

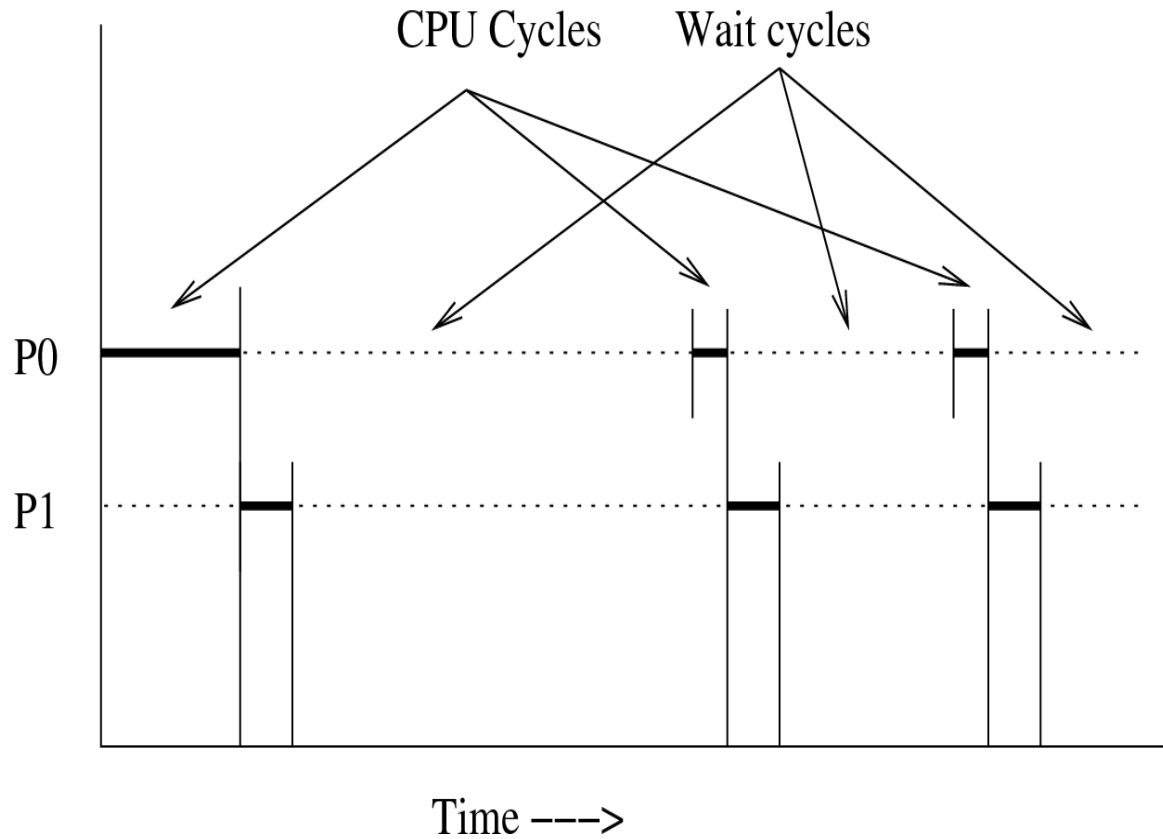
CPU Scheduling

R K Joshi
IIT Bombay

Requirements of CPU Scheduling

- CPU and IO cycles
- Short vs. long tasks
- Real Time vs. non-real time tasks
- Preemption vs. no preemption
- Priorities of tasks
- Utilization of idle cycles

CPU cycles and Wait/IO Cycles



Performance measures

Required time

- Per process:
 - Waiting time
 - Turnaround time
 - Penalty ratio (1/Response ratio)
- System measures
 - Throughput
 - Average waiting time
 - Average Turnaround time
 - Average penalty ratio (Response ratio)

Performance measures

■ Per process:

- Required time 20 seconds
- Waiting time 20 seconds
- Turnaround time 40 seconds
- Penalty ratio (1/Response ratio) $40/20 = 2$

■ System measures

- Throughput k processes per min.
- Average waiting time
- Average Turnaround time
- Average penalty ratio (Response ratio)

[Scheduling Policies]

