LEC-9 | Intro to Process.

* what is program? compiled code, that is ready to execute.
* what is process? program under execution.

.cpp → compile → compile → executable → program

program e.g TikTok → OS → program
↓
Process

why process? — use — work → way

program → process.

* **How OS creates a process?** Converting program into a process.

① Load the program & static data into memory.
↓
used for initialization.

② Allocate runtime stack.
→ part of memory used for local variable, i/p argument & return value.

③ Heap memory allocation.
— part of memory used for dynamic allocation.

④ I/O tasks

- i/p — handle
- o/p — handle
- errors — handle

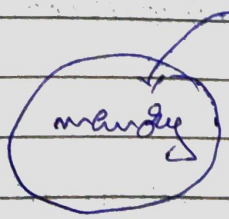
⑤ OS hands off control to main()

```
main()
{
   ——
   ——
   return 0;
}
```

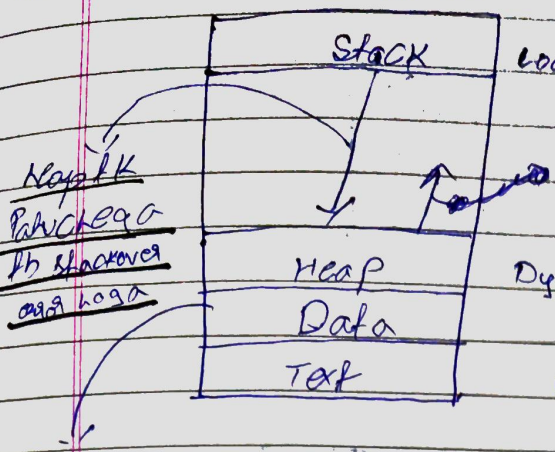
return 0; → successful execution hogya to check.

```
sum()
{
```

exit(-1) → ab return hoga th OS ko bolega ki kuch th gadbad h



* architecture of process:



- ⊖ stack overflow - Base case set
- ⊖ out of memory - deallocate unnecessary obj.

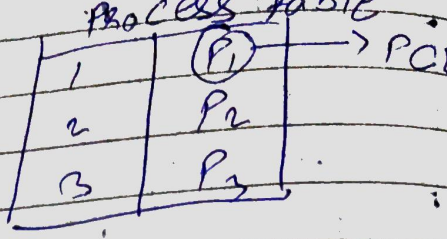
* Attributes of process.

- Feature that allows identifying a process uniquely.

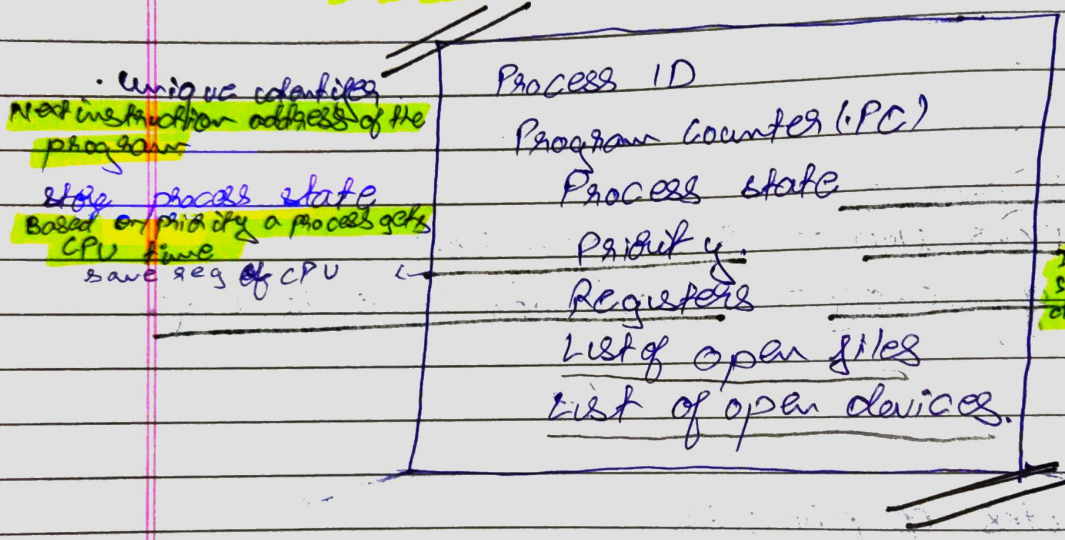
- Process table.
  - All processes are being tracked by OS using a table like data structure.
  - Each entry in that table is process control block (PCB).

- PCB: storing of attributes of a process.
  - Data structure used for each process, that stores info of a process such as process id,

Program counter, process state, priority etc.



PCB structure:



Program counter (PC)

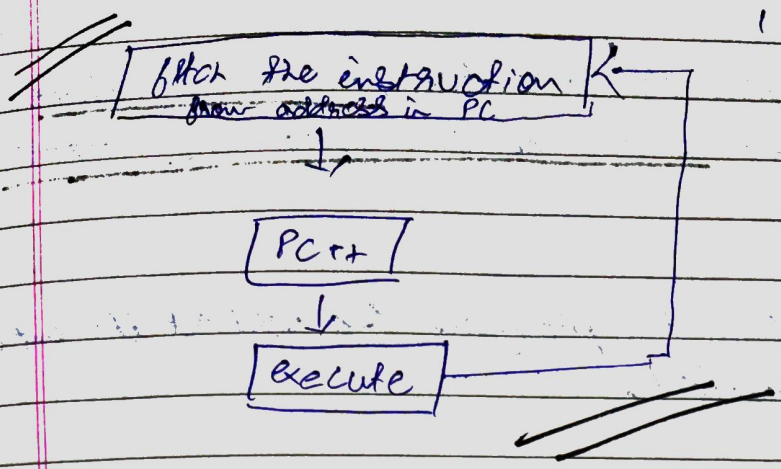
Compile → converted into instructions

1.2 ADD

program → collection of instructions

CPU want only instruction & it will execute it

In OS kata kar process se related ek program counter associate rahta h



1 2 Add &

Registers in the PCB, it is a data structure. When a process is running & it's time slice expires, the current value of process specific registers would be stored in the PCB & process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB & written to CPU registers. This is the main purpose of registers in PCBs.

