	14	14	8
2	1	4	6	5	1
3	2	1	3	1	0
4	3	3	9	6	3
5	4	5	19	15	10

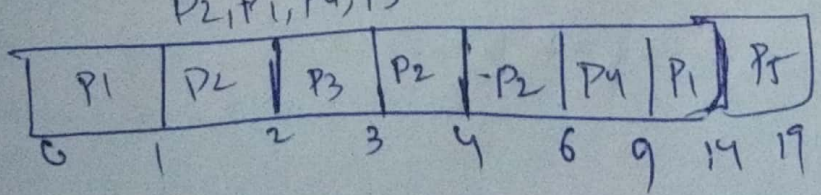
P3 P4, P5

P1, P2,



calculation wrong

P2, P1, P4, P5



Schedule length = 19-0=19

Submission time will vary. don't take

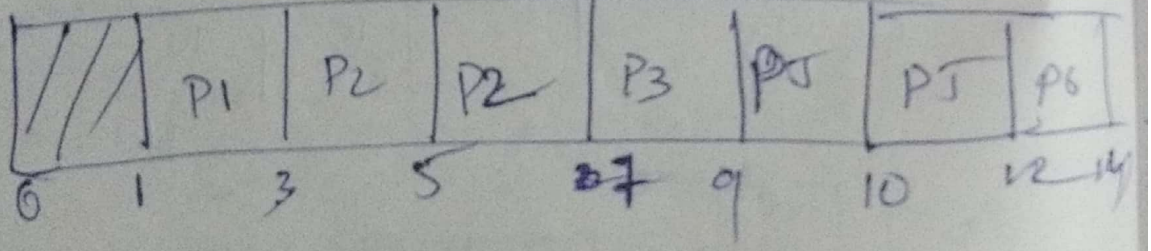
Submission time

PId	AT	BT	CT	TAT	WT
1	1	6	25	24	16
2	3	4	7	4	0
3	5	2	9	4	2
4	7	5	19	12	7
5	9	3	12	3	0
6	10	2	14	4	2

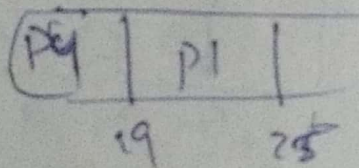
4

27/6 = 4.5

P1, P2, P3, P4, P5, P6



P1, P2



L = 25-1=24

(4) Priority Based CPU scheduling Algorithm

Criteria: Based on Priority

Performance of sjf (Sjf / SRTf) — difference in preemptive (or) non preemptive

→ SRTf have more context switching than Sjf

Advantages

(1) It favors to short jobs
(2) It is also referred as optimal CPU scheduling
(3) It generate Good throughput
(4) It minimum (1) TAT
(2) W.T

Disadvantages

(1) Longer job are starvated
(2) It's not suitable to implent practical cases since predicting CPU Burst time is not possible Hence it is not suitable for shortterm scheduler
(3) It can be implementable in long term sch
→ scheduler by predicting burstime.

→ predicted burstime is depends on (i) process size
(ii) process type ... so on

NOTE: SJF shortterm scheduler is a benchmark tool for measuring performance of CPU scheduling algorithms

→ we have starvation problem in SJF

(4) Priority Based CPU scheduling Algorithm

Criteria:-
Based on the priority of a process

priority is a integer value, assigned to each process when it is loaded into RQ

mode:- Non-preemption / preemption

Concept.

(1) process are kept in the RQ on the order of priority

(2) Highest priority process of prority is selected for exe

(3) If two or more process have having same priority then FCFS to break tie.

* RQ is implemented priority queue (Doubly)

Case ①

→ Priority value will not be changed until its execution.

→ Both Preemptive and Non-preemptive can be implanted

Case ①

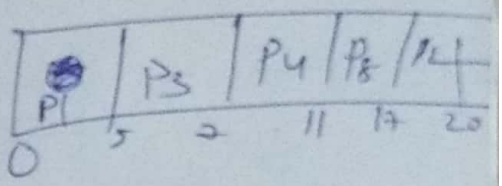
Non-Preemptive

Highest no — high priority

Priority	PId	AT	BT
10	1	0	5
5	2	1	3
15	3	2	2
7	4	3	4
6	5	4	6

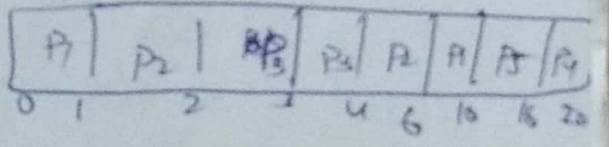
Priority Que

P2	P5	P4	P3



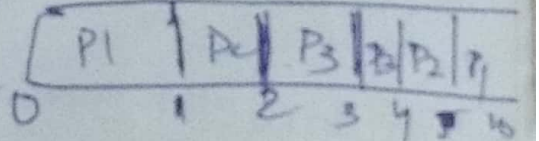
Preemptive:

