*Scheduling

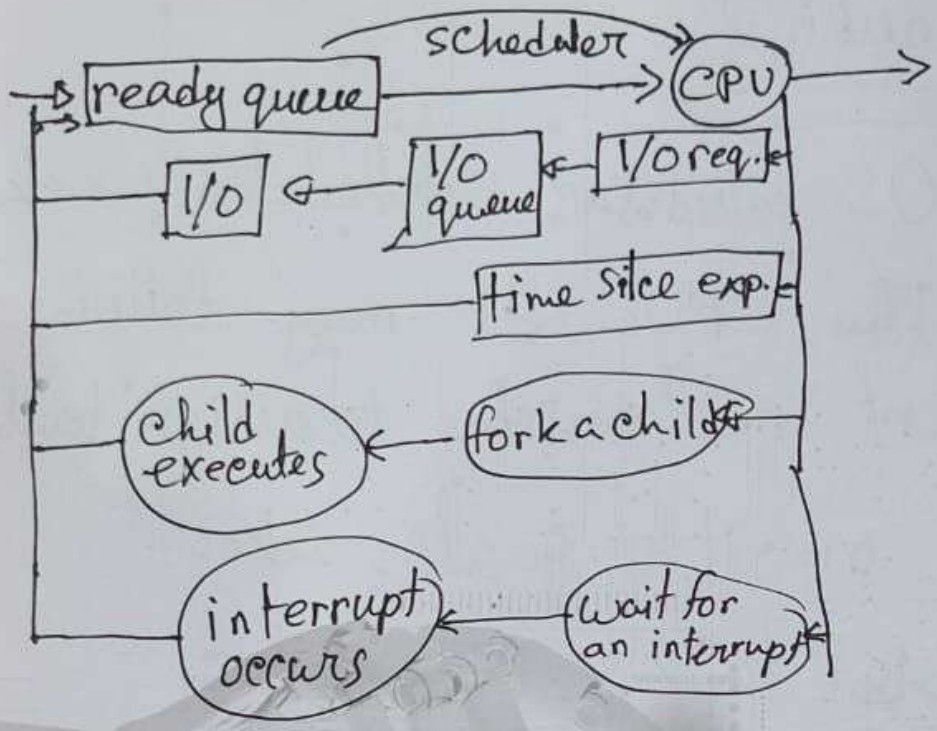
⇒An OS scheduler schedules work onto the CPUs. It may follow different mechanisms to assign tasks based on its goals, like:

(i) Assign tasks immediately
goal:- scheduling is simple (First come first serve (FCFS))

(ii) Assigns simple tasks first
- maximize throughput (shortest job first (SJF))

(iii) Assigns complex tasks first
- maximize utilization of CPU, devices, memory.

*CPU Scheduling



→ Chooses one of ready tasks to run on the CPU

→ scheduler runs when
  ⇒ CPU becomes idle
  ⇒ new tasks become ready
  ⇒ timeslice expired timeout.

→ Thread is dispatched on CPU.
  ↳ before: context switch, enter user mode, set PC & go!

In summary:

scheduling = choose tasks from queue.

⇒ which task should be selected?
→ depends on the scheduling policy / algorithm

⇒ How this is done?
→ dep. on the runqueue data-structure.

runqueue ⟺ scheduling algor.
↳ tightly coupled

(*) "Run-to-completion" scheduling

This type of sch. assumes as soon as a task is assigned to a CPU it will run & finishes or completes.

Initial assumptions:

→ group of tasks / jobs
→ known execution times
→ no preemption
→ single CPU

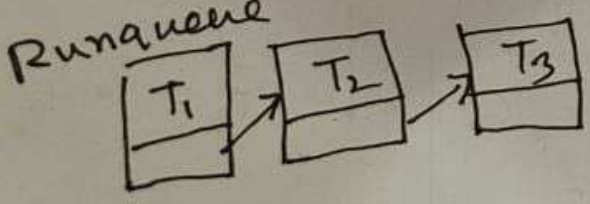
metrics for comp algo:

→ throughput,
→ avg. job completion time
→ avg. " waiting time
→ CPU utilization

(1) First-Come Fist-Serve (FCFS)

- schedules tasks in order of arrival

T_1 { T_2 { T_3 { runqueue == queue (FIFO)



$T_1 = 1s,\ T_2 = 10s,\ T_3 = 1s$ ... (i)

Throughput: $3/(1+10+1) = 0.25$ tasks/s.

