CPU Scheduling:-

selection of processes waiting in ready state to allocate CPU
followed process switching

per CPU only one process can be running state and other processes
will be in blocked or ready states

CPU scheduler -- typically called as short term scheduler

Mid term scheduler, long term scheduler
    -- memory scheduler,like swapper
    -- job scheduler (for delayed/periodical execution)

Who will enter ready state?
    newly created process/thread
    unblocked process/thread
    preempted process/thread

Ready Q/ Run Q

Scheduler Criteria:-

1.CPU utilization -- cpu cycles should be spent on some apps for max time, should be idle for min/zero time

2.Throughput -- no.of tasks(parts) completed per unit time

3.Turnaround time -- duration b'n creation and termination time spent ready,running,blocked states

4.CPU waiting time -- time spent in ready state due to non availability of CPU,scheduler not allowed also known as scheduling delay/latency/jitter

5.Response time -- time taken for first i/o and subsequent i/o

1&2 -- max, 3,4&5 -- min

scheduling latency/jitter/delay:-

t1 -- process arrives ready state
t2 -- process entered running state(scheduled)
(t2-t1) is accounted as scheduling delay

it's one of the signficant factor influencing real time scheduling

Real Time (system/computing/os/application)
deterministic behaviour -- response/completion
a finite upper limit on delay : (t2+delta)
eg:- vehicle safty, satellite/defense, medical emergency
telephonic apps. media streaming,

hard real time vs soft real time

expectations:-
no starvation -- get scheduled within finite time
fairness -- reasonable delay for every process
real time constraints

Process execution -- CPU Bursts, I/O Bursts
CPU Burst -- duration of running state
I/O Burst -- duration of blocked state

CPU Bound processes -- more time on CPU,less on I/O
background processes/scientific computing

I/O Bound processes -- more on I/O , less on CPU
interactive in nature, foreground

Scheduler can prefer I/O bound to CPU bound due to
fairness and nature of applications

Under what conditions switching occurs?
When scheduler will be invoked?

1.Termination -- assigned task is over
2.Blocking call -- i/o needs
3.High prio process arrived
4.Time limit expires
5.H/w interrupt occured, ISR execute

Non preemptive scheduling ==> 1&2 only
Preemptive scheduling ==> 3&4 also...1&2 of course

Classical algorithms:-
==> First Comes FIrst Serves(FCFS)
==> Shortest Job First(SJF), SRTF
==> Priority Based
==> Round Robin(RR) -- Time slice based

Gannt chat -- timing diagram -- delays

FCFS:-

based on arrival time, earliest arrival time as implicit priority
always non preemptive
simple to implement, less no.of context switchings (+)
average waiting time need not be minimal (-)
no fairness, but no starvation

convoy effect -- shortter processes getting more delay
due to longer processes ahead..eg:- p4,p5

arrival sequence, delays:-

	scheduled	arrival	delay	avg: 7.4
p1:	0	0	0	
p2:	4	1	3	
p3:	6	2	4	
p4:	16	2	14	
p5:	19	3	16	

SJF:-

based on least cpu requirement, min cpu burst as high prio
can guarrantee minimal avg waiting time
in case of equal CPU requirements, FCFS may be applied
may lead to starvation

practically not possible to implement, as knowing future CPU requirement is not possible..but the principle behind this will be basis for periodical executions like earliest deadline first(EDF) in real time systems

sequence, delay

process	scheduled	arrival	delay	avg: 3.6
p1	0	0	0	
p5	4	3	1	
p2	5	1	4	
p4	7	2	5	
p3	10	2	8	

Note:- every scheduling algorithm should select process with O(1) complexity

can be preemptive -- shortest remaining time first(SRTF)
-- preemptive SJF

Priority Based:-

Based on explicit priorities assigned by user(lib calls/sys calls)
Non preemptive algorithm -- not meaningful
Preemptive algorithm -- most practical
In case of equal prioirity -- FCFS or RR may be applied
Scheduling delay -- low for high prio process and vice versa
May lead to starvation -- low prio process may not be
scheduled or keep on getting preempted
due to arrival of high prio process
Aging technique can prevent starvation -- increasing
prio of process(inc/dec numeric) gradually
based on waiting time..
finite time=time_to_increase_prio * diff_in_prio_levels

process	delay	
p1	(0-0) + (14-1)	= 13
p2	(1-1) + (13-2)	= 11
p3	2-2	= 0
p4	17-2	= 15
p5	12-3	=9

Round Robin(RR) policy:-

Based on time slice/time quantum
No process can run beyond the time slice at a time
Preemptive...context switchings will increase
No starvation, acheives fairness to some extent
max delay = no_of_processes * time_slice
Lesser the quantum -- better fairness,but more switching overhead and vice versa
Useful multi tasking in general purpose systems

timing sequence:-

0 -- p1 scheduled
2 -- p2 scheduled, p1 preempts
4 -- p3 schedules, p2 completed/blocked
6 -- p4 scheduled, p3 preempts
8 -- p1 rescheduled, p4 preempts
10 -- p5 schedules, p1 completed
11 -- p3 reschedules, p5 completed
13 -- p4 rescheduled, p3 preempts
14 -- p3 rescheduled, p4 completes
16,18 -- p3 reschedules

Applicability of algorithms:-

PRIO ==> Real time scheduling
RR ==> fairness, multitasking GPOS
FCFS ==> background jobs

Combinations:-

PRIO+FCFS
PRIO+RR
pure RR (RR+FCFS)
pure FCFS -- normal

RR+PRIO ==> PRIO+RR

Multi level queue scheduling:-

More than one ready queue with independent policies

eg:- scheduling for smartphone OS
telephonic apps
user apps -- internet,media,games,PIM etc.
updates, sync

Multi queue feedback scheduling:-

processes may move across the queues

Multiple queues are maintained in PRIO policy
- -- different queues for each prio level
- -- with specialized h/w bit patterns highest level non empty Q can be identified
- -- This enables O(1) complexity.

Further:-

Scheduling in Linux -- Time Sharing, nice values

Scheduling in SMP -- cpu affinity, load balancing