Mainframe O.S. for hardware utilisation.
PC : complex games, business apps & everything in between.
Handheld Comp. O.S. : user can easily interface

DATE: / /

types of OS

O.S. Operations

trap or Interrupt execution unauthorised access.

to handle trap code is written

Since O.S. & user share hardware & software resources of the computer we need to make sure that an error in user program could cause problems only for one program running with sharing many processes could be affected by a bug prgm.

1. Dual mode operations : in order to ensure the proper execution of O.S. we must be able to differentiate b/w the execution of O.S. code & user defined code. The approach taken by computer system is to provide hardware support that allows us to differentiate among various mode of operations or execution.

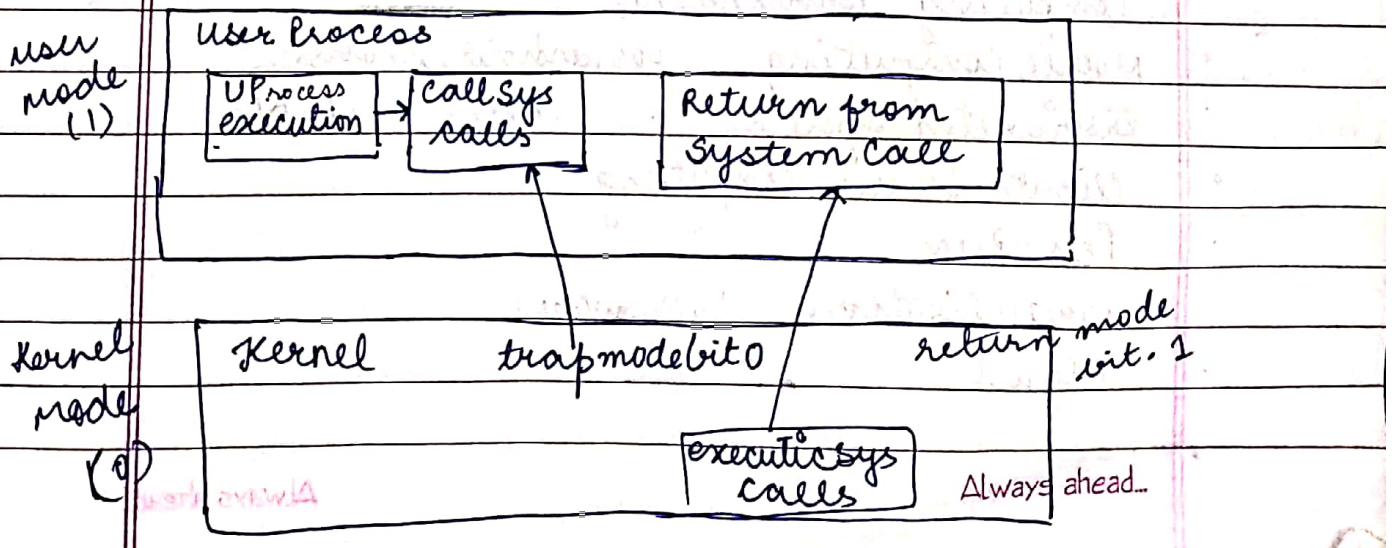
a) user Mode

a bit called modebit is added to hardware of computer to indicate the current mode.

Kernel mode = 0 bit

User Mode = (1).

b) Kernel mode.



at the boot-time hardware starts in kernel mode, O.S. is then loaded hence starts users apps or processes in user mode

DATE: __/__/__ PAGE NO. ____

whenever a trap/interrupt occurs hardware switches from user mode → kernel mode i.e. changes mode-bit to zero; whenever operating s. gains control of comp. it is in Kernel mode. Sys. allows switches to user mode before passing control to a user ~~mode~~ process.

TIMER : OS maintains control over CPU we can't allow a user program to get into an infinite loop or to fail to call sys services & never return control to the O.S. To accomplish this use of Timer is there.

* For specific interval of time process works

Before Kernel begins executing the user code, a timer can be set to interrupt the system or computer after specific time

PROTECTION & SECURITY

Protection involves ensuring that no process access or interfare with resources to which they are not entitled.

Security involves protecting sys from attacks, viruses & malwares

Computing Environment

- Mobile Computing : ios, android, windows
- Distributed System : LAN, WAN, MAN
- Client-Server Computing
- Peer-Peer "
- Virtualisation (VM ware)
- Cloud "

COMPUTING ENVIRONMENT:

DATE: / /
PAGE NO.

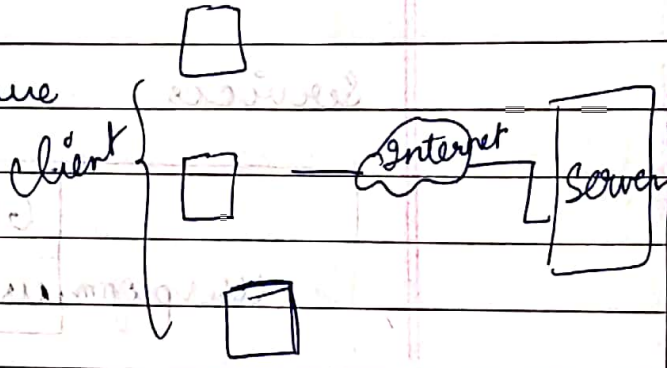
(a) Mobile Computing: is a technology that allows transmission of data, voice & video via a computer or any other wireless enabled device without having to be connected to a fixed physical link.
(Mobile host ⇒ link ⇒ mobile ~~host~~ support stations)

Principles: Portability, Connectivity, Social interactivity

b) Distributed sys: is a model in which components located on networked comp. communicate & coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

Characteristics: Resourcesharing, Openess, Concurrency, Fault tolerance, transparency.

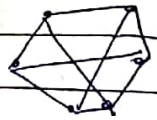
c) client- server
It is a distributed app structure that partitions tasks or workloads b/w providers of a resource or service called servers, & service requesters called clients.



d) Virtualization in cloud computing: is creation of a virtual (rather than actual) version of something, such as a server, a desktop, a storage device, an OS or network resources.

Peer to Peer: computing or networking is a distributed application architecture that partitions tasks or workloads between peers.

DATE: __/__/__ PAGE NO. ____



Cloud computing: practice of using a network of remote servers hosted on internet to store, manage & process data, rather than a local server or a personal computer.

provides storage on a network

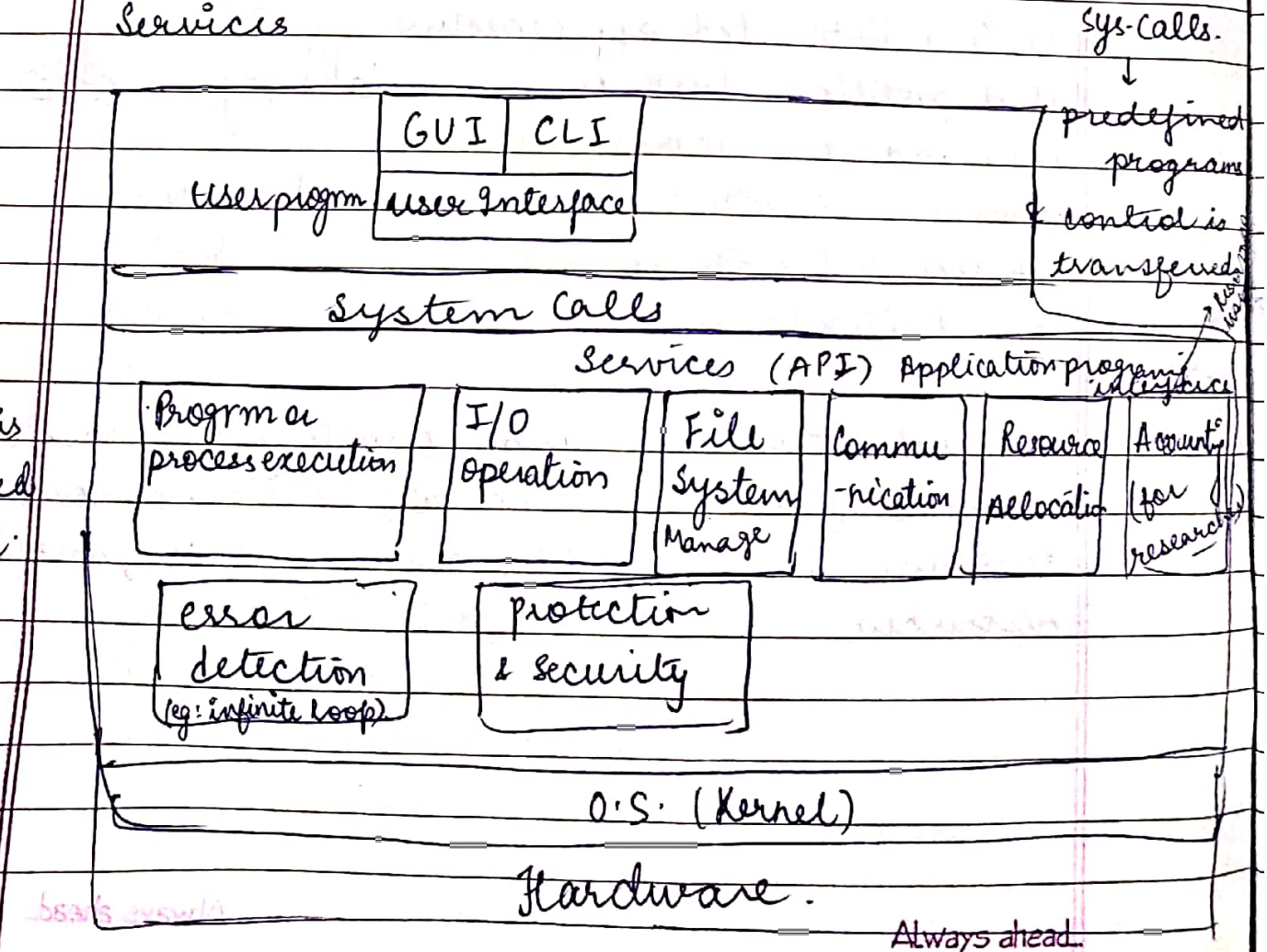
↳ Private, Public, Hybrid

↳ software, platform, infrastructure

Operating system services

O.S. interface:

1. GUI 2. CLI



Always ahead.

Accounting : Keep track of which user used how much & what kind of computer resources.

DATE: / / PAGE NO

O.S. Interfaces

1. Command Line Interface (CLI).
Shells some O.S. uses or includes command interpreter in kernel (O.S.), on sys. with multiple CLI interpreters to choose from interpreters are known as shells

2. (GUI) Graphical user interface : mouse-keyboard are used with right clicks on drop-down menus at backend we have commands running.

System Calls : Interface b/w an application prgm an O.S. is through sys. calls. when a prgm in user mode requires access to RAM or hardware resources it must ask the kernel to provide access to that resources.

Program → Sys. calls → O.S.

This is done by Sys. calls means programming Interface to services provided by O.S.
These are typically written in HLL (C or C++)
Mostly accessed by program via High Level API.
Most common API's are (i) WIN 32 API for windows
(ii) POSIX API for all versions of Unix, Linux, Mac O.S.
(iii) Java API for Java virtual machine

Types :
(i) process control (ii) File manipulation (iii) Device Manipul I/O
(iv) Information Maintainance (v) Communication
vi) Protection.

Always ahead

System Calls

DATE: __/__/__
PAGE NO. _____

```
#include<stdio.h>
main
{
printf(" ")
}
```

user program

user mode.

C standard library or System Call interface

fork()
read()
open()

Registers (smallest & fast)
↓
Cache mem
↓
RAM
↓
Hardisk

Kernel mode (for eg: write())

↓

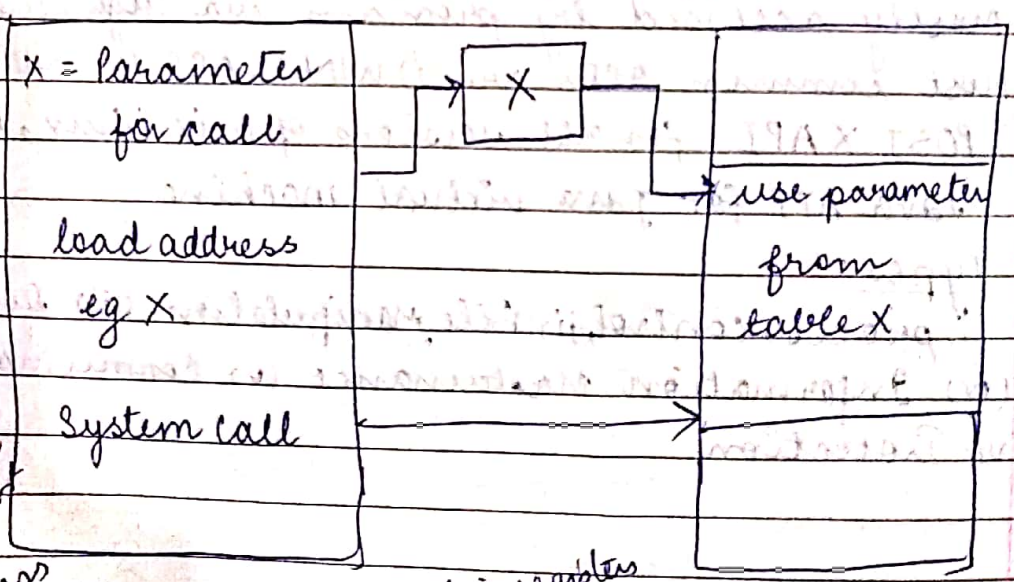
execute write() system call

System Call parameters :

to send parameter to O.S

Parameters can be passed in registers.

when more parameters than register than these can be stored in a block or table in a memory at address of that table or block is passed as a parameter in register.



In single parameter no requirement for storing memory address but for more parameters are passed in registers in O.S / memory address is sent in registers

Always ahead

block

Scanned by CamScanner

Pushing parameter in stack or copying parameters or data from stack.

DATE: / / PAGE NO.

Types of Sys. Calls

1. Process Control

following f"s of process control are :-

- create or terminate process
- load & execute processes.
- End a process or abort it.
- Get process attribute & set process attributes
- Allocate & free memory.

system ca
for proces
fork () = Crea
exit()
wait
(name, ids of proces

2. File Management

- open file, close file &
- create & delete file
- read, write & reposition of file
- get & set file attributes.

open()
read()
write()

3 Device Management

- Request device & released device
- Read, write & reposition device.
- Logically attached & deattach device.
- Get device & set device attribute.

read()
write().

4. Information Maintainance

- Get time or date & set time or data.
- Get sys data & set sys. data
- Get process file & device attributes
- set process " " " "

5 Communication

- Create, delete communication connections
- send, seciene msgs.
- transfer status information

Always ahead

System Programs

1. File Management : open, close.
2. Status information : (built in programs) memory life, multiple users, date & time
3. Programming Language Support.
   compiler, assembler etc. Eg : gcc compiler.
4. File Modification (gedit)
5. Communication
6. Program loading & execution (load in RAM)
7. Multiple Loader : for eg : GRUB in sys. loader or boot
   for file : ? which loader is used ? absolute.
8. Background services : work at boot time

Computer System allows multiple programs to be loaded into m/m & to be executed concurrently.

Process

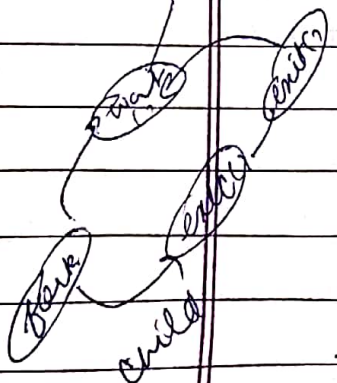
is basically a program in execution. The execution of a process must progress in a sequential fashion.

It is defined as an entity which represents the basic unit of work to be implemented in System

Procedure for program execution.

loading into main memory → execution → process

Execution of process in Memory.



stack	memory for temporary data store (method parameters, local variables)
heap	memory at run time
text	data in registers is stored ~~value of program counter~~.
data	value of pgrm counters globally declared variables.

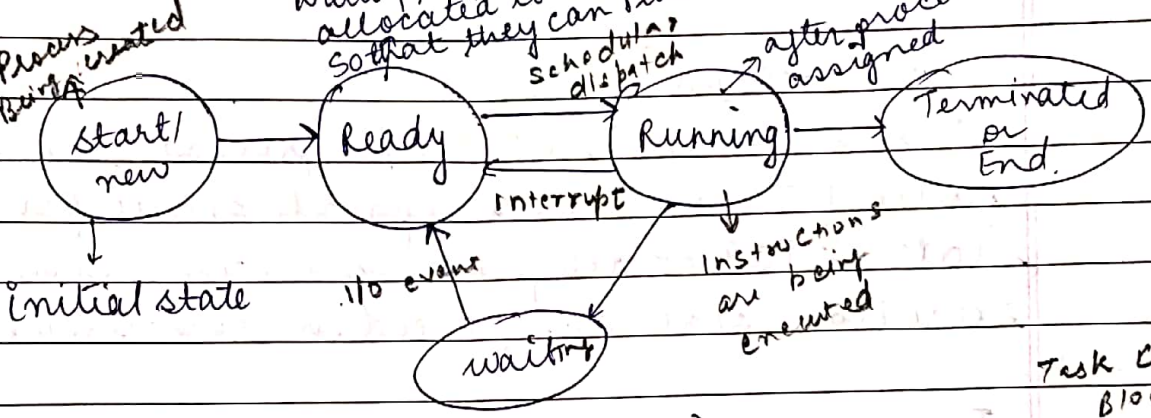
1. Stack: temporary data such as method / fn, parameters, return address & local variables.

2. Heap: dynamically allocated memory to a process during its run time.

3. Text: includes the current activity represented by the value of Program Counter & content's of the processor's registers.

4. Data: this section contains the global & static variables.

Process states



PCB (Process Control Block). (Task control Block)

(a type of register) ↳ Each process is represented in OS by PCB

A process it is a data structure maintained by the O.S. for every process.

Process id	→ unique
Process State	→ current state of process i.e. running, waiting etc.
Program Counter (PC)	→ counter indicates the add of next instruction to be executed
CPU Registers ← Register types: General purpose, Stack register, index register	* when interrupt occurs, to allow process to continue correctly
Memory Management ~~Limit~~	→ info about page table, memory limit depending upon OS (MM used by OS)
list of open files	→ I/O devices allocated to process
etc :-	

Always ahead...

PCB is maintained for a process throughout its lifetime & is deleted once the process terminates

DATE: / /
PAGE NO. Aa

Process Scheduling

to achieve multiprogramming & time sharing system objectives process scheduler select a process from a set of processes for program execution on CPU. We can say that process scheduling is the activity of process manager that handles removal of removal of running process from the CPU & selection of another process.

Process Scheduling Queues.

O.S maintains all PCBs in process scheduling queue O.S maintains a separate queue for each state & PCB of all processes in the same execution state are placed in same queue.

Job Queue : as processes enters sys. they are into job queue

Ready Queue : processes reside in main memory & ready to execute are kept on a list called ready queue. (linked list)

Device Queue : process in waiting state, are stored in device Queue (I/O) devices are required

waiting for I/O

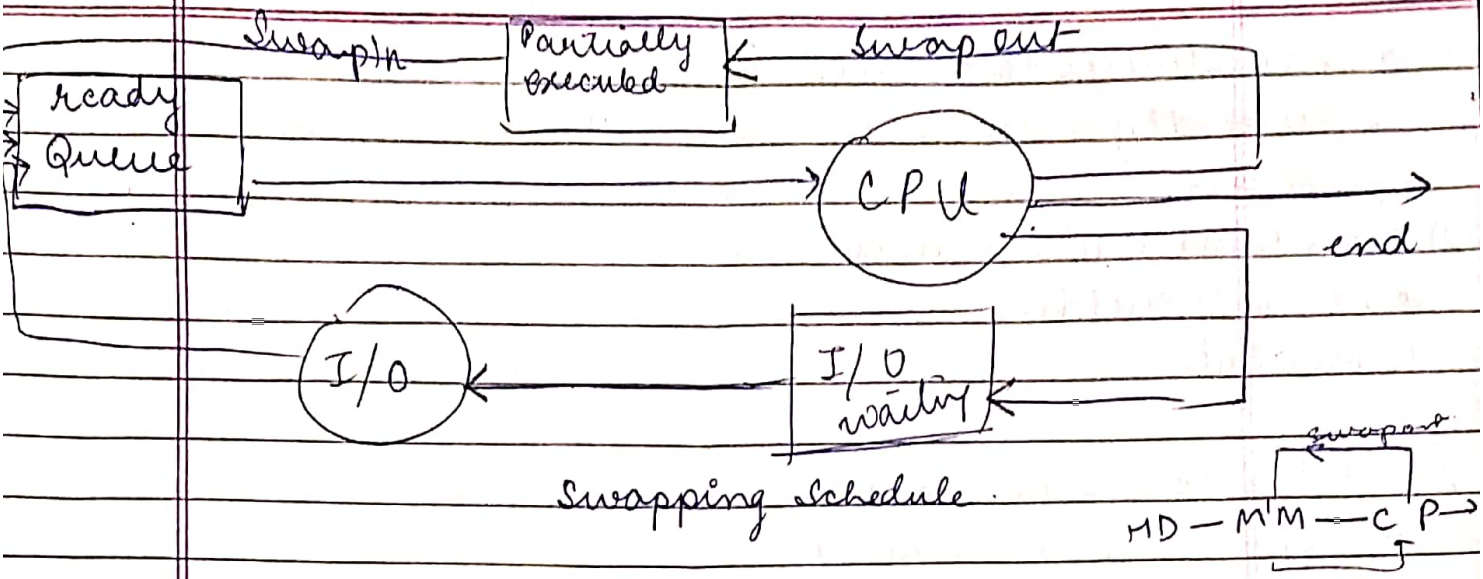
Schedulers : types of schedulers → select process & load them into m/m for execution

1. long term scheduler or job scheduler (Hard-D to Main M/m)
2. short term or CPU scheduler (memory → CPU)
3. Medium or Swapping Scheduler (remove process from m/m & thus reduce degree of multi)

↓ control degree of multiprogramming - no. of process in m/m

Always ahead

a process migrates among various scheduling queues throughout its lifetime, schedulers main task is to select job to be submitted into sys & to decide which process to run.



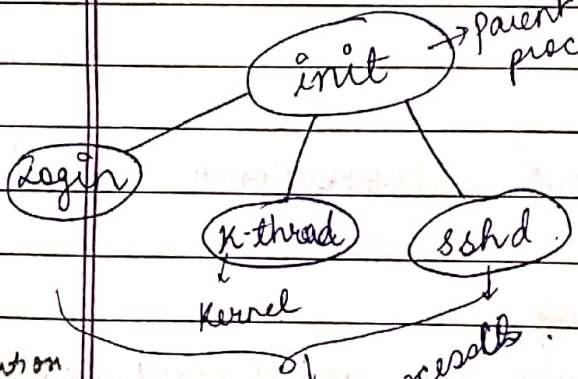
Swapping schedule.

Context switch :

Saves & start from last point for execution.

Operations on Process :

Creation of Process : Process may create several processes, creating a process is called parent process & rest is called child process.



automatic fork() is made process id is generated.