LEC - 10 | Diff process states in OS.

* Process states: As process executes, it changes state. Each process may be in one of the following state.

(a) **New**
OS is about to pick the program & converted it into process is being created.

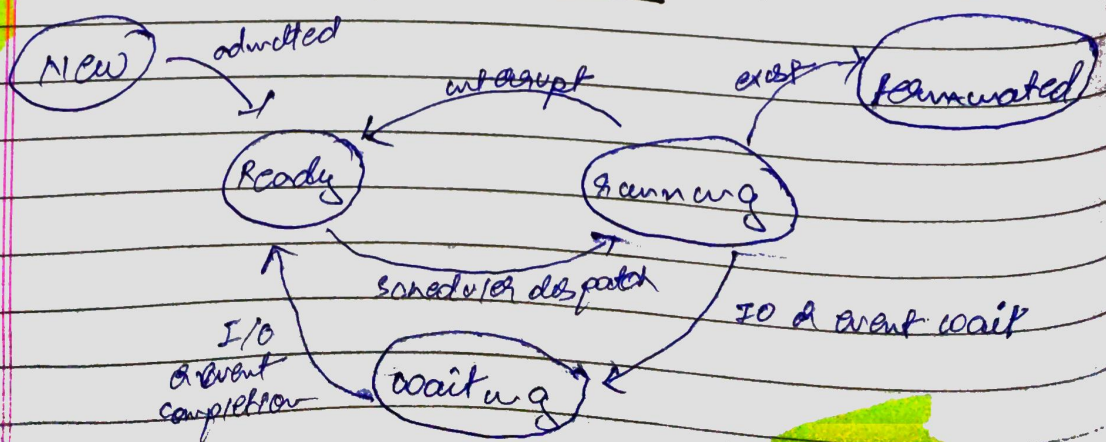
(b) **Run**
~~Instructions are being executed; CPU is allocated~~

(b) **Ready:**
Process is waiting for memory, wait
- is in ready queue.

(c) **Run:**
Instructions are being executed; CPU is allocated.

(d) **Waiting**
waiting for I/O

(e) **Terminated:** The process has finished execution. PCB entry removed from process table.



Page No.
Date:

* Process Queues

(a) Job Queue

I. Process in new state
II. present in Secondary memory → low freq, ideal time is high
III. Job schedular (Long term schedular (LTS)) picks process from the pool and loads them onto memory for execution. → i.e from new state → i.e. ye check karta h koi job ready queue mae dalni h.

| P1 | P2 | P3 |

(b) Ready Queue

I. processes in Ready state.
II. present in main memory. (STS)
III. CPU schedular (short term schedular) picks process from ready queue & dispatch it to CPU. → work on high freq → P3 → I/O h jaldi se P1 & P2 ko running state mae daal dega
i.e ideal time is low

(c) waiting queue

I. process in wait state.

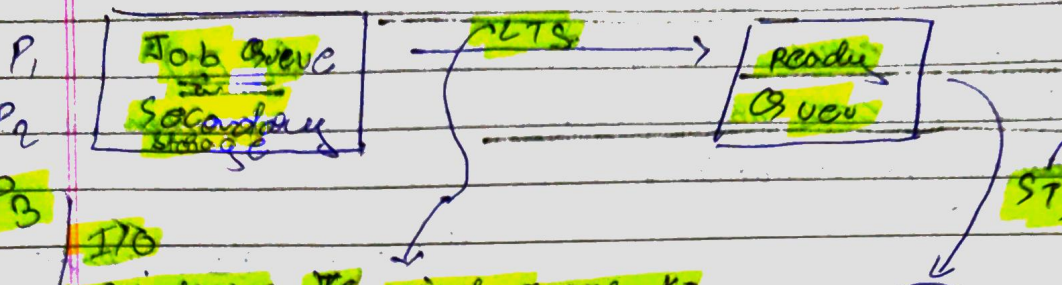
* Degree of multi programming: The no. of process in the memory.
  - (LTS) control degree of multi programming. → as yahi memory se utha ke new state mae daal rha h

* Dispatcher: The module of OS that gives control of CPU to a process selected by STS.

LEC 11 | Context switching | Medium Term sched. & phantom/zombie process

STS

LTS

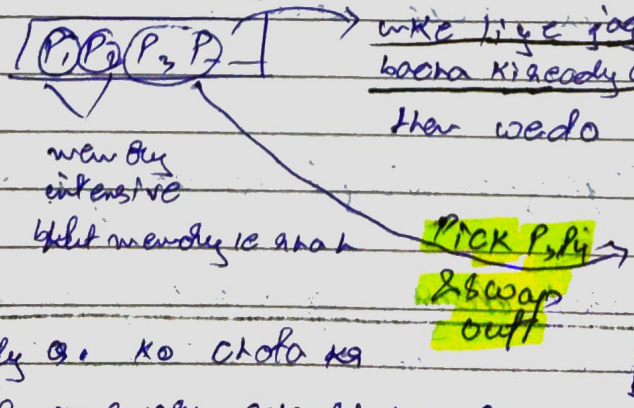


I/O CPU intensive ka mix of process ko choose karna taki ready queue mix of process ho taki Starvation na ho agar kewal IPU intensive ho to bhi starvation ka chance bn skta h

* MTS — Medium term scheduler

- degree of multi prog. ↑ by (LTS)

Ready Que.