Least no → High priority



priority	PId	AT	BT
5	1	0	4
2	2	1	3
0	3	2	1
10	4	3	4
9	5	4	6

P4 P1



P5 | P4 |

16 20

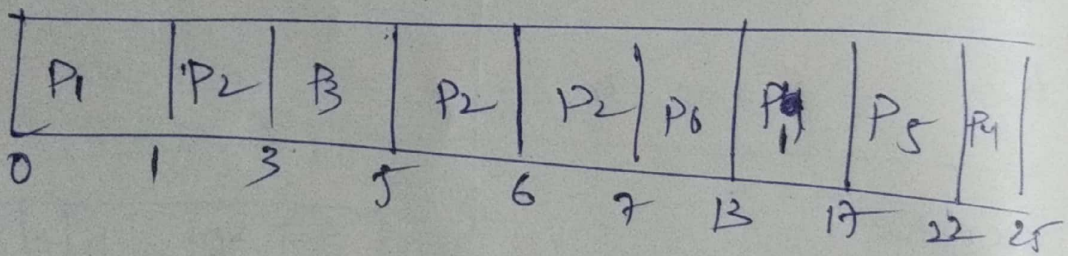
Ex①

Preemptive

~~Highest no~~ → Highest no → high priority

Priority	Pid	AT	BT
5	1	0	~~8~~ 4
10	2	1	4 ~~2 1~~ 0
15	3	3	~~2~~ 0
3	4	5	3
4	5	6	~~8~~ 0
8	6	7	~~6~~ 0 0

P1, P4, P5, P6



~~schedul~~ Schedule length = 25

NOTE:

① Once, the priority is assigned, it will not change during its execution

② Lower priority process may Starvate (or) infinite block

③ To avoid above problem, aging mechanism is used

④ Aging mechanism: (or) Algorithm

→ These mechanism (or) Algorithm is used to ↑ increase the priority of low priority process to get the chance for execution. (Gradually)

* Round Robin: CPU scheduling Algorithm:

→ M.P | M.T | Multiuser O.S

criteria: Based on the A.T + Time Quantum

mode: preemption

Concept:

① processes are kept in RQ in the order of "FIFO"

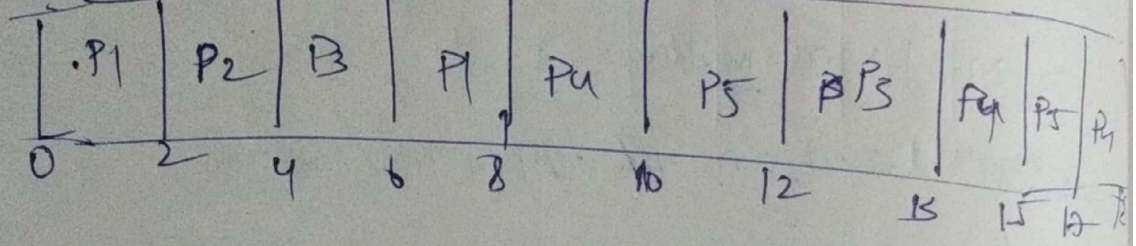
② processes have to share the CPU with time interval (TQ/TS) ↓ Time slice. ↓ Time quantum

NOTE:

1) It also refered as preemptive version of FCFS

pid	AT	BT	TQ=2
1	0	4 2 ~~1~~	
2	1	2 ~~1~~	
3	2	3 ~~2~~ ~~1~~	
4	3	~~8~~ 8 1	
5	4	4 2	

P1, P2, P3, P1, P4, P3, P4, P5



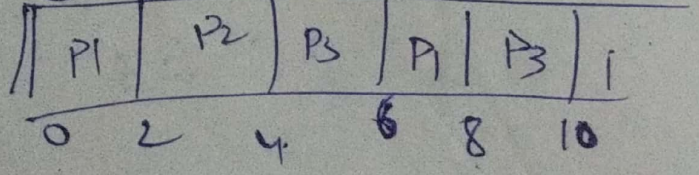
· schedule Length = 18 - 0 = 18

b)

PID	AT	BT	TQ=2
1	0	~~8~~ ~~6~~ 4	
2	2	~~6~~ 4	
3	4	4 2	
4	8	3	
5	10	5	

→ After completing P1, first we P2 and then P1 & all process

P1, P2, P1, P3, P2, P5, P1, P3



TQ=2

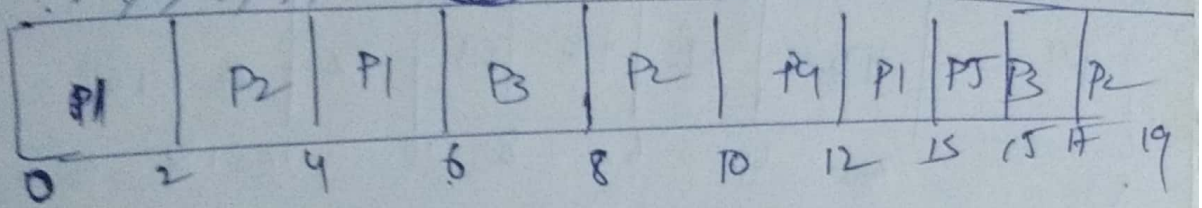
Pid	AT	BT	CT	TAT	WT
1	0	8 6 4	13	13	8
2	2	6 4 2	19	17	11
3	4	4 2 0	17	13	9
4	6	3 1	20	14	11
5	8	3 1	23	15	10