Avg. completion: $(1+11+12)/3 = 8$ sec

Avg. wait: $(0+1+11)/3 = 4$ sec.

poor if one task is long

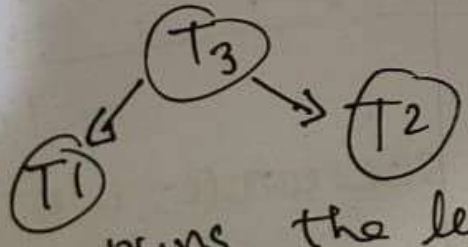
(ii) Shortest Job first (SJF)

→ schedules tasks in order of their execution time

→ (i) → T1 (1s) → T3 (1s) → T2 (10s)

runqueue = ordered queue.

or tree as.



runs the left first.

insertion is complex but retrival is not

Throughput : $(1+10+1)/3 = 0.25$ /s

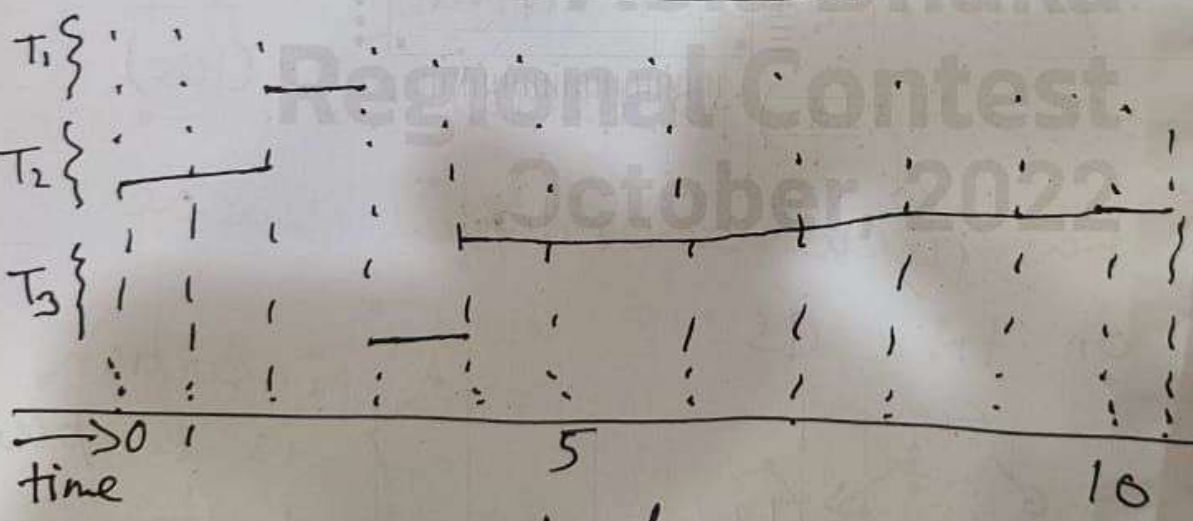
Avg. completion: $(1+2+12)/3 = 5$ s

Avg. wait : $(0+1+2)/3 = 1$

Preemptive scheduling

SJF + Preemption

Task	Exec time	Arrival time
T1	1s	2
T2	10 s	0
T3	1 s	2



~~if we~~ In actual scenario, we do not ~~ne~~ know the exec. time

so, we need a heuristic based on history => job running time
→ how long did a task run last time?
→ how long did a task run last n time? (window avg).

(iii) Priority scheduling (a part of preemptive sch.)

- tasks have different priority levels.
- run highest priority runs next.

Task	Exec	arrival	Priority
T_1	1	2	P1
T_2	10	0	P2
T_3	1	2	P3

P1 < P2 < P3

→running scenario:

0-2 → T_2

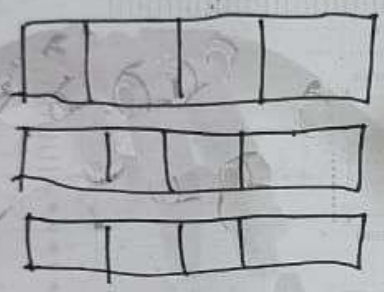
2-3 → T_3 (as this has the highest prio)

3-10 → T_2 (second highest)

10-11 → T_1 (as lowest)

runqueue for this is:

⇒ we can have multiple runqueues ds.



⇒ tree ordered based or Priority.

⇒ demerits:

starvation: as low priority task stuck in a run queue.

