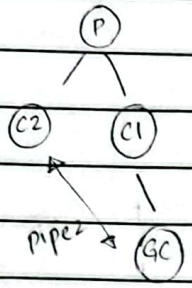
Dated:

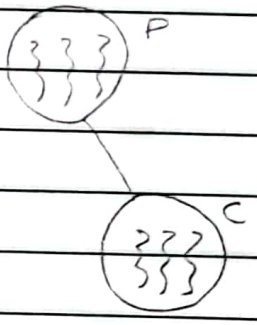


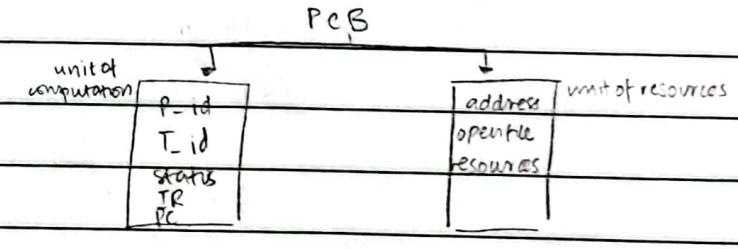
- two fork. first pipe then second pipe
- create pipe before fork.
- parent is the common.

- there cannot be multiple readers because it is a buffered area so data is lost for second reader.
- when buffer is full you'll be block on writes end.

Threads

- problem 1 : if at a time, we have to make multiple processes work it can be done through child process. additional cost is to be paid.
- problem 2. CPU time is wasted because of context switching.
- problem 3 : data sharing. multiple processes and we need to utilize each process for work IPC increases.
- thread is a lightweight process.
- threads exist inside the processes and address space.
- a process can create multiple threads



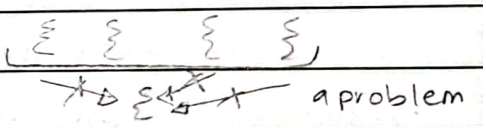


- in multiple core, each core has a thread to be executed, means a process can execute multiple threads concurrently.
- different processes can not execute threads.
- context switching between threads of the same process is faster. PCB is not loaded as they're shared. but for different processes it takes the same time.
- IPC exists by default. make global variables. as one data is shared with others in PCB.
- if one file closes, it closes for other

Updated:

- multicore is hardware component.
- multithread is software component
- thread is always assigned a function.
- pthread_creat → this function is used to creat
- container → thread id and return.
- attribut → to make joinable or not.
- function ptr → generic, any type of function
- argument to function ptr → arguments in function and then typecast.
- threads which are intrested in synchronization are called joinable threads. to send and receive signals.
- those which cannot send or receive signals is called detach.
- joinable threads can be converted into detach. but not vice versa.
- by default, attribut is joinable.
- void * is a data type in which any type of data type can be stored.
- directly void* data cannot be used and should be typecasted at first. *go back to original typecast.
- if a thread has to wait for another thread, then join is called and join tells which thread it is waiting for.
- same process so thread id will be told by user or the process - data sharing
- multiple threads cannot call join for one thread.
- if a thread is terminated then signals won't reach to every thread.

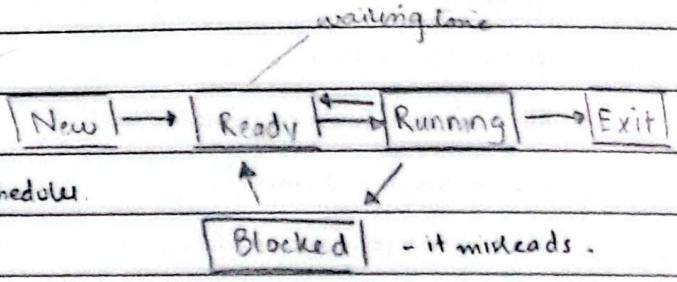
OPEN MP



- 5 state process model
- scheduling rule with que
- on which basis can
- scheduling objective.