9.8

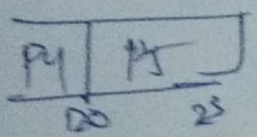
8

P1

P1, P2, P1, P3, P2, P4, P1, P5, P3, P2, P4, P5



P

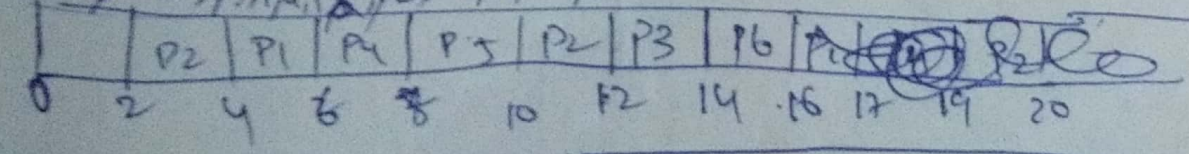


schedule length = 23-0 = 23

Pid	AT	BT
1	0	4 2 X
2	1	8 3 1
3	5	3 1
4	3	7 3
5	4	2 0 X
6	6	6 4

TQ=2

P1 P2 P1 P4 P5 P2 P3 P6 P2



Performance of Round Robin

Round Robin depends on the TQ/TS

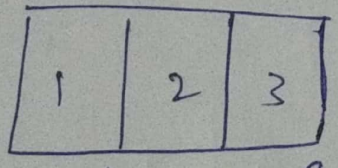
- If TQ is small
  - → good processor sharing
  - → But how CPU utilization
- if small
  - → more Context switches
  - → more interaction
- Large
  - → Less sharing
  - → Less interaction
- Very large
  - → It degerals to FCFS
  - In case it will act as FCFS

eg

Pid	AT	BT
1	0	4
2	0	2
3	0	3

case-①

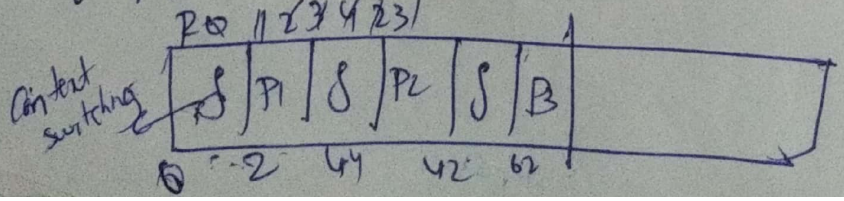
CPU



Very large

case ② TQ=0.1, S=2

to FCFS



$= \frac{03}{63} \times 100\%$

$= \frac{6.3}{63} \times 100\%$

$\cong 4.7\%$ → so only 4.7% of CPU utilization is done when TQ = 0.1 S = 2 and remaining all used for context switching

S = 2 is the context switching

* under round robin scheduling algorithm average waiting time and average "turn around time" are not minimal."

* RoundRobin is most suitable for the Timesharing os

NOTE:-

→ The most optimal Algorithm in all algorithm

→ SJF

next → RoundRobin

→ TLB is optimal but not much Reallue

Interprocesses Communication & Synchronization

Intra → within
Inter → between

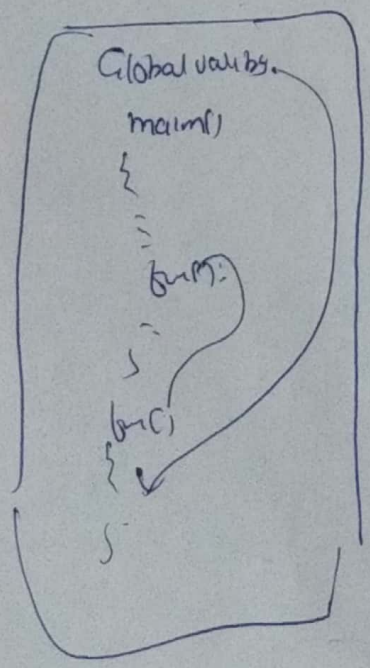
Purpose:

1. Data transfer
2. information sharing
3. Resource sharing.

Intra process v/s Inter process

- message passing
- shared memory

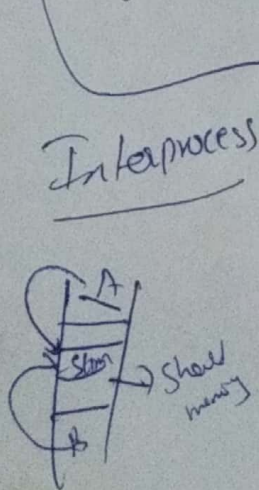
within process

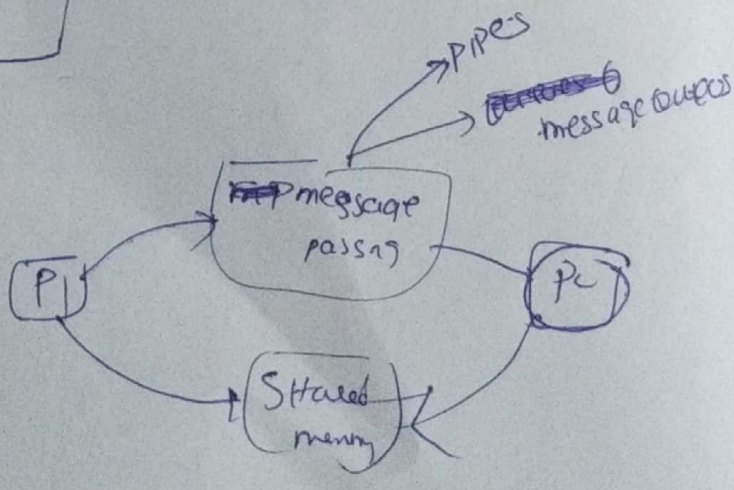


- Intraprocess achieved in a program through
  1. Existence of Global variables
  2. By passing the parameters