- Non-preemptive policies

- Once a process is scheduled, it remains scheduled till completion

- Preemptive policies

- A scheduled process may be preempted and another may be scheduled

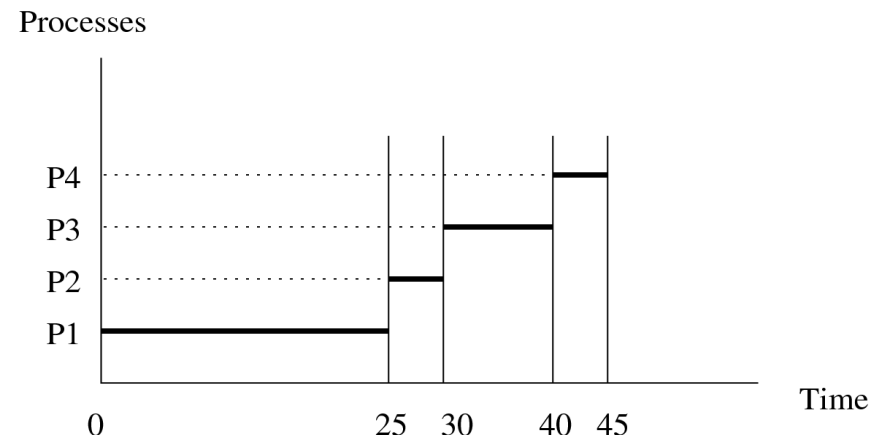
[When is a scheduler invoked?]

- Creation
- Completion
- Voluntary withdrawal
- Wait for a slower device
- Device Ready
- Policy dependent events

[First come first served (FCFS)]

Pid	CPU requirement
P1	25
P2	5
P3	10
P4	5

Schedule based on arrival time
Process executes till completion



FCFS Performance

Pid	Reqd. time	Waiting time	Turnaround time	Penalty Ratio = 1/Response ratio
P1	25	0	25	1
P2	5	25	30	6
P3	10	30	40	4
p4	5	40	45	9
averages		23.75		5

Throughput = $4/45$ processes per unit time

[FCFS on interactive processes]

- When a process waits or blocks, it is removed from the queue and it queues up again in FCFS queue when it gets ready
- Ordering in queue may be different in second serve

[Suitability and Drawbacks]

- Simple to implement
- Starvation free
- Examples: printer queues, mail queues

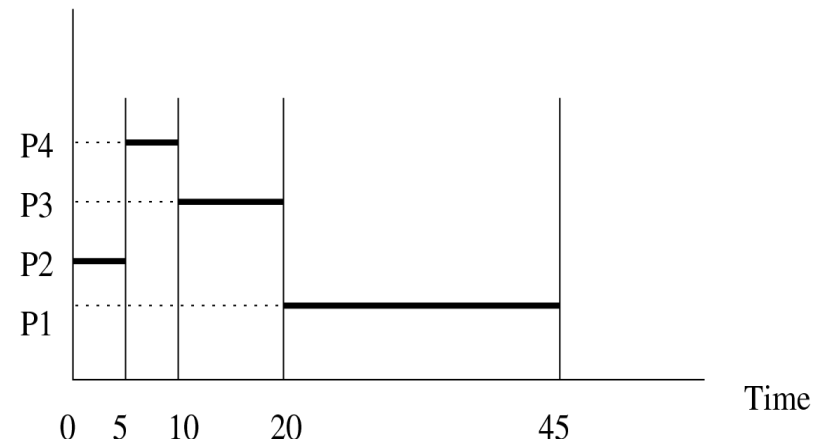
- Response time
- Suffers from Convoy Effect

[Shortest Job First (SJF)]

Pid	CPU requirement
P1	25
P2	5
P3	10
P4	5

Schedule based on job size
Process executes till completion

Processes



SJF Performance

Pid	Reqd. time	Waiting time	Turnaround time	Penalty ratio
P1	25	20	45	1.8
P2	5	0	5	1
P3	10	10	20	2
p4	5	5	10	2
averages		8.75		1.7

