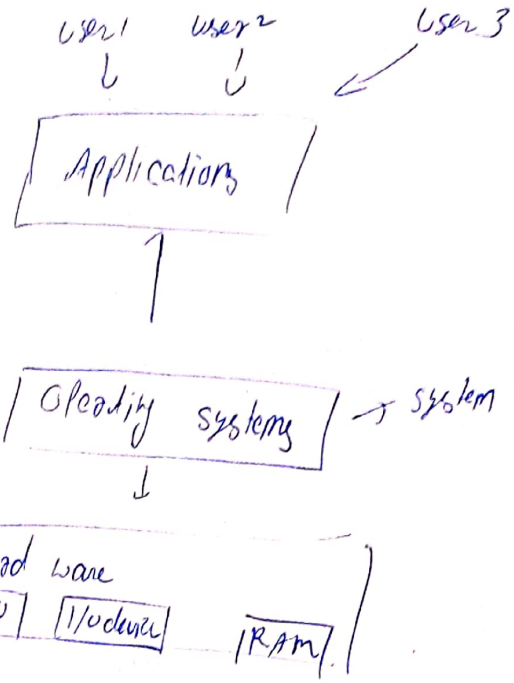
Operating System and its functions

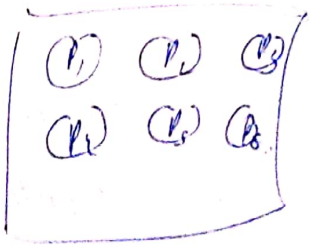


functions

1) Resource management
2) Process Management (CPU scheduling)
3) storage mgmt → file management (harddisk)
4) Memory Management (RAM)

Types of operating system

1) Batch → Punch cards were used for similar type of work / Batch used in 1960 by IBM
2) multiprogrammed
3) multitasking
4) Real time OS
5) Distributed OS
6) clustered
7) Embedded



2) multiprogrammed OS - Non preemptive

all tasks are loaded in Ram once and executed one by one fully until the program wanted to be interrupted / need I/O

- OS can't interrupt program

Multitasking / time sharing OS → Preemptive

it is a bit similar to the multiprogrammed only diff is it will not execute a program fully it will execute all program sequencially for a specific duration once will continue this cycle until all programs gets executed fully

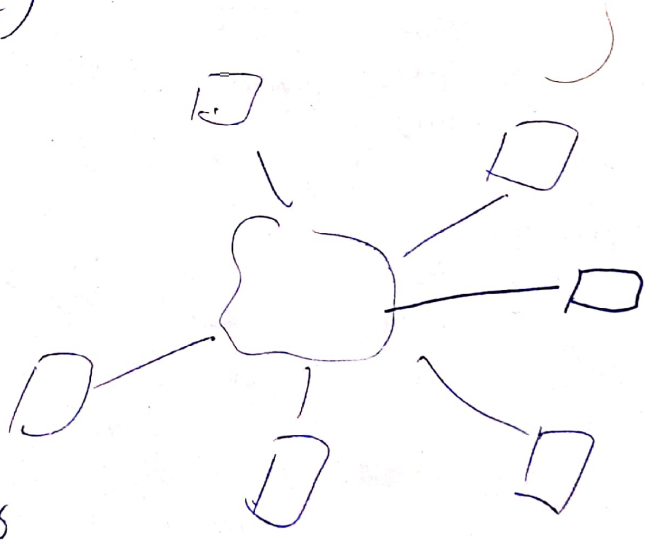
this os can preempt itself unlike multiprogrammed we get responsiveness through this multitasking

Real time OS → Hard
→ Soft

delays are not expected / accepted
can cause major loss
eg missile system

delays are accepted
minor delays won't cause much trouble
eg youtube live stream

distributed (environment)



geographically seperated

in case 1 system fails other system will handle the work of on first one

clustered

- multiple machines are added in 1 system to make it more powerful, more efficient
- connected in lan or by any other way
- concept of super computer

