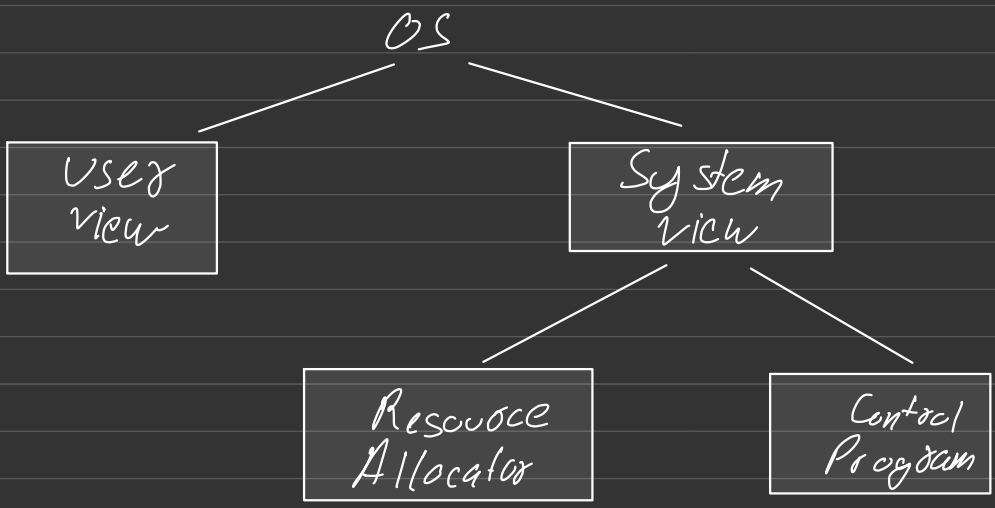
Program is cycle of CPU burst & I/O burst



Batch System: no direct interaction b/w the user & the program

multiprogramming: allows to keep multiple programs in the main memory

features:-
- I/O routine supplied by the system
- Memory management
- CPU scheduling
- Allocation of devices

Timesharing System: allows to share the computer resources rapidly, it switches so rapidly that the time lag isn't observable

short time slot is given to each user for their executions

Need memory protection

Multiprocessing System: Tightly coupled system, supports more than 1 CPU, sharing the computer resources.
Communication usually takes place through shared memory

Advantages: Increased Throughput, Economy of Scale, Increased Reliability
(load can be shared by other cores if one fails)

Symmetric	Asymmetric
if one fails, another will take the load	if failed, master changes
No master slave relation	Allocates work to slave processors
Most modern OS support SMP	More common in extremely large systems
Many processes can run at once without performance deterioration	Each processor assigned specific task; master assign processes to the slave
each processor run the task of OS	Only master runs the tasks of OS
all processor communicate with another by shared memory	need not communicate as they are controlled by the master processor

Real Time OS: well defined, fixed time constraint or system will fail

processing time is fixed
executes within the define time
delay is not possible, it will fail

e.g. controlling scientific experiments, medical imaging,

Soft Real Time OS: delay is possible

Distributed System: computation is distributed, resource sharing is helpful

disadvantages: complex, Security, Manageability, unpredictability

Dual-mode Operation: monitor & user, interrupt, set user mode

Priveledged instructions can be issued only in monitor mode

I/O -> monitor mode

Memory Protection: Protecting memory from diff. programs

29 -> FTP
80 -> HTTP

System Service, Program & call

System Service

- Resource Allocating
- User Interface: Command Line, Graphical
- Program execution: load into main memory to run it
- I/O operations: provide means to perform I/O
- File-System manipulation: program capability to perform ops
- Communications: exchange of info b/w processes
- Error detection • Protection[I] & Security[E]
- Accounting: Health check of cpu, memory &

System program

- Program loading & execution
- File modification
- Status info: system for info - date, time, etc.
- Programming Language Support
- Communications: provide mechanism for creating virtual connections among processes

Imp
System call: interface b/w running program & OS, provides an interface to the service made available by an OS.

OS services are used by programs by system call

interface to use the services provided by the OS

switches from user mode to Kernel mode or monitor mode on system call

Types of OS

Process Control: end, abort, load, execute, create

File management: Create, delete, Open, close, Read, write

Device management: Request, Release, Read write, get device attributes, set device attributes

Information maintainenance: get or set time & date

Communications: Create & delete, Send & Receive

Imp

Kernel is the core functionality which always stays in the main memory. responsible for functionality of OS

Operating System Structure

Monolithic Structure: whole program in one file
e.g. dos

Problems: complexity, protection

Shorter process wait for longer process to execute → Convoy Effect
or release the CPU

FCFS Scheduling Algorithm

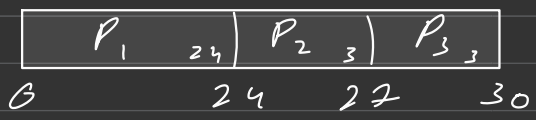
The process that request the CPU first is allocated the CPU first

it's always Non-preemptive

arrival time parameter to be seen for numerical

Process	Burst Time	Arrival Time
P_1	24	0
P_2	3	0
P_3	3	0

Grantt Chart



Avg. waiting time = TT − BT = 27
Avg. turnaround time = CT − AT = 17

	TT	WT
P_1	24 − 0 = 24	24 − 24 = 0
P_2	27 − 0 = 27	27 − 24 = 3
P_3	30 − 0 = 30	30 − 3 = 27

Grantt Chart

	TT	WT
P_2	3	0
P_3	6	3
P_1	30	6

ATT = 13
AWT = 3

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

	TT	WT
P_1	8 − 0 = 8	8 − 8 = 0
P_2	12 − 1 = 11	11 − 4 = 7
P_3	21 − 2 = 19	19 − 9 = 10
P_4	26 − 3 = 23	23 − 5 = 18

ATT = 15.25
AWT = 8.75

Process	BT	AT
P_1	4	2
P_2	3	3
P_3	2	0
P_4	1	5
P_5	3	4

P_1	$6 - 2 = 4$	$4 - 4 = 0$
P_2	$12 - 5 = 7$	$7 - 3 = 4$
P_3	$2 - 0 = 2$	$2 - 2 = 0$
P_4	$13 - 5 = 8$	$8 - 1 = 7$
P_5	$9 - 4 = 5$	$5 - 3 = 2$

ATT $= 26/5 = 5.2$

AWT $= 13/5 = 2.6$

Shortest Job First Algorithm

CPU is allocated to the process with next shortest burst time

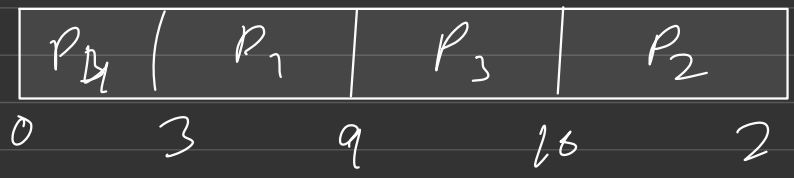
non-preemptive

preemptive ⟶ Shortest remaining time first

difficult to implement since it's not always to predict the execution time of the job
hence it's considered as a benchmark algorithm & not implemented in the OS

Optimal algorithm: avg or minimal waiting time

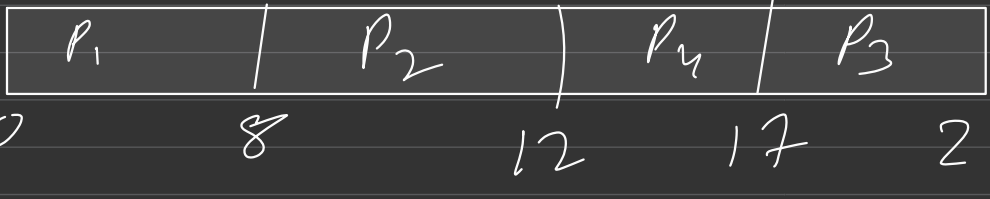
Process	CPU Burst
P_1	6
P_2	8
P_3	7
P_4	3



	TT	WT
P_1	$9 - 0 = 9$	$9 - 6 = 3$
P_2	$24 - 0 = 24$	$24 - 8 = 16$
P_3	$16 - 0 = 16$	$16 - 7 = 9$
P_4	$3 - 0 = 3$	$3 - 3 = 0$

Avg TT = 52/4 = 13
Avg WT = 28/4 = 7

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



	ATT	AWT
P_1	$8-0=8$	$8-8=0$
P_2	$12-1=11$	$11-4=7$
P_3	$26-2=24$	$24-9=15$
P_4	$17-3=14$	$14-5=9$

$ATT = 57/4 = 14.25$

$AWT = 31/4 = 7.75$

Preemptive

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

	ATT	AWT
P_1	$17-0=17$	$17-8=9$
P_2	$5-1=4$	$4-4=0$
P_3	$26-2=24$	$24-9=15$
P_4	$10-3=7$	$7-5=2$

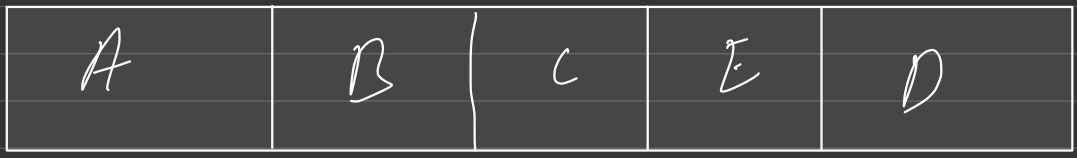
ATT = 13 AWT = 6.5

Process	AT	Execution Time
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Preemptive