Throughput = $4/45$

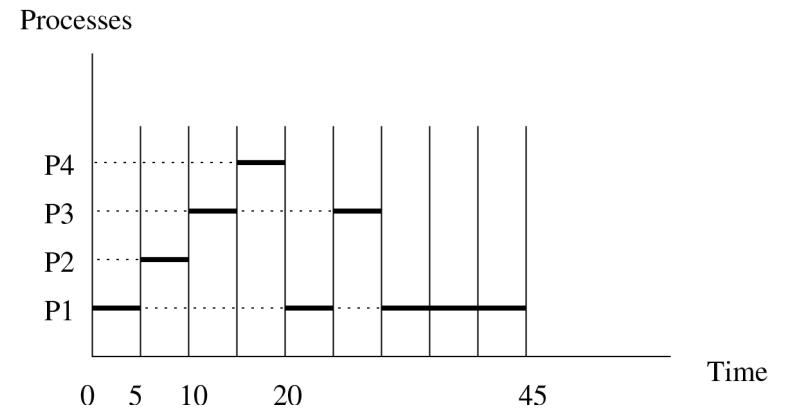
[Suitability and Drawbacks]

- Optimal for average waiting time
- Favors shorter jobs against long jobs
- If newly arrived process are considered at every schedule point, starvation may occur
- May not be possible to know the exact size of a job before execution

Round Robin (RR)

Pid	CPU requirement
P1	25
P2	5
P3	10
P4	5

Schedule based on time slicing



RR Performance

Pid	Reqd. time	Waiting time	Turnaround time	penalty ratio
P1	25	20	45	1.8
P2	5	5	10	2
P3	10	20	30	3
p4	5	15	20	4
averages		15		2.7

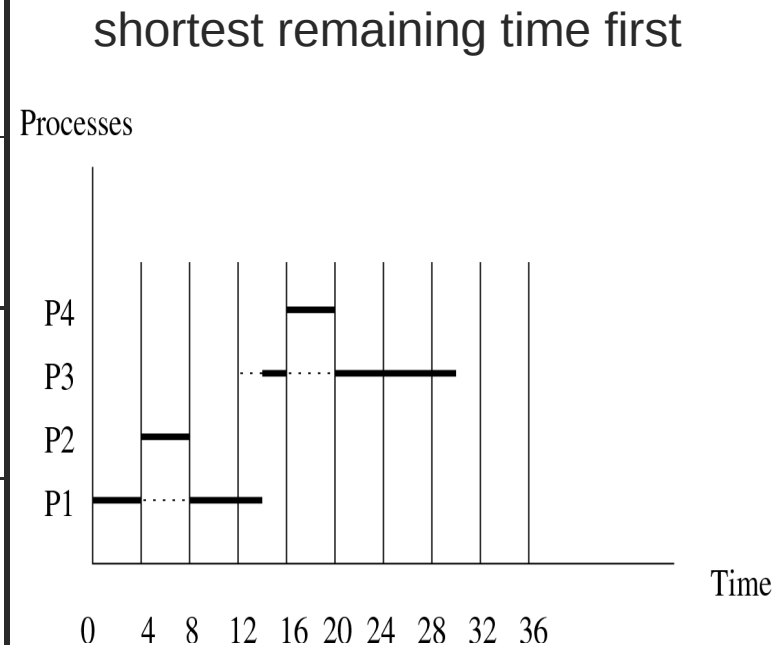
Throughput =

[Suitability and Drawbacks]

- Somewhere between FCFS and SJF
- Guarantees response time
- But it involves context switching
 - Attempt must be made to minimize context switch time
- Process needing immediate responses have to wait for $T * n - 1$ time units in worst case (calculate for 100 processes, 10 ms)

Preemptive Shortest Job First (or SRTF)

Pid	Arrival Time	CPU requirement
P1	0	10
P2	4	4
P3	8	12
P4	16	4



Preemptive SJF Performance

Pid	Reqd. time	Waiting time	Turnaround time	Response ratio
P1				
P2				
P3				
p4				
averages				

Throughput =

[Suitability and Drawbacks]