Termination of Process.

exit call is used.

child execution complet resources are demanded is excess → end.

parent process complet execution.

system call

return code for fork call — 0 (new child process)
non-zero (child return to parent)

Process creates a new process with two possibilities for execution

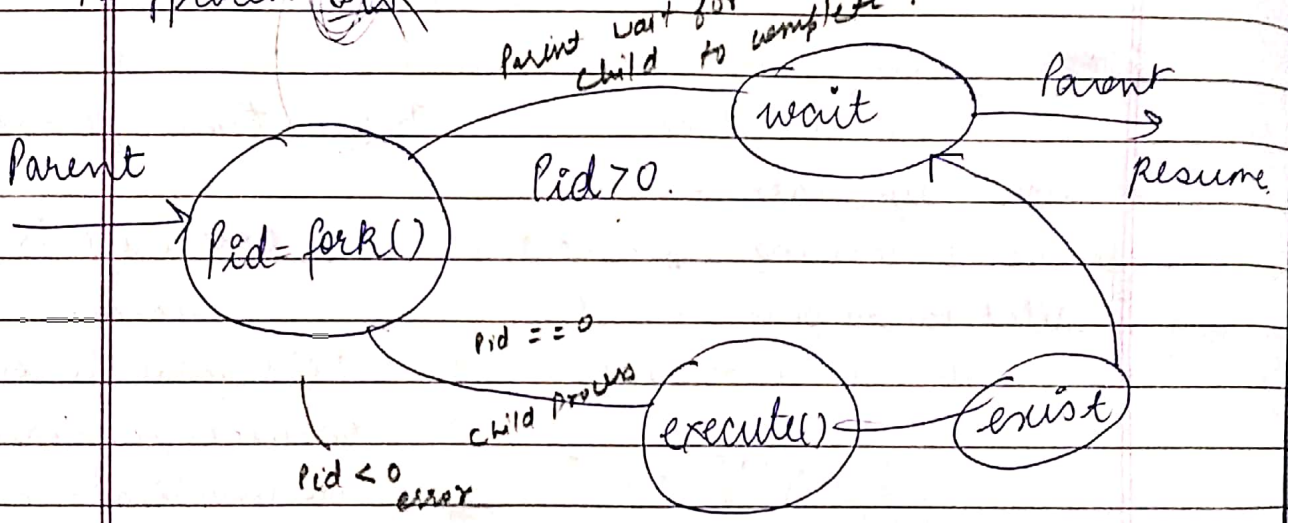
DATE: __/__/__
PAGE NO. _____

1) Parent continues to execute concurrently with child processes.
2) Parent wait until all or some of its children have terminated.

for ~~Total~~ fork calls ~~created~~.
no. of processes created

$2^n - 1$ (n = no. of sys calls) $= 2^n - 1$

if fork calls = 3
no. of process = 8



Process termination

* IPC (Inter Process termination Communication)

Types
- Independent
- Concurrent Co-operating

There are several reasons for providing an environment that allows process co-operation.

Use of co-operation
- information sharing
- Computation speed-up
- Modularity

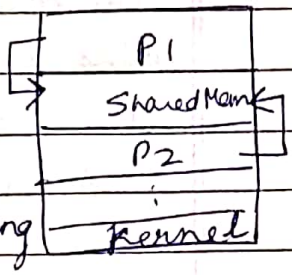
Always ahead

There are two methods.

- Shared memory :- Resources sharing use executing
- Message passing : Sender & Reciever

DATE: / /
PAGE NO.

↳ In shared memory a region of memory i.e. shared by co-operating processes typically a shared memory region reside in address space of process creating shared memory segment, Other processes that wish to communicate using this memory segment must attach to their address space.



Message Passing:

↳ Naming ↳ Synchronisation ↳ Buffering

provides a mechanism to allow space processes to communicate & without sharing same address space. eg: chat (useful in distributed environment)

Operations:

a) sending b) Recieving

(P, Q) are process

Send (P, msg)

Recieve (Q, msg)

Methods for msg passing

1) Naming (Direct or Indirect Communication)

in direct

define the sending & Recieving : symmetric.

define the sender but not Reciever asymmetric

in indirect

sending has defined add but not receiver as it can recieve from multipleprocesses.

Direct Communication:

- a communication link automatically b/w every pair of processes that want to communicate.
- Link with exactly two processes.

Always ahead...

Indirect communication :
communication b/w diff processes is done by a mail box or port.
a space

send (A, msg) P process sends
Recieve (A, msg). Q process recieves.

Synchronization
Communication b/w processes take place through calls to send & recieve permitives.
There are different design options for implementing each permitive, msg passing may be either blocking (synchronous) or non-blocking. asynchronous

Blocking send
sending process is blocked until msg is recieved by the recieving process or mail box.

Non-blocking send.
can send to others after resuming

Blocking recieve
recieves & then again blocks.

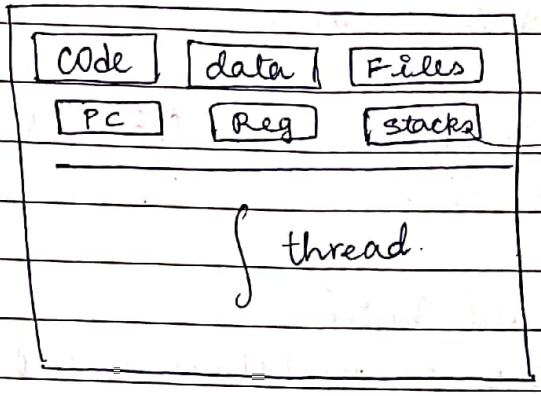
Non-blocking recieve
recieves retrieves either a valid message or null.
(whatsap)

Buffering : whether communication is direct or indirect, messages exchange b/w processes reside in temporary queues. These queues can be implemented by 3 ways :

1) 0 capacity queues only send/Receive not for storage.
2) bounded capacity queues n no. stores fix msg.
3) Unbounded capacity queues. ∞ stores msg (infinitely long).

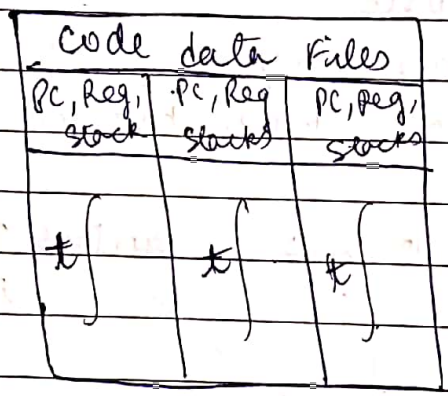
* threads (light weight process)

→ Single threaded
→ Multi threaded



→ executed prgm.

Single thread



Multi-thread

Benefits:
Fast, time less.
Resource sharing
Economy: Multi-thread.

Multiprogramming threads :.

core.

Core 0	T_1	T_3	T_1	T_3
Core 1	T_2	T_4	T_2	T_4

single.

T_1	T_2	T_3	T_4	T_1	- - -

Type of parallelism.

i) Data parallelism (data is distributed to cores)
ii) Task parallelism (to threads)

Multi-threaded Models.

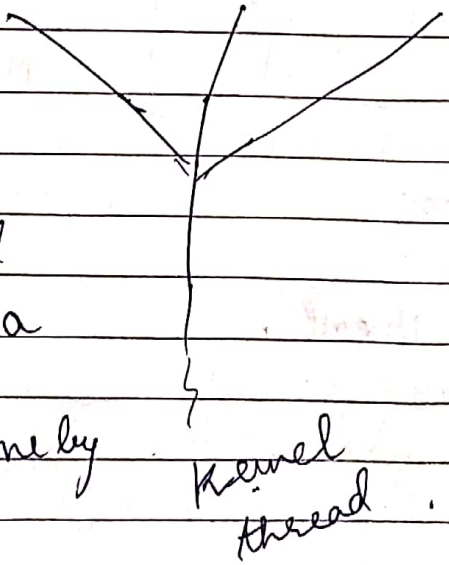
many to one multi-threaded model
one to one
many to many

b/w user & kernel : multi-threaded

Threads

User thread1 User thr2 User thr3 — User threads ↓ user handles.

Kernel threads O.S manages

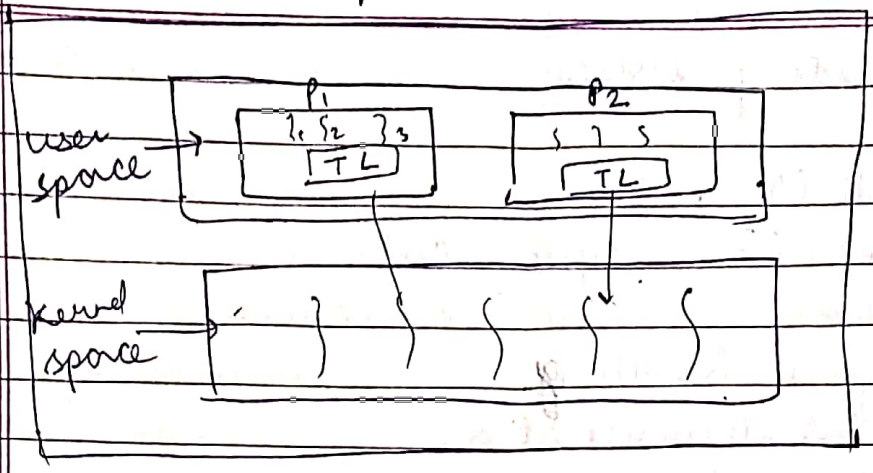


only one thread access kernel at a time thread Management is done by thread library

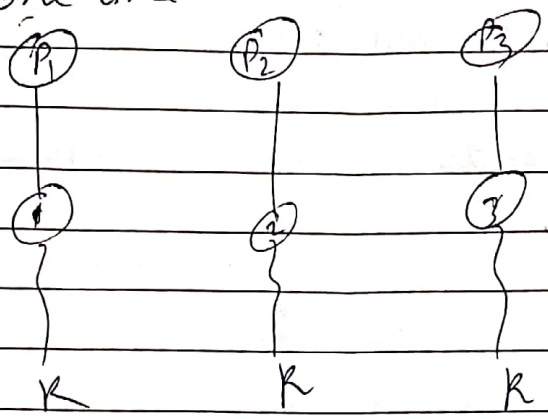
Kernel thread.

DATE: __/__/__
PAGE NO. ____

M - one



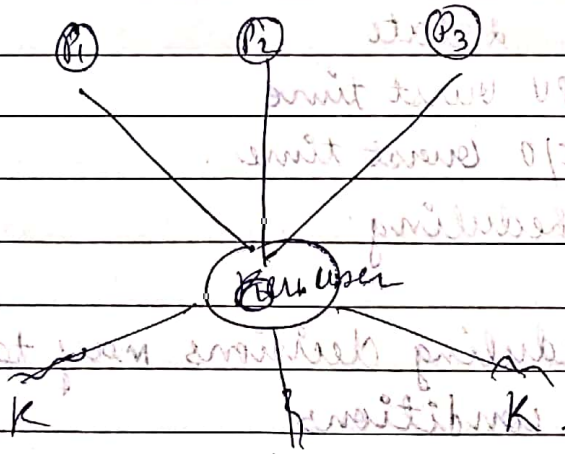
one - one



specific task is stopped

kernel thread is ~~used~~ generated everytime by OS in this but blocking situation is diminished

many to many



if any is blocked we can distribute it to another kernel thread

Always ahead...

Threading issues:

- Fork() and exec() system call.
- signal handling.
- thread cancellation
- thread local storage.
- scheduler activation

* linux threads

1. fork()
2. clone() flag (

data share

Process Scheduling.

Basic Criteria:

CPU : 2 state
CPU burst time
I/O burst time.

Primitive scheduling:

* CPU scheduling decisions may take place under four conditions
1. when process switches from running state to waiting state. R-W
2. when process switches from running to ready state R-R
3. when process switches from waiting state W-R.

Always ahead.

DATE: / / Aa
PAGE NO.

to ready state.

4. Terminates.

For situation 1 & 4 there is no choice in terms of scheduling.
A new process must be selected for execution whereas there is a choice for situation 2 & 3.

When scheduling takes place under cond^{n} 1 & 4. We say that scheduling is non-primitive or co-operative otherwise primitive.

Scheduling Criteria

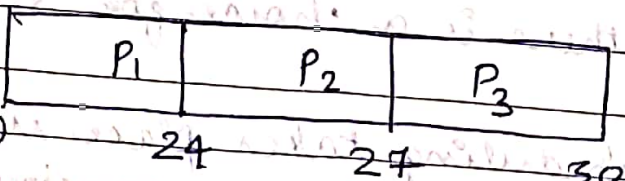
1. Max^{m} CPU utilisation
2. Throughput
3. Waiting time
4. Response time

Process Scheduling Algorithm

1. First come, First serve. FCFS
2. Shortest Job, First Scheduling. SJFS

DATE: __/__/__
PAGE NO. _____

Process	burst time
P_1	24
P_2	3
P_3	3

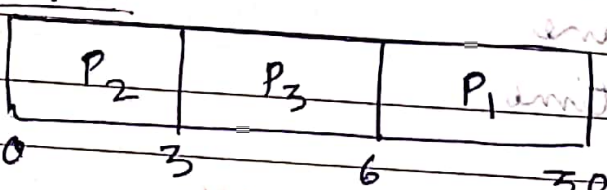


for P1 P2 P3. → for execution

Avg. waiting time: $\frac{0+24+27}{3}$

$= \frac{51}{3} = 17$ ms

2. type SJFS.



Wating time $= \frac{0+3+6}{3} = 3.$

DATE: / / PAGE NO. A2

Process	Burst time.
P_1	6
P_2	8
P_3	7
P_4	3.

GANTT diagram

P_4	P_1	P_3	P_2

0 3 9 16 24

Avg. waiting time $= \frac{0+3+9+16}{4}$

$= 7.$

type1:

P_1	P_2	P_3	P_4

0 6 14 21 24.

$\frac{0+6+14+21}{4} = \frac{41}{4} = 10.25$

Avg waiting time = burst time.

P_1	6
P_2	8
P_3	7
P_4	3

SJFS.

P_4	P_1	P_3	P_2

0 3 9 16 24

Avg time = 7.

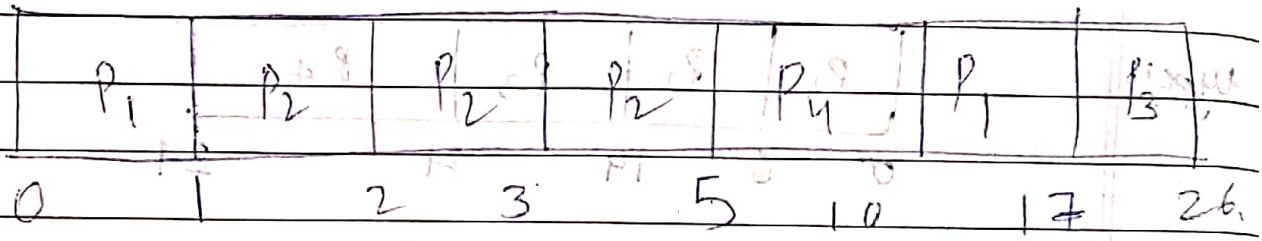
Always ahead..

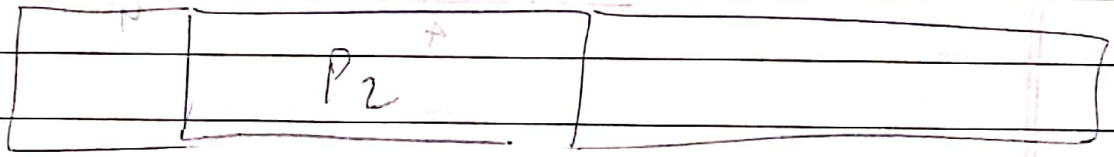
non-primitive method

DATE: __/__/__
PAGE NO. _____

Process	Burst time or execution	Arrival time
P_1	8	0
P_2	4	1
P_3	9	2
P_4	5	3

GANTT chart.





Turn around time =
Completion time - Arrival time

waiting time = turn around time - burst time

DATE: / /
PAGE NO.

Q2

	Completion	Turn around time
P_1	17 - 0	17
P_2	5 - 1	4
P_3	26 - 2	24
P_4	10 - 3	7

$$Avg = \frac{17+4+7+24}{4} = \frac{52}{4} = 13$$

P_1	17 - 8	9
P_2	4 - 4	0
P_3	24 - 9	15
P_4	7 - 5	2

$$\frac{9+0+15+2}{4} = \frac{26}{4} = 6.5$$

Q2

	Burst	Arrival
P_1	21	0
P_2	3	1
P_3	6	2
P_4	2	3

P_1	P_2	P_2	P_2	P_4	P_4	P_1
0	1	2	3	4	6	12	32

	Turnaround	Waiting
P_1	32 - 0 = 32	32 - 21 = 11
P_2	4 - 1 = 3	3 - 3 = 0
P_3	12 - 2 = 10	10 - 6 = 4
P_4	6 - 3 = 3	4 - 2 = 2

Avg = 11.5 Avg = 12

Always ahead...

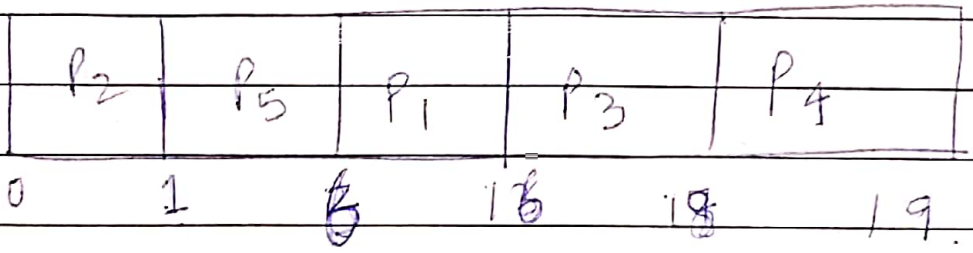
Scanned by CamScanner

SRTF : Preemptive
SJF : non - Preemptive

DATE: / /
PAGE NO.

Q3

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

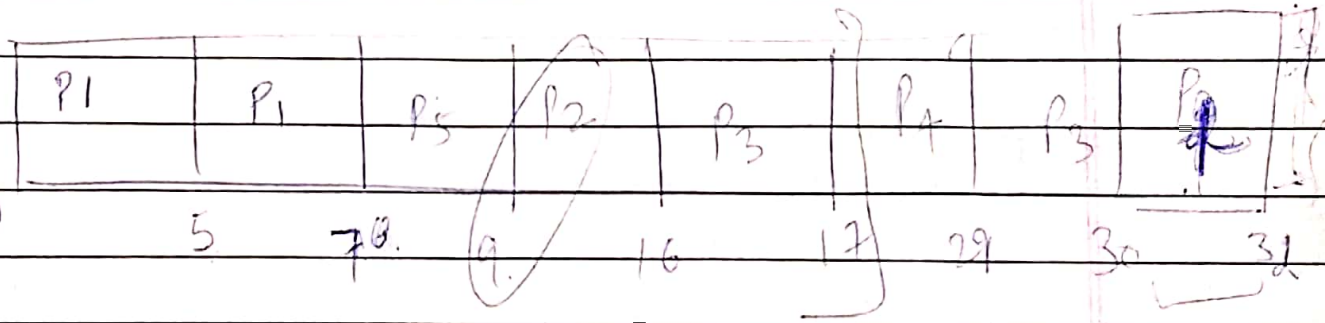


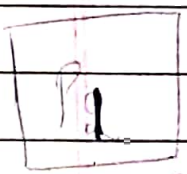
$$\frac{1+6+16+18+19}{5}$$

DATE: / /
PAGE NO.

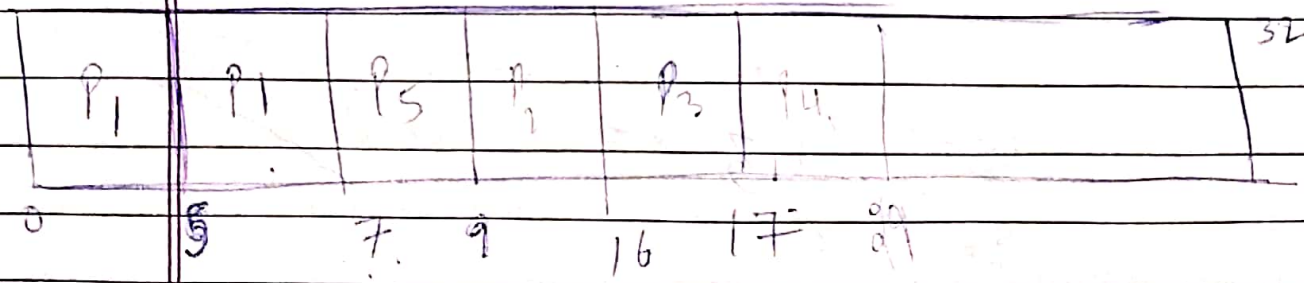
Q4

Process	burst	Priority	A.T
P_1	8	3	0
P_2	9	2	5
P_3	4.3	2	16
P_4	12	0	17
P_5	2	1	7





32 35

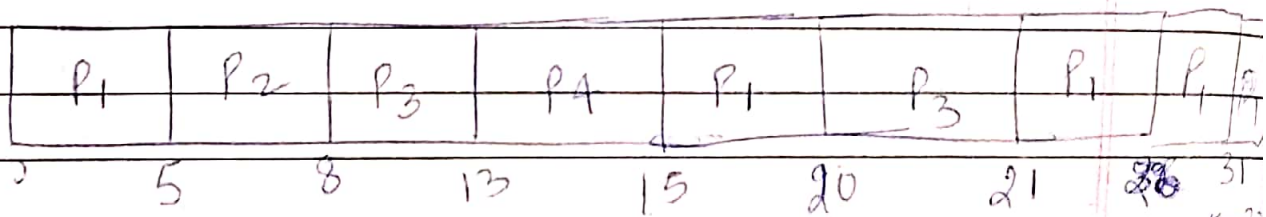


Round Robin

DATE: / /
PAGE NO.

It is for time sharing system.
It is similar to first come first serve but preemption is added to enable the system to switch b/w processes.
A small unit called time Quantum or time slice define.

Process	Burst time
P_1	21 16 11
P_2	3
P_3	6 1
P_4	2



Avg waiting time.

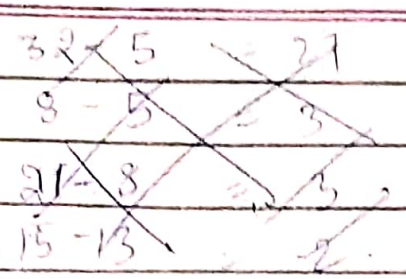
$$\frac{0+5+8+13+15+20+21}{4}$$

$0+15-5+21-20$ ⊕ $= 11$

	Waiting
P_1	0 + (15 − 5)
P_2	5
P_3	8 + (20 − 13) = 15
P_4	13

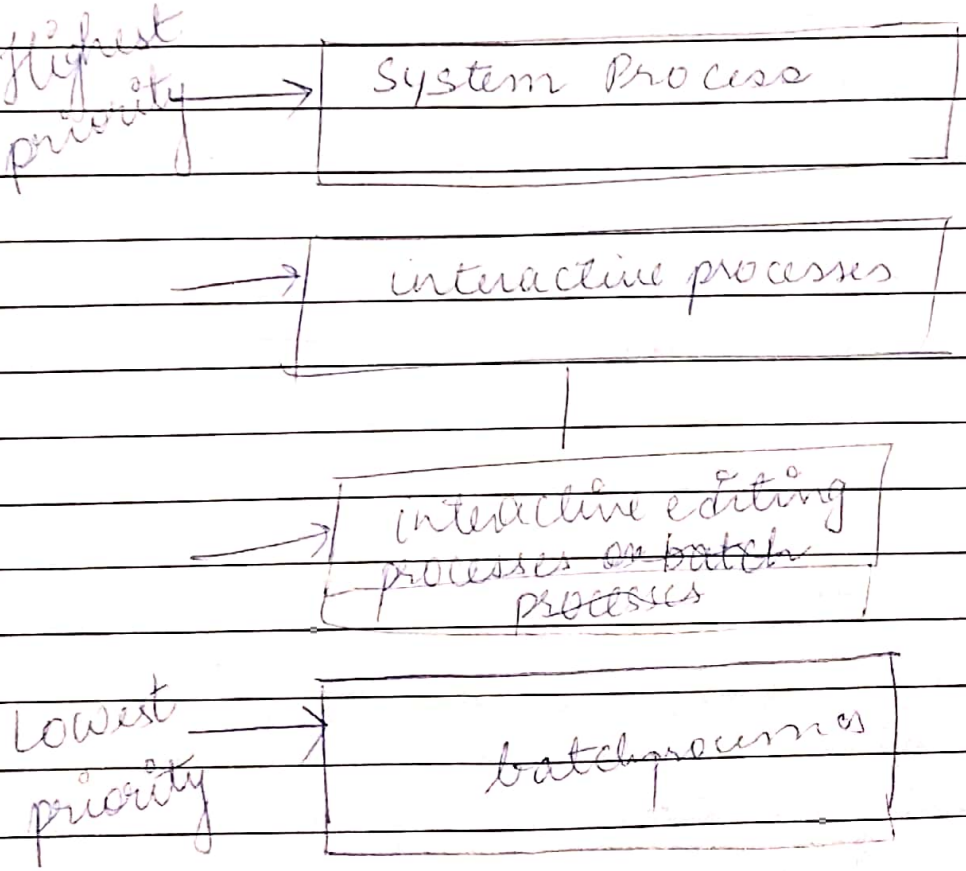
$44 = 11$

DATE: / /
PAGE NO.