P_1 and P_2 must be synchronised

Interprocess





Lack of synchronization:

1. misunderstanding, wrong results → Inconsistency
2. Loss of Data
3. Deadlock.
4. undesirable problem of operating system is deadlock

There are 2 types of synchronisation of process

(i) Cooperative synchronization type
(ii) Competing synchronization

→ Two (or) more process are said to be in cooperative synchronization if and only they effected They get effected each other

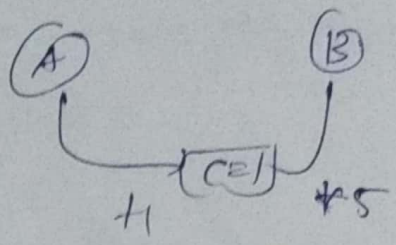
- Ex: producer - consumer problem

Competing synch:

Two (or) process are said to be under competing synch if and only if they contend to access a shared resource simultaneously

Inconsistency Problem:

Let 2 process (A) and (B) sharing a common variable 'C' . initial value of C. process want to increment value of C by 1 and process B wants multiply value of C by (*5)



A: C = C+1

B: C = C*5;

RQ A B

① A/B → C = 10

② B/A → C = 6

ex

ProcessA

I Load R1, M[C]

II INC R1, #1;

III store M[C], R;

Process B

I. Load R2, m[C]

II. MUL R2, #5

III store M[C], R2;

The process scheduling can be done in many way

A, B

& A

multi-layer

RQ. PA, PB

The expired exam

to: PA	I, II ↓ preemp
t1: PB	I, II, III $R_2 = K_5$
t8 = PA	III

A/B = C = 2 ≠ to

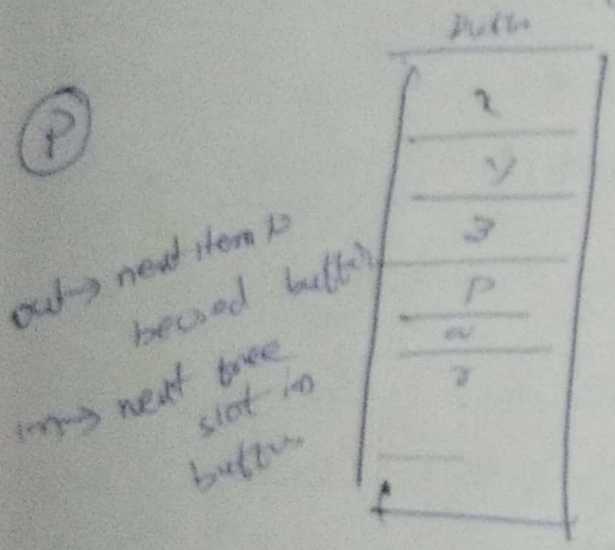
RR P1, P2

b	P2	I II ↓ preempted
b1	P1	I, II, III
t	P2	III

(1-5) Inconsistent

producer consumer problem (or) Bounded Buffer problem

Count 0 1 2 3 4



Inconsistent state

out-4 t2 Inconsistent state

#define Buffer [w]-100

int count = 0

→ Consumer will get affected when buffer empty

→ producer will get affected when buffer full

Producer code

void producer (void)
{

int in=0 , itemp;

while(1)
{

1. produce item (item); → if condition false
2. while (count==N) // Busy wait //
3. Buffer [in]= itemp;
4. in = (in+1) % N;
5. count = count+1;
}

I. Load R_i m[count]
II. INC R_i, #1
III. store M[count], R_i

Consumer.

```
void consumer(void)
{
    int out=0, itemc;
    while(1)
    {
    1. while (count==0); // Busy wait
    2. itemc= Buffer [out]
    3. out= (out+1)% N;
    4. count = count-1;
```

4. count = count-1; → I. Load R_j m[count]
II. Dec R_j, #1
III. Store m[count], R_i

int in=0 , itemp;

while(1)
{

1. produce_item (item); → if condition false
2. while (count==N) //Busy wait //

3. Buffer[in] = itemp;

4. in = (in+1) % N;

5. count = count+1;
}

I. Load R_i m[count]

II. INC R_1, #1

III. store M[count], R_1

Consumer.

void consumer(void)
{

int out=0, itemc;
while(1)
{

1. while (count==0); // Busy wait
2. itemc = Buffer[out]
3. out = (out+1)/N;
4. count = count-1; → I. Load R_j m[count]
II. Dec. R_j, #1
III. Store m[count], R_j

RG [P.C]

t_0 : P	1,2,3,4,5 (I, II, IV print)	$R_2 = 4$
t_5 : C	1,2,3,4 (I, II, III)	
t_{10} : P	5, (III)	

Result = 4 ≠ 5 wrong (counter)

Necessary and Occurence of synchronisation: Problems

* presence of critical section (or) critical region (due to shared variable)

Critical section (or) critical region :

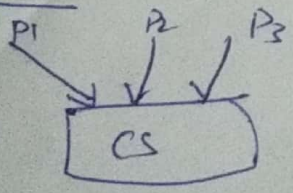
→ Critical section is a portion of programm code where shared resources are accessed

→ Non-critical section : is the portion of the program code where doesnot access any shared resource.