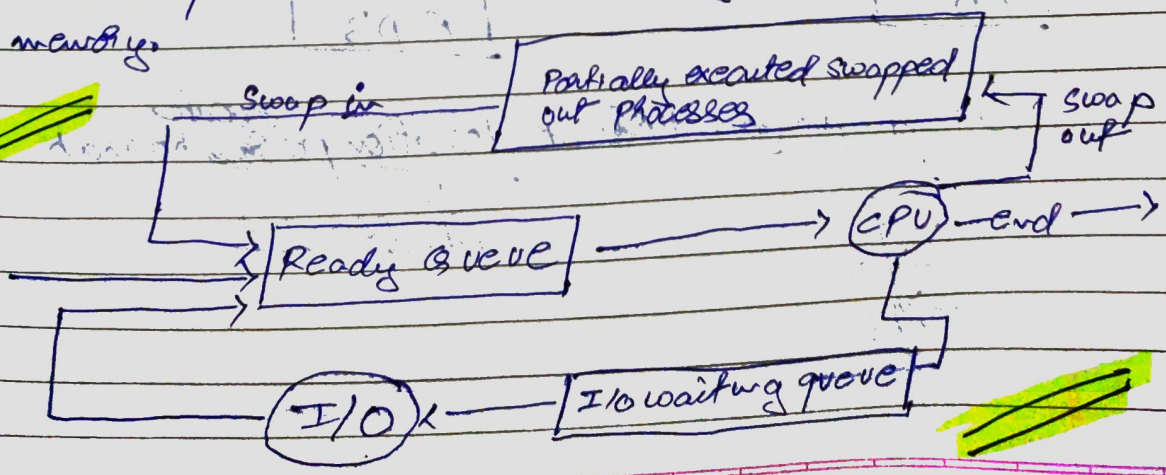


=> Ready Q. ko chota kr diya or memory exhaust hone se bacha liya

Also, let Ki P_i → term wale ho gaye
P_i jo Ki memory intensive thi th ab
P_3 or P_4 jo swapped out hogyi thi unko uthake
wapas ready q. mai le aao. i.e swapped in.

Swapping

- Time sharing system may have medium term scheduler (MTS)
- Remove processes from memory to reduce degree of multiprogramming.
- These removed processes can be reintroduced into memory & its execution can be continued where it left off.
  This is called swapping
- Swap out & swap in done by MTS.
- Swapping is necessary to improve process mix b/c a change in memory requirements has overcommitted available memory, requiring memory to be freed up.
- Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) & make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.



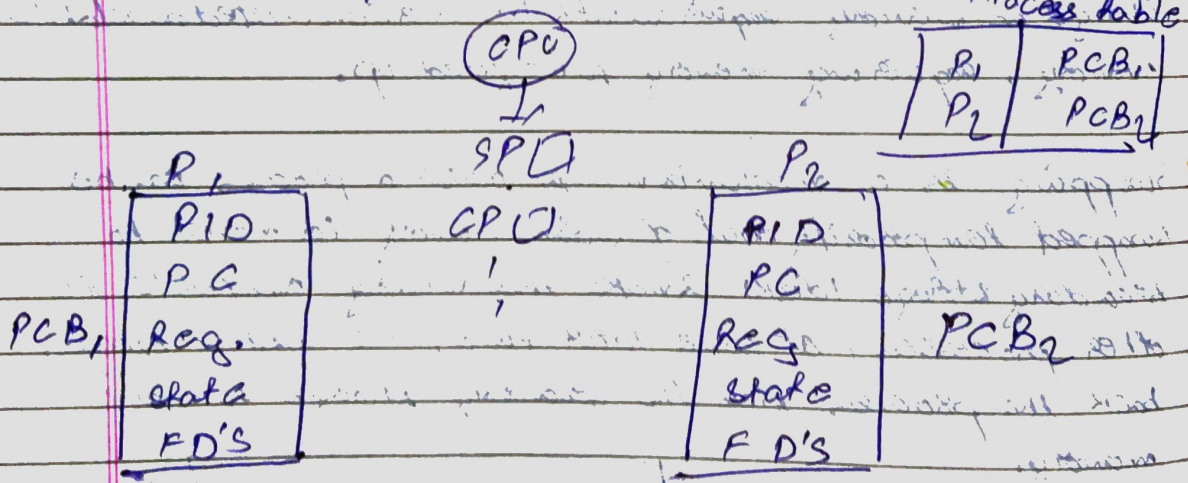
* Context Switching

- Switching the CPU to another process req. preserving a state of one of the current process & a state restore of a diff process.

- When this occurs the kernel saves the context of the old process in its PCB & loads the saved context of the new process scheduled to run.

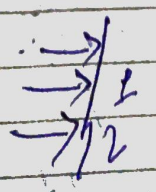
- It is pure overhead, b/c the system does no useful work while switching.

- Speed varies from machine to machine, depending on the memory speed, the no. of registers that must be copied etc.

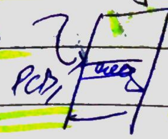


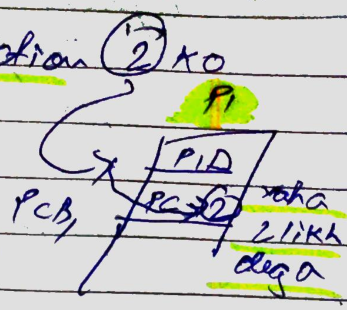
abhi ek program read ek location per execute horaha

ie address I



ab CPU kahega abhi ka context save karlo
save karne ke liye wo next instruction (2) ko
or jitne bhi CPU ke reg honge
utha ke jo abhi current value wo
use reg. wale entry mae save kar dega



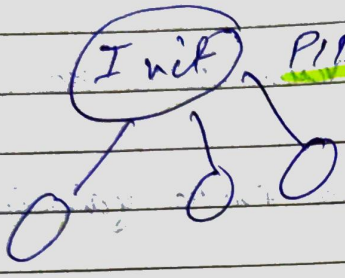


Ab usko P_2 schedule karni thi
P_2 bhi kabhi save hui hogi
th P_2 ke corresponding reg. ki value utha ke CPU ke reg. mae copy kar dega
or Reg program counter ko btaega ki usko yaha se execute karna h.
or Ab P_2 ke context ko restore kar liya P_1 ke context ko save kar liya.

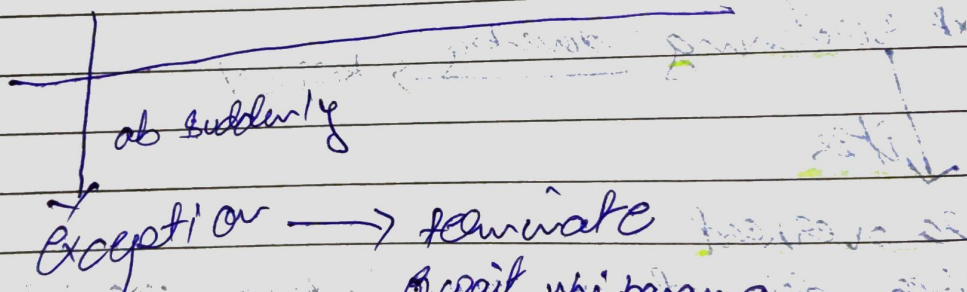
Context switching (done by → kernel)
↓ its
Pura overhead
i.e user defined processes in ready, no one is working, only context switching is happening

* Orphan process

- The process whose parent process has been terminated and it is still running
- Orphan processes are adopted by init process
- Init is the first process of OS.