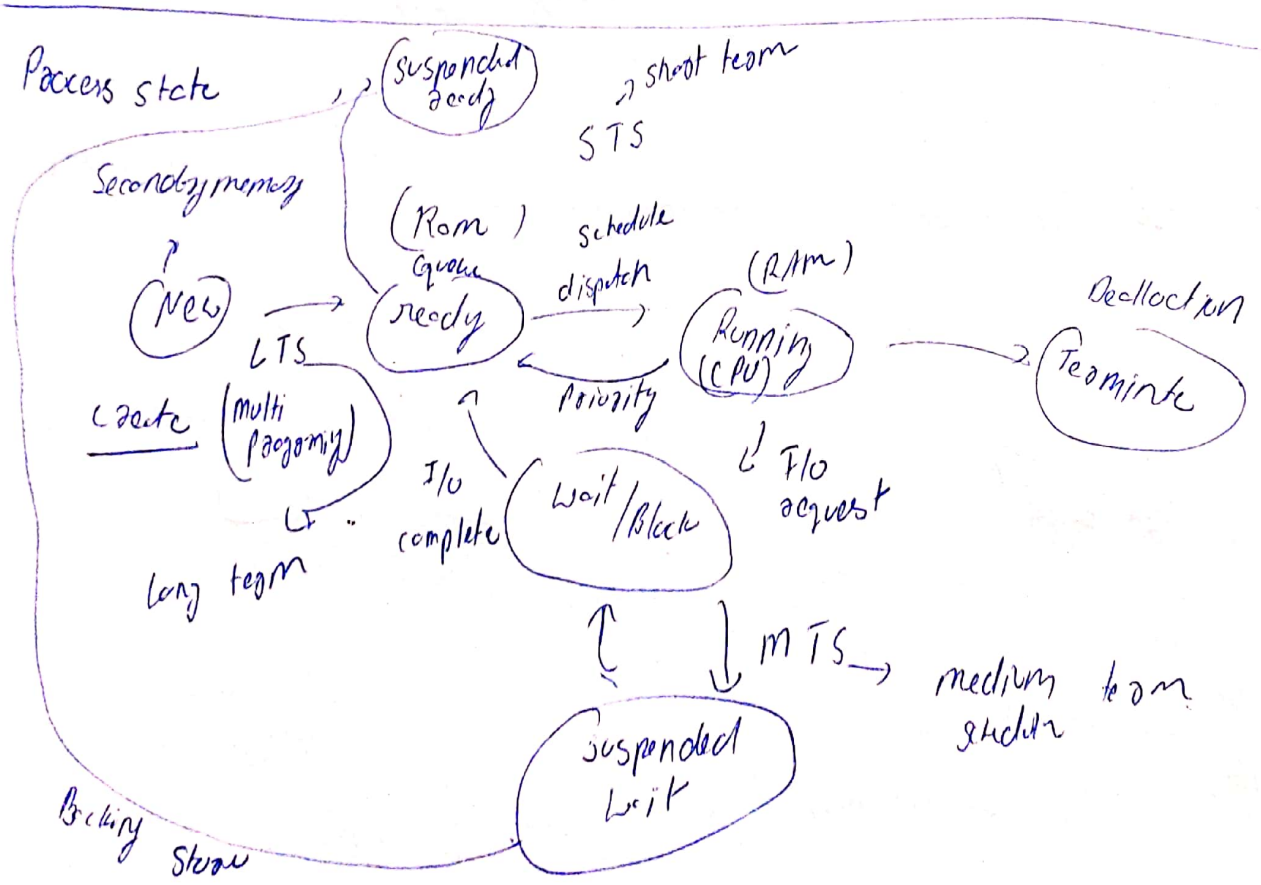
Processing power.
availability
computation power

everything increases

Embedded

- are programmed to do fixed task
we cant change its functionality

eg washing machine
micro wave

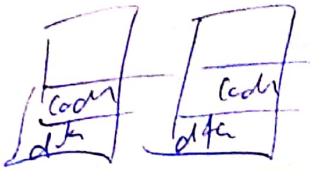


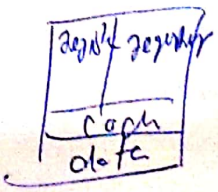
Long term scheduler - it is responsible for bringing more process in ready state from secondary memory

Short Term Scheduler - responsible for picking a process from Ready queue and giving it to cpu for execution

Medium term Scheduler - is responsible for sending process into suspend state due to queue overflow

Process	threads (user level)
1) system calls involved in process	There is no system call involved
2) os tacks diff process differently	All user threads treated as single task for os
3) diff process have diff copies of data, files, codes	threads share same copy of code and data
4) context switching is slower	4) context switching is faster
5) Blocking a process will not block others	- Blocking a thread will block entire process
6) Independent	- Inter dependent





User level thread	Kernel level thread
- User level threads are managed by user level library	Kernel level threads are managed by OS system calls
- User level threads are typically fast	Kernel level ~~ters~~ are slower than user level
- context switching is faster	- context switching is slower
- If one user level thread perform blocking operation then entire Process gets blocked	if one kernel level thread ~~performs~~ ~~blocked~~ blocked, no effect on others

Starvation — Higher priority long process keeps on executing which results in non execution of process with low priority

Synchronization mech

Primary ↳ conditions

1) Mutual Exclusion → ~~on~~ critical section only 1 program is allowed at a time multiple can enter in this section
2) Progress → one program not entering in critical section ~~but~~ and stopping other from entering in critical section
3) Bounded wait → one process keeps on using critical condition infinite time and another can't use
4) No assumption related to Hard ware and speed must be universal

· Semaphores

Semaphores is an integer variable which is used in mutual exclusive manner by various concurrent cooperative process in order to achieve synchronization

- counting ($-\infty$ to $+\infty$)
- Binary (0, 1)

P() , Down , wait

V() up , signal post , release

```
Down (semaphore S)
{
S.value = S.value -1;
if (S.value < 0)
{
put process(PCB) in
suspended list sleep();
}
else
return;
}
```

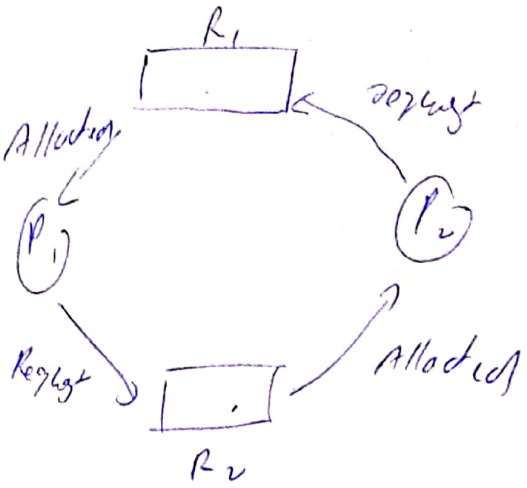
```
up(semaphore S)
{
S.value = S.value +1;
if (S.value <= 0)
{
select a process
from suspended list
wakeup();
}
}
```

```
Down (semaphore(S))
{
if (S.value == 1)
{
S.value = 0;
}
else{
Block this process
and place in
suspend list,
sleep()
}
}
```

```
up(semaphore S)
{
if (suspended list is empty)
{
S.value = 1;
}
else
{
select a process
from suspend
list and
wake up();
}
}
```