A program may consist critical and noncritical section

* Race condition :

2 (or) more process are racing to access the critical section simultaneously · concurently



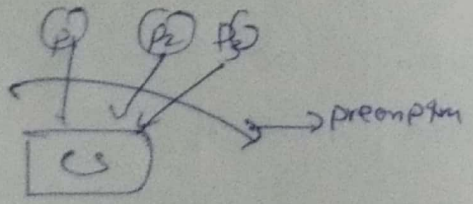
→ The value of the Critical section depends on the final process which executed

→ These situation where several process access and manipulate the shared data simultneously concurrently.

→ The final value of shared data depends on the process which executed (or) finishes Last

* Preemption Applied:

process allowed to preempt from the holding of critical critical section



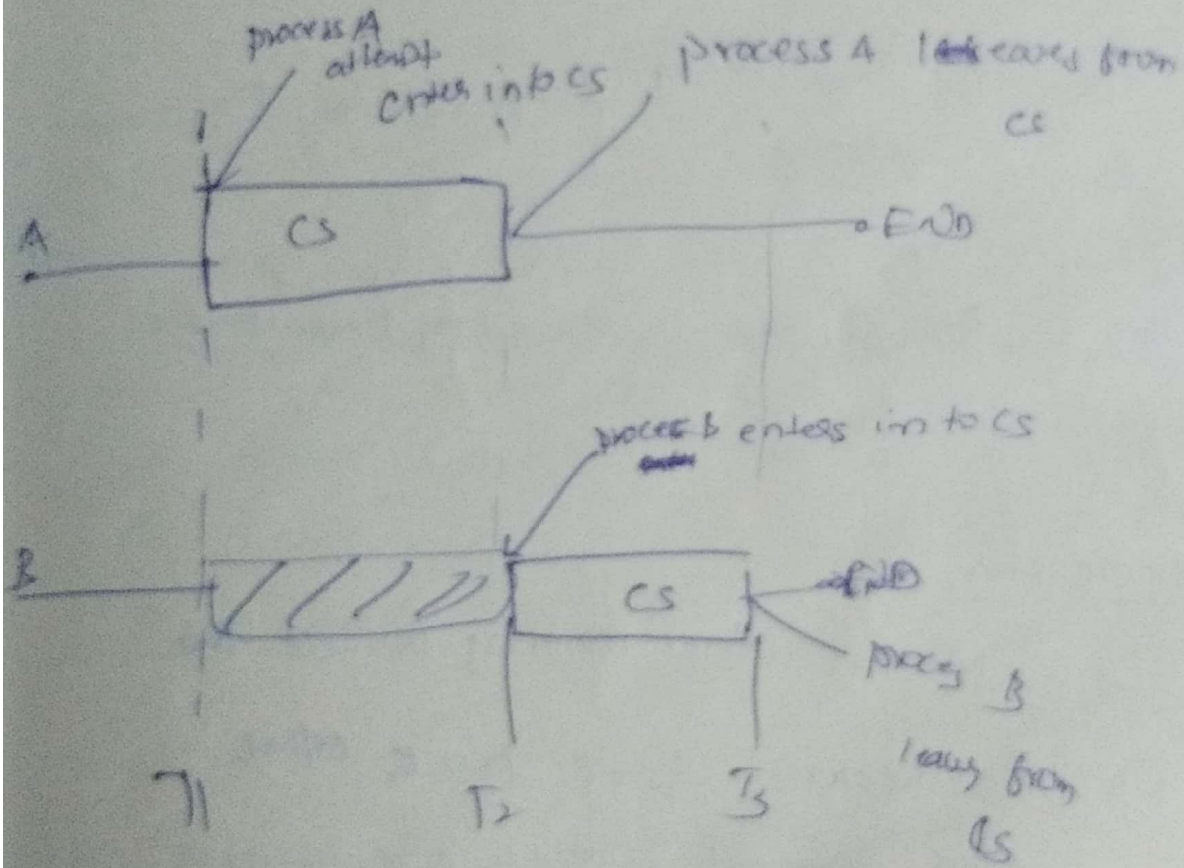
If The Above ALL 3 conditions occured simultneously from It leads to ~~Data Loss~~ "Data Inconsistency"

Mutual Exclusion:

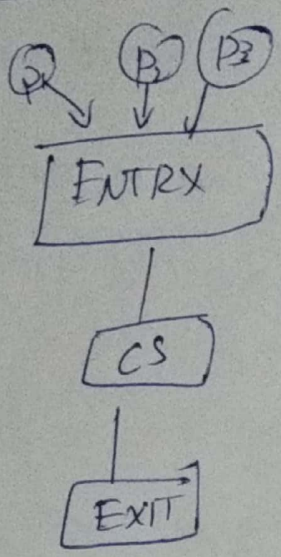
Principles: No 2 process present in the critical section at same time ie only one process is allowed to access critical section same time

Concurrent: first one P1 will execute next another P2 will execute and P1 will be at wait

Example:



Architecture for Synchronisation



Requirements for Synchronisation mechanism

(1) Mutual exclusion
(2) progression } mandatory Requirement

(3) Bounded wait } secondary Requirement

progression:

No processes running outside the critical section should block other ~~in rested~~ interested process for entering in to critical section whenever critical section is free.