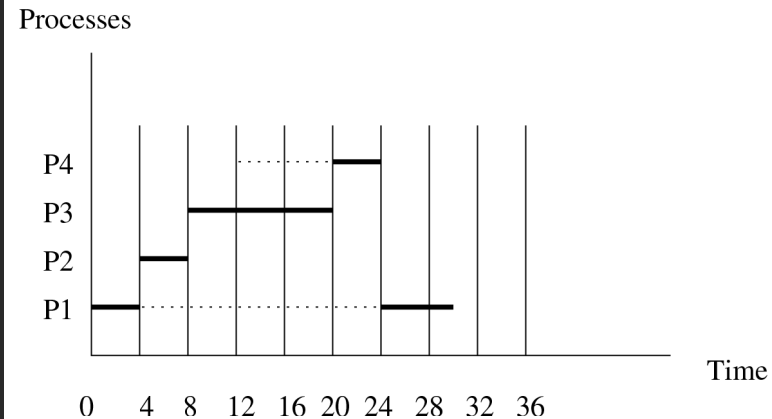
- SJF extended strictly considering arrivals at any point of time
- Optimal average waiting time in presence of dynamically arriving jobs
- The policy suffers from Starvation
- May not be possible to know the job size in advance → use prediction

Priority scheduling



Pid	Arrival Time	CPU requirement	Priority
P1	0	10	10
P2	4	4	12
P3	8	12	14
P4	12	4	12

Priority based schedule

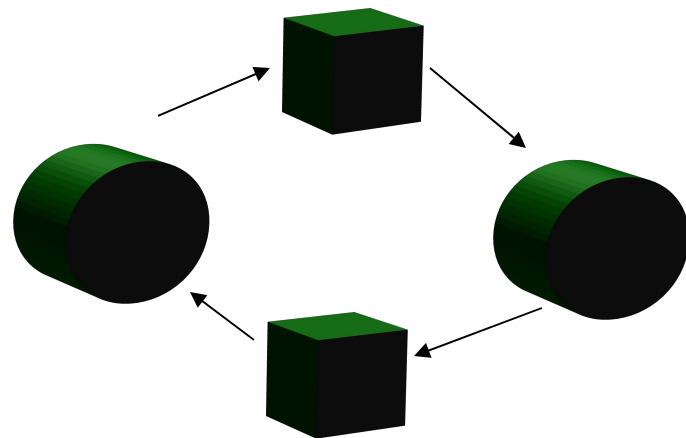


[Suitability and Drawbacks]

- One can combine several parameters in one priority value
- Computing priority is a challenging task : fairness must be guaranteed to various kinds of processes
- Tunable priorities: also from user space
- Deadlocks may occur in certain situations
- Priority Inversion problem!

Construct a deadlock case?

- P1 (pri=10) arrives
- P1 executes
- P2 (pri=12) arrives
- P1 is stopped and P2 executes
- Busy wait for P1



Priority Inversion

P1 (pri=10)	P2 (pri=12)
<ol style="list-style-type: none">1. Local computation2. Wait till R is locked3. Operations on R4. Release R5. Local computation	<ol style="list-style-type: none">1. Local computation2. Wait till R is locked3. Operations on R4. Release R5. Local computation

[Consider following case:]

- P3 arrives with priority=11
- P3 does not need resource R

[Point out Case of priority
inversion in above example?]

[Solution?

]

[Solution: Priority inheritance]

- Raise the priority of P1 to that of P2 till it finishes with the resource needed by P2

[Predictive SJF]

- Traditional UNIX scheduler uses:
 - $\text{Priority} = \text{seed priority} + (\text{Estimate}/4) + 2 * \text{nice priority}$
 - Lower the value, higher the priority
 - Seed priority: fixed at say 50
 - Every 10 ms: estimate of running process is incremented by 1
 - Estimate is reduced by a decay factor after every second (say by a decay of 0.5)

[For a process P1:]

Real time (sec)	0	1	2	3	4	5
System clock ticks	0	100	200	300	400	500
Estimate	0	50	75	87.5	93.75	96.875
priority	50	62.5	68..	71....	73..	74..