Dated:

- ready state means other than CPU, it has everything
- running state to blocked or exit.
- blocked state to ready.
- running to ready.
- degree of multiprogramming is controlled by the job scheduler. long term
- store blocked in hard disk, access will be easier. time taken will be less
- short term scheduler | CPU scheduler
- preemptive scheduling, non pre-emptive



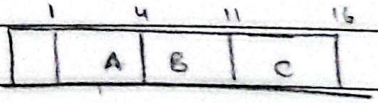
- it misleads.

to cannot come to ready to running

- non-preemptive -
- context switching takes place in both preemptive and non-preemptive.
- more in preemptive because of time sharing.
- whichever uses CPU is useful, not keeping it idle.
- number of processes completed in a unit amount of time. is throughput
- turnaroundtime: adding up all the time from new to exit
- waiting time, wait for CPU so ready queue takes time.
- response time: the first time we tell to start work, user engagement

- increased
- increased.
- decreased.

SCHEDULING



A = 0 B = 3 C = 9 : 0 + 3 + 9 = 4

1 - 1 4 - 1 11 - 2 3

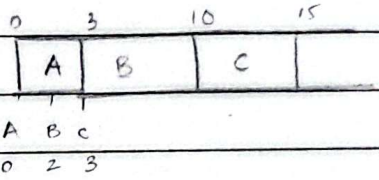
A 2

B

	PID	Arrival	CPU
A	3	1	3
B	1	1	7
C	2	2	5

Dated:

Name	PID	Arrival Time	CPU Burst
A	3	0	3
B	1	2	7
C	2	3	5



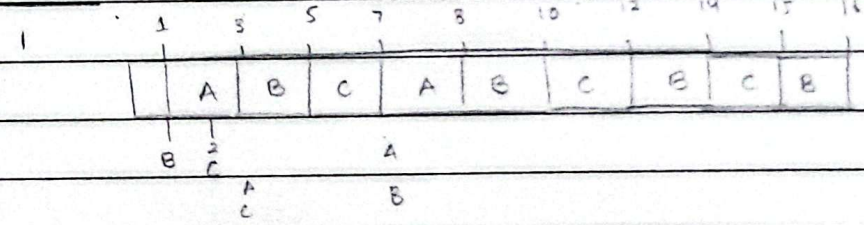
A B C

0-0=0 3-2=1 10-3=7

$= \frac{0+1+7}{3} = 2.67$ Turnaround Time

- FCFS is non-preemptive in nature
- cannot go from ready to running
- FCFS has an issue known as "convoy effect"

Round Robin



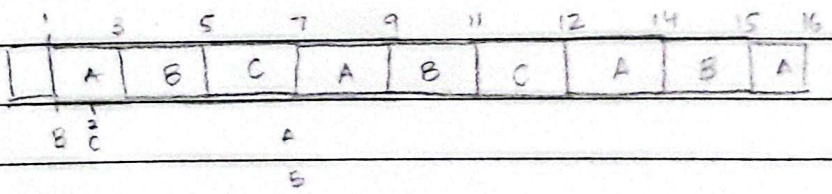
A = 0 + 4 = 4

B = 2 + 3 + 2 + 1 = 8

C = 3 + 3 + 2 = 8

CPU Burst	Name	Arrival Time
7	A	1
5	B	1
3	C	2

quantum 2



A = 0 + 4 + 3 + 1 = 8

B = 2 + 4 + 1 + 2 = 9

C = 3 + 4 = 7

Dated:

- Race condition where the code has no issue when 2 queue are used and inserted
- a code which is accessed by shared memory is called critical section.
- in critical section, solution designed has 4 requirements.
  1. mutual exclusion : when 1 process enters CS, a second process cannot enter it's shared memory.
  2. no assumptions.
  3. bounded wait : wait has upper bound., no process should be left undone.
     whichever process enters CS, should'n't be taken out.
  4. progress : it shouldn't stop

: solution : strict alternation
to decide turns between processes.

2nd solution : turns and interest both are seen. - in between A can show interest

: semaphores can never be a negative number.
to the operations are atomic - check and update, OS doesn't disturb

- counting and binary
  to min = 0, max = 1
  to they are mutex ; lock and unlock. semaphore / mutex

- deadlock : a state where 2 or more than 2 processes enter into an indefinite ~~wait~~ state because of shared resources.
  → if either one gives off resources, then every

- 4 thing to create deadlock
  → mutual exclusion :
  → non-preemptive in nature (some resources) : printer
  → hold and wait : those allocated then it is holding and not received is waiting
  → circular wait can be created by above 3 causes.
  to between 2 or more processes, waiting for itself and others to create a circle.

ted:

Deadlock Prevention. → deadlock nahi hota.