Bonded wait

No process have to wait forever (infinity) to access the critical section i.e It should be ~~bloc~~ hold finite

Synchronization mechanisms :-

(1) Mutual exclusion with Busy Wait mechanism

* Disable interrupt
* Lock variable
* Strict Alternation
* Peterson's solutions (Dekkers Alg)
* TSL Instructions (Test and set Lock)

(2) Mutual exclusion without busy wait mechanism

(a) Blocking wait :

(means keeping processes in secondary memory)

* sleep() & wakeup()
* semaphore
* monitors

Disable Interrupts:

→ It is a simple mechanism to implement i.e disable all interrupt just after enters into critical section and Re-enable them just before leaving from it. (Critical section)

Semaphore

→ It is mutual exclusion with non-busywait mechanism [Blocking]

→ It is multi-process solution at user mode

→ It uses e-commerce, Banking Applications

→ Semaphore is a variable (of semaphore type itself with property that it takes an integral values)

→ Based on the integral value

semaphore divided in

① Counting semaphore [$-\infty, +\infty$]

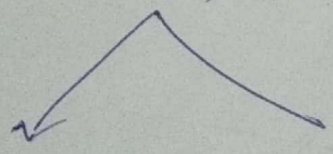
② Binary semaphore only (0,1)

→ semaphore is a operating system resource

→ "Dijstra" was given 2 semaphore operations (mathamatician)

which are implemented in kernel mode

→ Dijstra operations on semaphore operations are



Down (or) P (or) wait

UP (or) V (or) signal

→ These operations will be called in the kernel

Counting Semaphore: (−∞, +∞)

```
Struct SEMAPHORE
{
    int value; // current value of semaphore
    Queuetype L; // List of PCBs of those process who got blocked while performing "down" operation unsuccessfully
}
```

data

```
SEMAPHORE S;
S.value = 5;
Down <S>;
<CS>;
UP<S>
```

using

unsuccessful: After performing series of down operation (or) single down operation if we get -ve value

Succeful: After performing down operation we get +ve value.

```
Down(Semaphore S)
{
1. S.value = S.value - 1
2. if(S.value < 0)
   {
      put this process into
block  S.L()  & block it (put in seonday mem)
queue  (sleep();)
   }
}
```

Semaphore :

s value :

① P(S) → = down

s = 2 → success

② P(S)

s = 1 → success

③ P(S)

s = 0 → success

④ P(S)

s = 0 − 1 ⇒ unsuccessful

⑤ P(S)

s = 1 − 2 ⇒ unsuccessful

⑥ P(S)

s = 1 − 3 ⇒ unsuccessful

for down operation

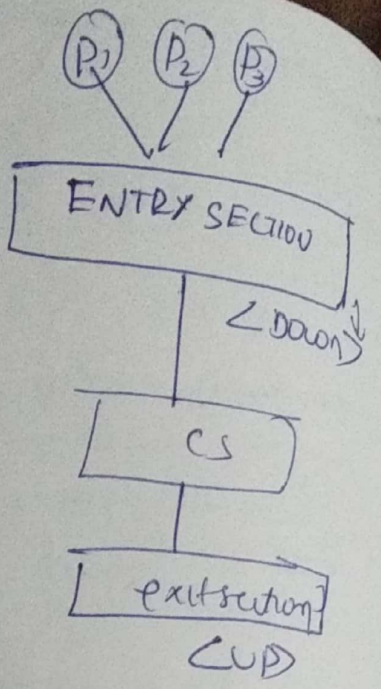
---

entry S

Back

UP = UP+1

– S=1



1. P (S)

S = 1 → 0 Successful

2. P(S) ; ← P2

S = 0 → −1 → unsuccessful

3. P3 → P(S)

S = −1 → −2 = unsuc

S = −2

P1: V(S) wakeup (P2)

S = −2 → −1

P2; V(S);

S = −1 → 0 wake(P3)

P3: V(S)

(= 0

UP definition

```
UP (Semaphore S)
{
  1  S.value = S.value + 1;
  2. if (S.value ≤ 0)
       { select a process from sleep() &
       } wakeup()
}
```

→ After performing series of down operation. If the value of count in semaphore(s) is "-ve" denotes those many blocked processes

→ The '+ve' value of counting semaphore denotes denotes those many successfull down operation will be carried out.

ex:

Consider a certain application in which the following operations are carried out

* 4P, 2V, 10P, 6V, 12P, & 1V operation

consider initial value of counting semaphore s=1. find out the no of block proces

Sol: s=1

4P

s=1, 0, -1, -2, -3

2V=

-3, -2, -1

10P: -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11

6V= -11, -10, -9, -8, -7, -6, -5

12P= -5-12 = -17

1V= -17 = 16

& final value of s = -16

No of process blocked = -16

I Binary Semaphore :- (0,1)

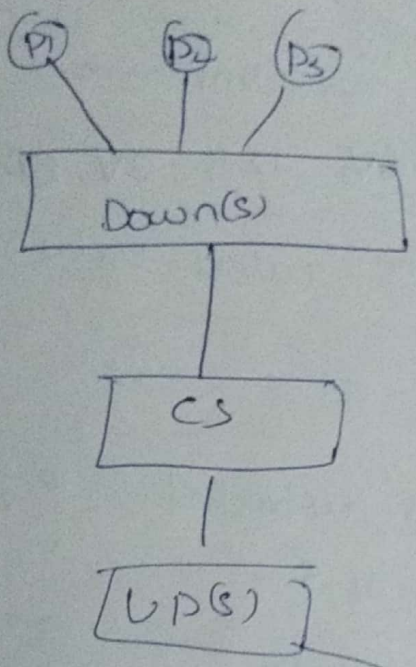
Structure definition

```
Struct BSEMAPHORE
{
    enum value(0,1)
    Queuetyp L;
}
```

```
BSEmaphore S;
s.value=1;
Down(S);
<CS>
Up(S);
```

```
Down(Bsemaphore S)
{
    1. if (s.value == 1)
        { s.value=0;
          return;
        }
```

else
{
put this process in to S.L() & block it
(sleep())
}
}



S=1

P1: P(S); Succ

S=0

P2: P(S);