Dead lock

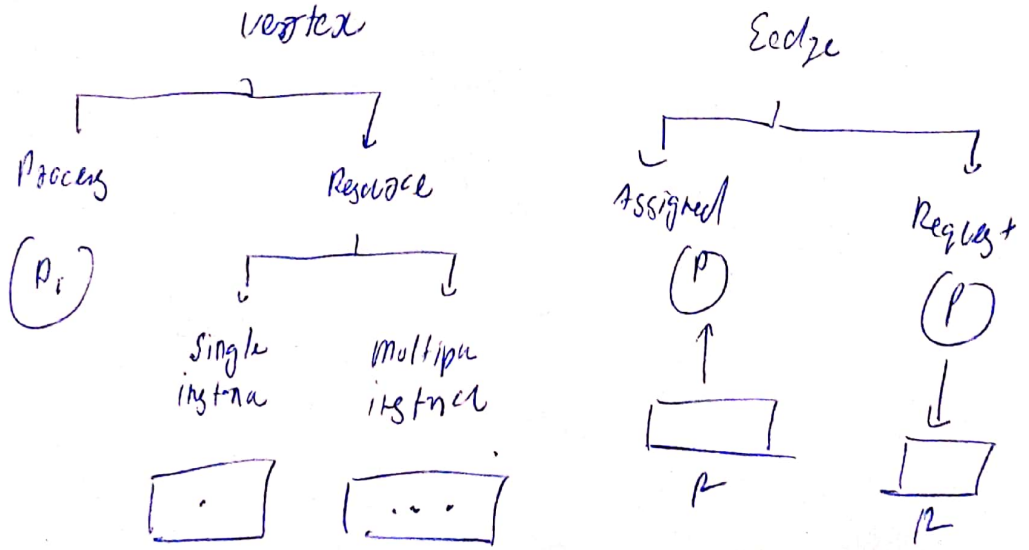
If 2 or more processes are waiting on happening of some event, which never happens, then we say these process are involved in deadlock then that state is called Dead lock



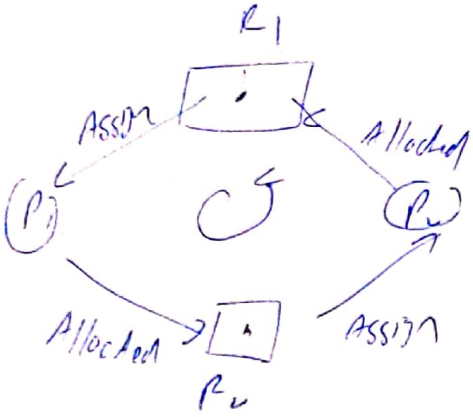
Condition for deadlock (necessary)

1) mutal exclusion
2) No preemption
3) Hold and wait
4) circular wait

Recourse Allocation Graph (RAG)



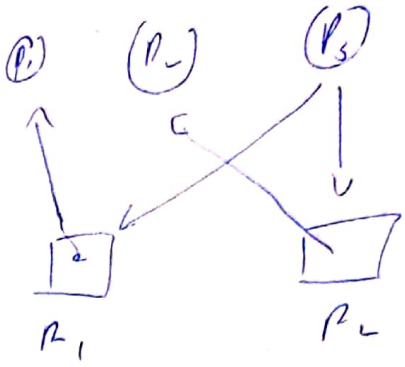
Single instance



circular wait

deadlock

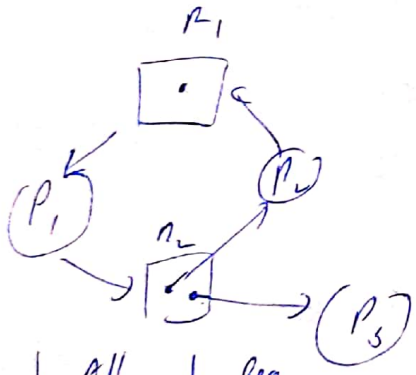
Multiple instance



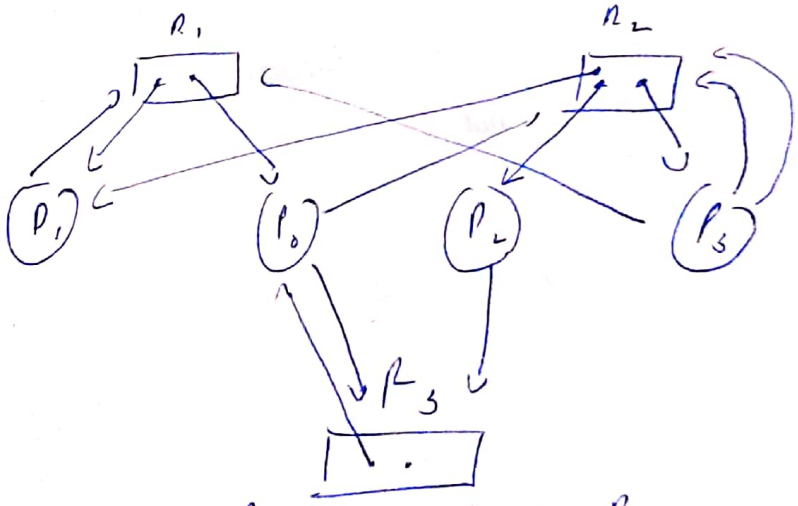
no deadlock

If RAG has circular wait (cycle) (single instance resource)
then will always be deadlock

multi instance RAG



	All		Req	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0



	All			Req		
	R1	R2	R3	R1	R2	R3
P0	1	0	1	0	1	1
P1	1	1	0	1	0	0
P2	0	1	0	0	0	1
P3	0	1	0	1	2	0

clear available (0, 0) → (0, 1) → (1, 0) → (1, 1)

P3 → P1 → P2

Available: 0 0 1 → 0 1 1 → 1 1 2 → 2 2 2

P2 P0 P1

Methods to handle deadlocks

(1) Deadlock ignorance (Ostrich method)