	ATT	AWT
A	8 - 0 = 8	8 - 6 = 2
B	5 - 3 = 2	2 - 2 = 0
C	12 - 5 = 7	7 - 4 = 3
D	21 - 7 = 14	14 - 6 = 8
E	13 - 10 = 3	5 - 3 = 2
	7.2	3

Non-preemptive



0 6 8 12 15 21

	ATT	AWT
A	6-0=6	6-6=0
B	8-3=5	5-2=3
C	12-5=7	7-4=3
D	15-7=8	8-6=2
E	21-10=11	11-3=8
	7.4	3.2

Priority Scheduling

Preemptive

Non-Preemptive

Drawback:

Starvation: higher priority process takes CPU from low priority process causing a possibility that the low priority process never gets the CPU for its execution

Solution:

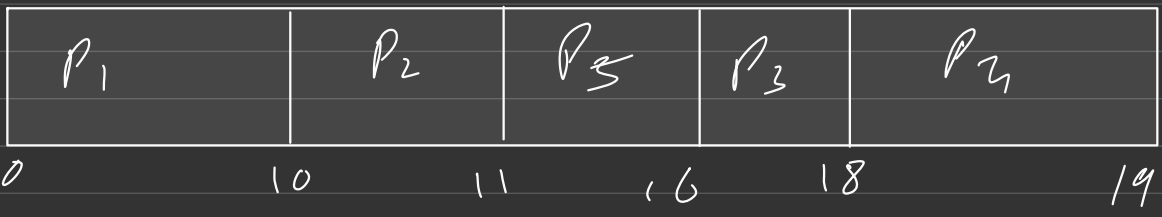
aging: gradually increases the priority of a process as time progresses so eventually low priority process will become a high priority process & will get a CPU

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

	TT	WT
P_1	16 - 0 = 6	16 - 10 = 6
P_2	1 - 0 = 1	1 - 1 = 0
P_3	18 - 0 = 18	18 - 2 = 16
P_4	19 - 0 = 19	19 - 1 = 18
P_5	6 - 0 = 6	6 - 5 = 1
	12	8.2

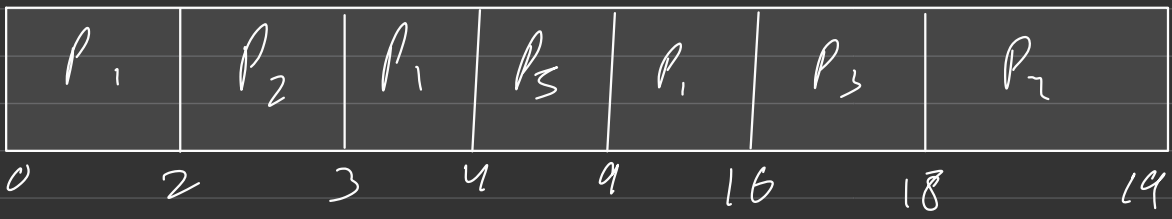
Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	1	1	2
P_3	2	4	3
P_4	1	5	3
P_5	5	2	4

Non - preemptive



	TT	WT
P_1	10 - 0 = 10	10 - 10 = 0
P_2	11 - 2 = 9	9 - 1 = 8
P_3	18 - 3 = 15	15 - 2 = 13
P_4	19 - 3 = 16	16 - 1 = 15
P_5	16 - 4 = 12	12 - 5 = 7
	12.4	8.6

Preemptive



	TT	WT
P_1	$16 - 0 = 16$	$16 - 10 = 6$
P_2	$3 - 2 = 1$	$1 - 1 = 0$
P_3	$18 - 3 = 15$	$15 - 2 = 13$
P_4	$19 - 3 = 16$	$16 - 1 = 15$
P_5	$9 - 4 = 5$	$5 - 5 = 0$
Avg	10.6	6.8

Process	Arrival Time	Burst Time	Priority
P_1	0	4	2
P_2	1	3	3
P_3	2	1	4
P_4	3	5	5
P_5	4	2	5

	TT	WT
P_1	$4 - 0 = 4$	$4 - 4 = 0$
P_2	$7 - 1 = 6$	$6 - 3 = 3$
P_3	$8 - 2 = 6$	$6 - 1 = 5$
P_4	$13 - 3 = 10$	$10 - 5 = 5$
P_5	$15 - 4 = 11$	$11 - 2 = 9$

7.4 | 4.4

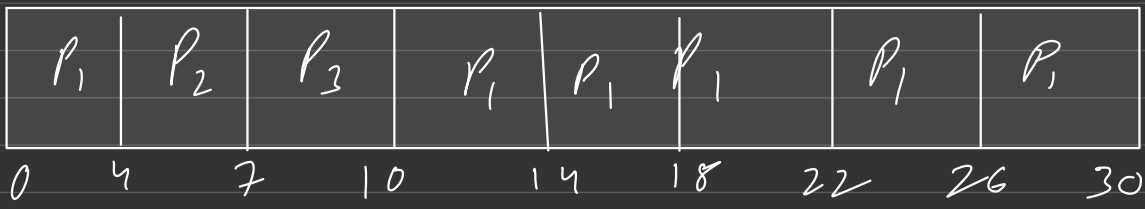
Round Robin Algorithm

fixed amount of time is given to each process for completing its execution

CPU is passed to the next process in queue after the alloted time or the execution is over

Process	Burst time
P_1	24
P_2	3
P_3	3

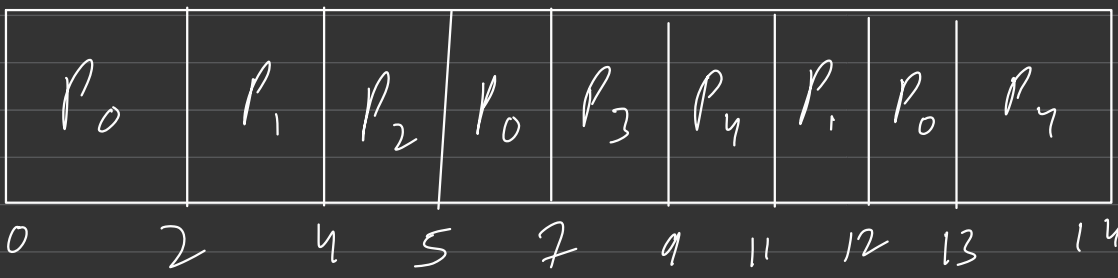
Time Quantum = 4



	TT	WT
P_1	30 − 0 = 30	30 − 24 = 6
P_2	7 − 0 = 7	7 − 3 = 4
P_3	10 − 0 = 10	10 − 3 = 7
Avg	15.67	5.67

Process	Burst time	Arrival time
P_0	5	0
P_1	3	1
P_2	1	2
P_3	2	3
P_4	3	4

Time Slice = 2 sec



	TT	WT
P_0	13 − 0 = 13	13 − 5 = 8
P_1	12 − 1 = 11	11 − 3 = 8
P_2	5 − 2 = 3	3 − 1 = 2
P_3	9 − 3 = 6	6 − 2 = 4
P_4	14 − 4 = 10	10 − 3 = 7
	8.6	5.8

Multilevel Queue

Modern OS maintain multiple queue for allocating diff process because different types of process require different type of algorithm

for e.g. background process require diff. OS than the interactive processes

OS also has to decide how much CPU needs to be allocated for diff. process at each level & which algorithm to be used

diff. OS uses diff algos.

Multilevel Feedback Queue

the processes move in between process at diff. level to complete the execution of each process (lower priority)

therefore, processes aren't particularly assigned to a single queue they move in diff. priority queue

Q. Consider a system which has a CPU bound process which requires the burst time of 40secs
multilevel feedback queue is used
TQ = 2secs
and at each level it's incremented by 5secs
How many interuptions?
& at which queue the process will terminate its execution.