(P1) running —fork()→ (P2) child running



ab (P2) ka kya hoga?

due to OS (P2) ka parent Init ho jaega.

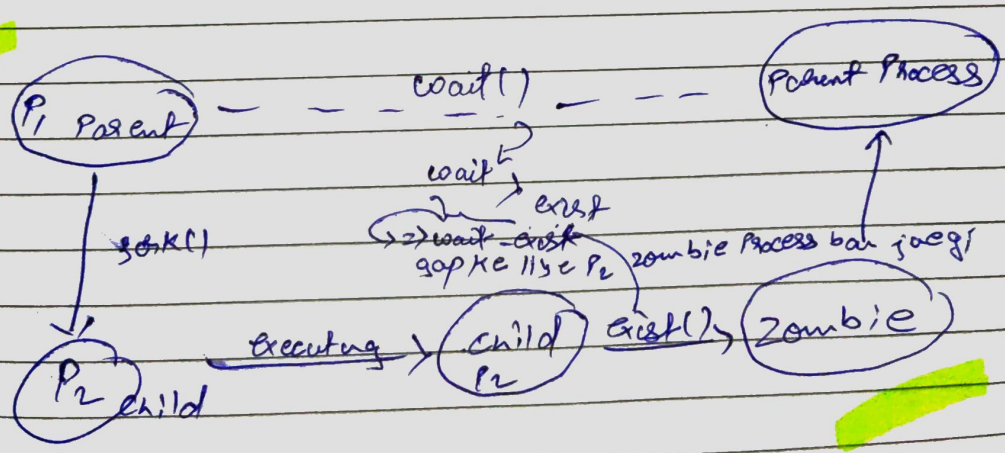
Zombie process / Defunct process

- A zombie process is a process whose execution is completed but it still has an entry in the process table.

- Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from process table. This is known as reaping the zombie process.

- It is b/c parent process may call wait() on child process for a longer time duration & child process got terminated much earlier.

- As entry in the process table can only be removed, after the parent process reads the exit status of child process. Hence, the child process remains a zombie till it is removed from process table



Y wait call nhi kiya parent ne
Process table exaust ho jaegi Zombie Process ki Pid ab
koi or process use nhi
kr skta

LE12| Process scheduling.

* **Process scheduling**
  - Basics of Multiprogramming OS.
  - By switching the CPU among processes the OS can make the computer more productive.
  - Many processes are kept in memory at a time. When a process must wait or time quantum expires, the OS takes the CPU away from that process & gives the CPU to another process & this pattern continues.

* **CPU Scheduler**
  - whenever the CPU become ideal, OS must select one process from the ready queue to be executed.
    Done by STS.

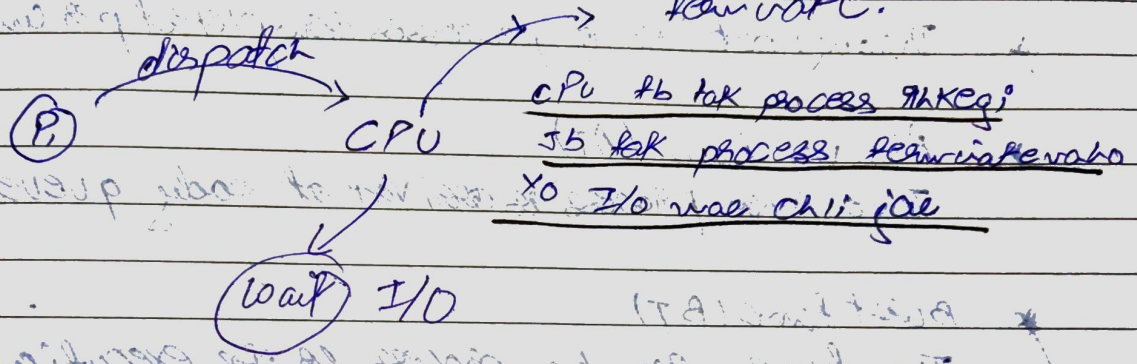
* ~~DO~~

* Non-preemptive Scheduling

- once CPU has been allocated to a process keeps the CPU until it releases CPU either by terminating or by switching to wait state.

- Starvation as a process with long burst time may starve less burst time process.

- Low CPU utilization.



* Preemptive scheduling

- CPU is taken away from a process after time quantum expires along with terminating & switching to wait state.

- Less starvation & high CPU utilization.

jb time quantum expire hota ho tb bhi CPU chod deta h

* Goals of CPU scheduling
- Maximum CPU utilization
- Minimum turnaround time (TAT)
- Min wait time
- Min response time
- Max throughput of system.

* Imp. Terms

* Throughput: No. of processes completed per unit time.

* Arrival time (AT)
Time when process is arrived at ready queue.

* Burst time (BT)
The time req. by process for its execution.

* Turnaround time (TAT)
Time taken from first time process enters ready state till it terminates. (CT - AT)

* Wait time (WT):
Time process spends waiting for CPU.
WT = TAT - BT

* Response Time: Time duration b/w process getting into ready que. & process getting CPU for first time

* Completion time (CT)
Time taken till process gets terminated.

* FCFS (First come first serve)

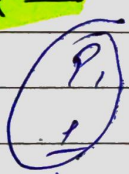
(a) Whichever process comes first in the ready que. will be given CPU first.

(b) In this, if one process has longer BT, It will have major effect on avg. WT of diff processes Called convoy effect

(c) convoy effect is a situation where many processes who need to use a resource for a short time, are blocked by one process holding that resource for a long time.