(2) Deadlock prevention

3) Deadlock Avoidance (Banker's Algo)

4) Deadlock detection & Recovery

Prevention → means designing such a system which violate atleast one of 4 necessary condition of deadlock and ensure independency from deadlock

Avoidance :- System maintain a set of data using which it takes decision whether to entertain a new request or not, to be in safe state

detection & recovery → Here we wait until deadlock occurs and once we detect it we recover from it

ignorance → We ignore the problem as if it doesn't exist

Prevention

X Mutual exclusion → make resources Sharable to remove/false this condition

X No preemption → preempt one of the process to make this condition false
(Time quantum can be used
When we preempt the process it releases all the resources which it is holding and these resources can be used by other process

X Hold & wait → give all the resources in ready state to avoid this condition

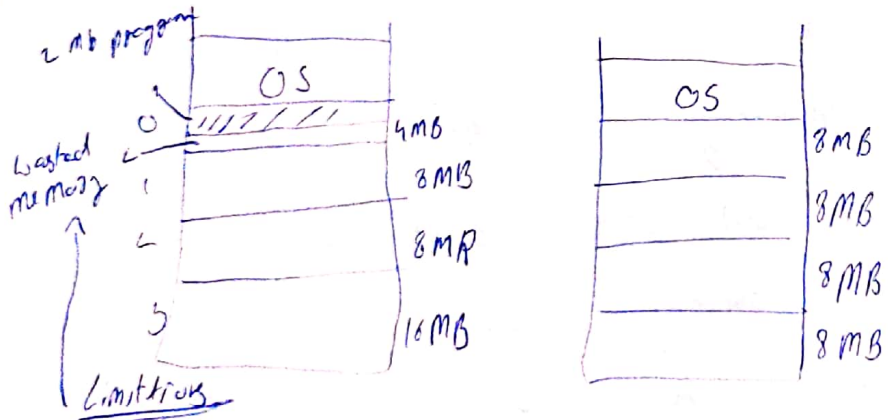
X circular wait → give numbering to resources and the process can only request for resource in increasing / decreasing order, random requests can't be made

Memory Management Techniques

- Contiguous
  - Static
    - Fixed Partition
  - Dynamic
    - Variable Partition
- Non contiguous
  - Paging
  - Multilevel Paging
  - Inverted Paging
  - Segmentation
  - Segmented Paging

Fixed partition (static)

- no of partitions are fixed
- size of each partition may or may not same
- contigous allocation so spanning is not allowed



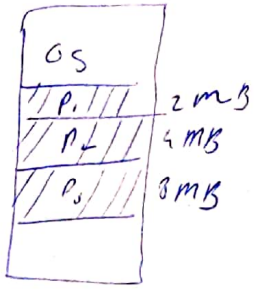
(1) Internal fragmentation - leftover memory gets wasted due to ~~diff size~~ difference in size of our Ram partition & Program memory

(2) Limit in Process size

(3) Limition on degree of multiprogramming

(4) External fragmentation - this happens when we have sufficient Total memory to execute the program but it is not contigous, it is in chunks

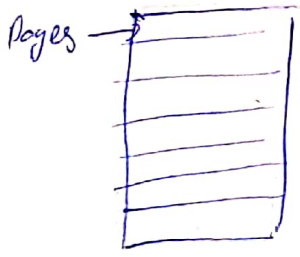
Variable partition
(Dynamic partition)



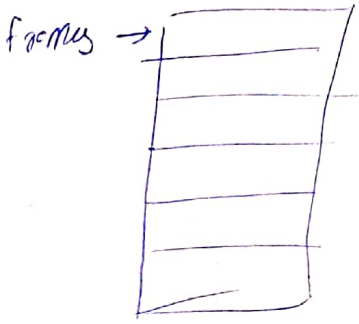
- no internal fragmentation
- no limitation of size of process
- no limition on no of process
- External fragmentation - collectly total space required for execution of program is available but its not contigous
- compaction can be used to remove External fragmentation
- Allocation & Deallocation is complex

discadvantage

Non contigious Mem Allocation



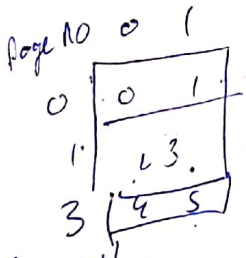
Process

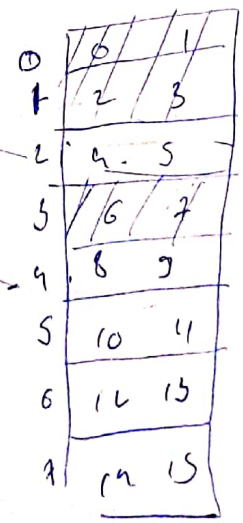


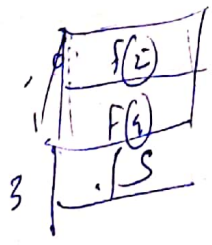
We divide process in secondary memory itself