Multilevel Queue Scheduling :

In this ready queue is divided into several separate queues. Processes are permanently assign to one que based on some property of process such as memory size, process priority or process type

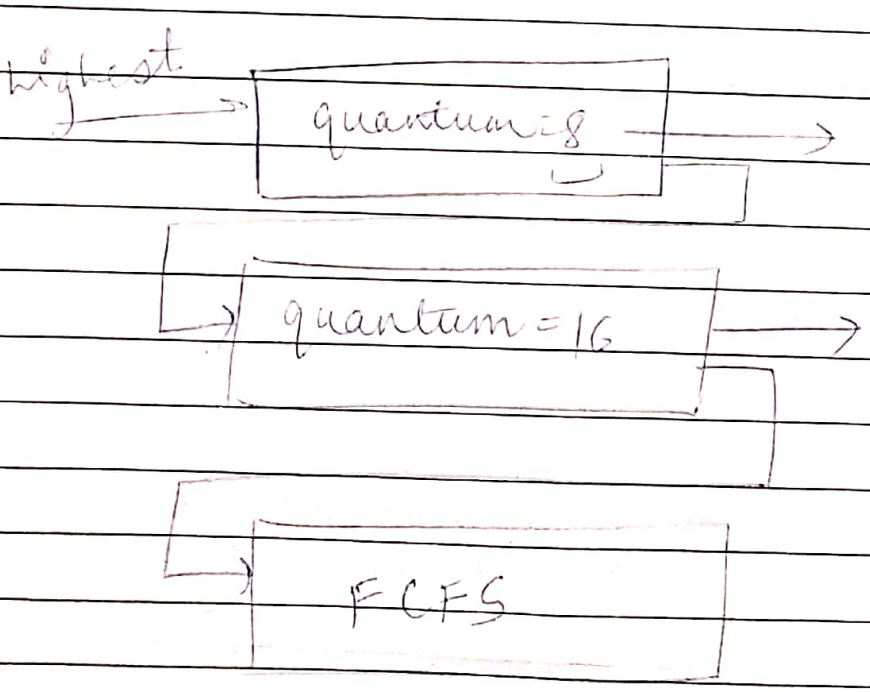


DATE: __/__/__
PAGE NO. _____

Multi-level feedback queue Scheduling

In this case, scheduling algorithm allows a process to move between queues. Idea is to separate processes according to characteristics of CPU burst time or execution time.

If a process uses too much time, it will be moved to lowest priority queue in addn to a process that waits too long in a lower priority queue maybe moved to a higher priority queue



Multi-processor Scheduling

Real time Scheduling

DATE: / /
PAGE NO.

5

Process	Arrival time		
P_0	0		1
P_1	1	3	2
P_2	2	6	1
P_3	3	6	3

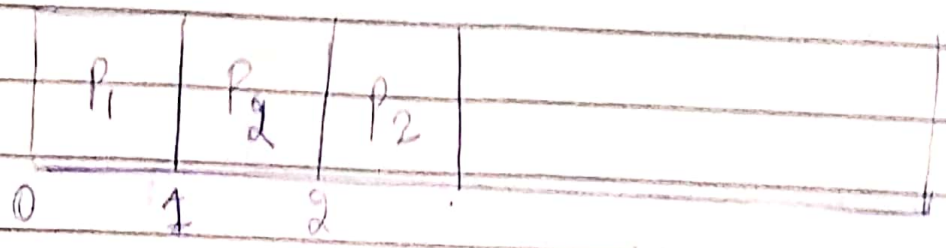
P_0	P_0	P_0	P_0	P_2	P_1	P_3	
0	1	2	3	5	17	16	20

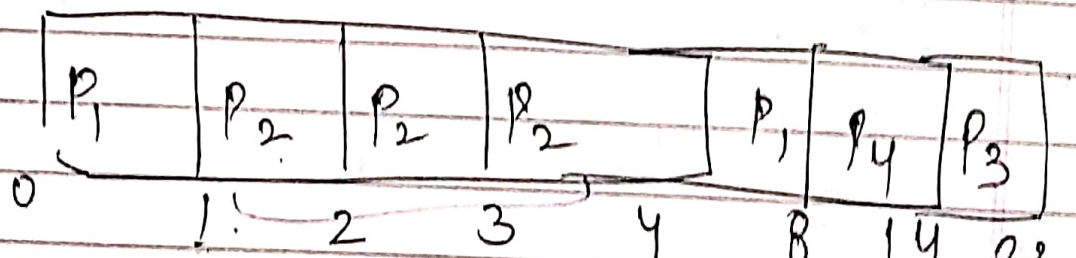
	Turn around time	Waiting time
P_0	5 − 0 = 5	5 − 5 = 0
P_1	16 − 1 = 15	15 − 3 = 12
P_2	17 − 2 = 11	11 − 5 = 3
P_3	20 − 3 = 17	19 − 6 = 13

6

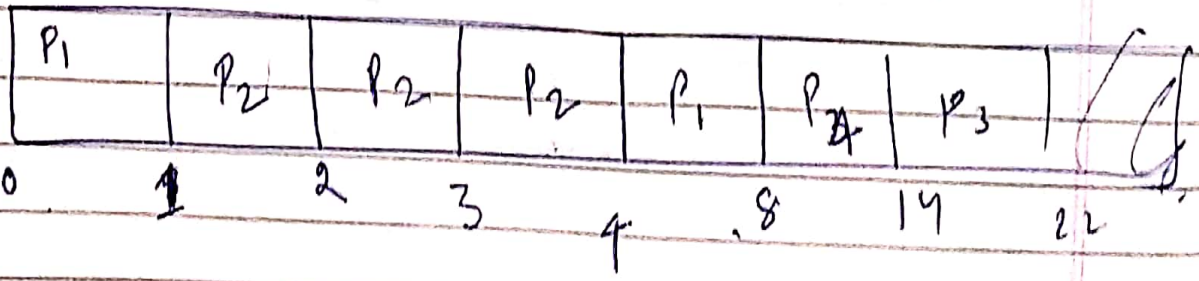
Process	Arrival time	Burst time
P_1	0	5
P_2	1	3
P_3	2	6
P_4	3	4

3+1





Process



Waiting time = P_1 1 + (8 − 1) = 5

P_2 2 + (4 − 3) = 3

P_3 22 − 14 = 8

P_4 14 − 8 = 6

TAT WA

DATE: / / PAGE NO.

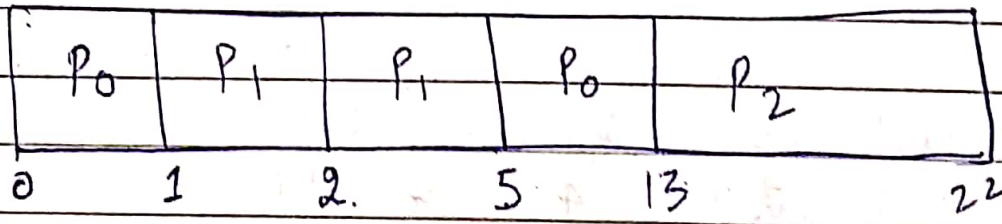
	Comp - Arr	T - B
P_1	8 - 0 = 8	8 - 5 = 3
P_2	4 - 1 = 3	3 - 3 = 0
P_3	22 - 2 = 20	20 - 8 = 12
P_4	14 - 3 = 11	11 - 6 = 5

12/8/20/4

Avg = 5

SJFS

Process	Arrival time	Burst time
P_0	0	9 . 8
P_1	1	4 = 3
P_2	2	9



13/9/22

	Comp - arrival	WT
P_0	13 - 0 = 13	13 - 9 = 4
P_1	5 - 1 = 4	4 - 4 = 0
P_2	22 - 2 = 20	20 - 9 = 11

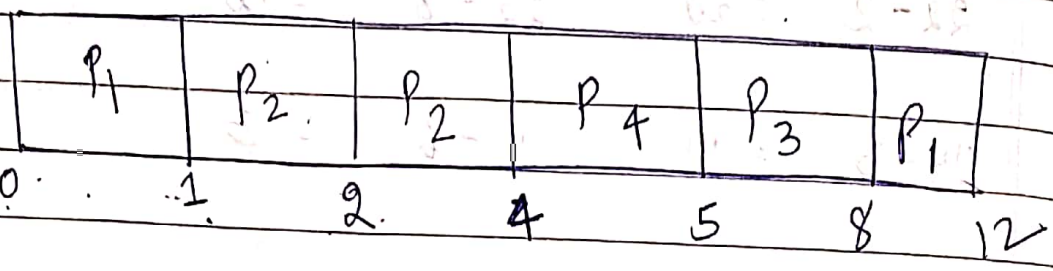
= 15/3 = 5 .

SC PS

DATE: / /
PAGE NO.

Q8

Process	A.T	Burst time
P_1	0	5 = 4
P_2	1	3 = 2 - 1 = 1
P_3	2	3
P_4	4	1



	TAT	WT
P_1	12 - 0 = 12	12 - 5 = 7
P_2	4 - 1 = 3	3 - 3 = 0
P_3	8 - 2 = 6	6 - 3 = 3
P_4	5 - 4 = 1	1 - 1 = 0

= 10/4 =

P_1	P_2	P_4	P_3	P_1
0	1 2	4	7	11

11 - 0 = 11
2 - 1 = 1
7 - 2 = 5
4 - 4 = 0

11 - 5
1 - 3

TAT = CT - AT

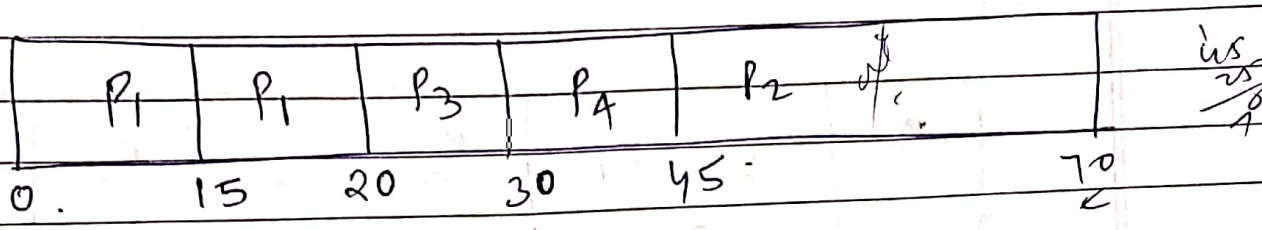
WT = TAT - BT

DATE: / /
PAGE NO.

Q9

	AT	Burst time	
P_1	0	20	= 5
P_2	15	25	
P_3	30	40	
P_4	45	15	

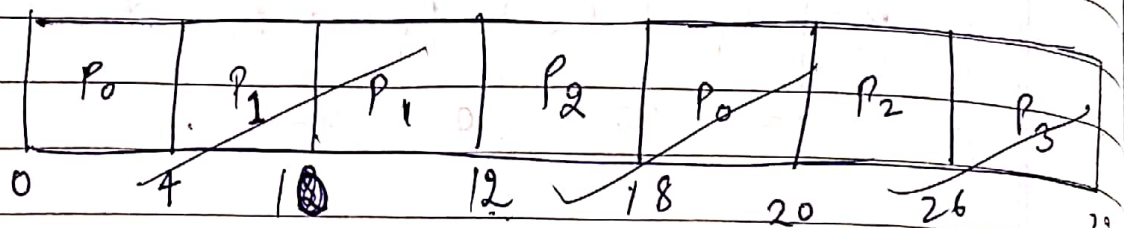
Waiting time for process P2.

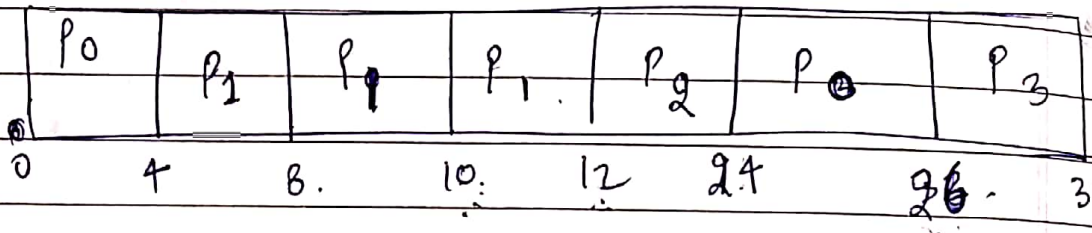


DATE: __/__/__
PAGE NO. ____

Q10 / By using priority algorithm,

Process	A	brust time	Priority
P_0	0	6-4=2	2
P_1	4	8-4=4	1
P_2	10	12	1
P_3	8	6	3

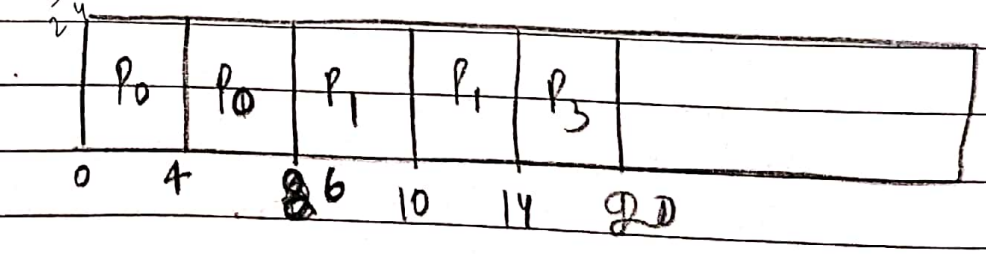




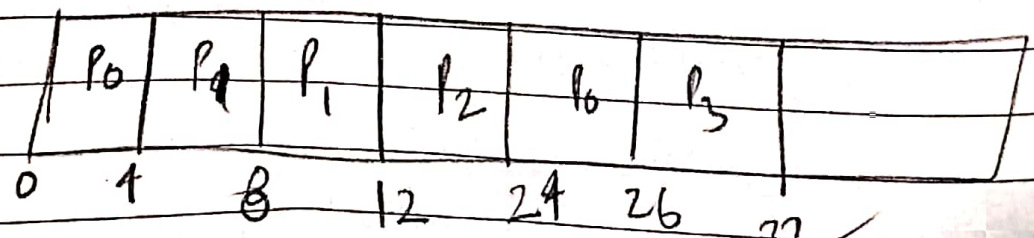
	TAT		AWT	
P_0	26-0	= 26	26-6	= 20
P_1	12-4	= 8	8-8	= 0
P_2	24-10	= 14	14-12	= 2
P_3	32-8	24	24-6	= 18

(26-8) = wait

40/4 = 10



do entry sec
CS
exit



code.

send & receive data. (communication)

DATE: 1 1 Aa
PAGE NO.:

Process Synchronization,

Critical Section problem :

Each process has a segment of code called critical section in which process may be changing common variables, updating tables, writing files & so on, imp. feature of system is that when one process is executing in its critical section, no other processes is allowed to execute its critical section.

Problems:

(i) Mutual exclusion (one at one time)

if process P_i is executing its critical section then no other processes can be executing its critical section.

(ii) Progress

If no process is executing in its critical section some processes which to enter into their critical section then only those processes that are not executing in their remainder section can participate in the ~~exit~~ deciding which will enter its critical section next.

(iii) Bounded waiting

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

Always ahead...

Solution to Critical Section problem. or Peterson algorithm.

DATE: __/__/__
PAGE NO.

Peterson solution or algorithm

Solution requires two processes to share two data items

i) int turn;

ii) Boolean flag [i] (2 processes)

variable indicate whose turn to enter into critical section

Two process.

turn = i means process Pi enter into critical section.

Pi, Pj

→ Pi will enter

Flag array is used to indicate if a process is ready to enter its critical section

```
Repeat
 flag [i] := true;

 turn = j;
 while ( flag [j] and turn = j ) do no-op;
  CRITICAL SECTION
   flag [i] = false;

  Remainder section
   until false;
```

DATE: __/__/__
PAGE NO. _____

Mutex Lock (Mutual exclusion lock):

```
acquire()
{ while (!available) ; (waiting)
  available = false;
}
release()
{ available = true;
}
```

Semaphore (S):
(integer variable)

wait:

```
{ while (S<=0);
  S--;
}
```

signal:

```
{ S++ ;
}
```

```
do {
  wait();
// C.S
  Signal()
} while (true)
```

DATE: __/__/__
PAGE NO. _____

Implementation of Semaphore.

CSP (Critical Section Problem).
two process (Peterson).
Multiprocess (Semaphore).
H/W.

to overcome the need for busy waiting, we can modify the definition of wait & signal operation. When a process execute wait operation & finds that the semaphore value is not +ve, it must wait.
Rather than engaging in busy waiting, the process can block itself
the block operation places a process into waiting queue associated with semaphore.
the state of process is switched to waiting state & then control is transferred to CPU scheduler which selects another process to execute.
a process that is blocked waiting on a semaphore S should be restarted when some other process execute a signal operation the process is restarted by a wake-up() or operation which changes process from waiting to Ready state.

DATE: / /
PAGE NO.

typedef struct
{ int value;
struct process * list;
} semaphore;

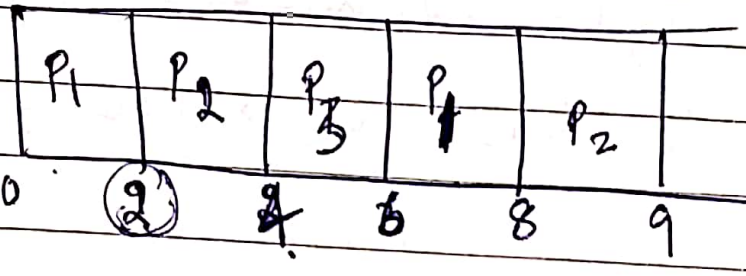
wait (semaphore *S) {
{ S -> value --;
if (S -> value < 0)
{ add this process to S -> list;
block()
}
}

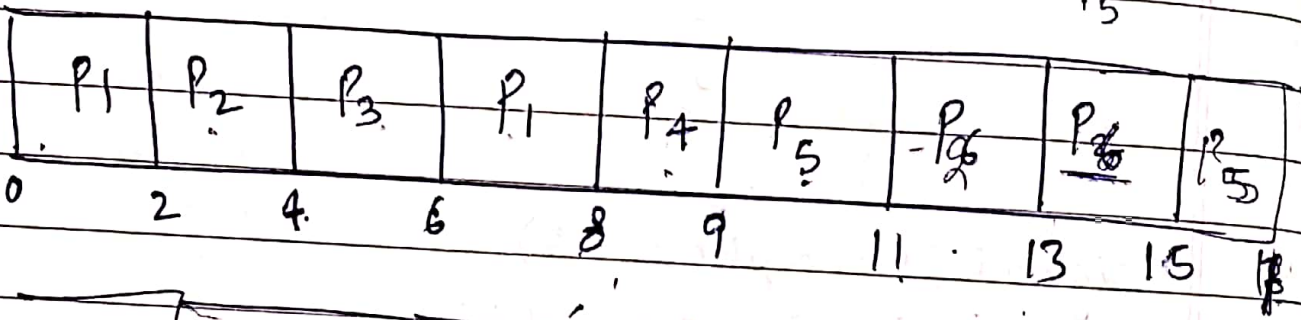
signal (Semaphore * S)
{ S -> value ++;
if (S -> value <= 0)
{ remove a process P from S -> list;
wakeup(P);
}
}

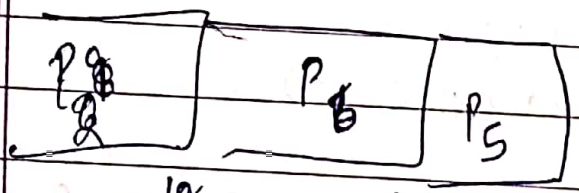
Time Quantum = 2

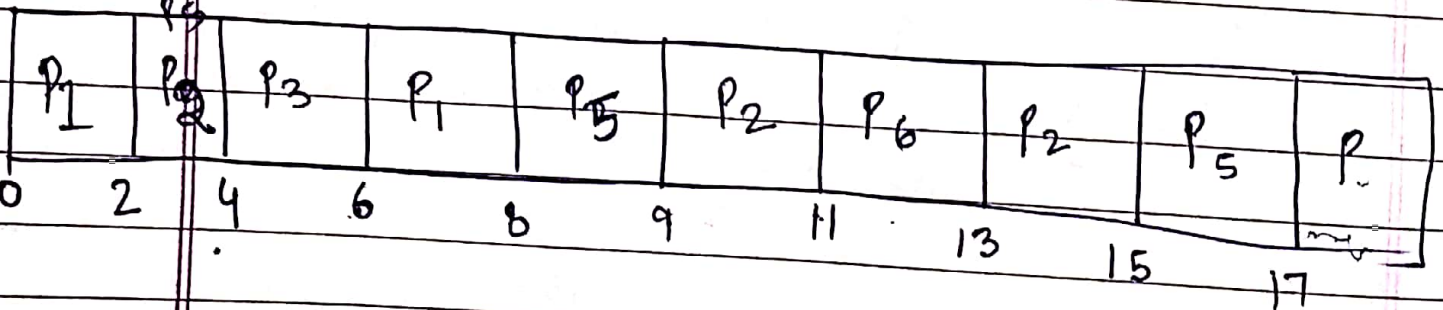
DATE: __/__/__
PAGE NO. _____

P	AT	BT
P_1	0	4
P_2	1	5
P_3	2	2
P_4	3	1
P_5	4	6
P_6	6	3





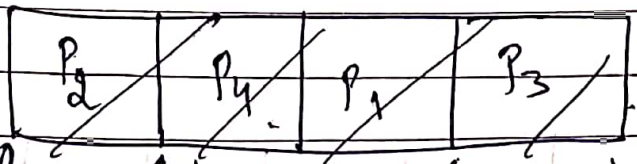




Consider a sys with 4 processes arriving in ready Q in same order at time 0, if burst time of these processes are:

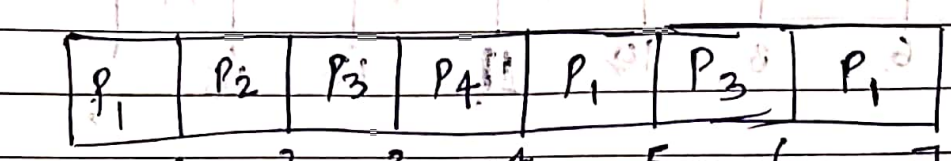
	AT	BT
P_1	0	4
P_2	0	1
P_3	0	8
P_4	0	1

then what is completion time of P_1 & avg time of all p

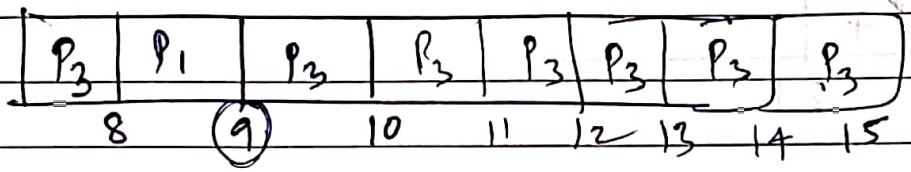


P_1
P_2
P_3
P_4

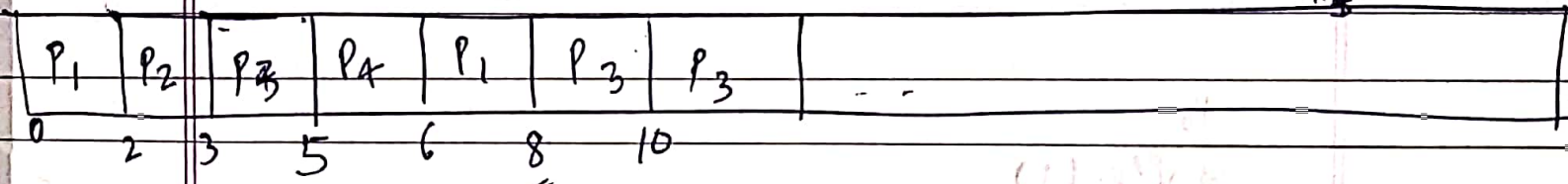
$P_1 = 3$.
$P_3 = 7$.