4 interuptions

terminated at 5th queue

Q Consider 3 processes all arriving at time 0 with total execution time of 10, 20, 30 resp. each process spends the first 20% doing I/O next 70% computation & last 10% I/O again

The OS uses shortest remaining compute time first scheduling algorithm & schedules a new processes either when the running process gets blocked on I/O or finishes it compute burst

Assume that all I/O ops can be overlapped as much as possible
For what % of time CPU remain idea) & Calculate avg. waiting time

Process	Burst time	Arrival time
P_1	7	0
P_2	14	0
P_3	21	0

Process	TT	WT
P_1	$-0=$	$11-7=4$
P_2	$25-0=25$	$25-14=11$
P_3	$47-0=47$	$47-21=26$
		13.67

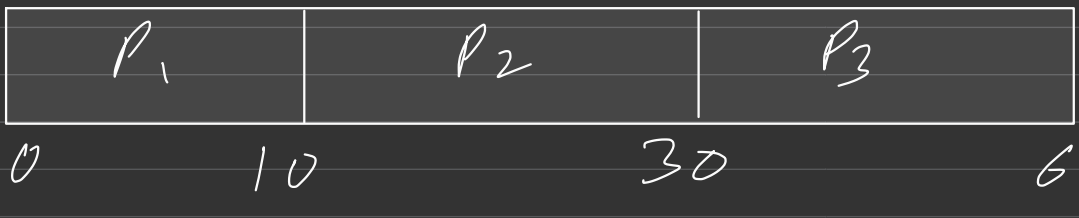
idle time % $= \frac{2+3}{47} \times 100$

$= 10.6\%$

Avg WT $= 5+19+$

6. Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

Process	AT	CPU
P_1	0	10
P_2	2	20
P_3	6	30



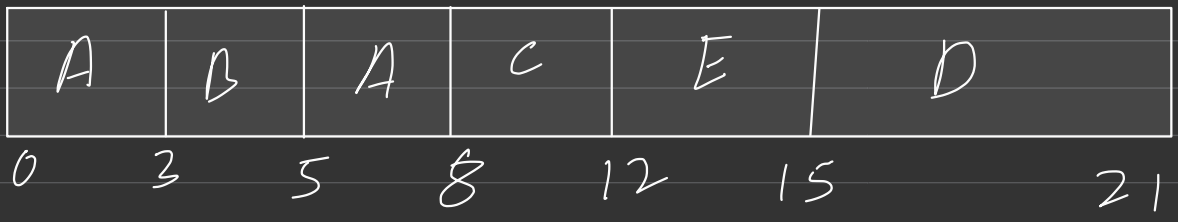
Context switching = 2

Q Consider the following set of processes that need to be scheduled on a single CPU. All the times are given in milliseconds? (GATE-2014) (2 Marks)

Process Name	Arrival Time	Execution Time	CT	TAT	WT
A	0	6	8	8	2
B	3	2	5	2	0
C	5	4	12	7	3
D	7	6	21	14	8
E	10	3	15	5	2

7.2

Using shortest remaining time first scheduling algorithm calculate average turnaround time



Deadlock: when a process is holding a resource and waiting for another resource held by a process

waiting process may never change its state as the required resources are held by other waiting processes

no premption; none of the resource can be forcefully taken back

occurs only when every process is waiting for anothe pre occupied resource

Mutual Exclusion: some of the resources should be non shareable to prevent deadlock

Hold & wait: process is holding a process and simultaneously waiting for another occupied resource

No preemption: process releases the resource itself only after the process is completed

Circular wait:

Resource Allocation - Graph

set of vertices & edges

if a cycle is present in the graph
it means the system is in deadlock state

if no cycle, not in deadlock

if multiple instances doesn't have a cycle
the system is not in deadlock

if multiple instances have a cycle
the system may or may not be in deadlock

Methods for Handling Deadlock

Prevention: ensure to never enter deadlock state by ensuring one of the four necessary conditions doesn't occur

Avoidance: ensure to never enter unsafe mode after memory allocation if the system is in safe mode then only memory will be allocated to next

Detection: allow to enter then recover runs frequently to check

Do Nothing: let the user or admin handle used by most OS, windows & UNIX

It's very expensive to use deadlock algorithms since it will be frequently using computation

Mutual exclusion can not be avoided since it's not possible to make all the resources as sharable

Hold & Wait: we must guarantee that whenever a process requests a resource, it doesn't hold any other resources

not efficient

First protcol: require a process to request & be allocated all its resources before it begins execution

Result. low resource utilization, starvation possible

An alternative protocol: process can request another resource only if it releases the current resource

No-preemption: process can not be taken

release process when it the process goes into waiting stale

Alternatively: releases resource when a process requests & the holding process is in waiting state

Circular Wait: imposes a total ordering of all resource types, & require that each process requests in an ↑ order of enumeration

Deadlock Avoidance: requires that the system has some additional *a priori* information available

simplest & most useful model requires each process to declare the max. number of resources of each type that it may need

dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

a resource-allocation state is defined by the no. of available & allocated resources & the max demands of the process

when a process requests an available resource, the OS must ensure that it's in a safe state

safe state => no deadlock

unsafe state => possibility of deadlock

avoidance => ensures that the OS never enters the unsafe state

it decides weather to give available resource to a requested process or not

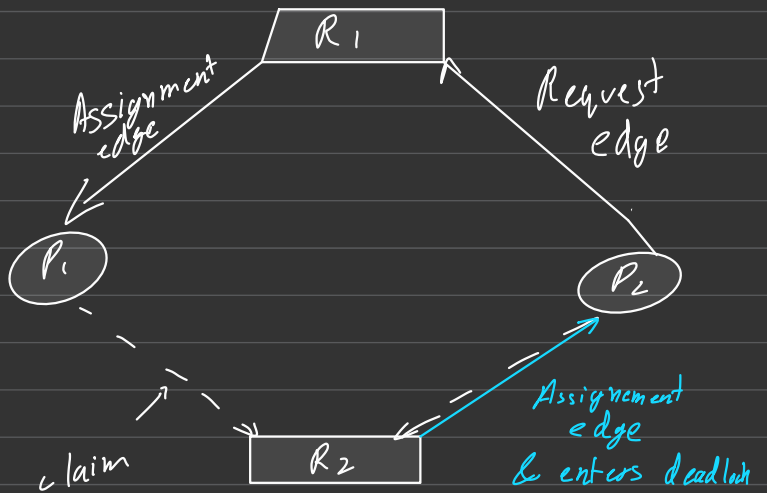
for a single instance -> resource-allocation graph

for a multiple instance -> banker's algorithm

Resource Allocation Graph Scheme

claim edge: future need of process

represted by a dashed line
converts to request edge upon requset
converts to assignment edge when assigned to the process



Q four instances A, B, C, D
A - 6, B - 3, C - 4, D - 2

Process	Allocation A B C D	Max A B C D	Need A B C D
P_1	3 0 1 1	4 1 1 1	1 1 0 0
P_2	0 1 0 0	0 2 1 2	0 1 1 2
P_3	1 1 1 0	4 2 1 0	3 1 0 0
P_4	1 1 0 1	1 1 0 1	0 0 0 0
P_5	0 0 0 0	2 1 1 0	2 1 1 0

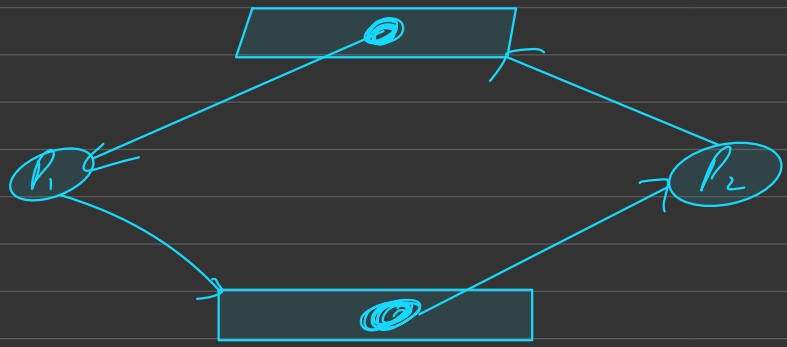
Available -> A - 1, B - 0, C - 2, D - 0

< P_4 P_5 P_1 P_2 P_3 >

6 3 4 2 -> 1 0 2 0 -> 0 0 1 0

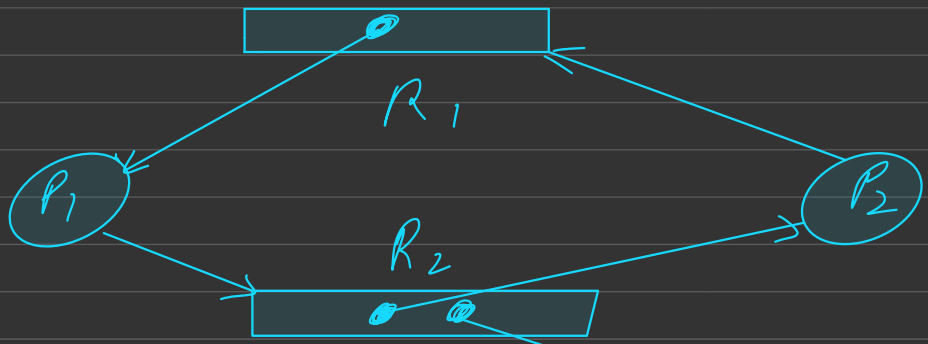
(a) Yes (b) Yes

Q



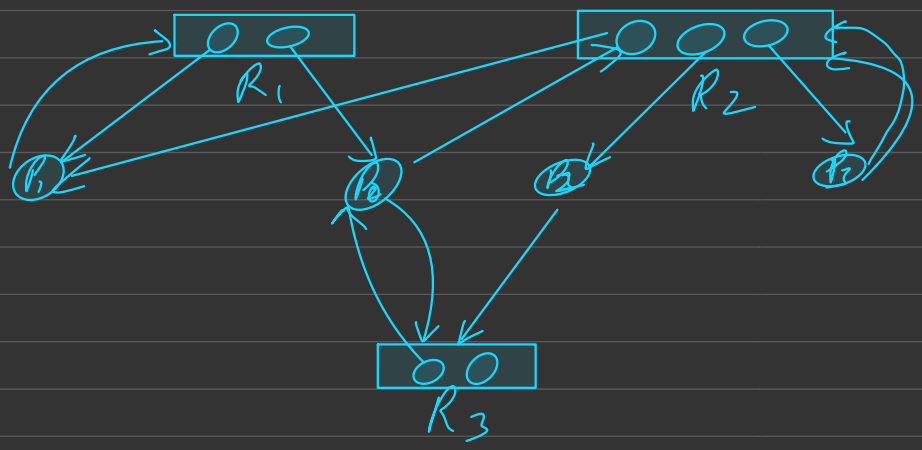
Deadlock

Q



Process	Allocation R_1	Allocation R_2	Need R_1	Need R_2	Available R_1	Available R_2
P_1	1	0	0	1	0	0
					0	1
P_2	0	1	1	0	1	0
					1	1
P_3	0	1	0	0	0	1
					1	2