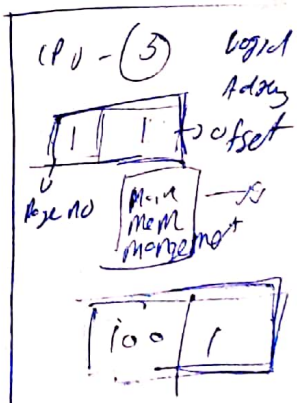
Page size = Frame Size

Page table





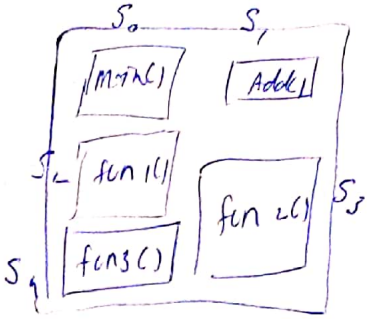


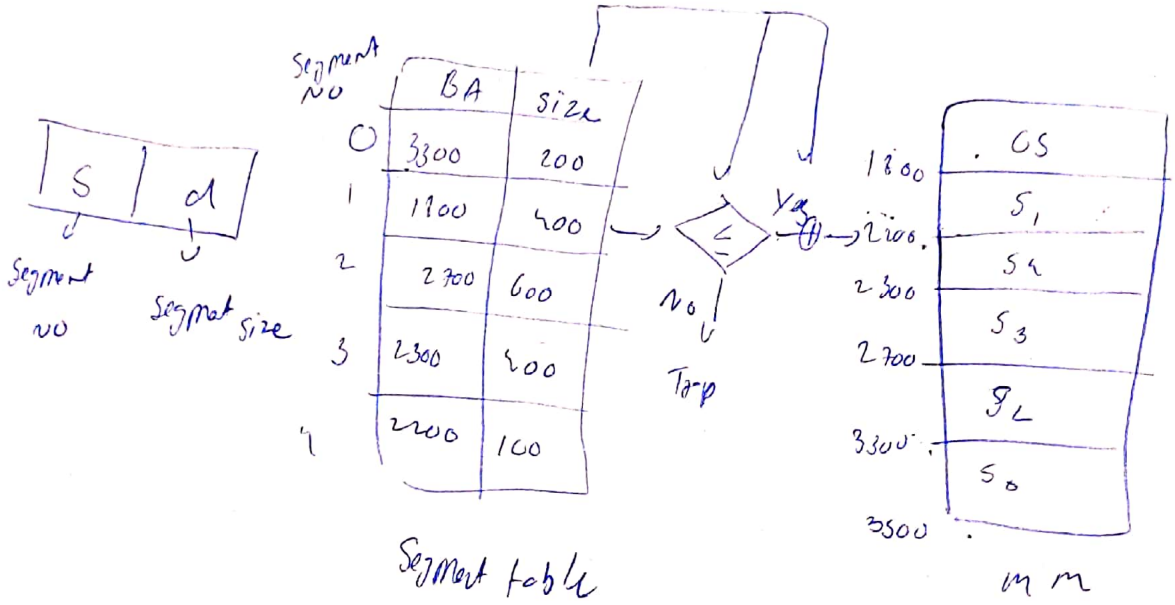


Segmentation

In paging all pages have equal size

- in segmentation we have no fixed size
size is decided by the code.





Segment table

m m

Virtual memory

non preemptive (1 complete job complete)
Preemptive process works in multiple parts

Page No.

FCFS

(CT - AT) turn around time — completion time

PNo	AT	BT	CT	TUT	WT	TUT-BT
1	0	4	4	4	0	
2	1	3	7	6	3	
3	2	1	8	6	5	
4	3	2	10	7	5	
5	4	5	15	11	6	

P1	P2	P3	P4	P5
0	4	7	8	10	15

TUT = WT + BT

TUT = CT - AT

SJF

PNO	AT	BT	CT	TUT	WT
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11

P1	P3	P4	P2	P5
1	8	9	11	16	24

Date / /
Page No.

Premptive

Shortest remaining time first SRTF

P No	A T	B T	C T	TAT	WT
1	0	76	19	19	12
2	1	54	13	12	7
3	2	3	6	4	1
4	3	1	4	1	0
5	4	2	9	5	3
6	5	1	7	2	1

P1	P2	P3	P4	P3	P3	P6	P5	P2	P1
1	2	3	4	5	6	7	9	13	19

Round Robin TQ = 2

P NO	A T	B T	C T	TAT	WT
1	0	8	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10

P1 P2 P3 P1 P4 P5 P2 P6 P5 P2 P6 P5

P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5
2	4	6	8	9	11	13	15	17	18	19	21

Date / /
Page No.

Tot = A = 10
B = 5
C = 7

Bankers Algorithm

Process	Allocation	Max need	Available	Max - Allocation Remaining
	A B C	A B C	A B C	A B C
P_1	0 1 0	7 5 3	3 3 2	~~7 4 3~~
P_2	2 0 0	3 2 2	5 3 2	~~1 2 2~~
P_3	3 0 2	9 0 2	7 4 3	~~6 0 0~~
P_4	2 1 1	4 2 2	7 4 5	~~2 1 1~~
P_5	0 0 2	5 3 3	7 5 5	5 3 1
	7 2 5		10 5 7	

Available = Tot - Allocation
= 10 5 7 - 7 2 5
= 3 3 2

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 - P_3$

Process	Allocation	Max	Available	Remaining
	E F G	E F G	E F G	E F G
P_0	1 0 1	4 3 1	~~3 3 0~~ 330	~~3 3 0~~
P_1	1 1 2	2 1 4	~~4 3 1~~ 431	~~1 0 2~~
P_2	1 0 3	1 3 3	~~5 3 4~~ 534	~~0 3 0~~
P_3	2 0 0	6 4 1	~~6 4 6~~ 646	3 4 1
			~~8 4 6~~ 846	

~~$P_0 \rightarrow P_2 \rightarrow P_1 - P_3$~~

$P_0 - P_2 - P_1 \rightarrow P_3$

18 CRECSOOL Girish

Q3

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

FIFO

6	7	0	2	1	2	3	4	2	1	5	6	2	1	2	3	7
6	6	~~6~~	2	2	2	~~2~~	4	4	~~4~~	5	5	~~5~~	1	1	1	~~1~~
	7	7	~~7~~	1	1	1	~~1~~	2	2	~~2~~	6	6	6	~~6~~	3	3
		0	0	0	~~0~~	3	3	~~3~~	1	1	~~1~~	2	2	2	~~2~~	7
x	x	x	x	x	Hit	x	x	x	x	x	x	x	x	Hit	x	x