$P_1 = 2$.



$P_3 = 6$ ✓
$P_1 = 1$
$P_3 = 5$
$P_4 =$

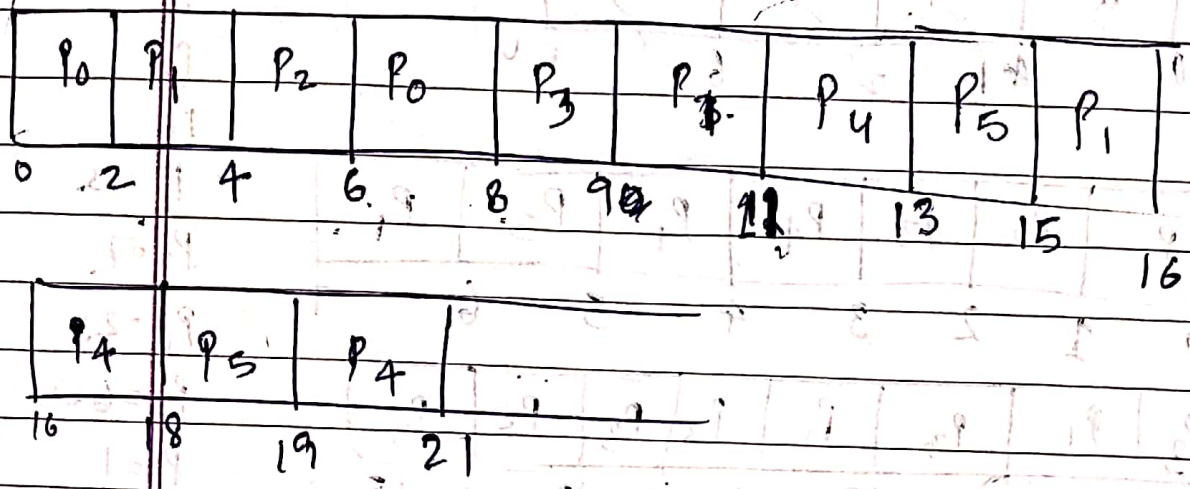


=

Scanned by CamScanner

P_0	0	4
P_1	1	5
P_2	2	2
P_3	3	1
P_4	4	6
P_5	6	3

$q = 2$



P_0 P_1 P_2 P_0 P_3 P_5 P_4 P_5 P_1

$P_0 = 2$

$P_1 = 3$

$P_4 = 4$

$P_5 = 1$

$P_4 = 2$

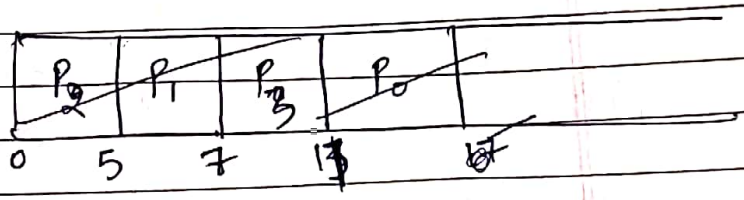
P_0

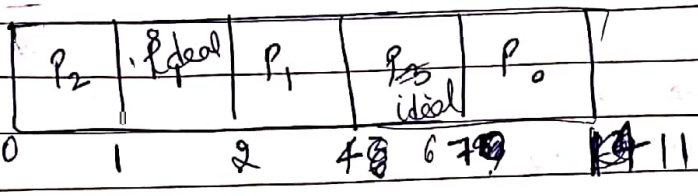
```
while (1)
{   flag[0] = True
    turn = 1
    while ( turn == 1 && flag[1] );
    C.S.
    flag[0] = False
    R.S.
} while(true);
```

Non Preemptive

DATE: DATE PAGE NO.

	BT	AT
P_0	4	7
P_1	2	2
P_2	4	0
P_3	6	6





Deadlock: (in case of processes)

a process request resources if resources are not available at that time, the process enter into waiting state, sometime a waiting process is never again able to change the state because resources it has requested are held by other waiting processes. This situation is called deadlock.

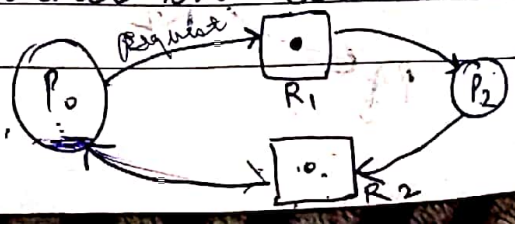
Four conditions to enter into deadlock:

i) Mutual Exclusion : one or more than one resource are non-shareable.

ii) Hold & wait : a process is holding at least one resource & waiting for resources

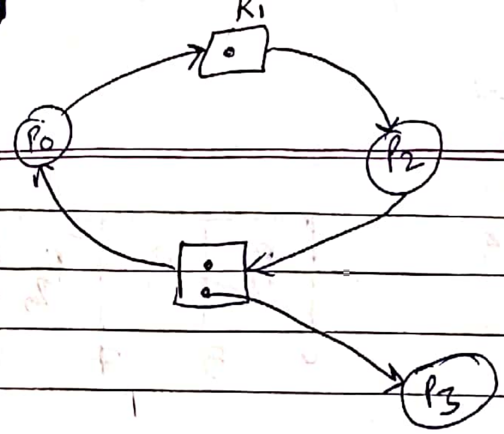
iii) No preemption : a resource cannot be taken from a process unless process release the resource.

iv) Circular weight : a set of processes are waiting for each other in circular form.



Always Resource allocation Graph

DATE: __/__/__ PAGE NO. ____



P3 only holds but not wait so (ii) is not verified

It is not in deadlock condⁿ.

Q. $R_1 \to 3$.
$R_2 \to 5$. hold

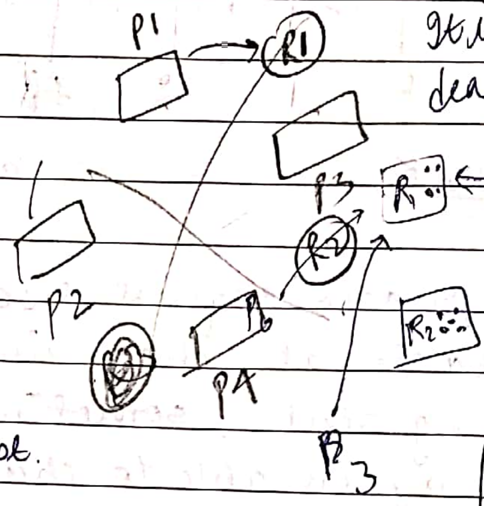
$P_0 - P_5$

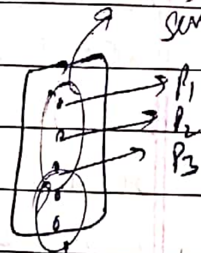
$P_0 \to R_1$

$P_3 \to R_1$

$P_4 \to R_2$ Request

B





Q

P1 ⟶ R1

P1 ⟵ R2

P2 ⟵ R1

P3 ⟵ R1

P3 ⟶ R2

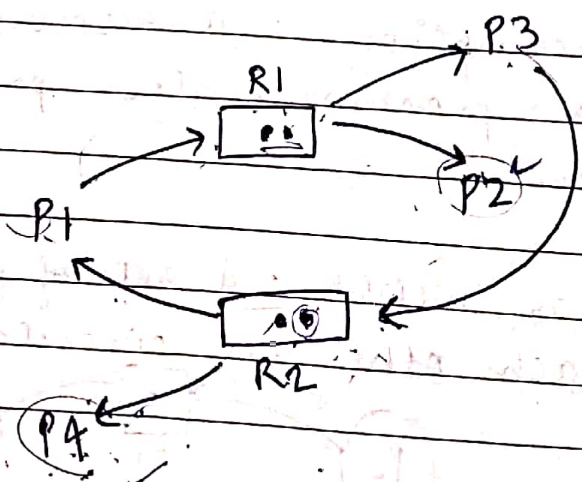
P4 ⟵ R2

there is a cycle but no deadlock bcoz P2 & P4 have all resources for completing

P2, P4, P1, P3

Cycle: $P_1 R_1 P_3 R_2 P_1$

RAG



no deadlock since P2 & P4 are only holding but not waiting

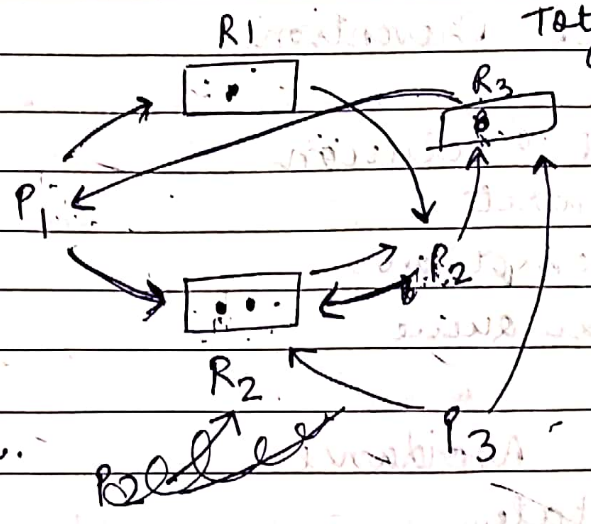
Always ahead

P1 cannot finish bcoz P1 needs R1 but can only acquire R2 while R1 is held by P2
P2 cannot finish bcoz R3 is with P1 but can only acquire R2 while R3 is with P1.
P3 needs R2 & R3 but R2 & R3 is with P1.

Q $P = \{P_1, P_2, P_3\}$, $R = \{R_1, R_2, R_3\}$,

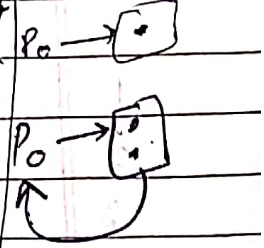
$E = \{P_1 \rightarrow R_1, P_1 \rightarrow R_2, P_2 \rightarrow R_2, P_2 \rightarrow R_3, P_3 \rightarrow R_2, P_3 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_1$

R1 ·
R2 ·
R3 ·



Total no. of cycles: 3

does not satisfy ME.



P2 is only waiting, no holding, so it is deadlock.

	AT	BT	
P1	0	7 = 6 ✓	
P2	1	5 = 4	
P3	2	3 = 2	✗
P4	3	1	✗ ✓
P5	4	2 = 1	✗
P6	5	1	✗ ✓

P1	P4	P6	P5	P3	P2
0	7	8	9	11	14

Preemptive:

P1	P2	P3	P4	P5	P6	P5	P3	P2	P1
0	1	2	3	4	5	6	7	9	13

(end: 19)

~~Non-preemptive~~

Non Preemptive

P1	~~P2~~	~~P3~~	~~P4~~	~~P5~~	~~P6~~
0	7	12	15	16	18

(end: 19)

DATE: __/__/__
PAGE NO. ____ Pa

Methods for handling Deadlock.

to ensure that deadlock never occur system can use either a deadlock prevention or deadlock avoidance scheme.

Deadlock Prevention

1. Mutual exclusion
2. Hold & wait
3. No preemption
4. Circular queue.

15 marks.

** Deadlock Avoidance:
→ Safe state
→ resource allocation graph algo
→ Banker's Algo.

(1.) Safe State

A state is safe, if system can allocate resources to each group in some order & still avoid a deadlock

A system is in safe state if there exist a safe sequence.

A sequence of processes $P_1, P_2, \ldots P_n$ is a safe sequence for the current allocation state if for each P_i, resource that P_i can request can be satisfied by currently available resources & resources held by P_j * where ($j < i$)

if safe : no deadlock
unsafe: deadlock

DATE: __/__/__ PAGE NO. ____

Process	Maxm need	current need	F-future Need-alloc
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Total = 9 ⇒ 12 − 9 = 3

We consider a sys. with 12 magnetic tape drive

$P_1 = 2+2 = 4+1 = 5$ ✓

12 − 9 = 3 (1) resources are left

$P_0 = 5+5 = 10$

(3)

$P_2 = 10+7 = 17$

$P_1 P_0 P_2$

Suppose there are four processes in execution with 12 instances of a resource R in a system. The maxm need & current allocation of processes is given in table. What is safe seq.

	Maxm	Current	F-A
P_1	8	3 + 4 = 12	5
P_2	9	4 + 5 = 9	5
P_3	5	2 + 3 = 5	3
P_4	3	1 + 3 = 4	2

10

12 − 10 = 2

$P_4 = 1+2 = 3$

$P_3 = 4+3 = 7$

$P_1 = 5+3 = 5$

P_2

$P_4 P_3 P_1 P_2$

$P_4 P_3 P_2 P_1$

Always ahead...

DATE: __/__/__
PAGE NO. ____

Resource Allocation Graph Algorithm

$P_i \longrightarrow R_i$ request

$R_i \longrightarrow P_i$ assignment

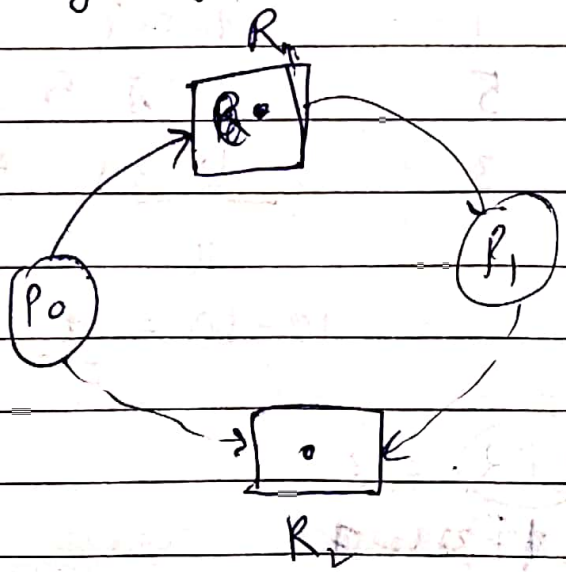
If claim edge : which can request in future

- - - - - - →

If we have Resource allocation System with only one instance of each resource we use resource allocation graph for deadlock - in add" req & assign

We introduce a new edge called claim edge. A claim edge P_i to R_j indicates process P_i may request a resource R_j in some time at future. this edge is represented by dotted or dashed line. when process P_i request resource R_j the claim edge P_i to R_j is converted to request edge

Ulty when a resource R_j is released by P_i, the assignment edge R_j to P_i is converted to claim edge.



DATE: __/__/__
PAGE NO. ____

Banker's Algorithm

A RAG algo is not applicable to a resource allocation system with multiple instances of each resource type. The deadlock avoidance algorithm (banker algorithm) is applicable to such a system but is less efficient than the resource allocation graph scheme.

1. Available : a vector of length m indicates no. of available resources of each type. if :
   if available[j] = k, then k instances of resource type Rj are available

2. Max : an n x m matrix defines max demand of each process.

3. Allocation : n x m matrix defines the no. of resources of each type allocated to each process.

4. Need : an n x m matrix indicates remaining resources need of each process.

Safety Algo:

Let work & finish be vectors of length m and n

1. work = available;
   finish[i] = false;
2. for (i = 0 to n-1)
3. Find an index i such that both a) finish[i] = false &
   b) need[i] <= work;
   if no such i exist goto step 4
4. work = work + allocation
   finish[i] = true.

Always ahead...

4. if finish[i] = true for all i
then sys. is in safe state.

DATE: / /
PAGE NO.

5×3

(5×3)

	Allocation A B C	Max A B C	Available A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0 (+0) 3 0 2	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	
~~P_5~~	~~+0 2~~		

Resource type A have 10 instances
B have 5 instances
C have 7 instances.

work = 3 3 2 (250) = 332

$W = W + A$

$332 + 200$

	Need A B C	
P_0	7 4 3	20
P_1	~~1 2 2~~	X 020
P_2	6 0 0	
P_3	0 1 1 ✓	
P_4	4 3 1	

work = [3 3 2] work = 5 3 2 ✓ (2 1 1)

~~finish[0] = false~~

P_0 false — 7 4 3 P_1

✓ P_1 — P_3

B work = work + allocation — P_4

[3 3 2] + [2 0 0]

[5 3 2]

work = [5 3 2]

P_0 → false

P_3 → ✓

work = work + allocation

~~7 3 2 + 3 3 2~~

~~[10 6 4]~~

= [7 3 2]

P_1 P_3 P_0 P_2 P_4

P_1 P_3 P_4 P_0 P_2

work

[7 4 3] / 6 1 0 = 7 5 4

Always ahead

000 : Request Detection.

DATE: / /
PAGE NO.

Suppose now the process P_1 request one additional instance of resource type A & 2 instances of resource of type B & type C. so request = [A B C] 102 we to decide whether this request can immediately granted.

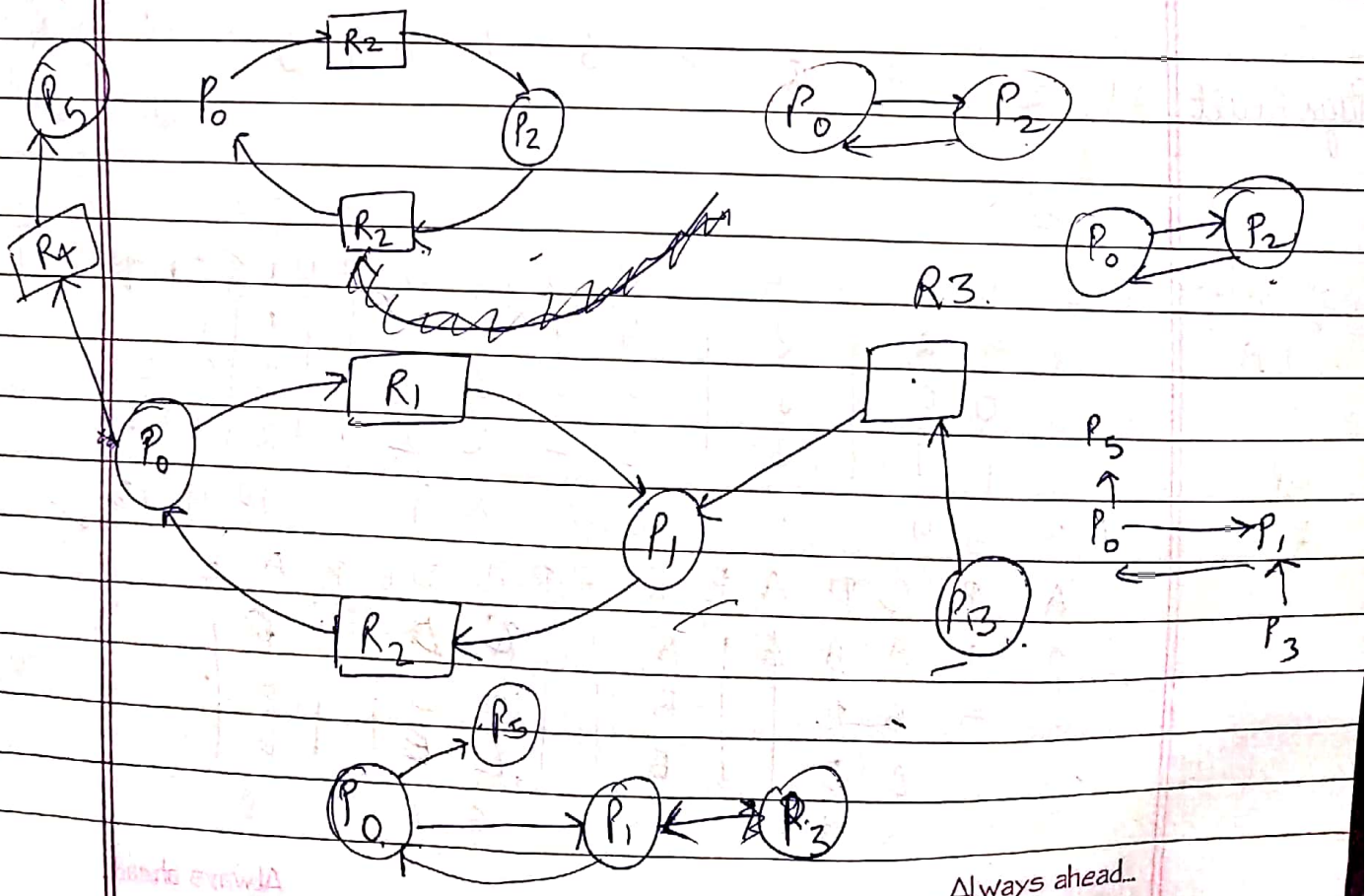
	Allocation	Max	Need	Available
P_0	010	753	743	~~230~~ 230 → 332
P_1	302	322	020	Yes.
P_2	302	902	600	
P_3	211	222	011	
P_4	002	433	431	

332 − 102 = 230

single instance ↳ circle ⇓ Deadlock Always

Deadlock Detection
Deadlock Recovery.

Wait for GRAPH:



Always ahead...

DATE: / /
PAGE NO.