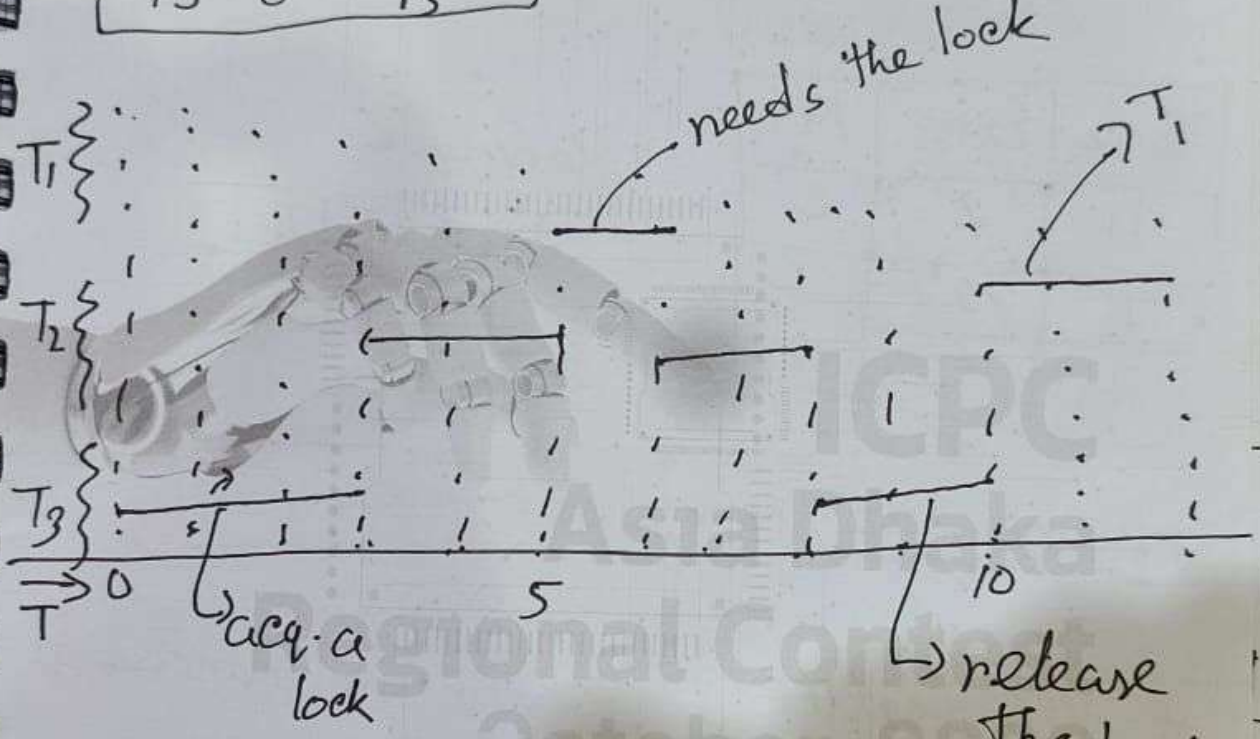
one possible solution: "priority aging". p = f(actual prio., time spent in run queue).

* (A math in the notes)

*Priority Inversion: assumes SJF here.

Task	Arr. time	Priority
T_1	5	P1
T_2	3	P_2
T_3	0	P_3

$P_1 > P_2 > P_3$



order of execution:

T_2, T_3, T_1

⇒ Priorities "inverted".

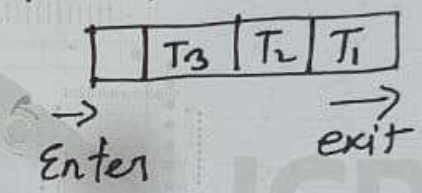
Solution:

- temp boost priority of mutex owner.

(iv) Round Robin Scheduling

- Pick up first task from queue. (FCFS)
- task may yield, to wait on I/O (unlike FCFS).

Task	Exec	Arr
T1	2	0
T2	2	0
T3	2	0

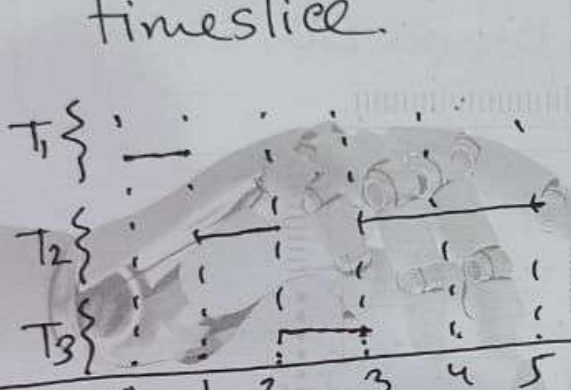


* more in notes for example

* Timeslice: maximum amount of uninterrupted time given to a task → time quantam

→ task may run less than timeslice time
- has to wait on I/O, sync...
  ⇒ will be placed on a queue

- higher priority task becomes runnable.
- using timeslices tasks are interleaved. ⇒ timesharing the CPU
- CPU bound task → preempted after timeslice.



E A
$T_1 \rightarrow 1 \rightarrow 0$
$T_2 \rightarrow 3 \rightarrow 0$
$T_3 \rightarrow 1 \rightarrow 0$

Alg	throughput	Avg. wait	avg. com
FCFS	0.25 t/s	4s	8s
SJF	0.25 t/s	1	5
RR(ts=1)	.25 t/s	1s	5.33

+ short task finish sooner
+ more responsive
+ lengthy I/O ops initiated sooner
⊖ overhead
⊕ sol: keep ts >> context switch

* How long should a timeslice be?

⇒ We have to keep in mind to balance benefits and overhead.

sc1: → for I/O bound

sc2 → for CPU "

* discuss these scenario with example