6	3	2	1	2	3	6	
6	6	6	6	~~6~~	3	3	
3	~~3~~	2	2	2	~~2~~	6	
7	7	~~7~~	1	1	1	1	
x	Hit	x	x	Hit	x	x	

Hit = 4

fault = 20

LRU

6	7	0	2	1	2	3	4	2	1	5	6	2	1	2	3	7
6	~~6~~	6	2	2	2	2	2	2	2	2	6	6	6	6	3	3
	7	7	7	1	1	1	4	4	4	5	5	5	1	1	1	7
		0	0	0	0	3	3	3	1	1	1	2	2	2	2	2
x	x	x	x	x	Hit	x	x	Hit	x	x	x	x	x	Hit	x	x

6	3	2	1	2	3	6	
3	3	3	3	3	3	3	
7	7	2	2	2	2	2	
6	6	6	1	1	1	6	
x	Hit	x	x	Hit	Hit	x	

Hit = 6

fault = 18

Optimal Page replacement

6	7	0	2	1	2	3	4	2	1	5	6	2	1	2	3	7
6	6	6	6	6	6	3	4	4	4	5	6	6	6	6	6	6
	7	7	7	1	1	1	1	1	1	1	1	1	1	1	3	3
		0	2	2	2	2	2	2	2	2	2	2	2	2	2	7
X	X	X	X	X	H,t	X	X	H,t	H,t	X	X	H,t	H,t	H,t	X	X

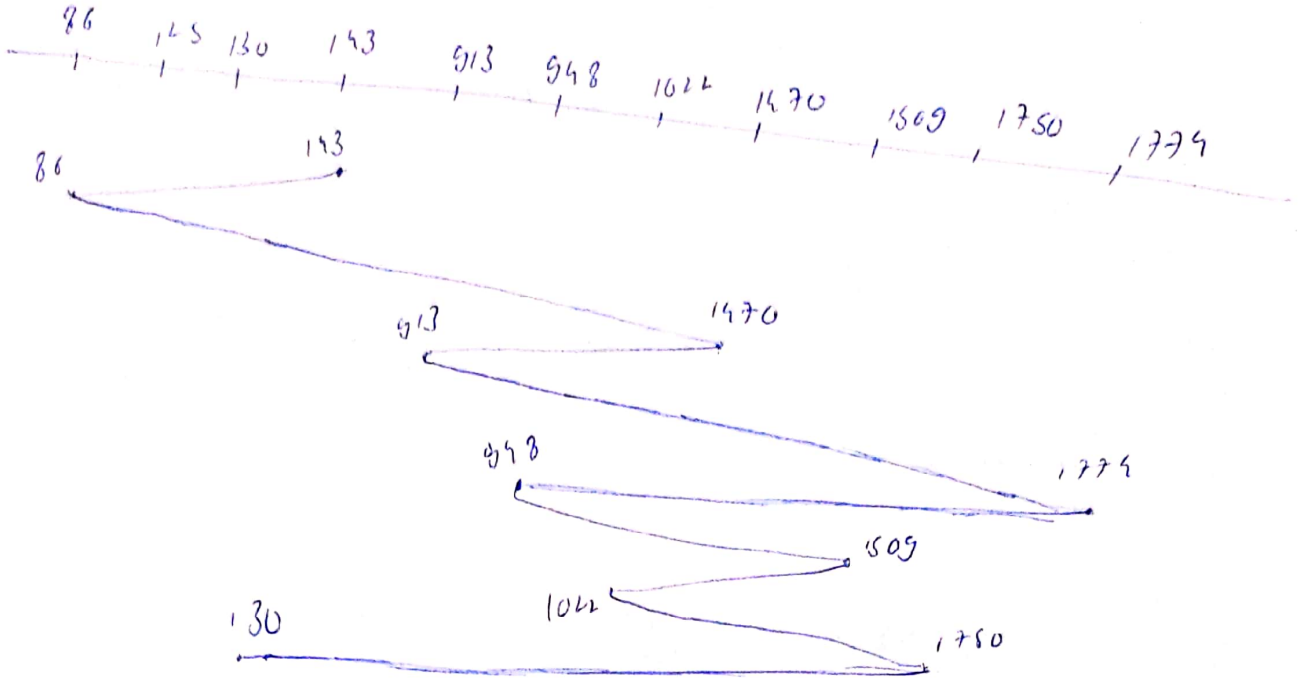
6	3	2	1	2	3	6
6	6	6	1	1	1	6
3	3	3	3	3	3	3
7	7	2	2	2	2	2
H,t	H,t	X	X	H,t	H,t	X