— This cause poor resource management

ex — Non Preempt



P2 P3
2 3

→ CPU

dispatcher

→ selected first

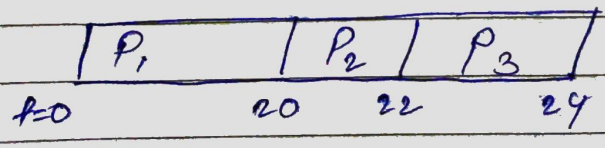
CT-AT

TAT - BT

Pro	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	2	22	21	19
3	2	2	24	22	20

39/3 → (13) → avg WT

Gantt Chart

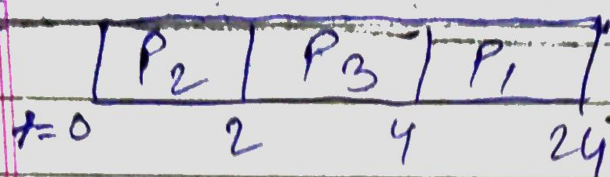


CT - AT → TAT - BT

Now.

Pro	AT	BT	CT	TAT	WT
P_2	0	2	2	2	0
P_3	1	2	4	3	1
P_1	2	20	24	22	2

avg (1) unit



i.e ek heavy process phle aagati h th baki process ko bhut wait karni parti h.

LEC-13 | CPU scheduling | SJF | Priority | RR.

* Shortest job first

(a) Non-preemptive.

- Process with least BT will be dispatched to CPU first.
- Must do estimation for BT for each process in ready que. beforehand, correct estimation of BT is an impossible task (ideally).

- Run lowest time process for all time then, choose job having lowest BT at that instance.

- This will suffer convoy effect as if the very first process which came is Ready state is having a large BT.
- Process starvation might happen
- criteria for SJF algos, AT + BT.

(BT → usually or estimation by OS)

EX-

Proc	AT	BT	CT	TAT	WT
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	26	24	15
P_4	3	5	17	14	9

avg → 7.75.

P_1	P_2	P_4	P_3

t=0, 8, 12, 17, 26

At 8: Ready que → [P_2 P_3 P_4] lab ↓ among them P_2 has lowest BT

At 12: [P_3 P_4] ↳ check less BT process i.e P_4

(b) SJF (Preemptive) (AT+BT) & Preemption

- Less starvation, No convoy effect

- Gives avg WT less for a given set of processes as scheduling short job before a long one decreases the WT of short job more than it increases the WT of the long process.

Ex:-

Process	AT	BT	CT	TAT	WT
P_1	0	~~8~~ 7	17	17	9
P_2	1	~~4~~	5	4	0
P_3	2	~~9~~	26	24	15
P_4	3	~~5~~	10	7	2

avg → 6.5

$P_{1BT} < P_{3BT}$

| P_1 | P_2 | P_4 | P_1 | P_3 |

t=0, 1, 5, 10, 17, 26

→ Now check process of less BT i.e. P_4

P_1 → 1s phir P_2 aajaega
then $P_{2BT} < P_{1BT}$
then P_2 preempt hoga

Now P_{1BT} → ~~8~~ 7

Agar P_2 apna 4s chalta h to sare process ready que aa jaenge but, P_3 & P_4 ka BT P_2 ke BT se kam h to P_2 pura ~~preempt~~ hojaega

Note:
Really impossible hota h kyuki BT pehle kaun pta se hi nhi ho skta.

Priority scheduling

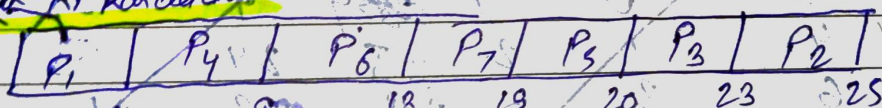
(a) Non preemptive

- priority is assigned to a process when it is created.
- SJF is a special case of general priority scheduling with priority inversely proportional to BT.

Ex

Process	Priority	AT	BT	CT	TAT	WT
P_1	2	0	4	4	4	0
P_2	4	1	2	25	22	22
P_3	6	2	3	23	22	19
P_4	10	3	5	9	6	1
P_5	8	4	1	20	16	15
P_6	12	5	4	13	8	4
P_7	9	6	6	19	13	7
					avg	9.714

No Process with t=0 so the process which arrives first is selected



t=0

at t=4 P1 ready

↓

P1

↓

P_2, P_3, P_4, P_5

↓

Now check high priority process among them. i.e $P_4 \rightarrow 10$

at t=4 P1 ready

↓

at t=9 P1

↓

check high priority process i.e P_6

Now check high priority among rest of process & schedule them.

(b) priority scheduling (Pre emptive)

- current RUN state job will be preempted if next job has higher priority.

- May cause indefinite waiting (starvation) for lower priority jobs. (possibility is they won't get executed ever). (True for both preemptive & non preemptive version)

  - Solution Aging is sol.
  - Gradually increase priority of process that wait so long. Eg increase priority by 1 every 15 minutes.

Ex—

Process	priority	AT	BT	CT
1	2	0	~~4~~ 3	25
2	4	1	~~2~~ 1	22
3	6	2	~~3~~ 2	21
4	10	3	~~5~~ 3	12
5	8	4	1	19
6	12	5	4	9
7	9	6	6	18

avg WT → 11.4

→ highest pri th pura sched. hoga

P1	P2	P3	P4	P6	P4	P7	P5	P3	P2	P1
0–1	1–2	2–3	3–5	5–9	9–12	12–18	18–19	19–21	21–22	22–25

t=0 → pre exh th at t=0 wale process ko exe. karte h next process ke AT tk & then checks its priority.

1 ↓ ready Q ↓ [P1 P2] ↓ check high priority i.e (P2) & P1, BT → ~~4~~ 3

3–5 → 2 sec kyuki vaht high priority process 2 sec baad aarhi h

9 ↓ Now in ready Q → [P1 P2 P3 P4 P5 P7] ↓ high priority among them i.e P4

avg WT non pre for priority < avg WT pre for priority

Convoy effect h → highest pri job aati jaegi or unka BT high jata ja rha h or jo lowest priority job thi woo niche jati ja rhi h. unko mauka nhi mil rha h.

→ Drawbacks for pre & non pre priority Sch.

— Indefinite waiting & Extreme starvation.