→ remove any one of the 4 causes.

① ✗ cannot remove because of race conditions.

② ✗

③ give all resources or do nothing. → throughput

④ ✓ direct alternation,

Avoiding Deadlocks. → asleta hai, mgr rokein kese?

→ Bankers Algorithm.

↳ available resource, maximum resource for each process, all resources

↳ check whether the system goes into unsafe state

↳ extra calculations so throughput is destroyed

Deadlock Detection

- processes are divided into partitions
- in PM and VM, partition size are same
- virtual memory is greater than PM
- pages and frame calculated, divided by frame/page
- when a frame is not in PM, get it from the VM because otherwise an operation will not be performed.

page = frame

PM ≠ VM

```
int z = x + y
        ↑   ↑   ↑
       P7  P5  P10
       F?  F?  F?
```

* if one frame doesn't exist, it first needs to uploade page in frame in PM.

* after updating z, reflect change in VM.

- Page Faults →
- Demand Paging → through page fault request is seen.
- Page Table → organized way to search a specific page, and then find in PM.
- page table in physical memory and in kernels address space, is protected.
- it is required before every operation.
- make convenience or cost, using present/absent bit

use as flag → 0 means no need to see page table, first load

1 means that in physical memory

IMPORTANT

inverted page table and page table advantages and disadvantages?

Dated:

1. Page size and frame size to be identified
2. determine size of PM and VM
3. calculate total number of pages in virtual memory and total frames in physical memory
4. convert these number / determine the number of bits required to ~~calculate~~ store these number
5. calculate offset bits.
6. ~~determine frame number~~ identify page number.
7. check page table for frame number
8. go to offset in frame.

offset: ik page k andar kitni bits save ki jaasakti hain.

page size = offset

* offset phle alag hongi from LSB
* offset ki bagi bits, padding krke zero krdein (page number)
* page ki offset bits aur frame ki same hongi.

Dated:

fragmentation: wastage of memory.

m = 16, n = 12

page number = 16 - 12 = 4 = m - n. (leftmost)

page offset = n = 12 (rightmost)

Logical address 8196 = 0010000000000100

page number is 2

page offset is 4

- check in page table index 2 and then look into frames.
- append the offset in physical address.

address = 0100 0000 1100 0000

m = 16, n = 12

page number = 4

page offset = 12

answer → 1000000011000000 offset is attached as it is.

Translation look aside buffer → special fast lookup hardware cache.

hit ratio

virtual memory: when data exceeds RAM limit

$2^n = 4$

$n = \log_2(4) = 2$

m = 12

n = 4

m - n = 12 - 4 = 8

00000 — 11001

$2^m = 6$

2^n

$2^m = 6 \times 4$

$m = 24$

$m - n = 24 - 4 = 20$

BANKER'S ALGORITHM

Instances → A=10, B=5, C=7

PROCESS	ALLOCATION			MAX			AVAILABLE			NEED		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	4	2	2				2	1	1
P5	0	0	2	5	3	3				5	3	1

1. Available : Initial Existing Instance - Allocation

A = 0+2+3+2 = 7 → 10-7 = 3

B = 1+0+0+1+0 = 2 → 5-2 = 3

C = 0+0+2+1+2 = 5 → 7-5 = 2

2. Need : Max - Allocation

P1 → 7-0, 5-1, 3-0 → 7, 4, 3

P2 → 3-2, 2-0, 2-0 → 1, 2, 2

P3 → 9-3, 0-0, 2-2 → 6, 0, 0

P4 → 4-2, 2-1, 2-1 → 2, 1, 1

P5 → 5-0, 3-0, 3-2 → 5, 3, 1

3. Safe Sequence.

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

Check if Need ≤ Available.