H,t = 10

fault = 14

most efficient is optimal page replacement

Q4 FCFS { 86, 1470, 913 1774 948 1509 1022 1750 130

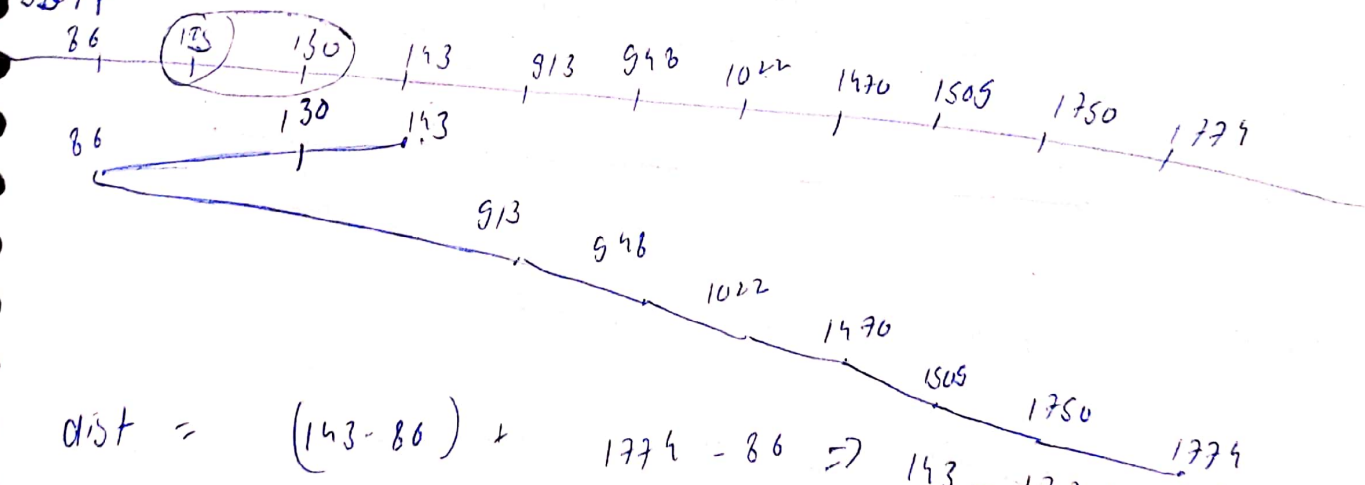


dist

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

$= 7081$

SSTF

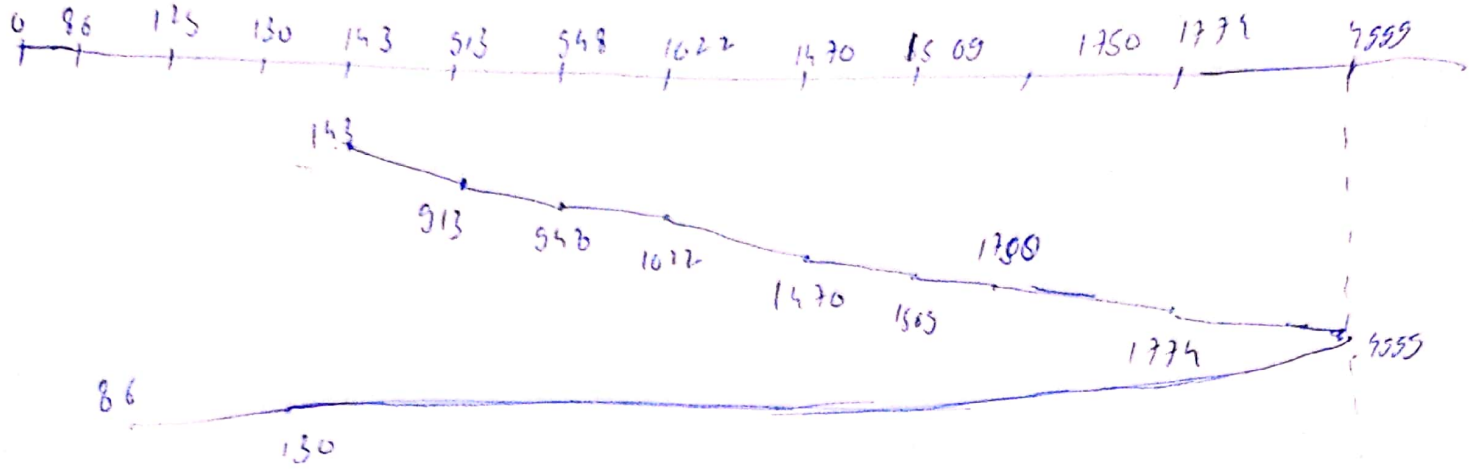


dist $= (143-86) + 1774 - 86 \Rightarrow 143 - 172 + 1774$

$= 1774 - 29$

$= 1745$

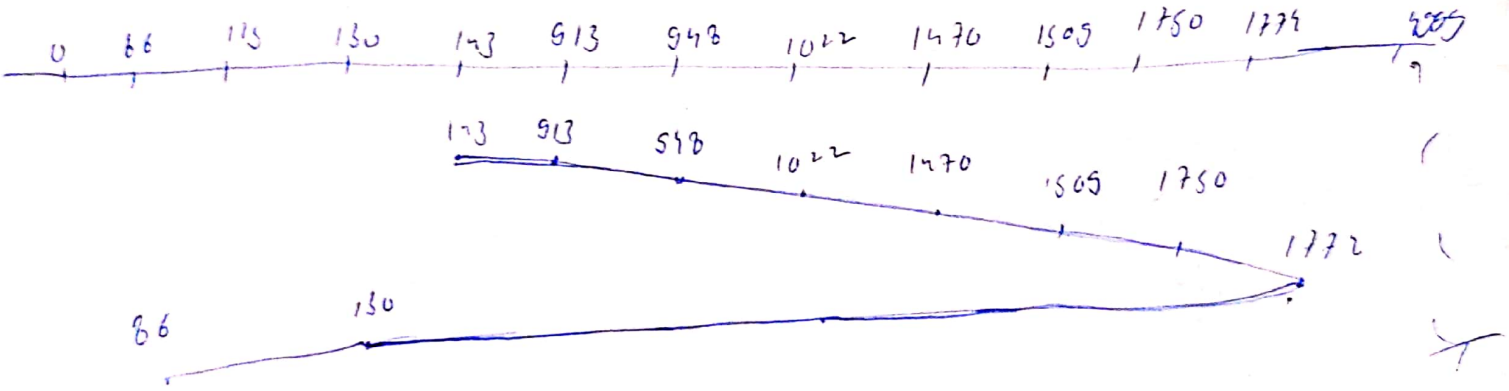
c) SCAN



dist = 4999 - 143 + 4999 - 86

= 9769

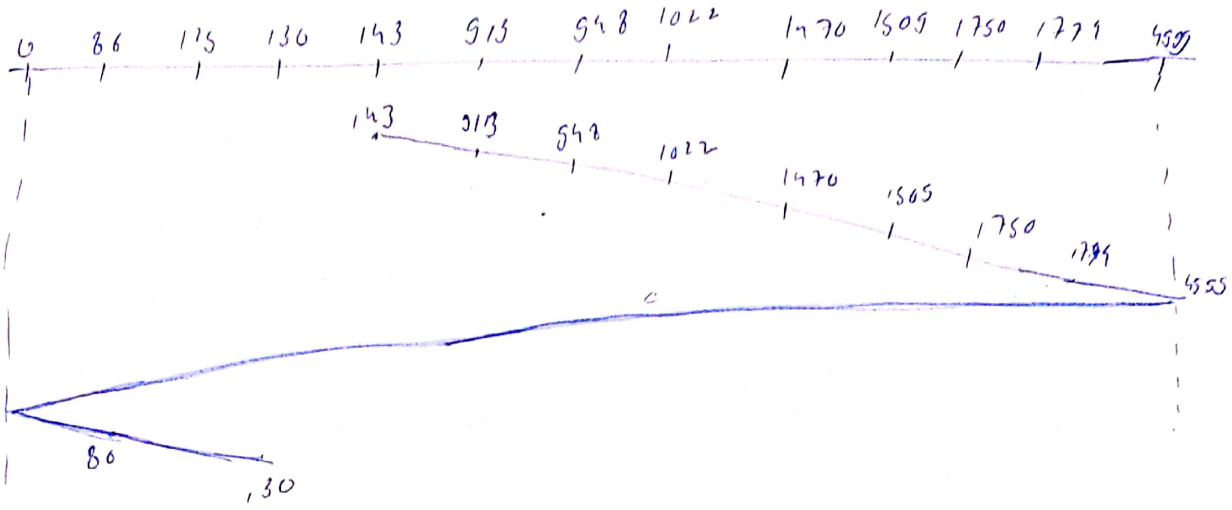
d) Look



dist = 1774 - 143 + 1774 - 86

= 3319

e) CSCAN

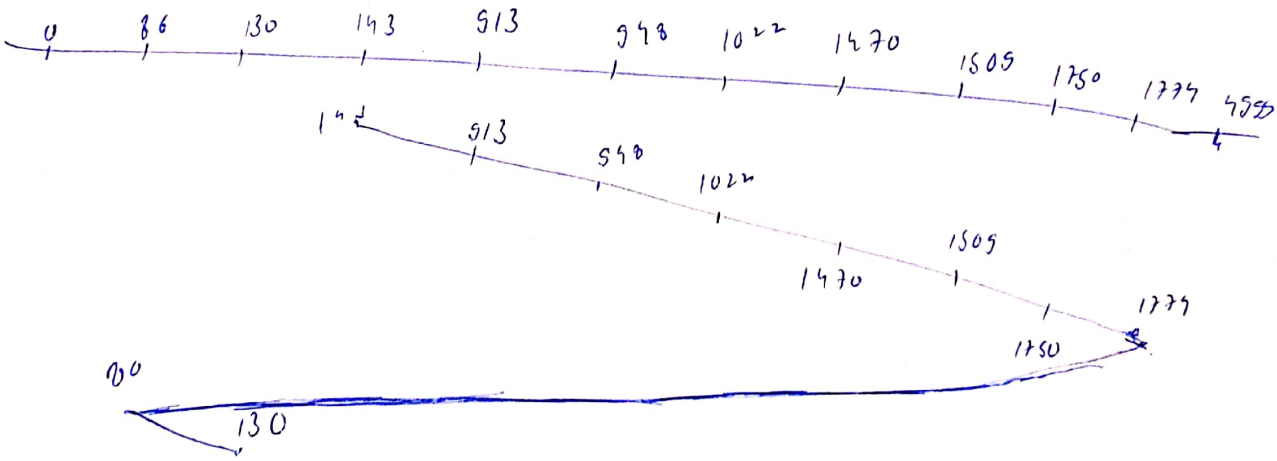


Toll dist = 4999 - 143 + 4999 + 130

= 9998 - 13

= 9985

f) Look



dist = 1774 - 143 + 1774 - 80 + 130 - 80

= 3363

Qs(1)

Thrashing

It is an issue cause when virtual memory is in use it occurs when virtual memory of computer is rapidly exchanging data for data on hard disk to an exclusion of most applications - level processing

→ If in page fault and swapping happening very frequently at higher rate then operating system has to spend more time to swap these pages.

This state is called thrashing

Q1

(a)

	Logical Address	Physical Address
→ Basic	generated by CPU	Location in memory unit
→ Address space	Logical Address space is set of all generated logical Addresses by CPU in reference to a program	Physical address is set of all Physical address mapped to corresponding logical Address
→ visibility	user can view logical address of program	user can never view physical address of program
Creation	generated by CPU	computed By MMU
Access	the user can use logical address to access the physical address	the user indirectly access Physical address but not directly

b) First Fit

212K → 500K memory

417K → 600K memory

112K → 288K (500 - 212K) memory

426K - needs to wait no empty space

Best fit

212K → 300K memory
417K → 500K memory
112K → 200K memory
426K → 600K memory

Worst fit

212K = 600K memory
417K = 500K memory
112K = 388K (600K - 212K) memory
426K needs to wait

Most efficient is Best fit

Preemptive
can be interepted
can be paused

non preemptive
can not be interupted
has to comple in Igo