4GB : game.
Ram = 2GB
2GB = Harddisk
↓
V/M

Memory:

(i) Physical memory (RAM)
(ii) Virtual memory (Logical)

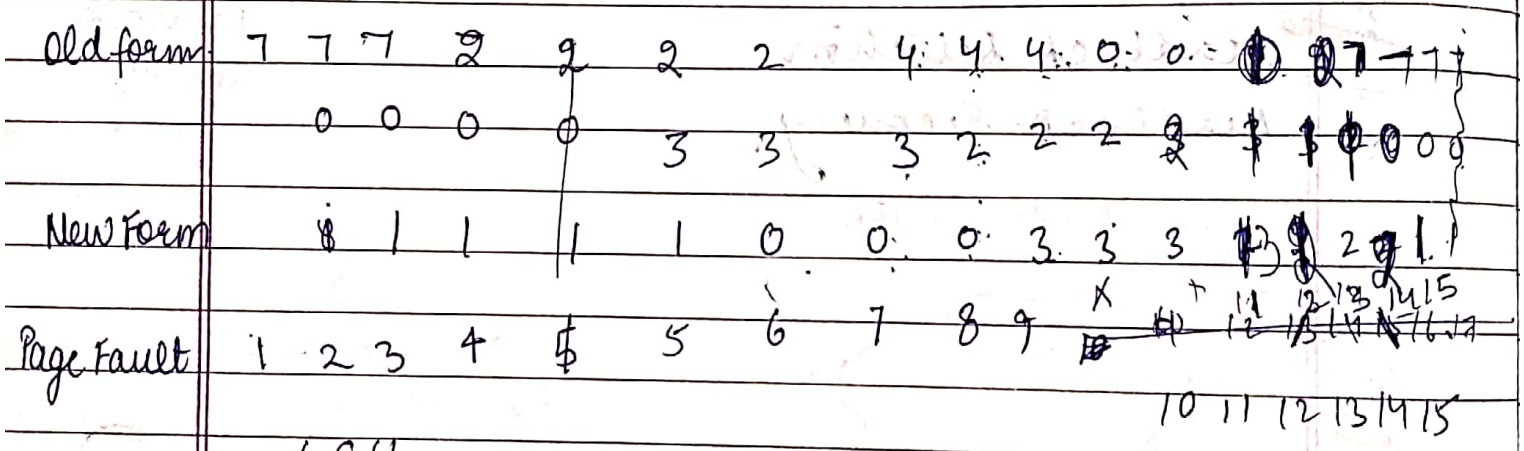
Page Replacement Algorithm

1: Page hit if already present
2: Page fault

FIFO; LRU, Optimal Page Replacement Algorithm
(Least Recently used)

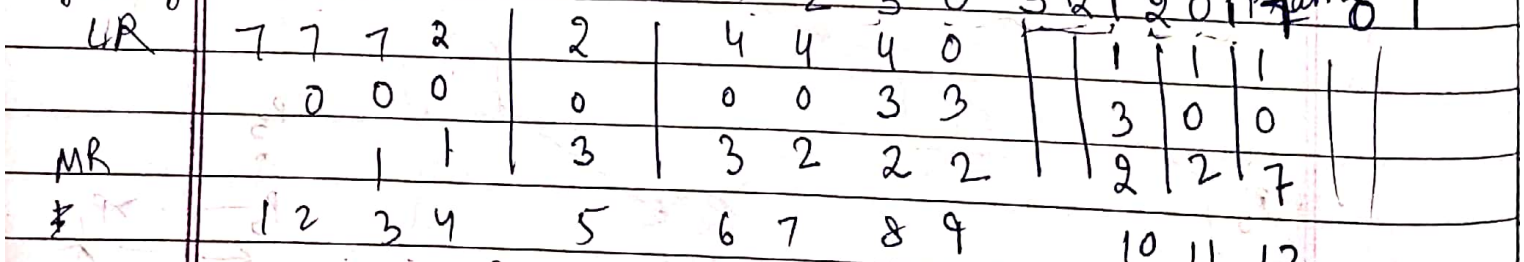
Frame: 3

Ref. string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



LSU

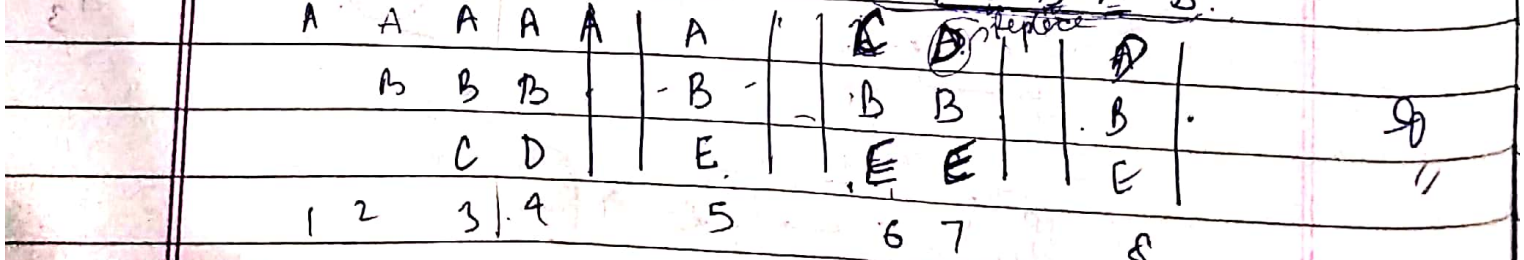
Ref. string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



LSU

Frame: 3

A B C D A B E A B C D E B A B



Optimal

Classic Problem of Synchronization

DATE: / /
PAGE NO.

A B C D A B E A B C D E B A B

Old Form	A	A	A	D	D	D	E	E	E	E	A
		B	B	B	A	A	A	C	C	C	C
			C	C	C	B	B	B	D	B	B
	1	2	3	4	5	6	7	8	9	10	11

FIFO

11

New Form By Faults	A	A	A	D	D	D	E	C	C	C	B	B
		B	B	B	A	A	A	A	D	D	D	A
			C	C	C	B	B	B	E	E	E	
	1	2	3	4	5	6	7	8	9	10	11	12

LRU

12

0 2 3 1 4 2 5 6 0 1 3 2 4 7 1 2

0	0	0	1	1	1	5	5	5	1	1	1	4	4	4	2
	2	2	2	4	4	4	6	6	6	3	3	3	7	7	7
		3	3	3	2	2	2	0	0	0	2	2	2	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

0 2 3 1 4 2 5 6 0 1 3 2 4 7 1 2

0	0	0	0	4	4	4	4	4	4	1	1	1	1	7	7
	2	2	2	2	5	5	5	5	3	3	3	3	2	2	2
		3	3	3	3	6	6	6	6	2	2	2	4	4	4
			1	1	1	1	0	0	0	0	4	4	4	1	1
1	2		3	4	5	6	7	8	9	10	11	12	13	14	

0 2 3 1 4 2 5 6 0 1 3 2 4 7 1 2 — LRU

0	0	0	0	4	4	4	4	0	0	0	0	4	4	4	
	2	2	2	2	2	2	1	1	1	1	1	7	7		
		3	3	3	5	5	5	5	3	3	3	3	1		
			1	1	1	6	6	6	6	2	2	2	2		
1	2		3	4	5	6	7	8	9	10	11	12	13	14	

Always ahead...

Belady's Anomaly

as no. of frames ↑, no. of pg fault ↓.

Method of storage.

CPU

↓

Register

↓

Cache L1, (L2, L3) → external to CPU

↓

RAM

↓

HDD.

Address Binding: A program reside on a disk as a binary executable file to execute a program it must be brought into Main memory & placed within a process.

Depending on the memory management in use the process may be moved b/w disk & memory during its execution. The processes on disk that are waiting to ~~brought~~ being into memory for execution from Input Queue

→ Compile time
→ Load time
→ Execution binding time

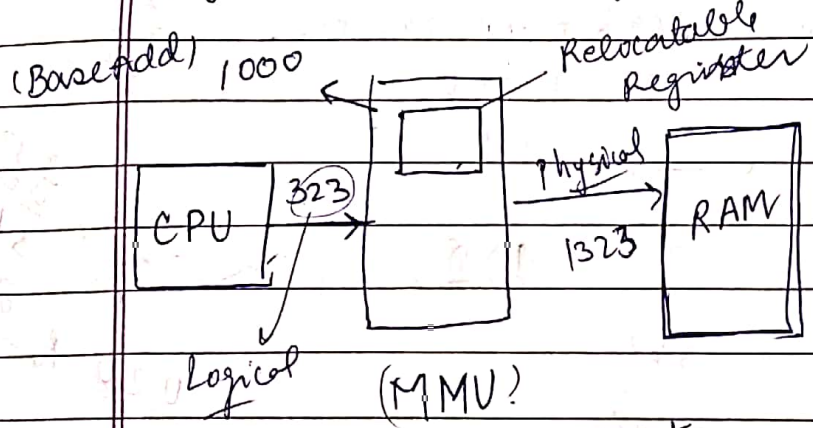
(1) Compile time : if you know at compile time where the process will reside in memory then absolute code can be generated.

Always ahead..

2. Load time : if it is not known at compile time where the processes will reside in memory than the compiler must generate ~~relocatable code~~.
In this case final binding is delayed until load time.

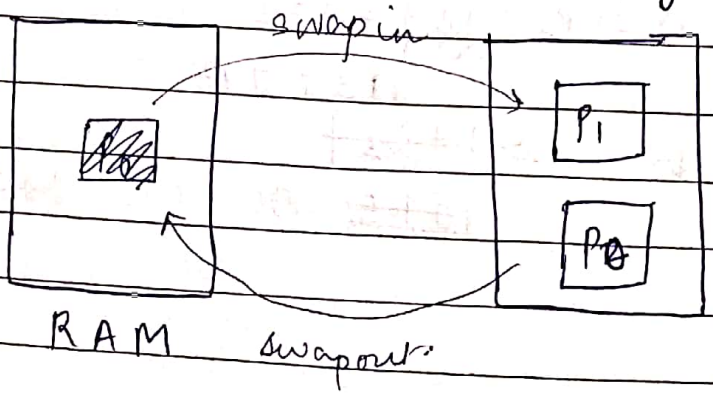
3 Execution time : if the process can be moved during its execution from one memory segment to another than binding must be delayed until the run time

Logical add v/s Physical address Space



(MMU)
memory management unit.
→ runtime mapping from virtual to physical address is d

SWAPPING: A process must be in memory to b executed. A process can be swapped out of memory to a backing store (secondary memory) & then brought back into memory for continued execution. Swapping makes it possible for total physical add spa of all process to exceed the real physical memory of sys.



draw in diff states.
or
draw like this

Non/ ~~contagious memory Allocation~~

Banker's Algo

~~Available = work;~~

work = available
Finish[i] = False;
Need[i] ≤ work;
Finish[i] = true;
work = work + Allocation

	Allocation	Max	Avail	Need
P0	0 1 0	7 5 3	3 3 2 / 3 3 0	7 4 3
P1	2 0 0	3 2 2	0 0 2	1 2 2
P2	3 0 2	9 0 2	3 3 2 / 0 2 0 / 3 1 2	6 0 0
P3	2 1 1	2 2 2		0 1 1
P4	0 0 2	4 3 3	1 0 1	4 3 1

+ 0,20 by P0

330 by P4 no

work = 332

P1 ✓ ; work = 200 + 332 = 532

P3 ✓ work = 211 + 532 = 743

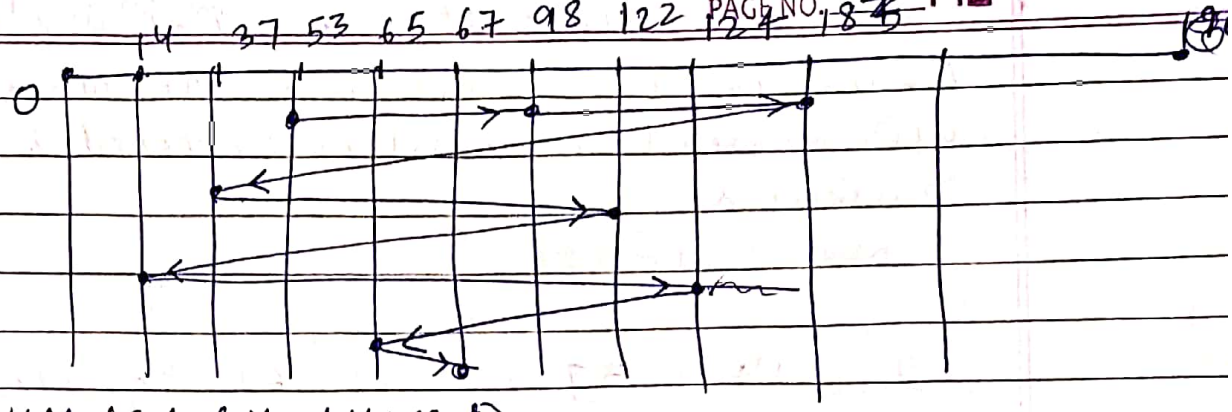
P4 ✓ w = ~~211 + 834~~ = ~~1045~~ 002 + 743 = 745

P0 ✓

P2

Safe sequence: P1, P3, P4, P0, P2

DATE: / /
PAGE NO.



THM (Total Head Moment)

(98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

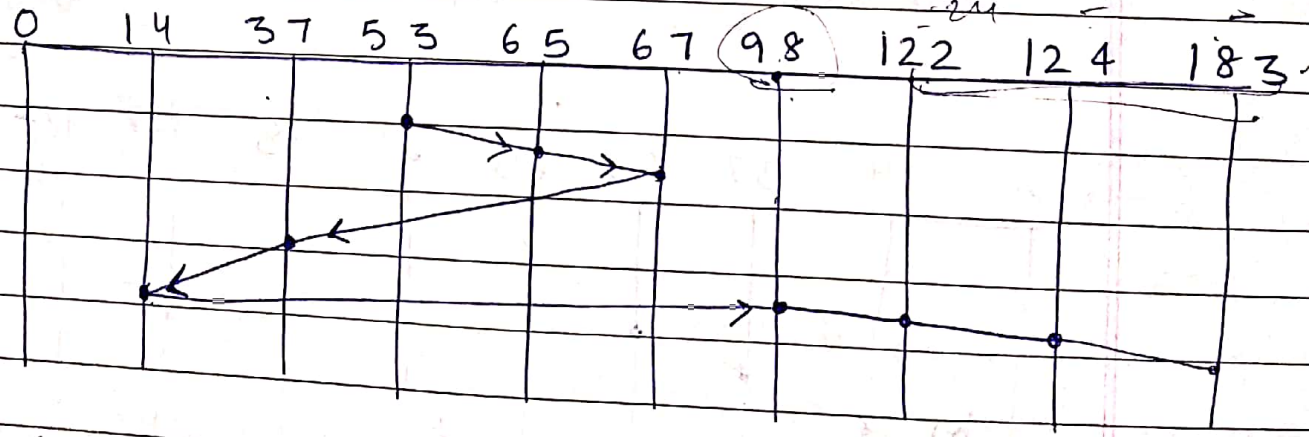
45 + 85 + 146 + 85 + 108 + 110 + 59 + 2

= 640. Ans.

(2) SSTF (Shortest Seek-time first).

Select the request with min^m seek time for current head position better than FCFS Algo bcoz less no. of head movements.

98, 183, 37, 53, 122, 14, 124, 65, 67.



(65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124)

12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236

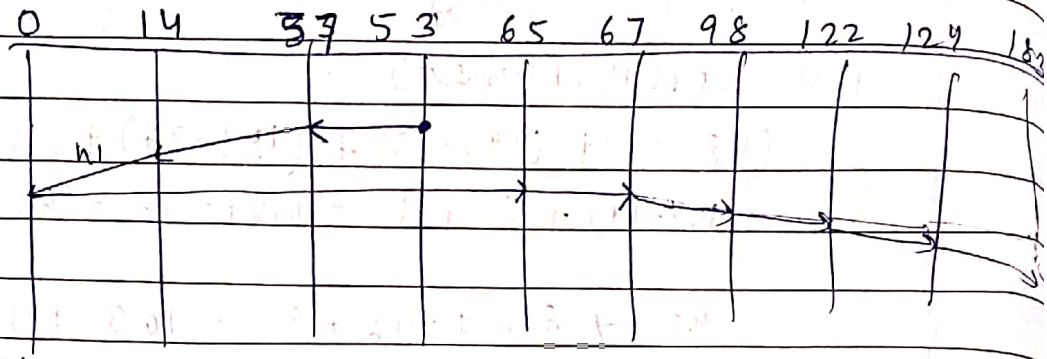
3. **Scan Algorithm:**

Also known as elevator

DATE: __/__/__
PAGE NO.:

Disc arm start at one end of disc & moves toward other end. At other end dir of head moment is reversed & searching continues.

~~head start at 40~~



THM

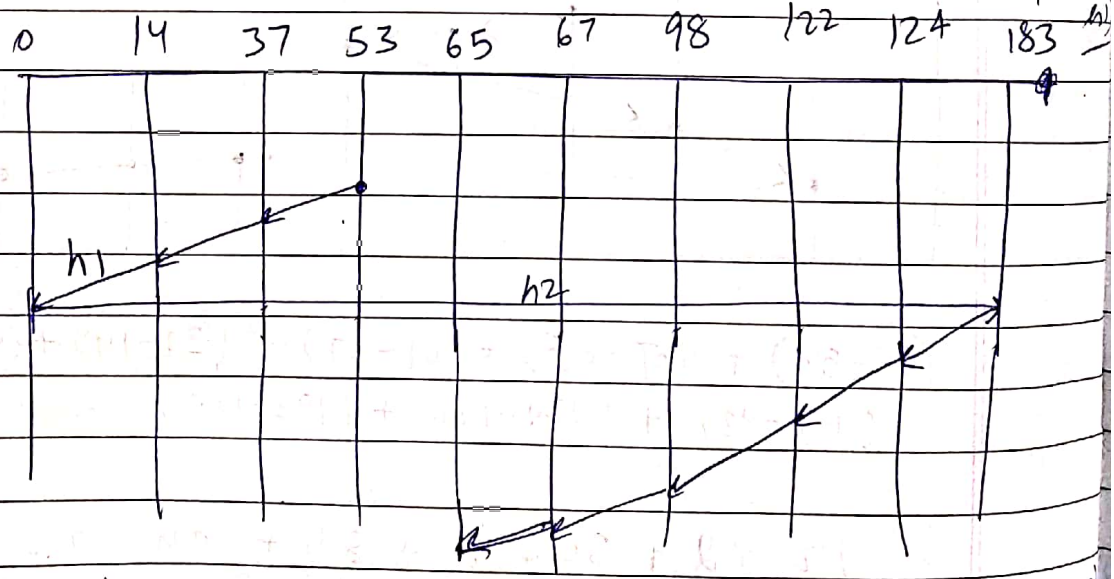
(53-37) + (37-14) + (14-0) + (65-0) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124).

⇒ 16 + 23 + 14 + 65 + 2 + 31 + 24 + 2 + 59

= 236.

C-Scan Algorithm

If given range till 190 then go to 190 to 0



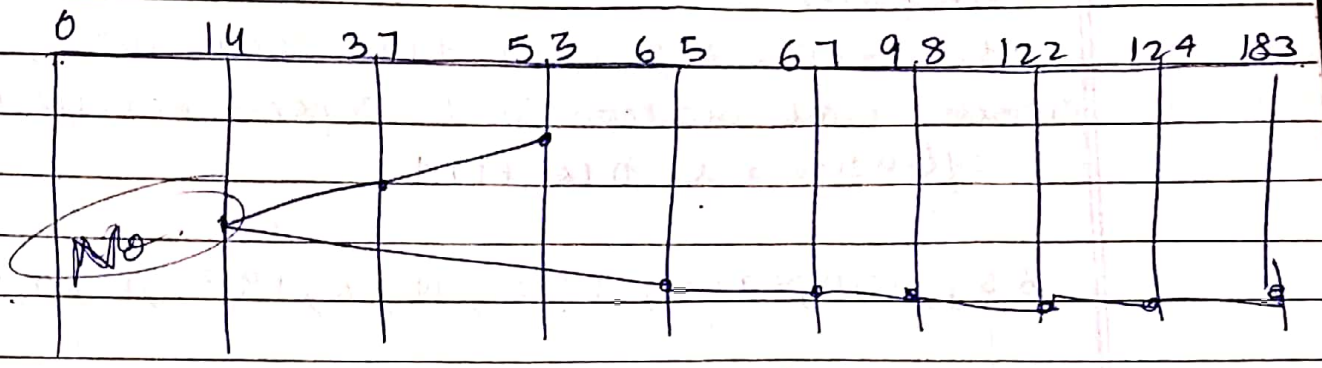
(53-37) + (37-14) + (14-0) + (183) + (183-124) + (124-122) + (122-98) + (98-67) + (67-65)

16 + 23 + 14 + 183 + 59 + 2 + 24 + 31 + 2 = 354

Always ahead...

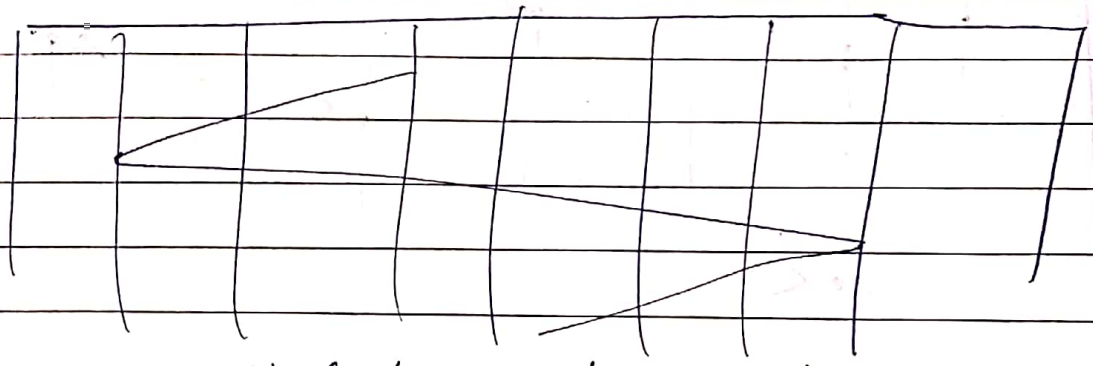
4. Look Scheduling Algorithm.

DATE: / /
PAGE NO.



= 208.

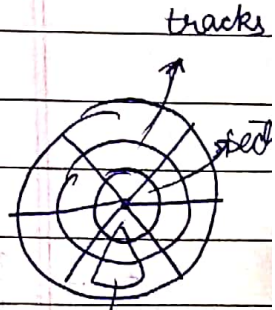
C-look.



No Extra head movement.

Assignment:

i) Mass storage Structure Magnetic disk
ii) Solid state disk
iii) Magnetic tapes.
iv) Solid state drive
v) Disk Scheduling.
vi) Disk Management (disk formatting, boot block)
vii) Swap space management.



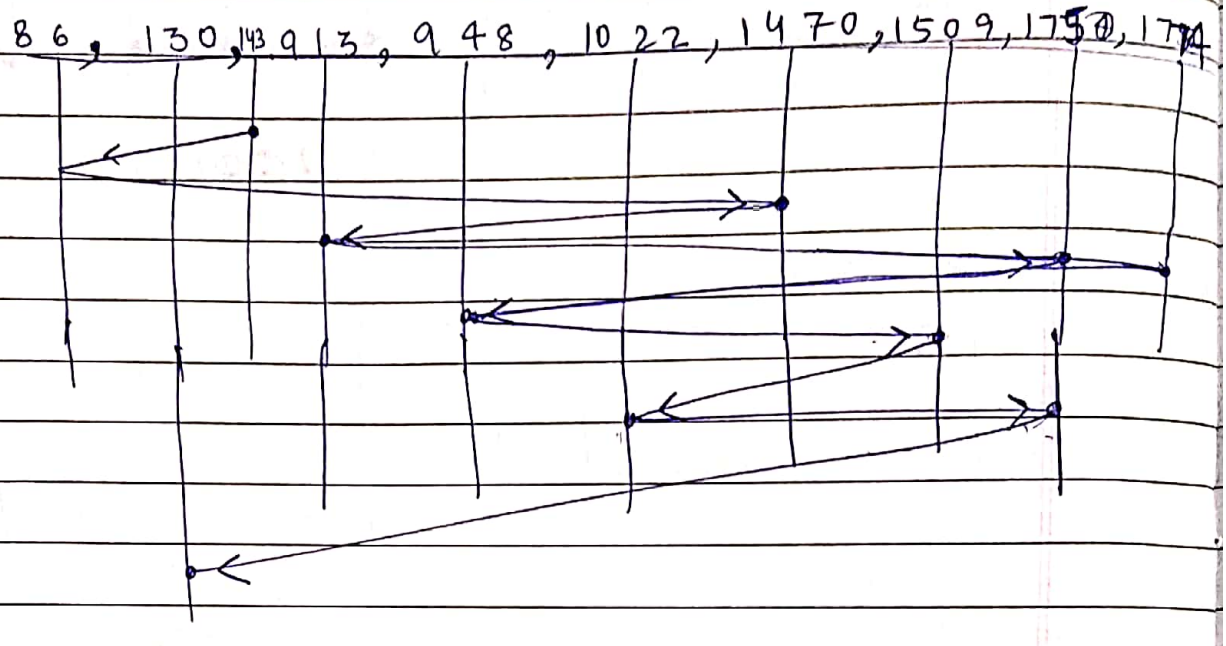
SATA
PATA.