⇒ unsucc

S=0

P3: P(S); ⇒ unsucc

P1: CS → leave

P1: V(S)

S.L()

P2: V(S): P2 P3

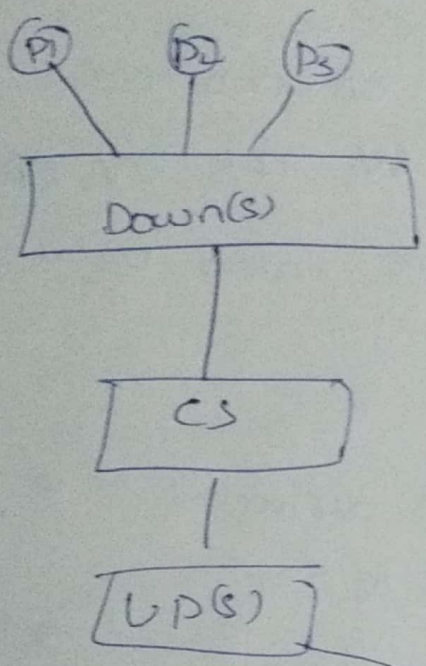
P3: V(S) ⇒ S=1

last process

makes=1

UP (B semaphore S)
{
1. if (S.L() is empty)
{
S value=1;
}
else
{ select a process ~~from~~ from S.L() &
call wakeup()

```
else
{
    put this process into S.L() & block it
    (sleep())
}
}
```



S=1

P1: P(S); succ

S=0

P2: P(S);

⇒ unsucc

S=0

P3: P(S); ⇒ unsucc

P1: CS → leave

P1: V(S)

S.L()

P2: V(S); [P2 P3]

P3: V(S) ⇒ S=1 — last process

makes=1

```
UP (B semaphore S)
{
1.  if (S.L() is empty)
    {
        S.value=1;
    }
    else
    { select a process from S.L() &
      call wakeup()
```

→ if will have already

NOTE 1/5

→ when "queue data structure" is used to used to store blocked process that is considered to be strong semaphore"

→ Semaphore is said to be "weak semaphore" if stack datastructure is ~~stor~~ used to store blocked process

ex - S = T = 1

P0	P1
P(S)	P(T)
P(T)	P(S)
CS	CS
V(T)	V(T)
V(S)	V(S)

T0 : P0	1 ↓	S = X0
T3 = P1	1	T = X0
T5 : P0	2	T = Φ0 Blocked
T7 = P1	2	S = Φ0 Blocked

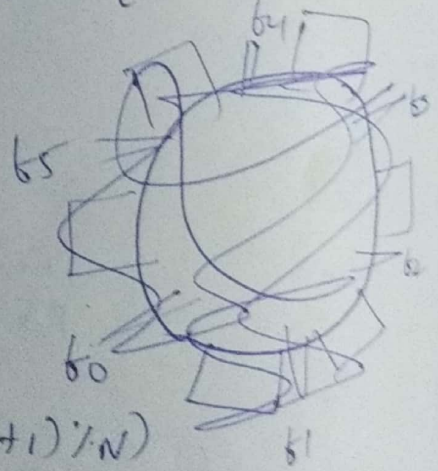
Deadlock

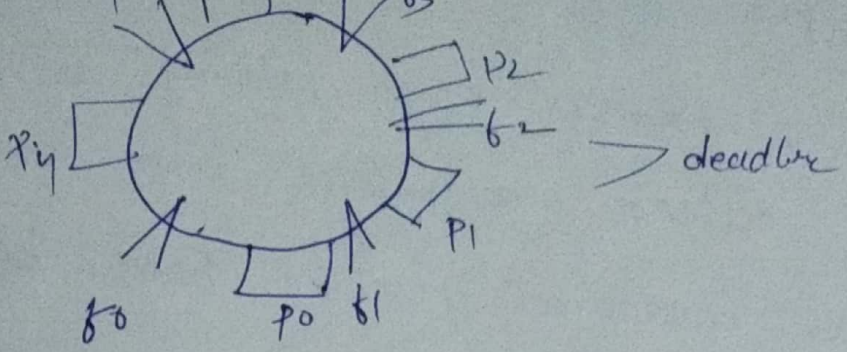
8 Semaphore based solutions for classical IPC problems.