↓

Soln

↓

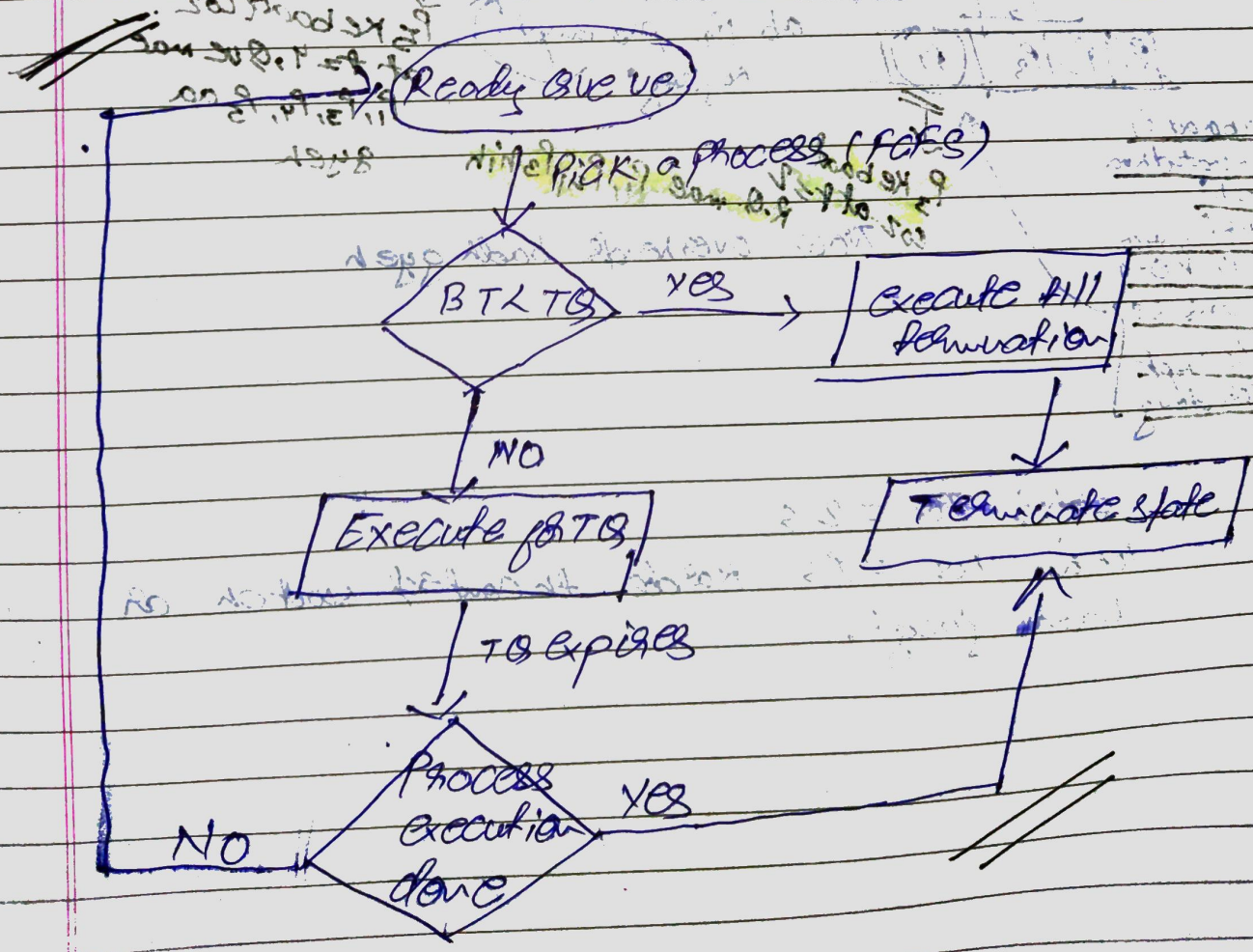
Aging

Gradually ↑ing priority of lowest priority job.

Jab 15 min 15 sec → lowest priority job ka unki priority ko (+1) karta jayega & ek lowest priority job ki unki priority dheere dheere bhad jayegi.

Round Robin Scheduling (RR)

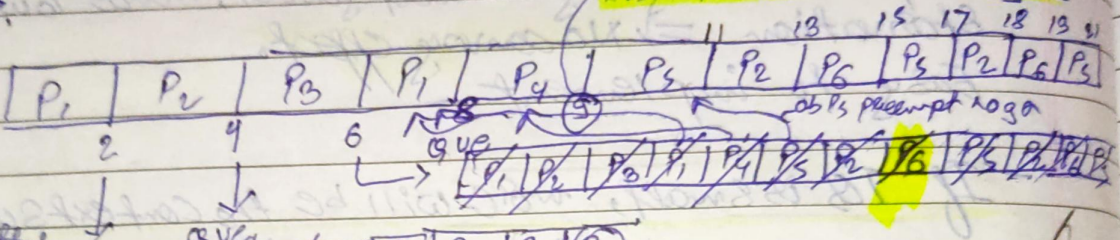
- Most popular.
- Like FCFS but preemptive.
- designed for time sharing systems.
- criteria: AT + time quantum (TQ), doesn't depend on BT.
- No process is going to wait forever, hence low starvation ⇒ No convoy effect.
- Easy to implement
- If TQ is small, more will be the context switch (more overhead).

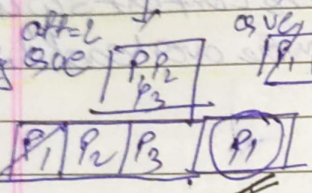


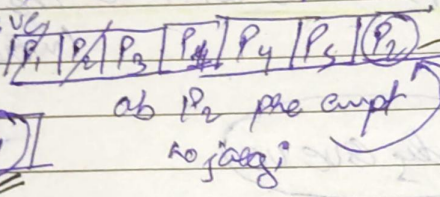
Ex

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

TQ = 2s







P5 ke baad coz at t=4, 3 ye mae P1, P3, P4, P5 aa gaye

P3 ke baad coz at t=7 R.Q. mae P1, P2, P3 with overheads

Now overheads badh gaye

es baad P1 terminated hua
⇒ uski wahi queue ka P1 ko wapas se pre empt kar dena

abhi TQ = 2s
lekin TQ = 1s karde to context switch or badh jaegi.