Q1



Process	Allocation R_1 R_2 R_3	Need R_1 R_2 R_3
P_0	1 0 1	0 1 1
P_1	1 1 0	1 0 0
P_2	0 1 0	0 0 1
P_3	0 1 0	0 2 0

2 3 2

Available R_1 R_2 R_3

0 0 1
0 1 0
0 1 1
1 0 1
1 1 2
1 1 0

$< P_2$ P_0 P_1 $P_3 >$

2 2 2
0 2 0
2 0 2
0 3 0
---
2 > 2

Safe

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
T_0	3	0	1	4	5	1	1	7	2	1	0	3
T_1	2	2	1	0	3	2	1	1	1	0	0	1
T_2	3	1	2	1	3	3	2	1	0	2	0	0
T_3	0	5	1	0	4	6	1	2	4	1	0	2
T_4	4	2	1	2	6	3	2	5	2	1	1	3

$\langle T_1\ T_2\ T_3\ T_4\ T_0 \rangle$

```
  1  0  0  2
+ 2  2  1  0
  ---------
  3  2  1  2
+ 3  1  2  1
  ---------
  6  3  3  3
  0  3  1  0
  ---------
  6  8  4  3
+ 4  2  1  2
  ---------
 10 10  5  5
  3  0  1  4
  ---------
 13 10  6  9
```

Safe

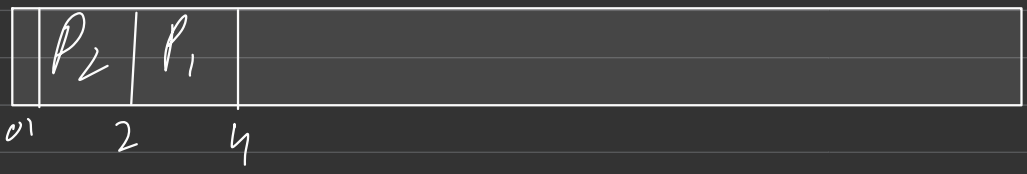
$\langle T_2\ T_1\ T_3\ \ \rangle$

```
  0  3  0  1
- 0  2  0  0
  ---------
  0  1  0  1
+ 3  3  2  1
  ---------
  3  4  2  2
- 1  0  0  1
  ---------
  2  4  2  1
+ 3  2  1  1
  ---------
  5  6  3  2
+ 0  5  1  0
  ---------
  5 11  4  2
```

Deadlock

Process	BT	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	3	3

FCFS



Compaction: shuffle memory content to place all free memory together in one block

Paging: non-contiguous memory allocation where physical memory is divided into fixed size block, known as frame and logical memory is divided into same size block, known as pages

Pages of a process are loaded into the memory wherever free frames are available

OS has to keep track of each page
so it creates a page table, tracking where each page is loaded
stores the base address of the pages in physical memory (frame address)

The logical address generated by CPU is divided into two parts:

Page Number: used as an index in page table

Offset: within a page, combined with base address to define the physical memory address which is sent to the memory unit

frame address + offset = Physical address
or
base address

If logical address of n-bits and m-bits are required for offset

Page number = m-n

Page size = 2^n

Q Consider a logical address space of 64 pages of 1024 words mapped into a physical memory of 32 frames.
How many bits are reqd. in a LA & PM

Physical → offset = 10 = 1024 = 2^{10}
Frame = 5 = 32 = 2^5
15

LA = 10+6 = 16 $2^6 = 64$

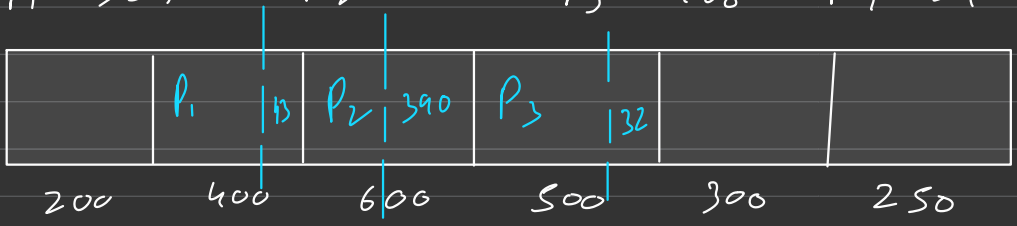
Q If 18 address bit. What is memory size?
if every loc is byte addressable
i.e. each instruction is of 1 byte

Algo for storage allocation

- first fit: allocate the first available hole
- best fit: allocate the smallest available hole
- woost fit. allocate the largest available hole

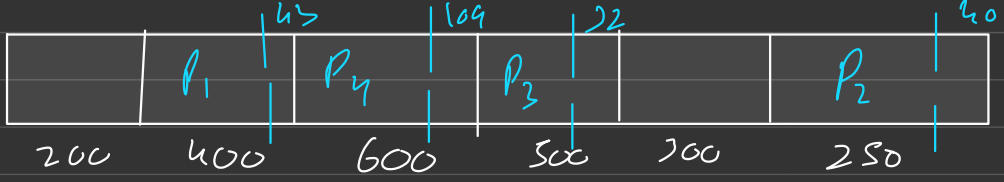
Q Given 6 memory partitions of size 200, 400, 600, 500, 300, 250 in order. How would the first fit, best fit and woost fit algo place processes in order. Which algo makes the most efficient use of memory

$P_1 - 357$ $P_2 - 210$ $P_3 - 468$ $P_4 - 491$



P_4 Has to wait

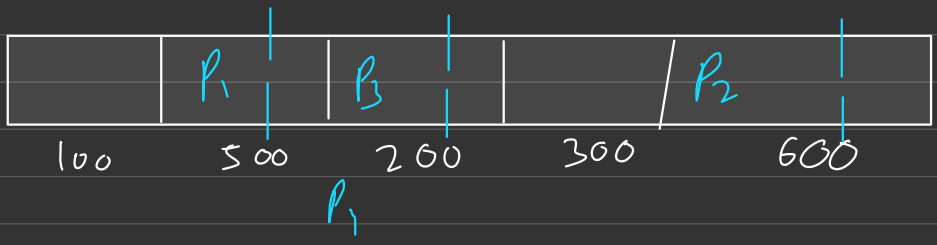
internal fragmentation = 43 + 390 + 32 = 465



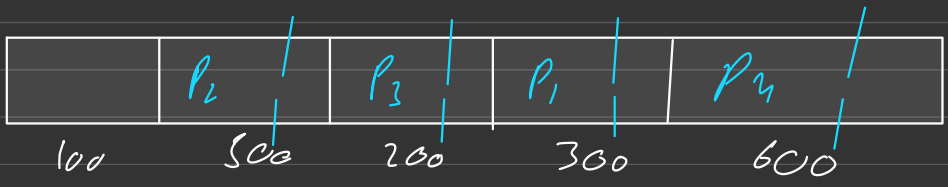
internal fragmentation = 43 + 109 + 32 + 40

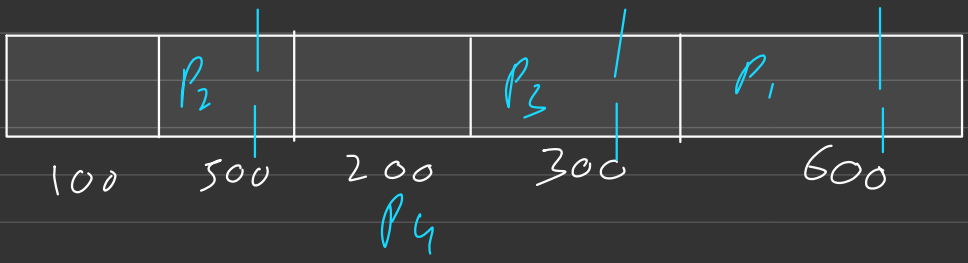
=

Q Given 5 memory partition of 100, 500, 200, 300, 600, in order How would all the algos place processes of 212, 417, 112, 426