1. producer-consumer problem
2. Reader-writer problem
3. Dining philosopher problem

Dining philosopher problem :-

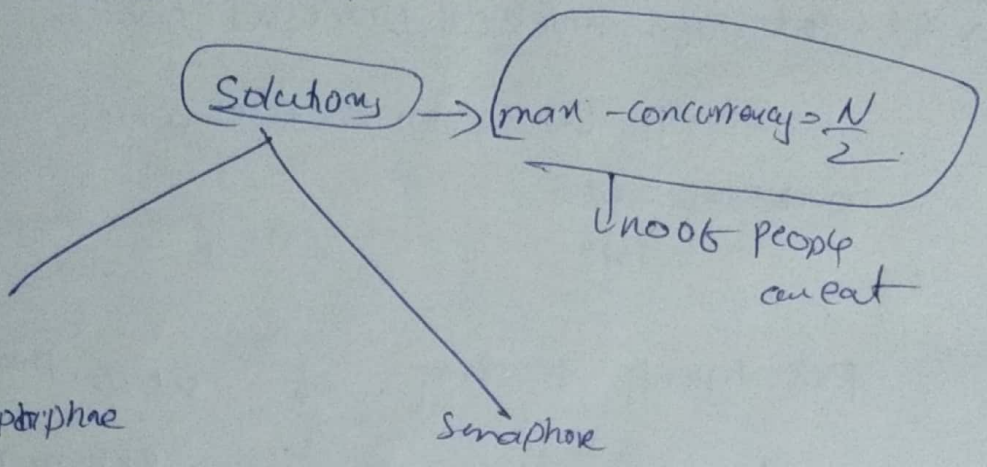
```
void Dinning philosopher(int i)
{
    while(1);
    {
        1. think
        2. Take fork(i)
        3. Take fork ((i+1)%N)
        4. eat(i)
        5. put fork(i)
        6. put fork ((i+1)%N)
    }
}
```





Dining philosphy problem

Solutions to break ~~deadlo~~ only deadlock chain



(1) 1
Left - Right
Reverse
N-1
R-L

(2) odd
L-R
even
R-L

Deadlock :

→ Set of processes Each holding a resource

(or) atleast one resorce and waiting for additional resource held by other waiting processes in that set

(or)

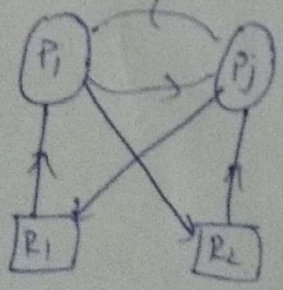
→ A set of blocked processes which are waiting for even to occur which could never occur

(or)

permanent blocking of set of processes that either compete for system resource (or) communicating each other

examples

(1)



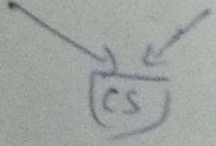
(2) Binary Semaphore S=1

P1

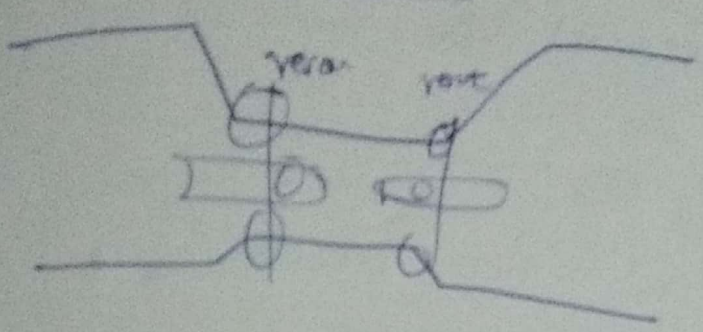
p(s);
p(f);

P2

p(f);
p(s);



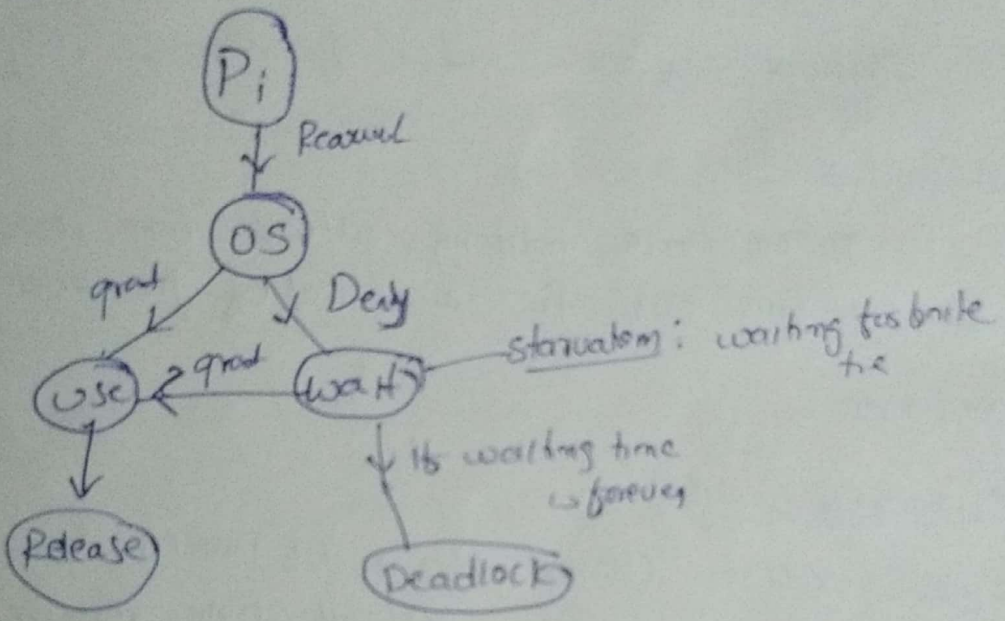
Bridge crossing



System model :

Let n - process ($P_1, P_2, P_3 \cdots P_n$)

m - Resources ($R_1, R_2 \cdots R_n$)



Characteristics of Deadlock

Necessary & Sufficient condition for occurance of Deadlock :

*(1) Mutual exclusion
(2) hold and wait
(3) Circular wait
(4) NO preemption