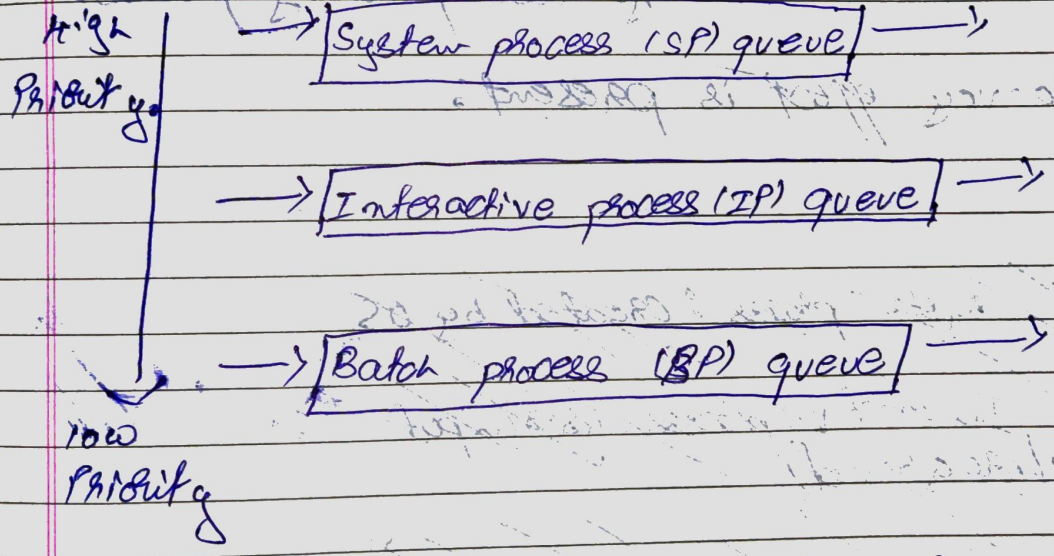
LEC-14 | MLQ | MLFQ.

* Multi level queue scheduling (MLQ).

a. Ready queue is divided into multiple queues depending upon priority.

b. A process is process is permanently assigned to one of the queues (inflexible) based on some property of process, memory, size, process priority & process type.

c. Each queue has its own scheduling algorithm
E.g. - SP → RR, IP → RR & BP → FCFS.



d. System process: Created by OS (Highest priority)

- Interactive process (foreground process): Needs user input (I/O)

Batch process (background process): Runs silently no user input req.

c. Scheduling among diff sub queries is implemented as fixed priority preemptive scheduling E.g. foreground queue has absolute priority over background queue.

f. If an interactive process comes & batch process is currently executing then, batch process will be preempted.

g. problem: only after completion of all the process from the top level ready queue, the further level ready queues will be scheduled. This cause starvation for lower priority process.

h. Convoy effect is present.

=> abhi P2 jo ki BP ka usko preempt kar lunga and jo new process ka P1 - IP usko wai schedule kar lunga

Jab tak top pri SP ki process terminate nhi ho jati baki waale process IP & SP ko mauka nhi milega.

=> Starvation hogi

Multi level feedback queue scheduling (MLFQ)

(a) Multiple sub queues are present.

(b) Allows the process to move b/w queues. The idea is to sep. processes according to the characteristics of their BT. If a process uses too much CPU time it will be moved to lower priority queue. This scheme leaves I/O bound & interactive processes in the higher priority queue.

In addⁿ, a process that waits too much in a lower priority queue may be moved to a higher priority queue. This form of ageing Prevent starvation.

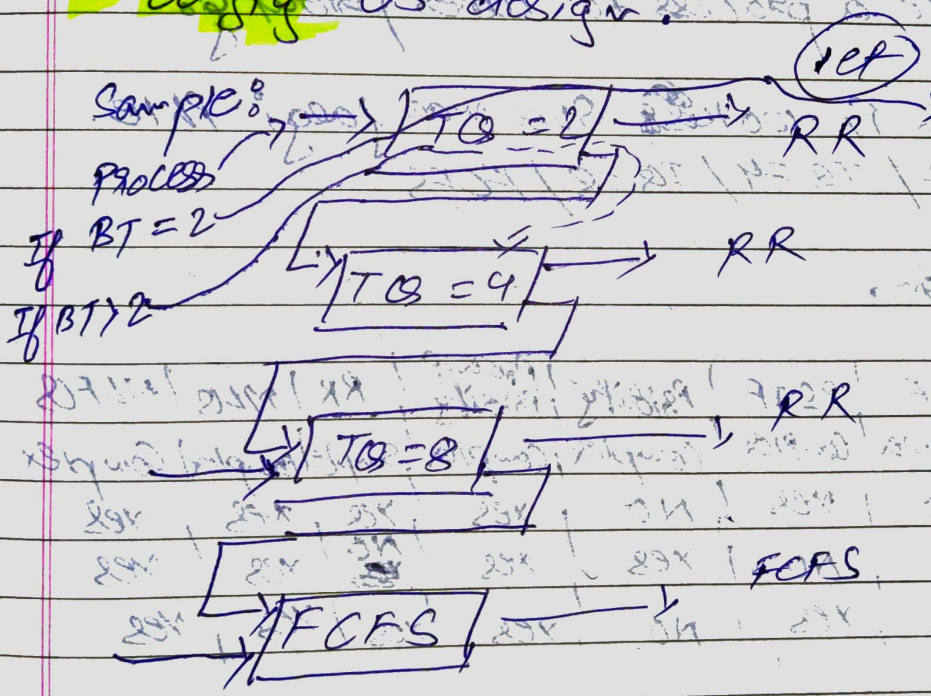
c. Less starvation than MLQ & it is flexible

d. Can be configured to match a specific system requirement sample MLQ design.

- Multiple sub queues
- Inter queue movement is allowed.
- Sep. process based on BT.
- BT ↑ => lower queues.
- I/O bound & interactive process => higher priority.

- aging method lower priority process ko priority increase karta jaega to avoid convoy effects.