DATE: / /
PAGE NO.:

Work queue :

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

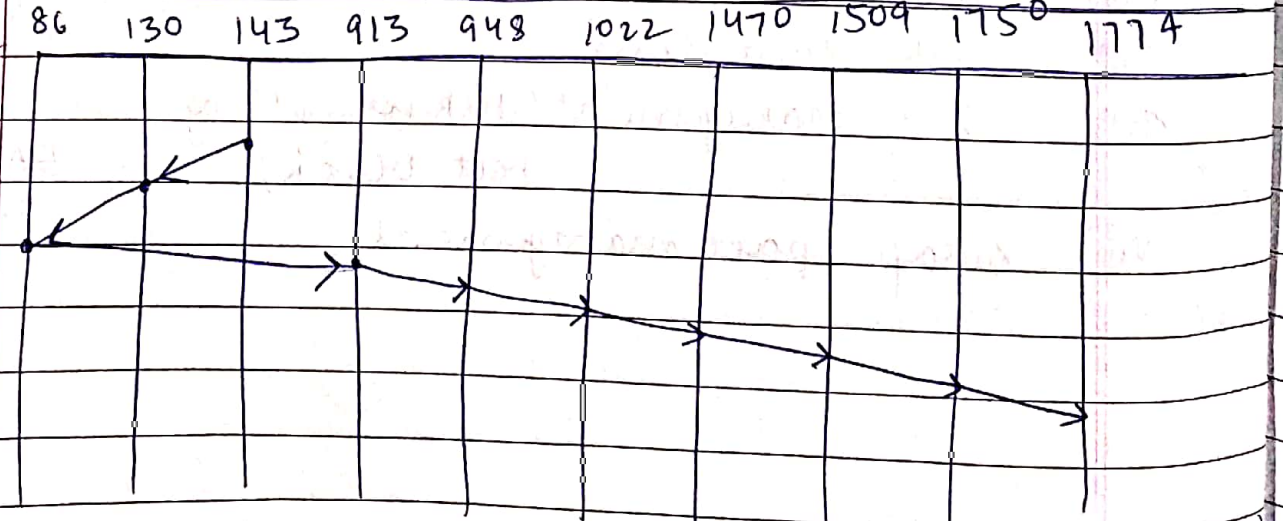
Current head position is 143 & prev Req = 125 & total cylinders are 0 to 4999



(143-86) + (1470-86) + (1470-913) + (1774-913) + (1774-948) + (1774-1509) + (1509-948) + (1509-1022) + (1750-1022) + (1750-130)

57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620 = 7081.

⑧ #



(143-130) + (130-86) + (913-86) + (948-913) + (1022-948) + (1470-1022) + (1509-1470) + (1750-1509) + (1774-1750)

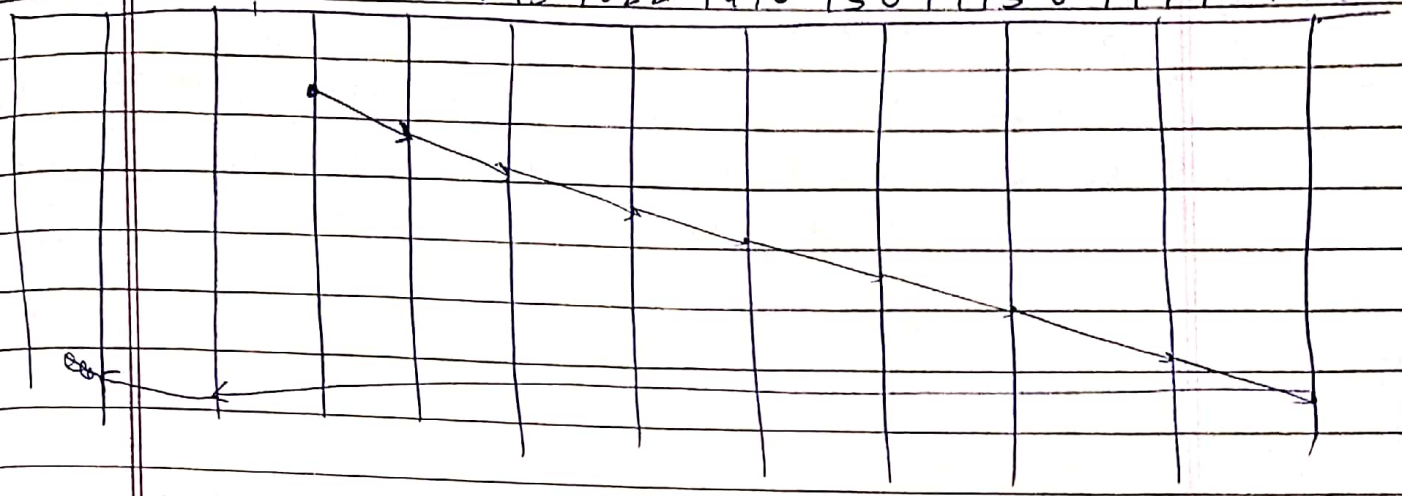
always ahead...

$13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24$
$= 1745.$

DATE: / / Aa
PAGE NO.

#. 86 130 143 913 948 1022 1470 1509 1750

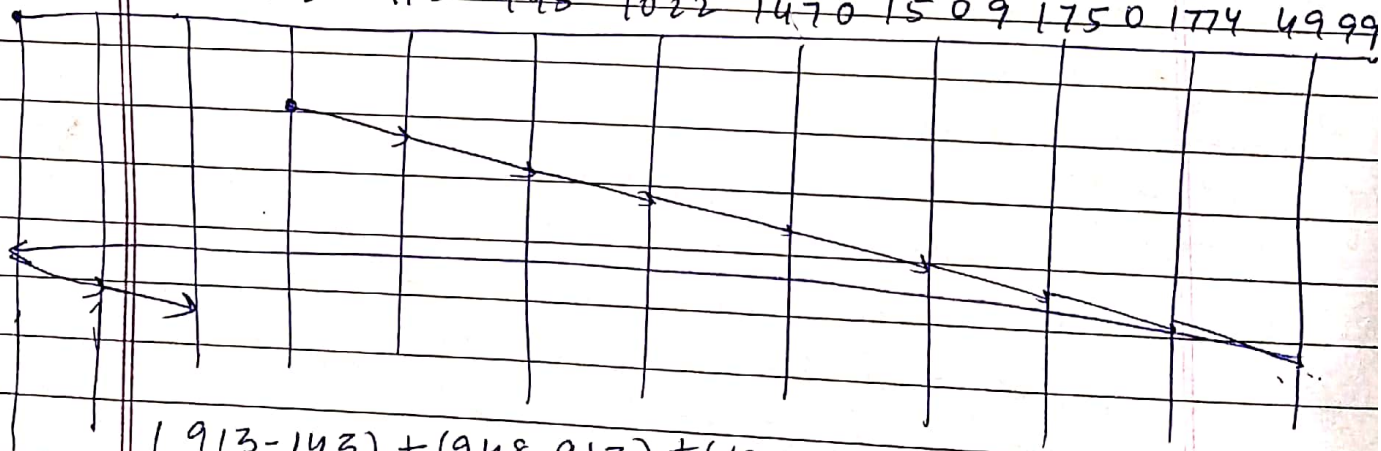
0 86 130 143 913 948 1022 1470 1509 1750 1774 4999



$(913-143)+(948-913)+(1022-948)+(1470-1022)+(1509-1470)$
$+(1750-1509)+(1774-1750)+(4999-1774)+(4999-130)$
$+(130-86)$

$770 + 35 + 74 + 448 + 39 + 241 + 3225 + 24$
$+ 4869 + 44$
$= 9769.$

0 86 130 143 913 948 1022 1470 1509 1750 1774 4999



$(913-143)+(948-913)+(1022-948)+(1470-1022)+$
$(1509-1470)+(1750-1509)+(1774-1750)+(4999-1774)$
$(4999-0)+(86-0)+(130-86).$

$770+35+74+448+39+241+24+3225+86+4999$
$+44.$

4856
$4999+130$

9985

Always ahead...

buffer: is a region of memory used to temporarily hold data while it is being moved from one place to an

DATE: PAGE NO.

& cache: is a temporary storage area where frequently accessed ~~data can be stored~~

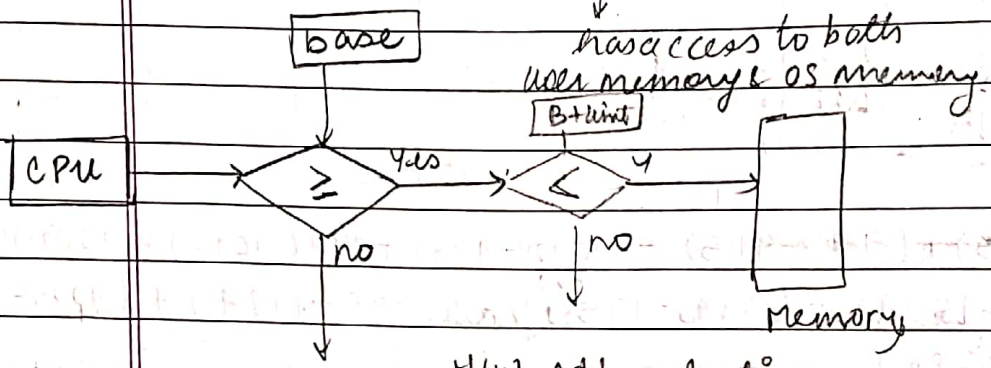
Basic H/W : CPU can access registers & MM only even instructions are passed using MM's add as arguments. Registers are usually inside CPU which can be accessed early than MM.

Correct operation

Process may access only legal address : add provided specifically for Proc

This is achieved by 2 Registers : Base Register & limit

Loaded by only OS in Kernel mode. Smallest add. largest of range.



H/W Add protection.

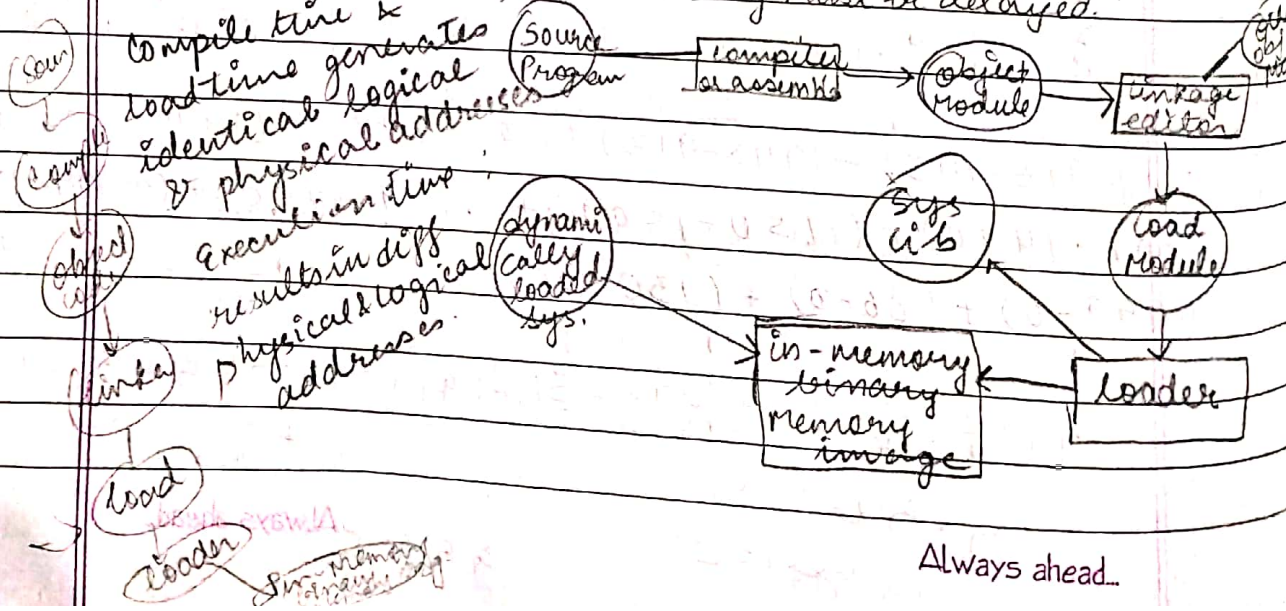
Address Binding: Process wait in input queue & as ~~they can be~~ process from queue gets loaded into MM, it is then executed & finally terminates & its M space is declared available.

before executed program goes through many steps: source program add have symbolic links are which are bind together to relocatable add. This linkage will in turn bind the relocatable add to absolute address.

→ Compile time: if know where process is; then absolute code is generated

load time → if at compile time it is not known where process is then compiler generates relocatable code, Final binding is delayed.

Execution Time: if process can be moved during its execution from one memory segment to another, then binding must be delayed.



Always ahead...

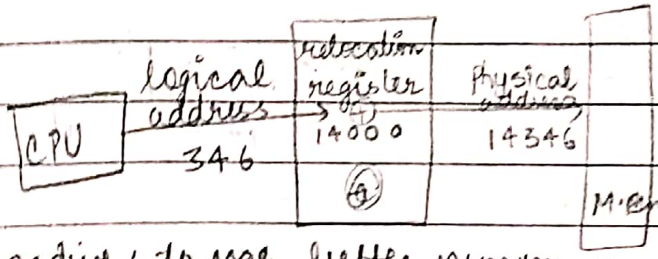
address by CPU : logical address → virtual address
loaded into the memory address register : physical add
execution time, logical & physical address spaces differ.

DATE: / /
PAGE NO. Aa

mapping from virtual to physical addresses is done by a h/w device called MMU (Memory management unit

Logical add is generated by user & real physical address cannot be seen by user.

user can create pointer to loc (346), store it, manipula it, & compare it with other addresses.



Dynamic loading : to use better memory ~~memory~~ space utilization

Routines are not loaded until called by program, all routines are kept on disk in a relocatable load format & when there is requirement of that routine, first it is checked if it has loaded or not, then it is loaded into memory & update programs address tables to reflect this change.

For refering symbol, we need their address, but at time of writing code or compiling, we have no idea, where these symbols are will be placed. So they are given a relative address w.r.t. starting of a section & later resolved, during loading phase to absolute memory address.

these relative address given to symbols are relocatable address. they can be later assigned an absolute address, hence relocatable.

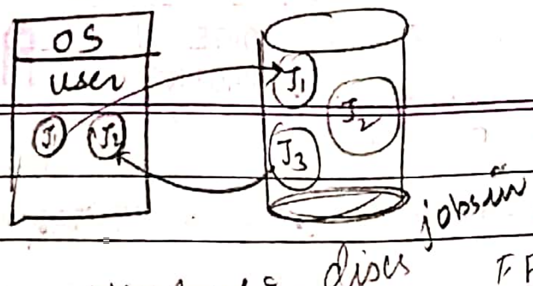
Swapping :

requires backing store : a fast disk.

when CPU scheduler decides to execute a process, it calls the dispatcher. checks to see whether the next process in the queue is in memory & then it swaps out & swaps in desired process.

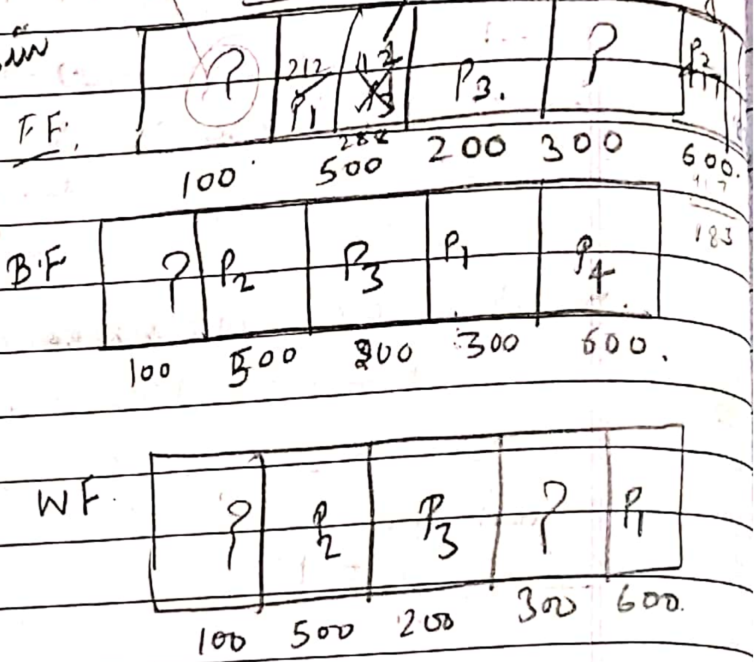
Q. user process = 100 MB, transfer rate of 50 MB/sec
context switch time = 100/50 = 2 sec
Always ahead.

Swapping



External Frag → Internal Fragment

In case of fixed Partitioning



Contigous	Non Contigous
Centralised	Decentralised.
as unit is stored	are distributed in mem.
partitioning Method	with Paging, Segmnt, V.M.

Partitioning → Fixed / variable

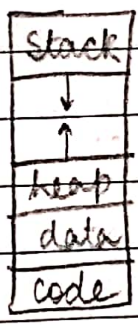
Policies:
- First Fit : allocates
- Best Fit
- Worst Fit

Internal : when it is left behind by process. in paging: p-1

external fragmentation : memory which is not contigous & left as such.

Paging : non-contigous memory allocation
→ Allows physical add space of a process to be non-contiguous
→ Logical add space is divided into equal size pgs.
→ Physical add space is divided into equal size frames

How process is stored in memory



Non-preemptive
process will complete
(i.e. either wait or terminate)
after that CPU will be deallocated

Preemptive
CPU can be taken away.

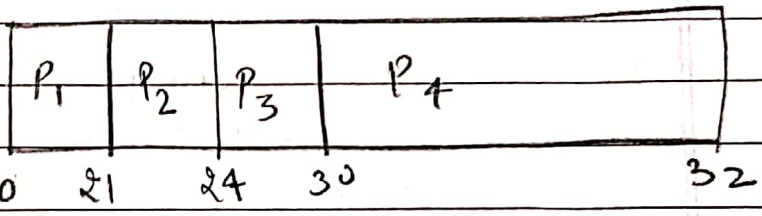
DATE: __/__/__ PAGE NO. ____

TAT	waiting time	Response time
Submission & completion	Average period of time a process spends waiting	Amt of time it takes when a req. was submitted until first resp is

For best sys. performance
CPU utilization, throughput } max^n.
TAT, WAT, RT } min

FCFS (Avg Waiting time is high) (non preemptive)

P	Burst time
P_1	21
P_2	3
P3	6
P4	2



Waiting time

$P_1 = 0$

$P_2 = 21 - 0$

$P_3 = 21 + 3 = 24$

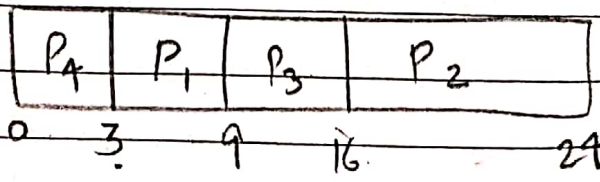
$P_4 = 30$

$$\frac{0 + 21 + 24 + 30}{4}$$

SJF (both Non & preemptive) min. wait

SRTA.
(SJP)

	Non preempt
P_1	6 ✓
P_2	8
P_3	7
P_4	3 ✓



wt.

P_4 0.

P_1 3

$P_3 = 9$

$P_2 = 16$

$$= \frac{0 + 3 + 9 + 16}{4}$$

Always ahead...

Preemptive SRTF.

	AT	BT
P_1	0	8 = 7
P_2	1	4 - 1 = 3
P_3	2	9.
P_4	3	5

$P_1 = 0 + 10 - 1 = 9$

$P_2 = 0$ ✓

$P_3 = 17 - 2 = 15$

$P_4 = 5 - 3 = 2$

DATE: / /

PAGE NO.

P_1	P_2	P_2	P_2	P_4	P_1	P_3

0 1 2 3 5 10 17 26

Avg: $\frac{9+0+15+2}{4} = \frac{26}{4}$ = 6.5 ms

RR (Always Preemptive)

P_1	21 = 16 = 11 Quantum = 5
P_2	3 ✓
P_3	6 = 1
P_4	2

P1	P_2	P_3	P_4	P_1	P_3	P_1	P_1

0 5 8 13 15 20 21 26 31

$P_1 = 0 + (15-5) + (21-20) + (32-21)$

$P_2 = 5$ ✓

$P_3 = (8-0) + (20-13) = 8 + 7 = 15$ ✓

$P_4 = (13-0) = 13$ ✓

$\frac{5 + 13 + 15 + 11}{4} = \frac{44}{4} = 11$

	BT	Priority
P_1	21 ✓	2
P_2	3 ✓	1
P_3	6	4
P_4	2.	3

(Preemptive & NP).

NP

P_2	P_1	P_4	P_3

0 3 24 26 32.

$P_1 = (32) = 11$ 34

$P_2 = 0$ 0 ✓

$P_3 = (26 - 4)$ 22 26

$P_4 = (24 - 3)$ 21 24

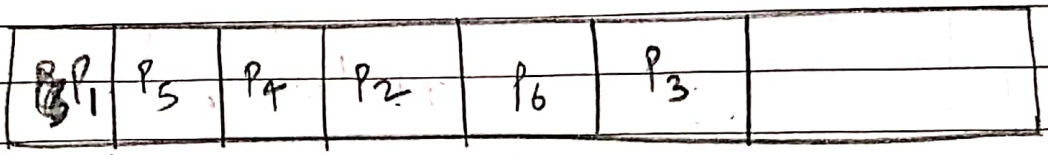
low priority process is blocked in starvation

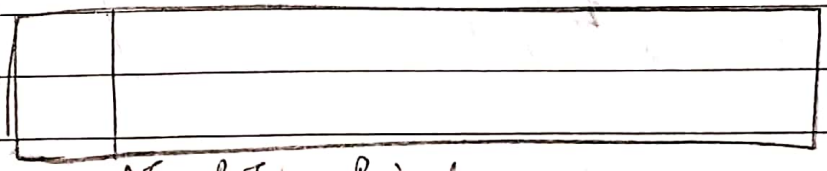
Aging ↑ priority of a process.

Non.

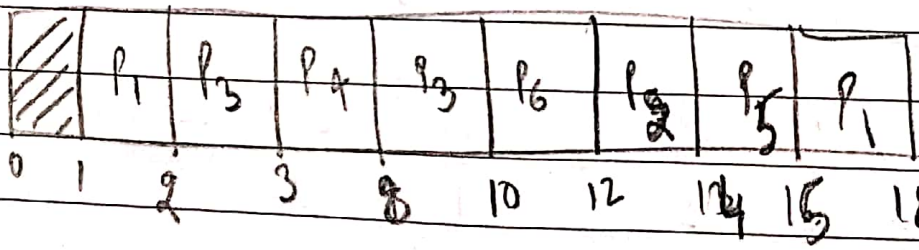
DATE: __/__/__

	AT	BT	Priority	Completion Time	TAT	WT	RT
P1	0	4	4	4	4-0=4	0	0
P2	1	5	5	14	14-1=13	8	8
P3	2	1	7	27	27-2=25	24	24
P4	3	2	2	9	9-3=6	4	4
P5	4	3	1	7	7-4=3	0	0
P6	5	6	6	20	20-5=15	9	9





	AT	BT	Priority
P1	1	4-1=3	4
P2	2	2	5
P3	2	3-1-2	7
P4	3	5-1-4	8
P5	3	1	5
P6	4	2	6



	AT	BT	WT
P1	0	7-1=6	0
P2	1	5-1=4	(2-1)+(9-2) = 1+7 = 8
P3	2	3	(3-2)+(7-3) = 1+4 = 5
P4	3	1 ✓	(4-3) = 1 = 1
P5	4	2	(5-4)+(6-5) = 2 = 2
P6	5	1 ✓	(6-5) = 1 = 1

DATE: / / PAGE NO.: A2

P1	P2	P3	P4	P5	P6	P5	P3	P2	P1

0 1 2 3 4 5 6 7 9 13 19

Avg WT = 17/6 = 2.