Estimate

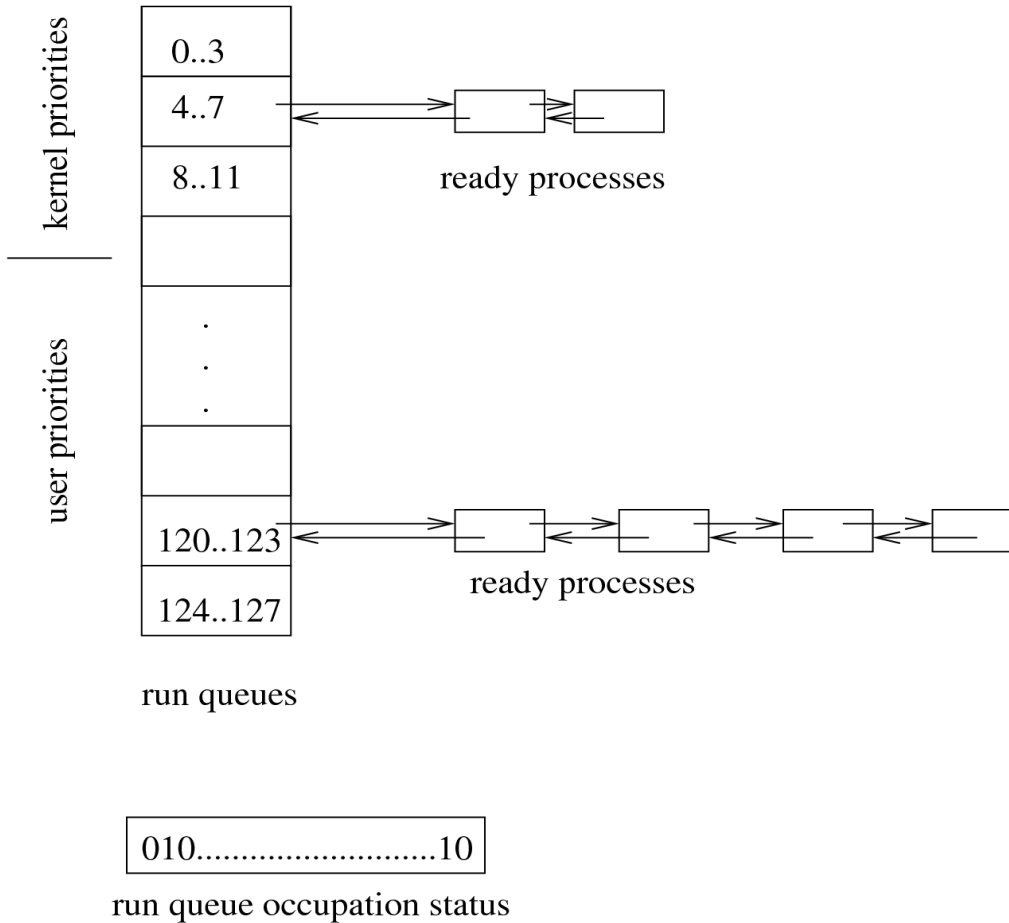
- Estimate = $\frac{1}{2}$ (CPU usage over last 1 second + Last estimate)
- $E_n = \frac{1}{2} (U_n + E_{n-1})$
- $E_1 = \frac{1}{2} (U_1 + E_0)$
- $E_2 = \frac{1}{2} (U_2 + E_1)$
- $E_2 = \frac{1}{2} U_2 + \frac{1}{4} U_1 + \frac{1}{4} E_0$
- $E_3 = \frac{1}{2} U_3 + \frac{1}{4} U_2 + \frac{1}{8} U_1 + \frac{1}{8} E_0$

Predictive SJF

$$T_{n+1} = x T_n + (1-x) T_{n-1}$$

$$0 \leq x \leq 1$$

Multilevel feedback queues



[4.4 BSD]

- Decay factor = $2 * \text{load} / (2 * \text{load} + 1)$
- 0-127 priority levels
- 50-127 user mode
- 32 run queues
- Queue no = priority / 4

[4.4 BSD]

- Sleeping process:
 - P_sleeptime is set to 0
 - Incremented every second
 - Estimate =
 - decay factor $p_sleeptime$ * estimate
 - Ignore nice priority

[4.4 BSD]

- Recompute priorities per second
- Round robin time slice 10 times per second
- Process in highest priority queue runs
- Hardclock() : 10ms