- Config OS design.



ye karne se lower BT processes ko starve nhi hona chahiye

Agar koi process TQ=2 → TQ=4 → TQ=8 → FCFS

issue starvation ka

Design of MLFQ

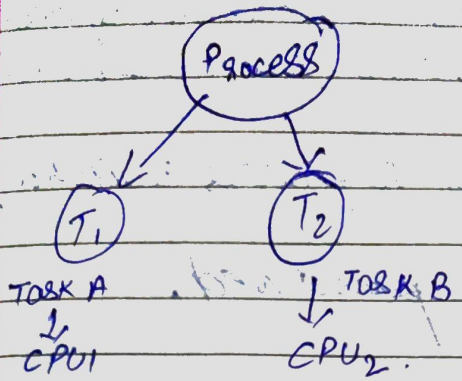
- No. of queue
- Scheduling Algo in each que
- Method to upgrade a process to higher que..
- demote a process to lower priority.
- Process P_1 woke up que mai jaegi phle. TQ=2 / TQ=4 / TQ=8 / FCFS

* Comparision.

	FCFS	SJF	PSJF	Priority	Precup Priority	RR	MLQ	MLFQ
Design	simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Precup	NO	NO	yes	NO	yes	yes	yes	yes
convoy eff	yes	yes	NO	yes	yes	NO	yes	yes
overhead	NO	NO	yes	NO	yes	yes	yes	yes

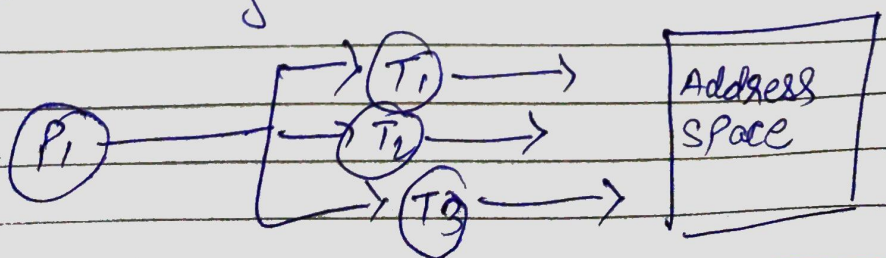
LEC-15 | Concurrency | Threads.

* Concurrency is the execution of the multiple instruction sequence at the same time. It happens in the OS when there are several process threads running in IIPI.



* Thread
- Single seq stream within a process.
- An independent path of execution in a process.
- Light weight process.
- Used to achieve parallelism by dividing a process's task which are independent path of execution.
- Eg- Multiple tabs in a browser, text editor.

* Thread scheduling
Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the OS.

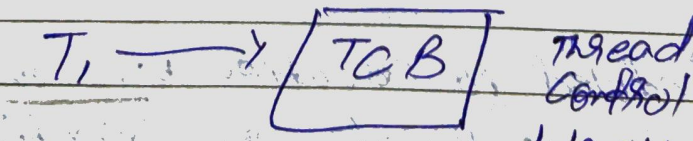


Thread context switching

- OS saves current state of thread & switches to another thread of same process.
- Does n't includes switching of memory address space. (But program counter, registers & stack are included)
- Fast switching as compared to process switching
- CPU's cache state is preserved.

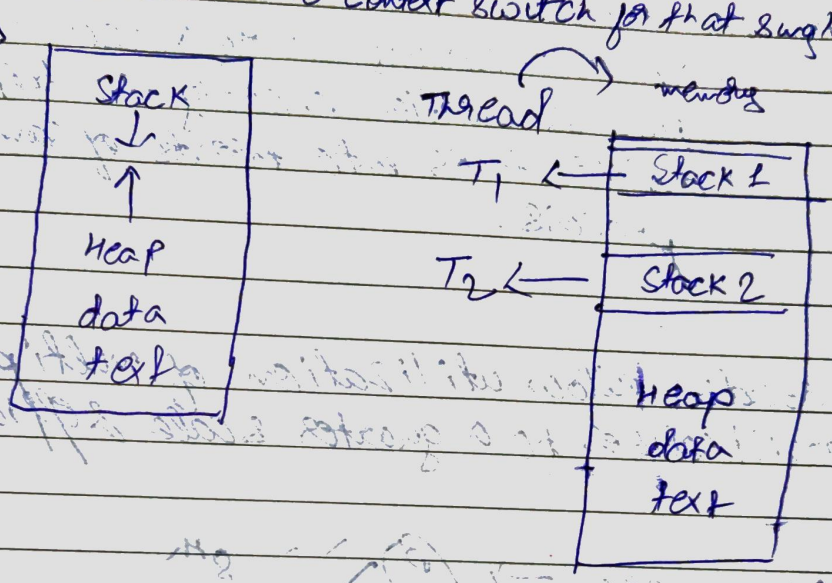
How each thread get access to CPU?

- Each thread has its own program counter.
- Depending upon the thread scheduling algo. OS schedule these threads ~~depending upon the thread scheduling algo~~
- OS will fetch instructions corresponding to PC of that thread & & execute instruction.



* I/O & TCB, based context switching is done here is well.
- we have TCB like PCB for state storage management while performing context switching.

* Will single CPU system could gain by multi threading technique?
- Never.
- As two threads have to context switch for that single CPU.



* Benefits of Multithreading
① Responsiveness
Ex- interactive app → taking input from user
→ I/O bhi ho aka
→ Internet se fetching bhi ho rhi
yaha independent but same path dekhe
user se input lene ka path
download krne ka diff path
backup krne ka diff path
* wai inn sbko threads wale divide kr dete hn
⇒ Agar ko koi thread I/O wale block bhi ho
baki threads formatting jaise task- bhi kr rhe
th wo particular process block nhi hui

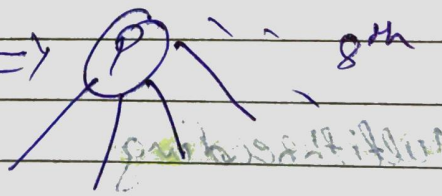
② Resource sharing

address space of same process threads are same so allow resource sharing.

③ Economical

- ~~process context~~
- It is more economical to create & context switch threads.
  => Also, allocating memory & resources for process creation is costly, so better to divide tasks into threads of same process.

④ Threads allow utilization of multiprocessor architectures to a greater scale & efficiency.

Ex. 8 core => 

Can be divided into 8 independent tasks.

Can be provided to 8 independent CPUs.