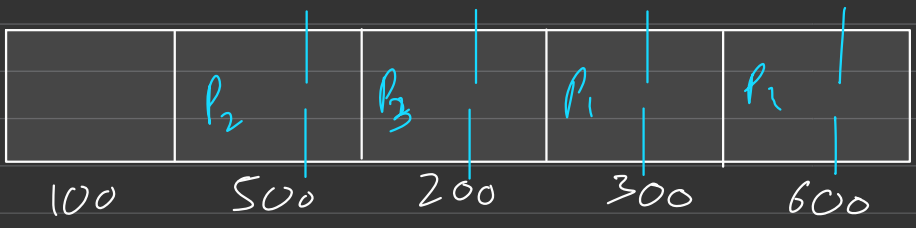
P1

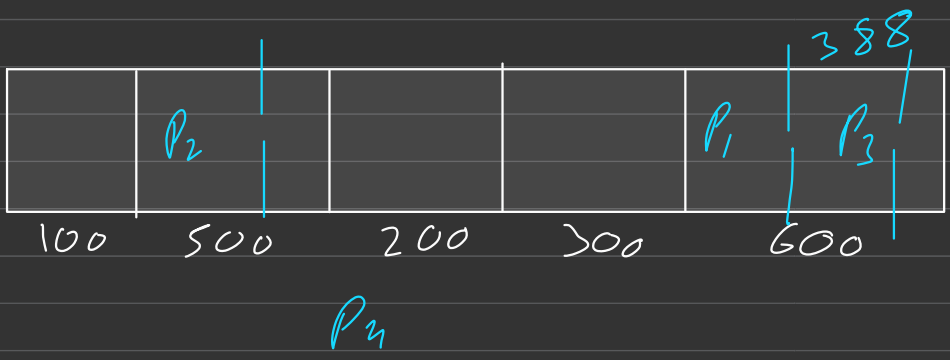




P4

212 , 417 , 112 , 426





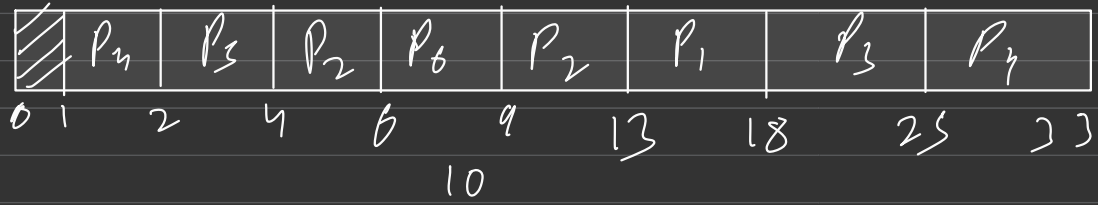
Process	Burst	Priority
P_1	8	4
P_2	6	1
P_3	1	2
P_4	9	2
P_5	3	3

FCFS

Process	Allocation	Max	Need
P_0	0012	0012	0000
P_1	1000	1750	0750
P_2	1354	2356	1002
P_3	0632	0652	0020
P_4	0014	0656	0642

```
  1 5 2 0
- 0 0 2 0
-------
  1 5 0 0
+ 0 6 5 2
-------
  1 11 5 2
- 0 6 4 2
-------
  1 5 1 0
    0
```

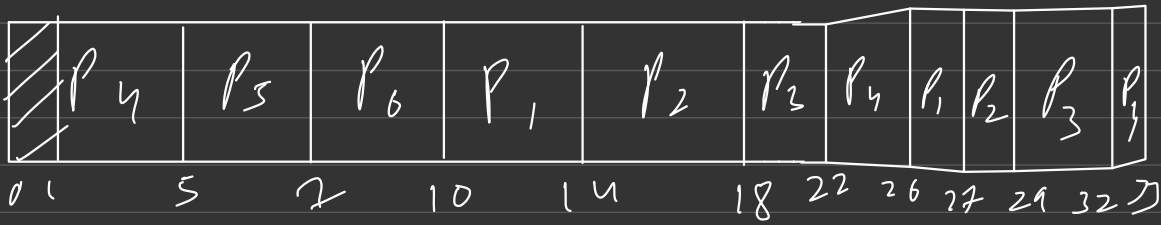
Process	BT	AT
P_1	~~5~~	5
P_2	~~6 - 4~~	4
P_3	~~7 -~~	3
P_4	~~8~~	1
P_5	~~2~~	2
P_6	~~3~~	6



	TT	WT
P_1	$18 - 5 = 13$	$13 - 5 = 12$
P_2	$13 - 4 = 9$	$9 - 6 = 3$
P_3	$25 - 3 = 22$	$22 - 7 = 15$
P_4	$33 - 1 = 32$	$32 - 8 = 24$
P_5	$4 - 2 = 2$	$2 - 2 = 0$
P_6	$9 - 6 = 3$	$3 - 3 = 0$
		$\Rightarrow 9$

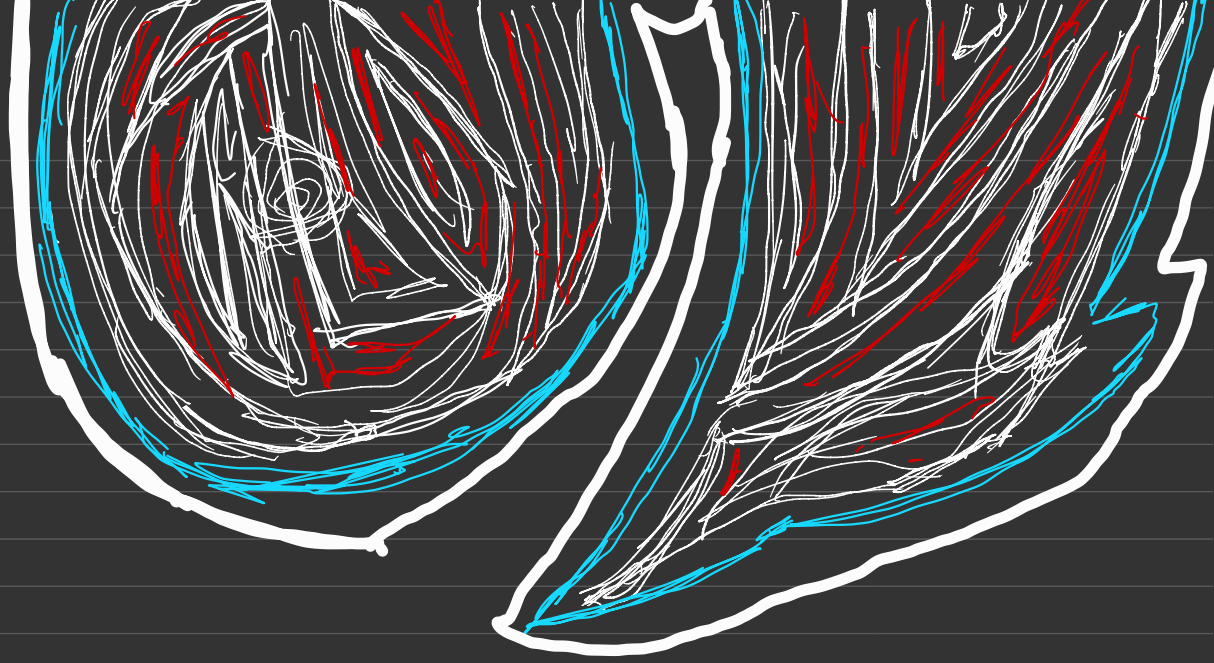
$\frac{54}{6}$

3 4 3 1 2 6



	TT	WT
P_1	$27-5=22$	$22-5=17$
P_2	$29-4=25$	$25-6=19$
P_3	$32-3=29$	$29-7=22$
P_4	$33-1=32$	$32-9=23$
P_5	$7-2=5$	$5-2=3$
P_6	$10-6=4$	$4-3=2$

$\Rightarrow 14.$

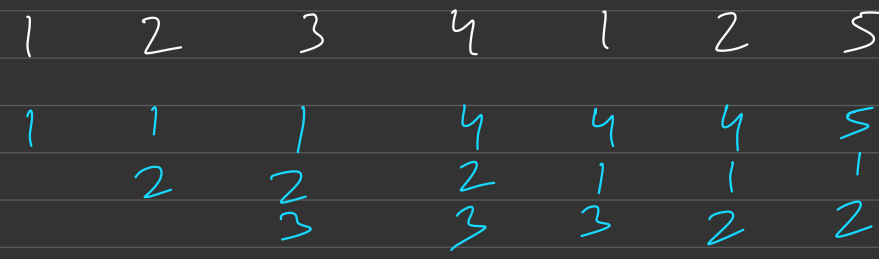


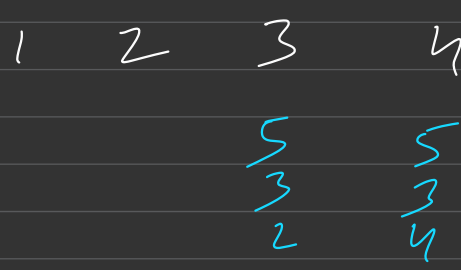
7 0 1 2 0 3 0 4 2

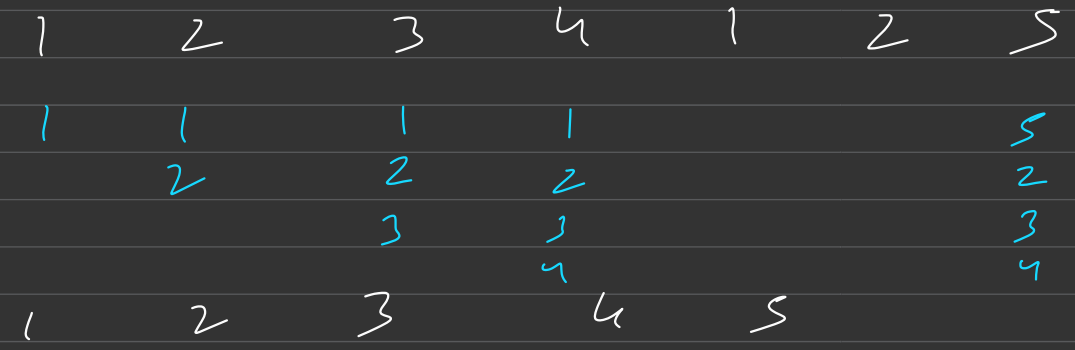
7 7 7 2 2 4 4
0 0 0 3 3 2
1 1 0 0 0

3 0 3 2 1 2 0 1 7 0 1

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


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


1 2 3 4 5
5 5
3 3
2 4

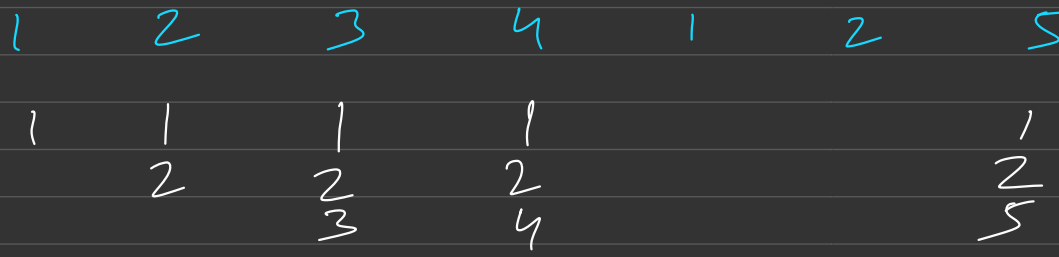

1 2 3 4 1 2 5
1 1 1 1 5
2 2 2 2
3 3 3
4 4
1 2 3 4 5
5 5 5 4 4
1 1 1 1 5
3 2 2 2 2
4 4 3 3 3