P1 → [7,4,3] ≤ [3,3,2]

P2 → [1,2,2] ≤ [3,3,2] → P2 will now release allocated resources. [3,3,2]+[2,0,0]

P3 → [5,3,2] → [6,0,0] ≤ [5,3,2]

P4 → [2,1,1] ≤ [5,3,2] → P4 will now release resources. [5,3,2]+[2,1,1]

P5 → [5,3,1] ≤ [7,4,3] → P5 will now release resource [7,4,3]+[0,0,2]

P1 → [7,4,3] ≤ [7,4,5] → P1 will now release resource [7,4,5]+[0,1,0]

P3 → [6,0,0] ≤ [7,5,5] → P3 will now release resource [7,5,5]+[3,0,2]

= [10,5,7]

Example: P2 requests (1,0,2)

Available [3,3,2] → [3,3,2] - [1,0,2] = [2,3,0]

P2 - Need [1,2,2] → [1,2,2] - [1,0,2] = [0,2,0]

P2 - Allocation [2,0,0] → [2,0,0] + [1,0,2] = [3,0,2]

new state is safe.

Example: P4 requests (3,3,0)

since P1 had requested (1,0,2), we now have available resources (2,3,0) which is not sufficient for P4, hence P4 has to wait.

Example: P1 requests (0,2,0)

Available [2,3,0] → [2,3,0] - [0,2,0] = [2,1,0]

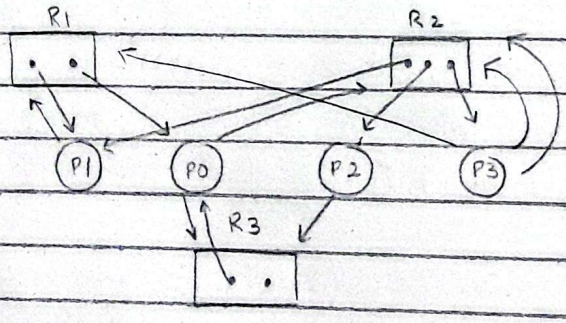
P1 - Need [7,4,3] → [7,4,3] - [0,2,0] = [7,2,3]

P1 - Allocation [0,1,0] → [0,1,0] + [0,2,0] = [0,3,0]

no row in the table has need ≤ available, hence unsafe state.

P0 should wait.

MULTIPLE INSTANCE RESOURCE ALLOCATION GRAPH



PROCESS	ALLOCATE			REQUEST		
	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

Current Availability →

R1	R2	R3
0	0	1

check resources

1. safe sequence

P0 [0,1,1] ≤ [0,0,1] ×

P1 [1,0,0] ≤ [0,0,1] ×

P2 [0,0,1] ≤ [0,0,1] ✓ [0,0,1] + [0,1,0] = [0,1,1] → P2

~~P3 [1,2,0] ≤ [0,1,1] ×~~ ~~[0,1,1] + [0,1,0] = [0,2,1] → P2 P3~~

P0 [0,1,1] ≤ [0,1,1] ✓ [0,1,1] + [1,0,1] = [1,1,2] → P2 . P0

P1 [1,0,0] ≤ [1,1,2] ✓ [1,1,2] + [1,1,0] = [2,2,2] → P2 . P0 P1

P3 [1,2,0] ≤ [2,2,2] ✓ [2,2,2] + [0,1,0] = [2,3,2] → P2 P0 P1 P3

LEAST RECENTLY USED

Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Frames = 4

F4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1
F2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	7	7	7
	PF	PF	PF	PF	H	PF	H	PF	H	H	H	H	H	PF	H	H	H	PF	H	H

- least recently used page in past ←

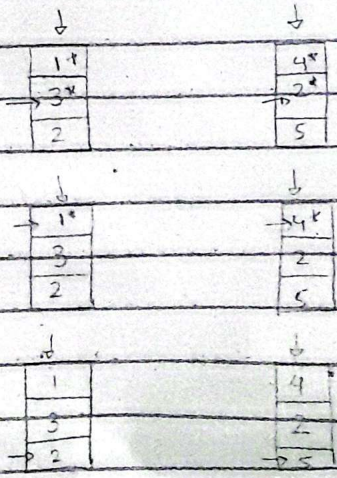
- page fault : 8
- page hit : 12

CLOCK POLICY

Reference string : 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

Frames = 3

F3	→	→	1*	1	→ 1	4*	4*	4	4	5*	5*	
F2 →	3*	3*	3*	→ 3	2*	2*	2*	→ 2	→ 2*	2*	2*	
F1	2*	2*	2*	→ 2*	5*	5*	→ 5*	→ 5*	3*	3*	→ 3*	→ 3*
	PF	PF	PF		PF	PF	PF	PF	PF		PF	



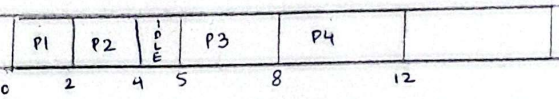
- if all bits are set to 1, then change it to 0.