P1	P2	P3	P4	P3	P6	P5	P2	P1

0 1 2 3 4 6 7 9 13 19

	AT	BT
P1	2	5-1=4
P2	3	13-2=11
P3	0	8-1=7
P4	5	4
P5	1	10-1=9

P3	P5	P1	P2	P4	P1	P3	P5	P2

0 1 2 3 5 9 13 20 29 30

	AT	BT
P1	8	1
P2	5	1
P3	2	7-1=6
P4	4	3-1=2
P5	2	8
P6	4	2-1=1
P7	3	5-1=4

P1	P2	P3	P4	P5	P6

0 7 12 15 16 18 19

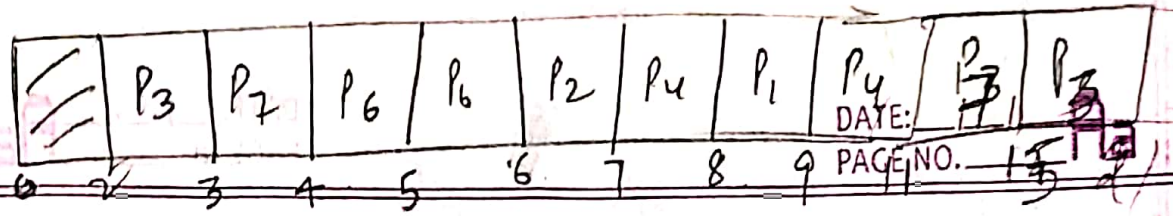
P3	P1	P2	P6

2 3 10 12

P3	P7	P6	P2	P6	P4	P4	P7	P3

0 1 3 4 5 6 8 9 12 16

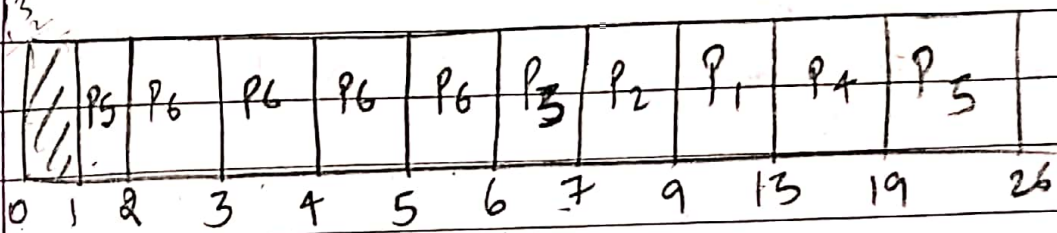
Always ahead...



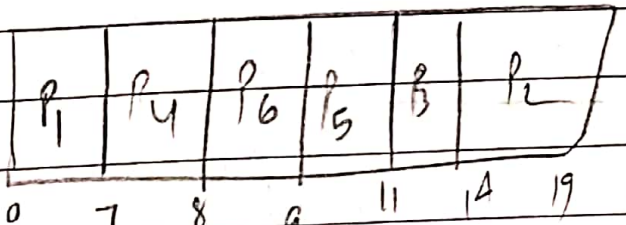
DATE:
PAGE NO.

Q.

Process	AT	BT.	CT	TAT (CT-AT)	WT (TAT-BT)
P1	3	4	13	10	6
P2	4	2	9	5	3
P3	5	1	7	2	1
P4	2	6	19	17	11
P5	1	8-1=7	(26)	25	17
P6	2	4-1=3-1=2-1=1	6	4	0

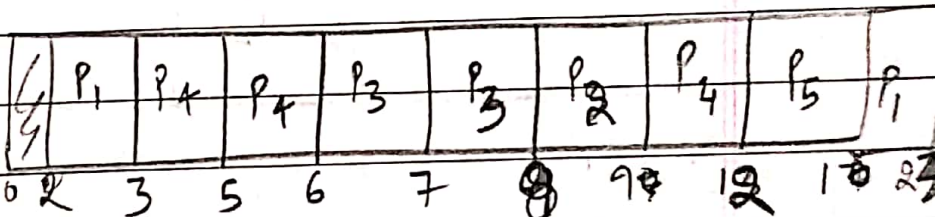


10
17
11
38



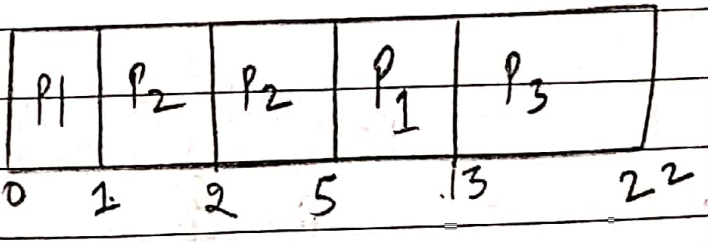
Q

	AT	BT
P1	2	8-1=7
P2	7	1
P3	6	2-1=1
P4	3	6-2=4-1=3
P5	5	4

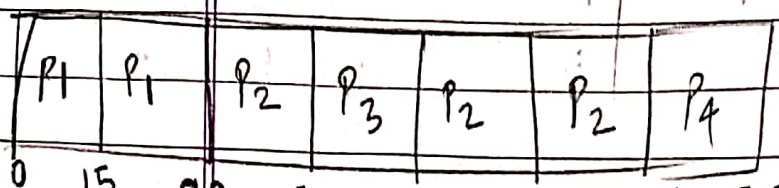


Q

	AT	BT
P1	0	9-1=8
P2	1	4-1=3
P3	2	9

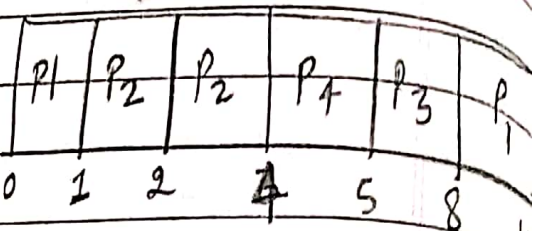


	AT	BT	CT
P1	0	20-5-10	20
P2	15	25-10-15	55
P3	30	10	40
P4	45	5	70



0 15 20 30 40 45 55 70

P1	0	5-1=4
P2	1	3-1=2
P3	2	3
P4	4	1

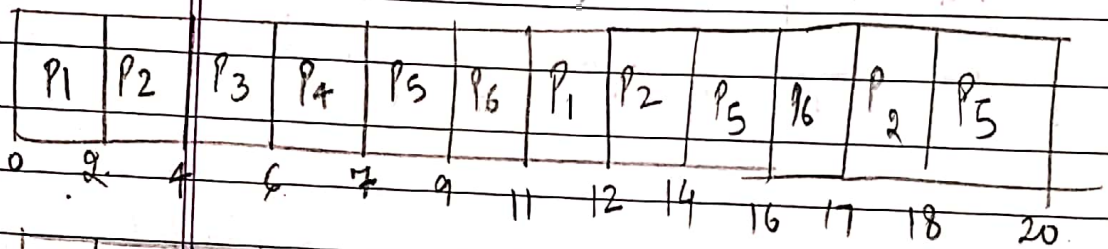


0 1 2 4 5 8

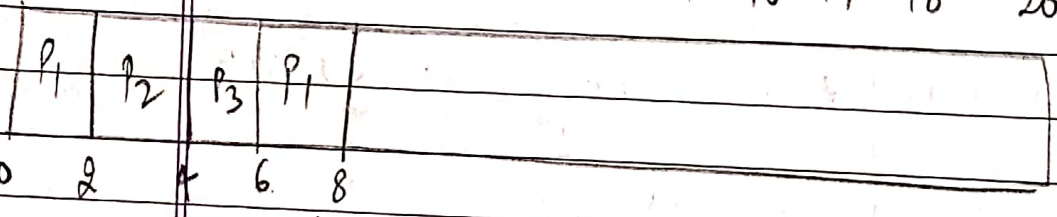
DATE: / /
PAGE NO.

Q-1.

	AT	BT
P1	0	4-2=2 ✗
P2	1	5-2=3-2=1
P3	2	2 ✗
P4	3	1 ✗
P5	4	6-2=4-2
P6	6	3-2=1 ✗

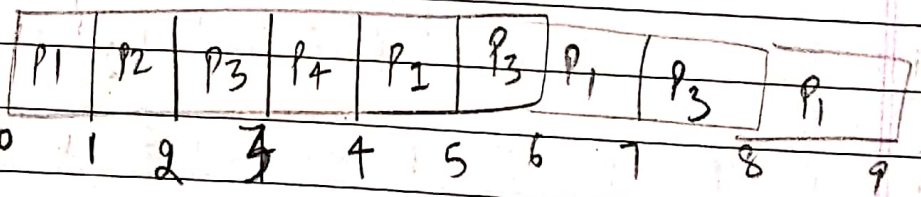


0 2 4 6 7 9 11 12 14 16 17 18 20



0 2 4 6 8

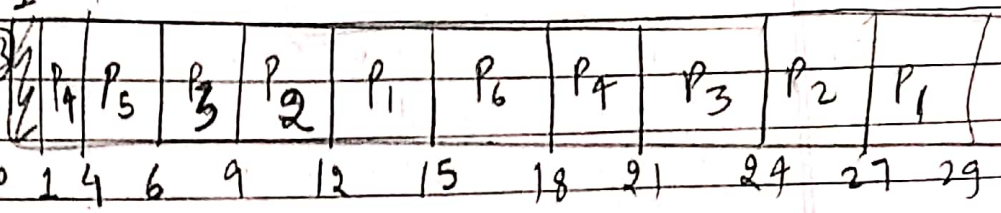
P1	4-1=3
P2	1
P3	8-1=7
P4	1

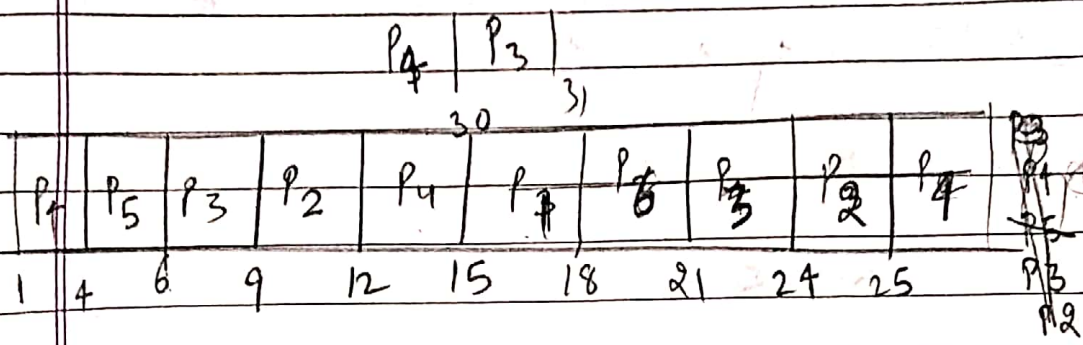


0 1 2 4 5 6 7 8 9

	AT	BT
P1	5	5-3=2
P2	4	6-3=3
P3	3	7-3=4 : 1
P4	1	9-3=6-3=3
P5	2	2
P6	6	3

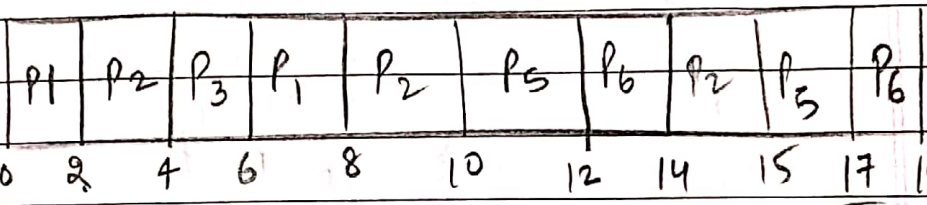
DATE: / /
PAGE NO.



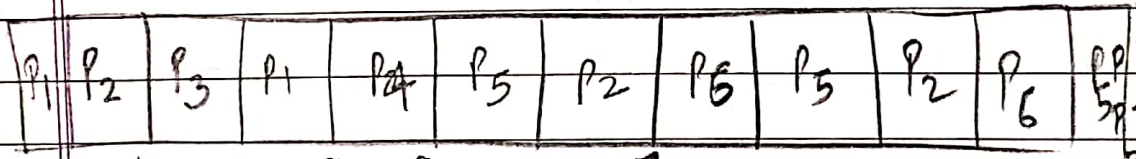


P4 = 6
P5
P3
P3 = 7-3=4
P2 = 2
P4 = 3
P1
P3 16
P2 =
P4 = 3
P3 = 1

	AT	BT
P1	0	4-1=3
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

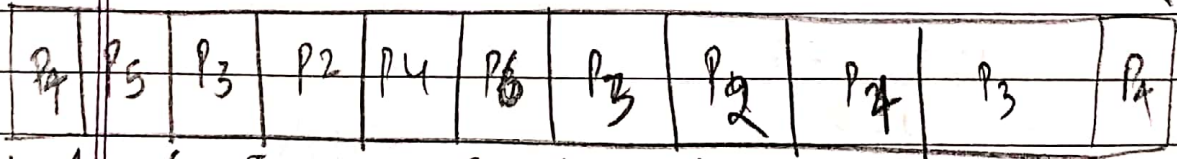


P5



P1 = 2
P2 = 3
P5
P6
P2 = 1
P5 = 4
P6 = 1

P1
P2
P3
P1 = 2
P4
P5
P2 = 2
P5 = 4
P2 = 1



P4, P5, P3, P2, P4, P6, P3=4, P2=3, P4=3, P3=1, P4=1

Banker's

DATE: __/__/__
PAGE NO.: Aa

	Allocation	Max	Need	Available
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

i) Work = Available
= 332.

ii) Need ≤ Work Finish[i] = true.
122 ≤ 332 P_1
F[1] = T.

iii) Work = Work + Allocation
332 + 200 = [532]

∴ P_1, P_3, P_4, P_0, P_2

Need = 1, 2 2 Need ≤ Available
P → Req. Need(P_1) → 122 ✓ T
Avail = 332 − 102 Available = Available − Req
= 230 Allocation = Alloc + Req.
200 + 102 Need = Need − Resour.
= 302
122 − 102
020

Need Matrix:

743
020
600
011
431

WORK = 230.
020 ≤ 230
F[1] = T
W = W + Alloc = 302 + 230
532

	Alloc A	B	C	Max	Need	Avail
P1	2	1	1	4 2 3	2 1 2	8 1 2
P2	2	3	~~4~~	7 3 10	5 0 9	
P3	1	~~4~~	4	5 2 13	4 1 9	
P4	1	~~4~~	2	10 5 4	9 1 2	
	6	9	8			

Total 14 10 10

6 9 8

8 1 2

W = Available

= 812

P1 ≤ 912.

W = W + A

= 211 + 812 = 1023

P4 912 ≤ 1023

W = 1023 + 142

1165

P1 < P4 : (deadlock).

Page Replacement Algos:

7 7 7 2 2 2 4 4 4 →

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0

7	7	7	2		2		4	4	4	0							
	0	0	0		0		0	0	3	3							
		1	1		3		3	2	2	2							
1	2	3	4	5	6	7	8	9	10		11	12					

12.

1 2 3 4 1 2 5 1 2 3 4 5.

DATE: __/__/__ Aa
PAGE NO.: 1

1	1	1	4	4	4	5		5	3
	2	2	2	1	1	1		3	3
		3	3	3	2	2		2	4

1 2 3 4 5 6 7 8 9

1	1	1	1		5	5	5	5	4	4
	2	2	2		2	1	1	1	1	5
		3	3		3	3	2	2	2	2
			4		4	4	4	3	3	3

1 2 3 4 5 6 7 8 9 10

Belady's Anomaly.

LRU

No. of Frames, Pg. Fault

0 4 8 2 0 2 4 3 4 4 2 6 8 7 2

0	4	8	2				0	0			6	6	6		
	4	4	4				3	3			3	8	8		
		8	8				8	4			4	4	7		
			2				2	2			2	2	2		

1 2 3 4 5 6 7 8 9

7 — 0 — 1 — 2 optimal

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		2			2			2			7		
	0	0	0		0		4			0			0			0		
		1	1		3		3			3			1			1		

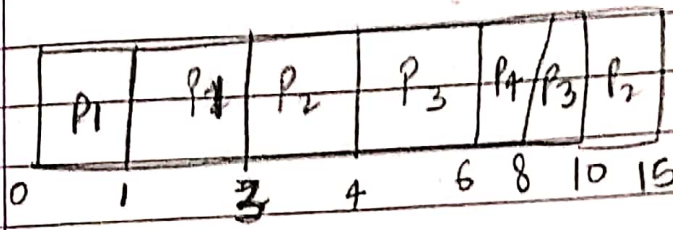
1 2 3 4 5 6 7 8 9

DATE: / / PAGE NO.

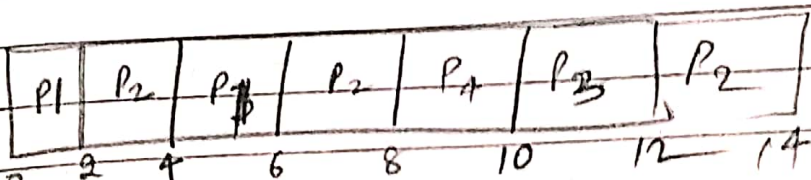
	AT	BT	CT	TAT
P1	0	3-2=2 ✗	3	3
P2	1	6-1=5	15	14
P3	4	4-2	10	6
P4	6	2 ✓	8	2

25 = 6·2
1

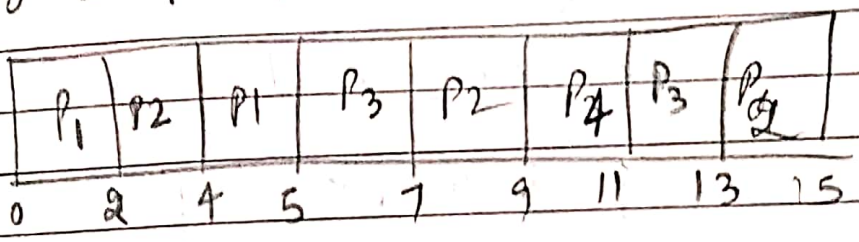
FCFS



P1 0 3-2=1
P2 1 6-2=4
P3 4 4
P4 6 2



P1
P2
P2=1
P3
P2=2
P4
P3=2
P4=4



	AT	BT	CT	TAT
P1	0	5	5	5
P2	1	15	14	14
P3	4	13	9	9
P4	6	11	5	5

1
19
14
33

P1
P2
P3
P4

P	BT	AT	Priority
1	4	0.000	3
2	3	1.0001	4
3	3	2.0001	6
4	5	3.0001	5

	BT	AT
P1	5	2
P2	13	3
P3	8	0
P4	4	5
P5	10	1

Process	BT	AT
1	11	5
2	4	0
3	14	0
4	9	1
5	21	2

Always ahead...

1200

DATE: / /
PAGE NO.

Allo A B C D	Max	Need	Avail
4 0 0 1	6 0 1 2	2 0 1 1	2 0 1 1
1 1 0 0	1 7 5 0	0 6 5 0	
1 2 5 4	2 3 5 6	1 1 0 2	
1 6 5 3	1 6 5 3	0 0 0 0	
1 4 1 2	1 6 5 6	0 2 4 4	

Work = Avail

W = 2011

12,00 ≤ P1 ≤ 2011

W = W + A = 2011 + 4001

6012

P[3] ≤ 6012

W = W + A

W = 6012 + 1653

7665

P[4] ≤ 7665

W = W + A

W = 7665 + 1412

8 10 7 7

P[2] =

8 10 7 7 + 1 1 0 0

9 11 7 7

P[3]

12 5 4 + 4 1 1 7

10 13 12 11

code in ROM & data in HD is always there unless you remove it.

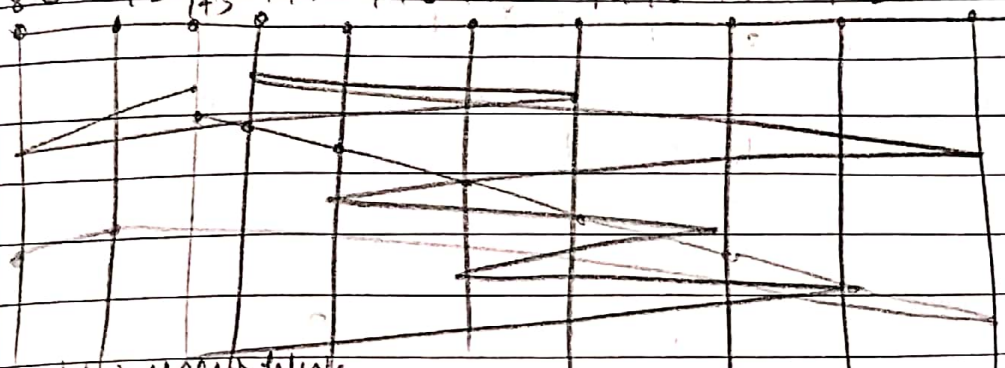
DATE: / /
PAGE NO.

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Current head pos 143.

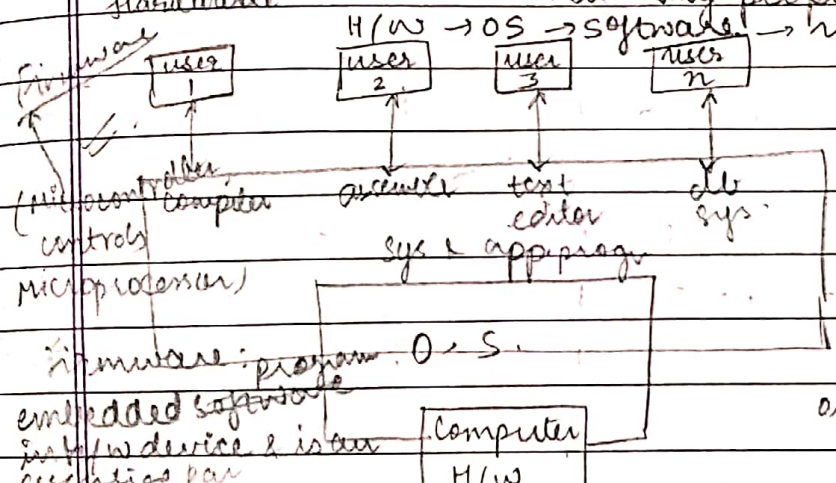
Scan prev Req = 125 (0 to 4999)

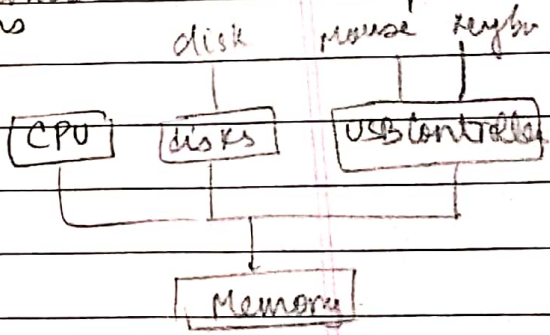
86 130 143 913 948 1022 1470 1509 1750 1774



Interface b/w users & h/w.