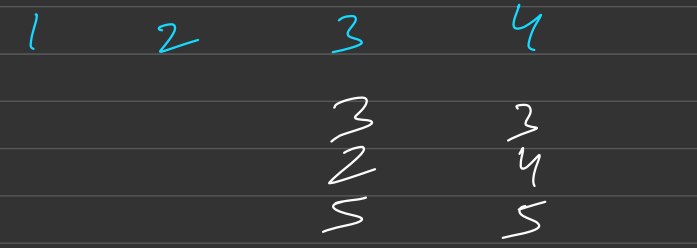
Belady's Anomaly: in case of FIFO, by increasing no. of frames of a process the page fault increases

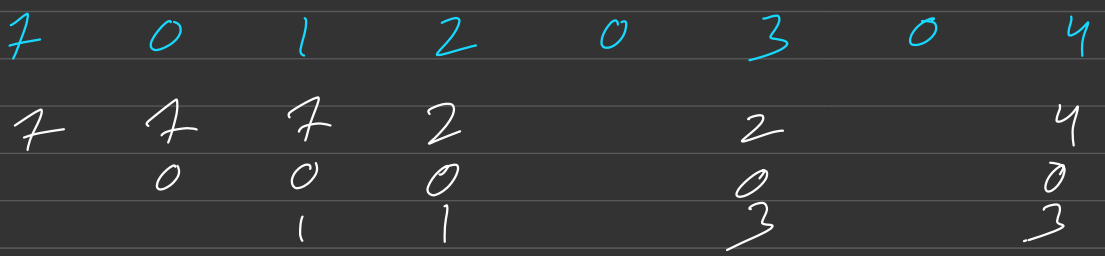
7	0	1	2	0	3	0
7	7	7	2		2	
	0	0	0		0	
		1	1		3	

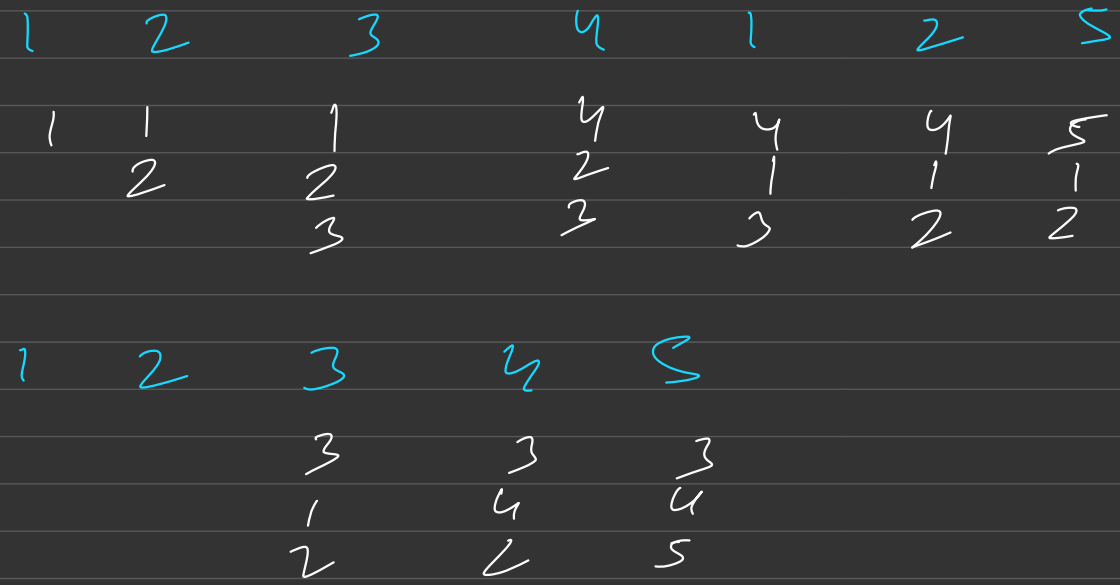
4	2	3	0	3	2	1	2
2			2			2	
4			0			0	
3			3			1	

0	1	7	0	1
		7		
		0		
		1		


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


1 2 3 4 5
3 3
2 4
5 5


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



Imp

Thrashing

high paging activity -> page fault rate is very high

working set model
page-fault frequency

PFF

take frame from a process which generate
page fault < lower bound, and
allocate the frames to the process generating
page fault > upper bound

WSM: based on the concept of locality model

set of pages that a process is currently using is called working set

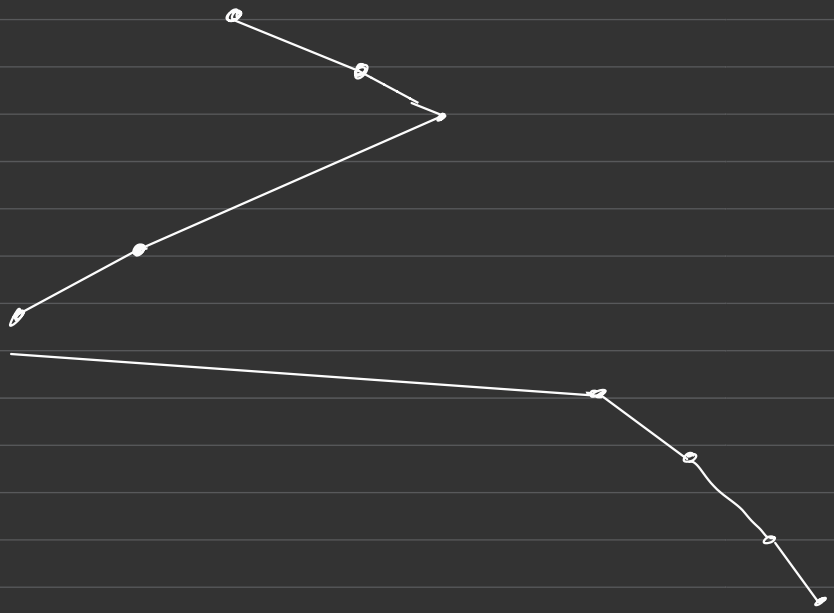
SSTF (Nearby requests)

Shortest Seek Time First

it may cause starvation of some requests

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

0 14 37 53 65 67 98 122 124 183



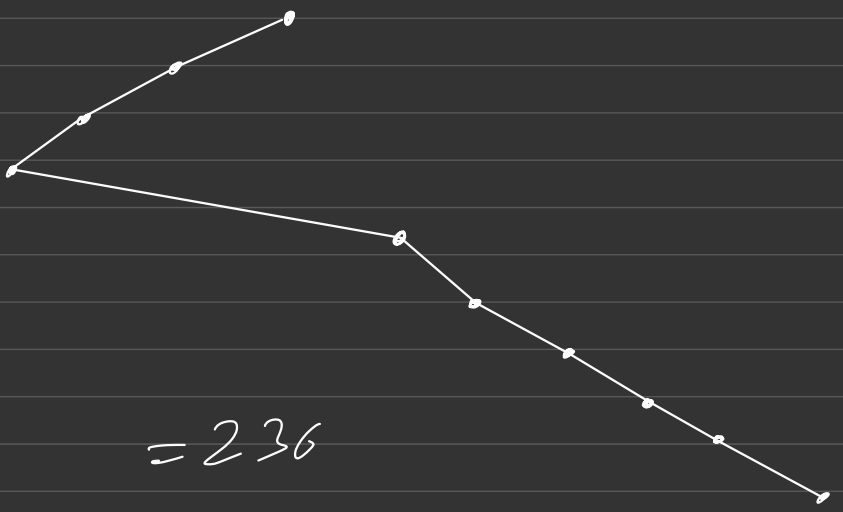
67 - 53 + 67 - 14 + 14 - 98
14 + 13 +

= 236

SCAN (Elivator requests)