SCHEDULING POLICIES

1. FIRST COME FIRST SERVED - NON PREMPTIVE

PROCESS	ARRIVAL	BURST	COMPLETION	TURN AROUND	WAITING TIME	RESPONSE TIME
P1	0	2	2	2	0	0
P2	1	2	4	3	1	1
P3	5	3	8	3	0	0
P4	6	4	12	6	2	2

gantt chart



turnaround time = completion - arrival

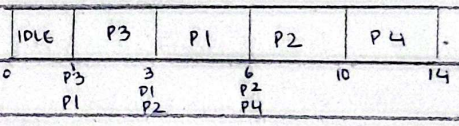
waiting time = turnaround - burst

response time = 1st CPU alloted - arrival time

2. SHORTEST JOB FIRST - NON PREMPTIVE

PROCESS	ARRIVAL	BURST	COMPLETION	TURN AROUND	WAITING TIME
P1	1	3	6	5	2
P2	2	4	10	8	4
P3	1	2	3	2	0
P4	4	4	14	10	6

gantt chart

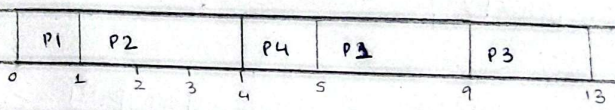


- criteria is burst time
- check for arrival time = 0, if not keep it idle
- shortest burst time

3. SHORTEST REMAINING TIME FIRST - PREMPTIVE

PROCESS	ARRIVAL	BURST	COMPLETION	TURN AROUND	WAITING TIME	RESPONSE TIME
P1	0	5	9	9	4	0
P2	1	3	4	3	0	0
P3	2	4	13	11	7	7
P4	4	1	5	1	0	0

gantt chart



P1 → 5-1 = 4-4 = 0

P2 → 3-1 = 2-1 = 1-1 = 0 DONE

P3 → 4-4 = 0

P4 → 1-1 = 0

• criteria is burst time.

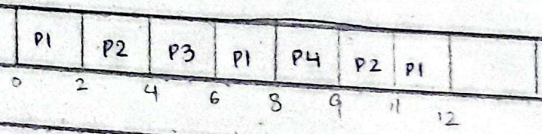
4. ROUND ROBIN

PROCESS	ARRIVAL	BURST	COMPLETION	TURN AROUND	WAITING TIME	RESPONSE TIME
P1	0	5	12	12	7	0
P2	1	4	11	10	6	1
P3	2	2	6	4	2	2
P4	4	1	9	5	4	4

TQ = 2

• criteria is time quantum ready queue P1 P2 P3 P1 P4 P2 P1

gantt chart



P1 → 5-2 = 3-2 = 1-1 = 0

P2 → 4-2 = 2-2 = 0

P3 → 2-2 = 0

P4 → 1-1 = 0

PAGING

logical address space of 256 pages with a 4KB page size

physical memory of 64 frames.

page size = $4 \times 1024 = 4096$ bytes.

frame size = page size.

a) bits for logical address

size of logical address space = 2^m

# of pages $(2^{m-n}) \times$ page size (2^n)

$2^n = 4096 \rightarrow \log_2(4096) = n \rightarrow 12 = n$ (offset)[bits]

$2^m / 2^n = 256$

$2^m = \frac{256 \times 4096}{4096}$

$m = \log_2\left(\frac{256}{4096}\right)$

$m = 20$

$m - n = 20 - 12 = 8$ (page number)[bits]

b) bits for physical address

offset = $n = 12$ [bits]

$\frac{2^m}{2^n} = 64$

$2^m = 64 \times 4096$

$m = 18$

$m - n = 18 - 12 = 6$ (frames)[bits]