Resource allocator: OS acts as resource allocator when many people share the same computer

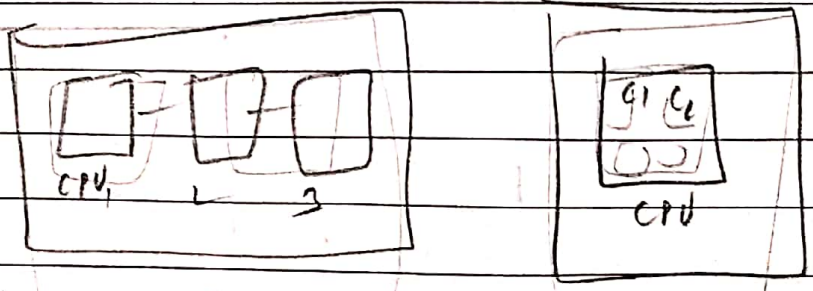




Memory controller are provided for orderly access

Firmware: program embedded software in h/w device & is an essential part

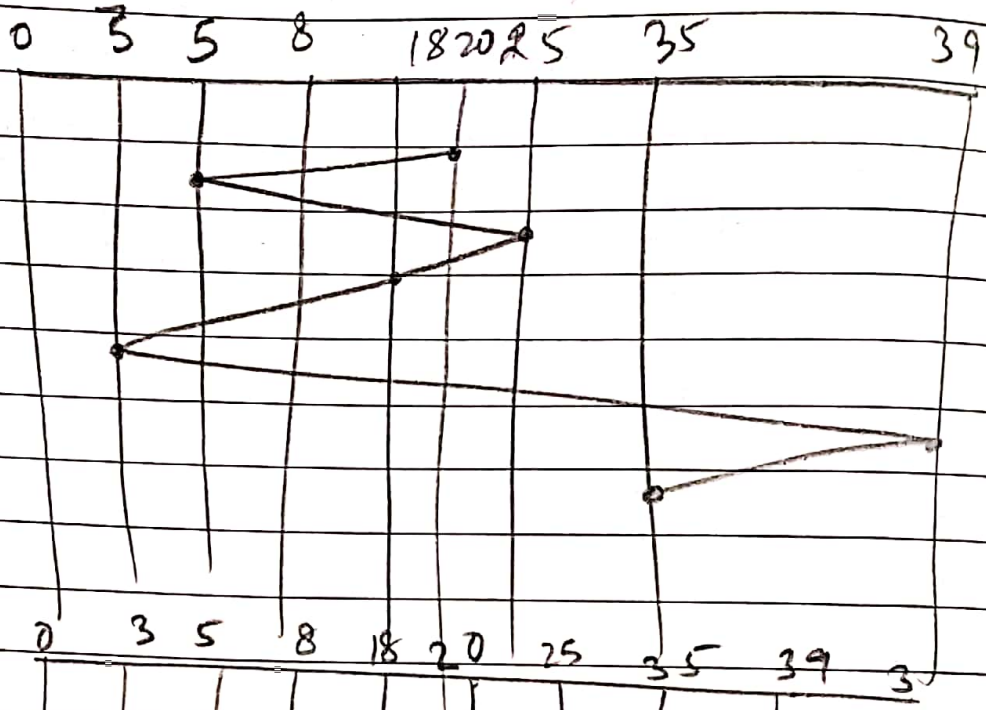
For booting: initial prgm is reqd: bootstrap prog stored in ROM or EEPROM

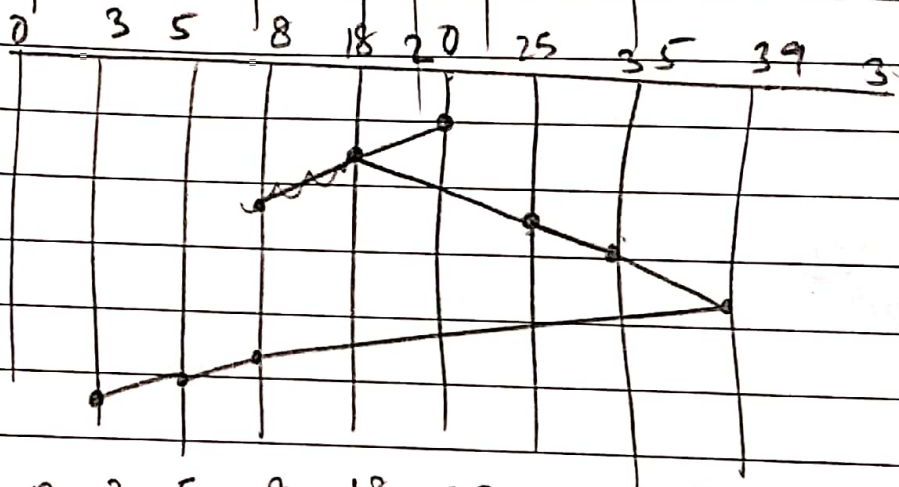


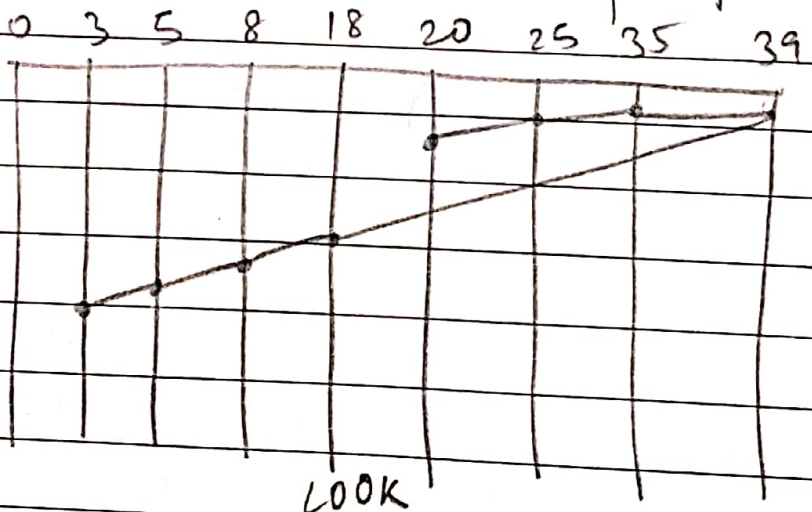
Multiprocessing	Multi-core
costlier	cheaper
distributed time	less time
little configuration multiple OS	No config. Single CPU

5 25 18 3 39 8 35

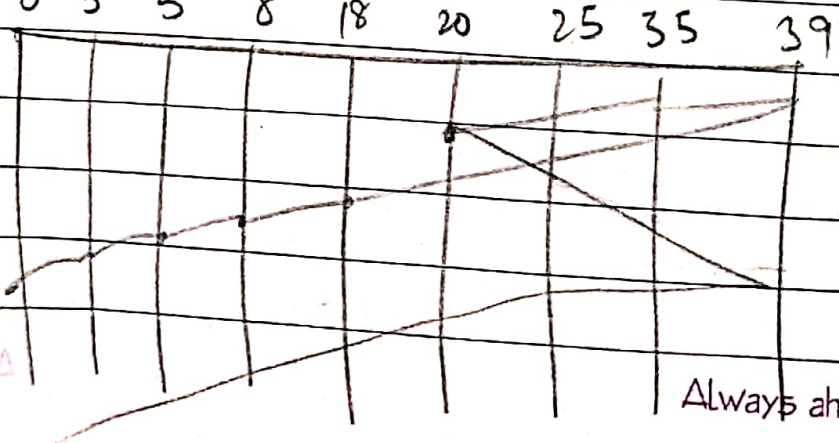
FCFS







LOOK



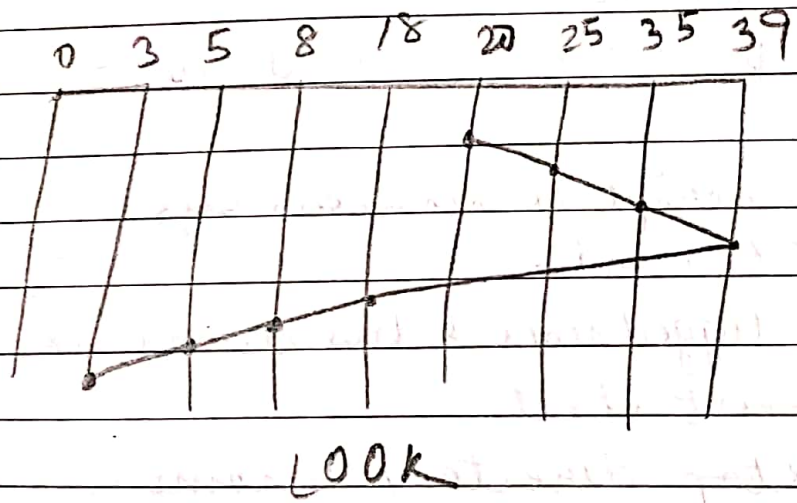
Always ahead...

DATE: / / Ha
PAGE NO.

case statement
+
case word in
+) $a+$b; ;
-)
*)
/)
*) default.

if
fi
write a shell script to do all arithmetic operations using case statement.

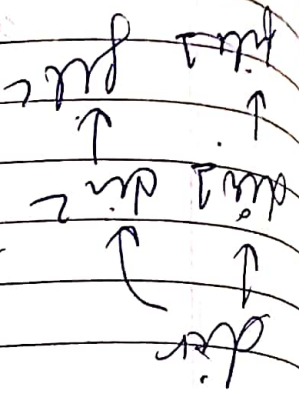
6)



DATE: / / PAGE NO. Ha

Q $\frac{dy}{dx} = 2 + \sqrt{xy}$

$y = 1, x = 1$, at $x = 2, y = ?$



```
mkdir dir7
cd dir7
for var in {1..5..1}
do
mkdir dir-$var
cd dir-$var
touch file-$var
cd..
done
```

$x^2 + y$

```
#!/bin/bash
echo $1 + $2.        bash abc.sh 5 6
```

Exp Write shell script to various sys

~~USER : keyword~~

1. Currently logged user & his username
2. Your current shell.
3. Your desktop directory or home.
4. Your O/s type
5. Your current path setting
6. Your current working directy.

USER	
LOGNAME	
SHELL	
HOME Directory	
OSTYPE.	
PATH	Current working Directory.

Always ahead

Command line argument;

DATE: / / PAGE NO.

abc.sh 5 3

$0 $1 $2

arg1 arg2 arg3.

$1

$2 bash -sh $12 2 3

$0 first var

echo $# = 5.

counting of arguments

$* all arguments

$@ same as * but stores in array

for i in command line argument

Store arguments that were entered (bash. sh $15 xyz

$* all = 15 xyz

$3 xyz

expr

array:

$@ abc = $@

abc[0].

suppose you are in home directory which contain a directory dir now you have to create a script which perform oper dir contains 5 subdirectin dir1 2 3 4 5 & each dir contains ~~file1,2,3,4,5~~ file1

dir

dir1 dir2 dir3

file1 file2 file3.

for loop

① for var in {1..5}..3}

do

—

—

done

for (i=0; i<5; i++)

do

—

done;

#

cd dir

for(i=1; i<5; i++)

mkdir dir-i

cd dir-i

touch file i

} c..

Always ahead..

Exp: 21 Intro to networking

ifconfig portname netmask ip of netmask

DATE: / / PAGE NO.:

ifconfig

sys setting : NL

Non-graphical : Linux

w3

links

elinks.

crown

Nice

Renice

Exp: 22 Sharing of files

Process Oriented Commands

→ ps → displays current active processes in current cell

ps -A : display all active processes in sys.

display all processes in BSD format.

ps aux → displays processes not attached to terminal

a → displays processes

u → owner.

x → processes for all users

ps au

print all processes running as route

ps -U root -u root.

display processes by process id.

ps -fp 101

display processes by TTY (terminal)

ps -t /pts/0

process tree (in structure)

ps -e tree

Display process id.

pstree -p

Sort by numeric process id

pstree -np.

Stop a command (kill & process id)

Kill Pid

Forcefully kill -9.

kill -9 Pid

killall gedit.

Always ahead...

Reader-writer

Reader

```
wait(m)
read_count ++
if (read_cont == 1)
    { wait (w)
      // read
      signal (m)
    }
wait(m).
read_count--;
if(readcount == 0)
    signal(w);
    [Last Reader];
    signal (m);
}
```

```
wait (w)
// read
signal (w)
```

```
wait(w)
// read
signal(m).
wait(m)
signal (w)
signal(m)
```

```
wait (m)
read_count++
if (read_count == 1)
    { wait (w)
      // read
      signal (m)
    }
wait (m);
read count --)
```

Is user forcefully

tatus into

How to create groups,

groupadd -name of group

DATE: / / PAGE NO. A2

to add linux users to group

usermod - aG group username

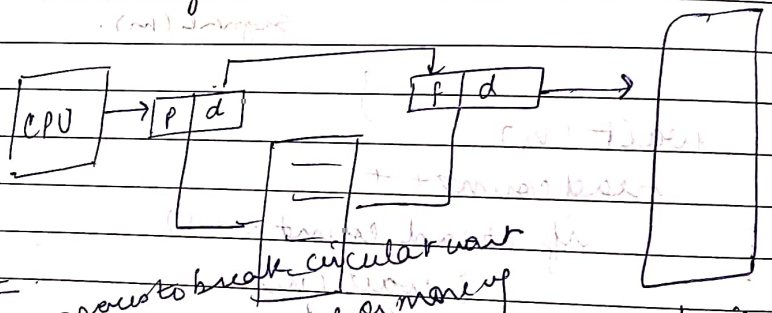
gpasswd gpadd groupname.

to remove gpasswd -r.

delete a group.

groupdel name of group.

Paging is a memory-management scheme that permits physical add space of a process to be noncontigous.



Recovery

→ abort one process to break circular wait

→ preempt some R from one or more deadlocked process.

(1). → Abort all deadlock process

→ abort one process at a time until deadlock cycle is eliminated after each check whether others are deadlock or not

tion.

ps-tree

Display process id.

pstree -p

Sort by numeric proce

Stop a command (K

ocess. (no of Rollbacks)

echo : is used to print characters ~~string in a single line~~
Basically it is used in shell scripting or C shell
for display.

echo "$x" or echo $x echo '$x'
as it is

echo -e
e option is used in linux for the interpretation of
escape character
/t horizontal tab
/v vertical tab
/c trailing

Exp 19 creating & deleting users (How to manage)
How we can create a user users
useradd name of user

useradd -e : date when account will expire
specifying user's full name when creating a user.
useradd -c "x y"

3456554 adduser name of user
6574
How to remove a user account
userdel name of user
-r all files deleted
-f user forcefully
password command.
passwd abc
display password status info
passwd -s.
-Sa

VCS repository git

tutorial, bit.do/git-block
To join community herokuapp.com
dsc-invite.herokuapp.com

Allocation: n x m defines no. of resources of each type currently allocated.

Need - ~~Max - All~~

Deadlock detection

→ an algo that examines the state of sys to determine whether deadlock
⇒ algo to recover from deadlock.

github link goo.gl/r7Zerx

https://education.github.com/pack

0	2	3	1	4	2	5	6	0	3	2	4	7	1	2
0	0	0	0	0		0	0		0	0	7	7	7	
	2	2	2	2		2	6		6	6	6	6	8	
		3	3	4		5	5		5	5	5	1	1	
			1	1		1	1		2	4	4	4	4	

1 2 3 4 5 6 7 8 9 10 11 12.

Single instance:

If all resources have single instance, then we can define deadlock-detection algo using WFG.
removing resource nodes & collapsing appropriate edges.

If WFG has cycle :- deadlock

to detect cycle :- n^2 operations, n is no. of vertices.

to call detection algo at every moment ↑ overhead. Alternate is to invoke algo at defined intervals :- once per hour or whenever CPU utilization drops below 40%.

When or how often deadlock occurs & how many processes will be affected.
Deadlock when occurs process make requests & it cannot be granted

Ex: 17-18

DATE: / /
PAGE NO.

7) 11 & communication oriented commands.

whois google.com

finger space root : info of users

if config : interface configuration

CH → R CH

~~Class A~~

eth0 192.60.60.1

How to enable ~~an~~ network interface

if config eth0 (port name) up
down.

How to assign a netmask to network interface,

Mask : 255.255.255.0

if config portname netmask ip of netmask

How to assignip netmask broadcast to network interface

if config eth0 10.0.0.1 netmask 255.255.255.0
broadcast 192.10.10.20.

How to check route table.

ip — route show

ping ipadd.

wget command recieve files from www using protocols like: http (hyptext transfer protocol), ftp (file transfer protocol)

wget -c http://———— . to download

Route : command

mail : command.

Exp-16

DATE: __/__/__
PAGE NO.

Text editors :

generates text file

(i) nano editors (ii) gedit (iii) vi editor

(iv) v-max.

(i) Nano editor : nano is a text editor licensed by GPL → free of cost.

I How to Run it

nano	open
nano abc.c	
ctrl + O	save
ctrl + X	Exit.
ctrl + n	next line
ctrl + v	next Pg
ctrl + p	Prev. line
ctrl + y	Prev Pg.
ctrl + A	Beg. of Line
ctrl + E	end of Line
ctrl + C	current line on which cursor
ctrl + w	search string in editor
ctrl + d	delete character currently under cursor
ctrl + K	to delete whole line, cut specific string ko replace karege new string se
ctrl + u	: to get back string.
ctrl + j	: justify
ctrl + b	: go back one char (cursor)
ctrl + f	" fwd " " "
ctrl + space	: one word fwd
alter - space.	back

esc + shift + colon

command mode by default. (:x)

i = 0 j = 4
i = 1 j = 3
i = 2 j = 2
i = 3 j = 1
i = 4 j = 0

312456

for (i = n; i != 0; i--)
for (j = 0; j != n; j++)
{ if. (i == n).

Produce only compile code
gcc -C macro.c
it will update the file a.out

enable all warnings:
gcc -Wall macro.c

use compile time macro using D option

gcc -D nameofmacro macro.c

provide gcc options through object file.

· gcc main.c @ ~~abc~~ h

gcc ~~micro~~ c -save-temps -wall -o macro.c

How to debug C program using gdb command

gcc → debugger.

gcc -g macro.c	gdb a.out

set up a break pt inside C program.
run

- l · list code.
p print
c continue till next breakpoint
n next line

Exp: 15

DATE: / /
PAGE NO.: Pa

Working with gcc compiler, debugging & compiling object file using gnu

gcc main.c -o vinod

a.out — by default executable file

Steps: (4 in total for compiling)

(i) Pre-processing : .i — preprocess
(ii) Compilation : .s — asm
(iii) Assembly — machine lang.
(iv) Linking — a.out — extra code to run a program.

```
#include <stdio.h>
#define mul a*b        ⇒ Macro fn in preprocessor
void main()
{ int a,b;
  scanf("%d%d",&a,&b); //
  printf("%d",mul);
}
```

flag
-save-temps
↓
store intermediate files.

gcc @ macro.c -o - macro

gcc -save-temps macro.

cat macro.i
cat macro.o
's.

size.o filename .o.
size a.out

produce only preprocessor file

gcc -E macro.c > abc.i.
gcc -S macro.c > abc.s

for Resource allocation of multiple instances
Bank should not keep money in vain that no customer is unsatisfied.
Data structures required.
Available : m indicates available
Max: n×m matrix defines the max demand of each process

BANKER

① W = avail
Finish[i] = False
② b) Need[i] < work
Finish[i] = true
③
④ W = W + Allocation
Request Algo.
Req ≤ Avail[i]
Req ≤ Need[i]
AV = AV - R.
AL = R + AL.
N = N - R.

Always ahead...

Scanned by CamScanner

P2

DATE: / / PAGE NO.

Q 3

Exp: 1

Deadlock may arise if all 4 satisfies simultaneously.

ME only one process at a time must hold the lock.
(i.e. non shareable).

2. **Hold & wait** : process must ~~acquire additional~~ be holding atleast one Resource & waiting to acquire additional that are held by another.

3. **No preemption** : Resources cannot be preempted, that is resource can be released by process only which is holding it.

4. **Circular wait** : $\{P_0, -- P_n\}$ of waiting processes must exist s.t. P_0 is waiting for R held by P_1, P_1 for another

RAG : for describing deadlocks.

cycle in RAG is necessary & sufficient cond'n if one instance is there.

but more than one; then it may or may not.

Methods for handling Deadlock.

We can use a protocol to prevent/avoid deadlocks, ensuring that the system will never enter a deadlocked state.

We can allow the system to enter a deadlocked state, detect & recover.

like UNIX & win, we can ignore the problem & pretend like deadlock never occur.

Deadlock Avoidance:

Deadlock prevention algos.

Side effect : low device utilization & reduced system throughput.

Eg:- A process req. printer tape drive sys might need to which P req. first P the TD but Q for TD & the P.
this complete knowledge of reason

* [0-9] *

[vpq] *.

locate abc *

DATE: / /

it won't point

doesnot give proper result

this means starting from abc ?

abcd

abcd.

^ carat = begining.

$ = end

[100 - 1123or 300] . X

[1000 - 2000] 3000 - 4x00]

Name of files start with abc f

abc * 100

ghi & j

[abcd] ? a in 3rd

[abcd] ? a * x

Restrict locate output

-l → length
-n → no. of files

DATE: / / Aa
PAGE NO.

locate -l 5 linux

Refresh mlocate database -
sudo updatedb.
↳ update database

locate yzx
find -name yzx
sudo update db
locate yzx

Separate o/p of entries without new line
-0 (zero)

Review your mlocate database

locate -S

locate -r linux $.

[abcd] ? a * X

Wildcard operators:

① * (as % in DB)

L*
*L

② ? (_ in db).
single character
searching, ls ??

③ Limit [] (range operator)

start a-f
end y-z

[a-f y-z]

find files modified in last hour

find -mtime -60.

DATE: / / PAGE NO.: A2

(17) find access files find -atime -60

find / (-mtime -60) → starts searching from root

(18) Find files & directories based on size.

c → bytes

k → KB.

M → megabytes

G → Gigabytes

find files of 50 MB.

find -~~50M~~ size 50M

find files b/w 50 KB to 100 MB.

find -size +50K -~~100M~~ size -100M.

~~find~~ all files which are empty.

find / ~~-empty~~ -type f -empty.

find -name "abc*" -a -name "*.c"

find: command: from orginal start abc & end .c

Locate: command. m locate database.

locate linux (absolute path)

1. search the file using locate.
   locate linux

2. Count no of files
   locate -c linux.

3. to remove case sensitivity
   locate -i linux

Search Only files or directories

. → (previous)

DATE: / / PAGE NO.

For files.

1) find -type d -name Linux for directories

2) find -type f -name Linux for files.

Search multiple directories together

find . / . / -name Linux.

Find hidden files (starts with .)

find -name ".*"

Search files & directories based on modification date & time

mtime modify time = days.

atime access time

n min

a min

c min change, files min

Find files modified n days ago (back)

=5

find -mtime -n

↳ +5

-15,

50 to 100

+50 -100

↓ ↓

after 50 less than 100.

Find files accessed n day ago

find -atime +5

Q in a range of 15 & 100 days ago

find -mtime +15 -mtime -100

+15 - 100

Changed in last 10 min.

find -mmin -10

Which firefox. A X B 5 X 4 DATE: / / PAGE NO.

Select * ename, sal from empe, empm join where e.empno = m.mgr.

Exp: 12 Searching files or directory using
1. Find.
2. Locate.

1 List all files in current directory & Subdirectory
If we are on Desktop # find

2 Search specific file or directory
find linux.

3 To search whereever Linux name file is saved.
find -name linux
./
↳ last directory.

4. to remove case sensitivity use -iname.
find -iname linux

5 limit depth of the directory traversal
find -max depth n -name linux.
↳ n=2

6. Invert match (-not)
find -not -name
find -name not linux

combine Multiple search criteria.
find -name abc [-name, c. file abc & file without .c
find -name abc -o -name, c. file abc or file with .c
find -name abc -a -name, c

Always ahead

List content of tar.gz2 archive file

DATE: / /
PAGE NO.

untar single file from tar file

tar - xvf linux.tar "abc.c"

untar multiple files from tar, tar.gz (gzip), tar.gz2 (gzip2) file

tar -

add files or directories to tar file

tar - rvf linux.tar

- tar
- tar.gz (gzip)
- tar.bz2 (gzip2)

gz & bz files are appended

tar - jcvf linux.tar.bz2 .

tar - ztvf linux.tar.gz