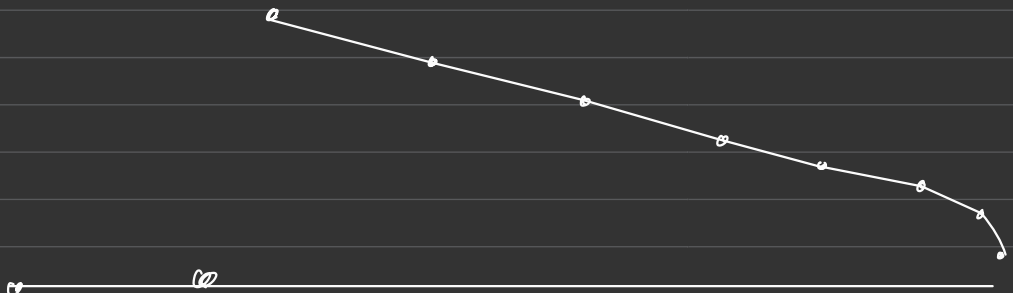
The disk arm starts at one of the disk & moves toward the other end, servicing request until it gets to the end where the head movement is reversed

0 14 37 53 65 67 98 122 124 183 199



= 236

0 14 37 53 65 67 98 122 124 183

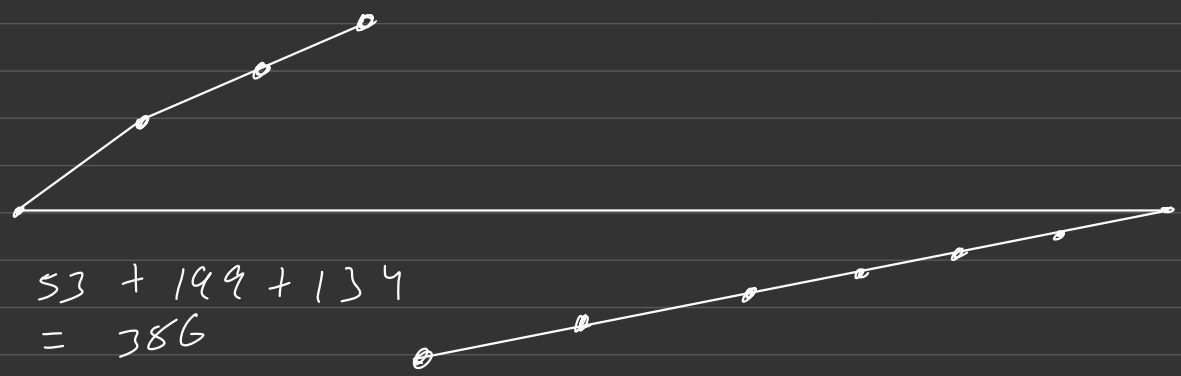


= 299

C-SCAN

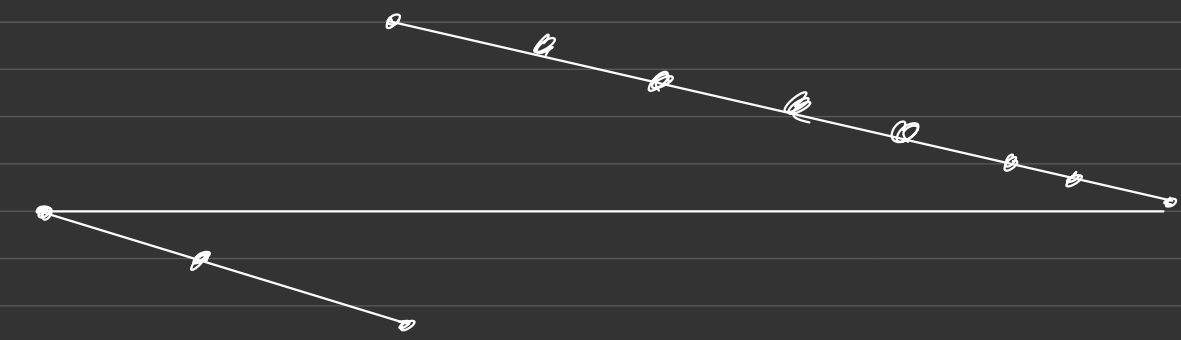
The head moves from one end of the disk to the other, when reaching to one end, returns to the beginning

0 14 37 53 65 67 98 122 124 183 199



53 + 199 + 134
= 386

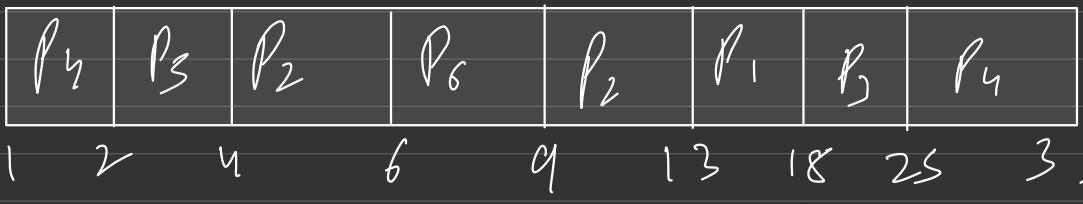
0 14 37 53 65 67 98 122 124 183 199



= 145 + 199 + 77
= 382

SJF

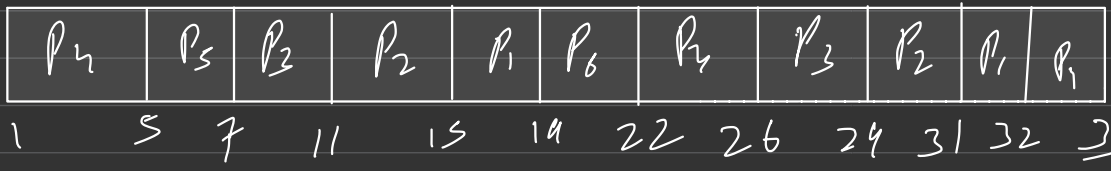
P_1	5	5
P_2	6	4
P_3	7	3
P_4	9	1
P_5	2	2
P_6	3	6



P_1	$18 - 5 = 13$	$13 - 5 = 8$
P_2	$13 - 4 = 9$	$9 - 6 = 3$
P_3	$25 - 3 = 22$	$22 - 7 = 15$
P_4	$33 - 1 = 32$	$32 - 9 = 23$
P_5	$4 - 2 = 2$	$2 - 2 = 0$
P_6	$3 - 6 = 3$	$3 - 3 = 0$

P_4 P_5 P_3 P_2 P_1 P_6

8 2 7 6 5 3



P_1	$32 - 5 = 27$	$27 - 5 = 22$
P_2	$31 - 4 = 27$	$27 - 6 = 21$
P_5	$29 - 3 = 26$	$26 - 7 = 19$
P_4	$33 - 2 = 30$	$30 - 9 = 21$
P_5	$7 - 1 = 6$	$6 - 2 = 4$
P_6	$22 - 6 = 16$	$16 - 3 = 13$

	Allocation	Maximum	Needed	Available
P_0	0012	0012	0000	1520
P_1	1000	1750	0750	0012 / 1532
P_2	1354	2356	1002	2356 / 3888
P_3	0632	0652	0020	0652 / 3140 (13) 10
P_4	0014	0656	0642	0656 / 3201 (8) 16
				1750 / 4272 (23) 16

P_0 P_2 P_3 P_4 P_1

64 pages 1024 words 32 frames

$2^6 \times 2^{10} = 2^{16} = 16$ bits

$2^5 \times 2^{10} = 2^{15} = 15$ bits

1 2 3 4 2 1 5 6

1 1 3 3 2 2 5 5
2 2 4 4 1 1 6

2 1 2 3 7 6 3 2

2 2 2 2 7 7 3 3
6 1 1 3 3 6 6 2

1 2 3 6

1 1 3 3
2 2 2 6

1 2 3 4 2 1 5 6 2
1 1 1 4 4 4 4 6 6
2 2 2 2 1 1 1 2
3 3 3 3 5 5 5

1 2 3 7 6 3 2 1 2 3 6
6 6 3 3 3 3 2 2 2 2 6
2 2 2 7 7 7 7 1 1 1 1
1 1 1 1 6 6 6 6 6 3 3

1 2 3 4 2 1 5 6

1 1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 4 4 4 5 6

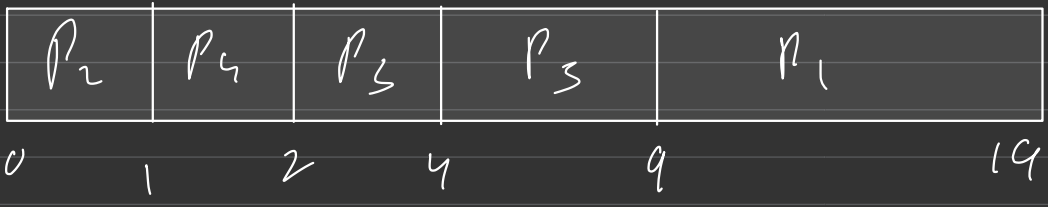
2 1 2 3 7 6 3 2

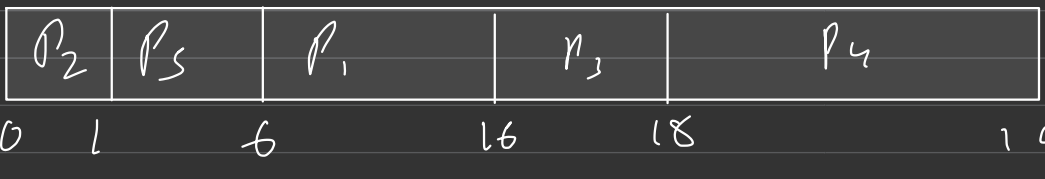
1 1 1 3 3 3 3 3
2 2 2 2 7 7 7 2
0 0 6 6 6 6 6 6

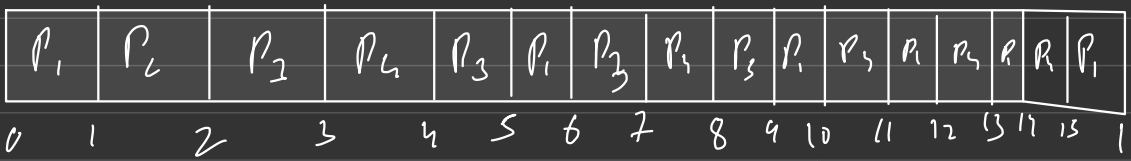
1 2 3 6

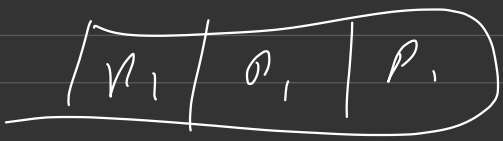
3 3 3 3
2 2 2 2
1 1 1 6

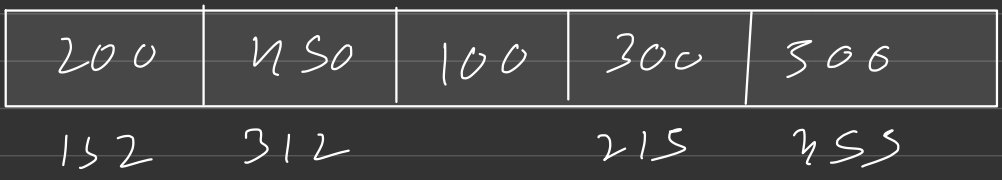
	BT	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2



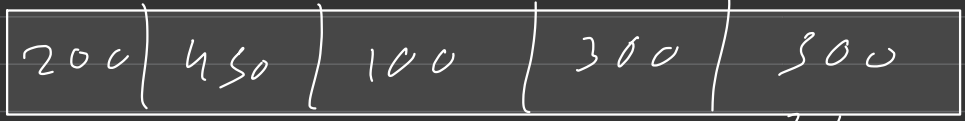







200
450
100
300
500
132
312
215
455


200
450
100
300
500
132
312
215
455


200
450
100
300
500
215
132
312

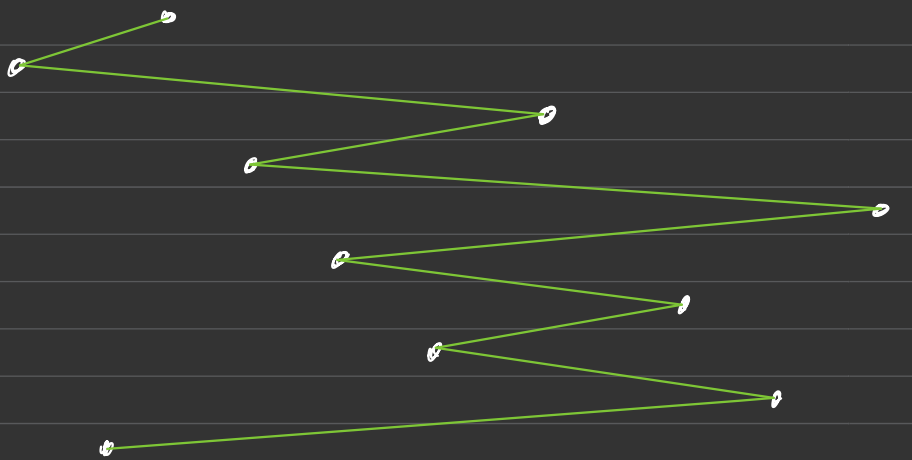
Process	Allocation	Maximum	Needed
P_1	1 0 2 2	3 2 5 2	2 2 3 0
P_2	0 2 1 2	3 4 1 2	3 2 0 0
P_3	2 4 5 0	2 7 7 3	0 3 2 3
P_4	3 0 0 0	5 5 0 7	2 5 0 7
P_5	4 2 1 3	6 2 1 4	2 0 0 1

P_5 P_2 P_3 P_4 P_1

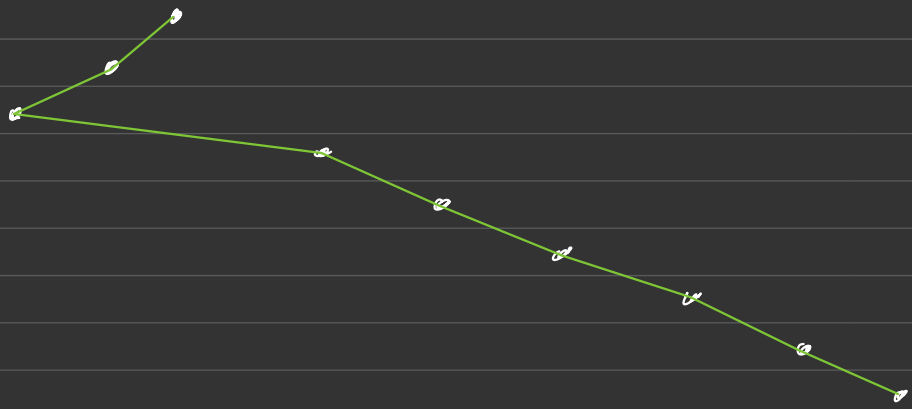
3 0 0 1
6 2 1 4
---
9 2 1 5
3 4 1 2
---
12 6 2 7
2 7 7 3
---
14 13 9 10
5 5 0 7
---
14 18 9 17

FCFS

86 130 143 913 948 1022 1470 1509 1750 1774



86 130 143 913 948 1022 1470 1509 1750 1774



7 0 1 2 0 3 0

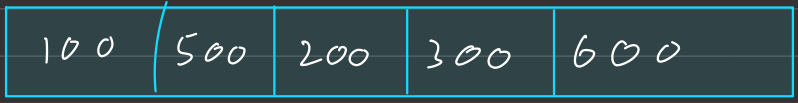
7 7 7 2 2
0 0 0 0
1 1 3

4 2 3 0 3 2 1

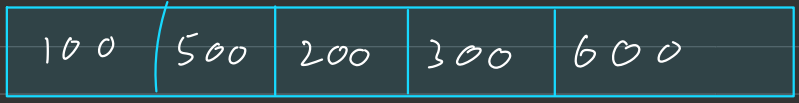
4 4 4 0 1
0 0 3 3 3
3 2 2 2 2

2 0 1 7 0 1

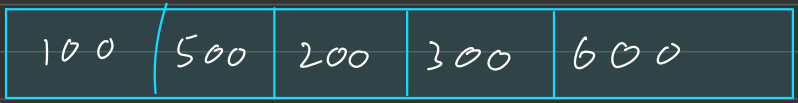
1 1
6 0
2 7



212 112 417
426



417 112 212 426



417 112 212

FIFO

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

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

0	1
6	0
2	1
3	3

缺页次数 3 0 1

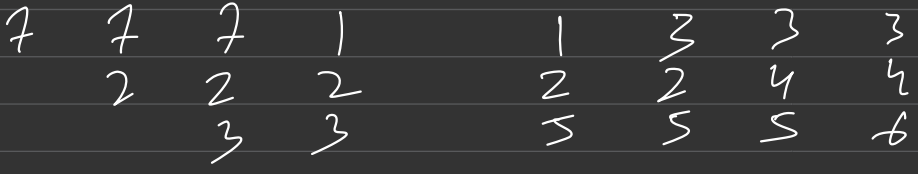
Optimal

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

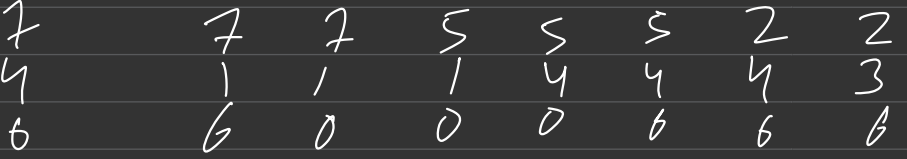
7	7	1	0	5	4	6	2	3
1			1		1	1	1	1
5			5		4	6	2	3
7			0		0	0	0	0

0	1

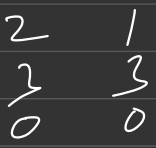
7 2 3 1 2 5 3 4 6



7 7 1 0 5 4 6 2 3



0 1



Process	Allocation	Maximum	Needed
P_0	3 0 1 4	5 1 1 7	2 1 0 3
P_1	2 2 1 0	3 2 1 1	1 0 0 1
P_2	3 1 2 1	3 3 2 1	0 2 0 0
P_3	0 5 1 0	4 6 1 2	4 1 0 2
P_4	4 2 1 2	6 3 2 5	2 1 1 3

P_2 P_1 P_3 P_4 P_0

P_1 P_2 P_3 P_4 P_0

0 3 0 1
3 3 2 1
---
3 6 2 2
3 2 1 1
---
6 8 3 3
4 6 1 2
---
10 14 4 5
6 3 2 5
---
16 17 6 10

1 0 0 2
3 2 1 1
---
4 2 1 3
3 3 2 1
---
7 5 3 4
4 6 1 2
---
11 11 4 6