Lecture 13

Question

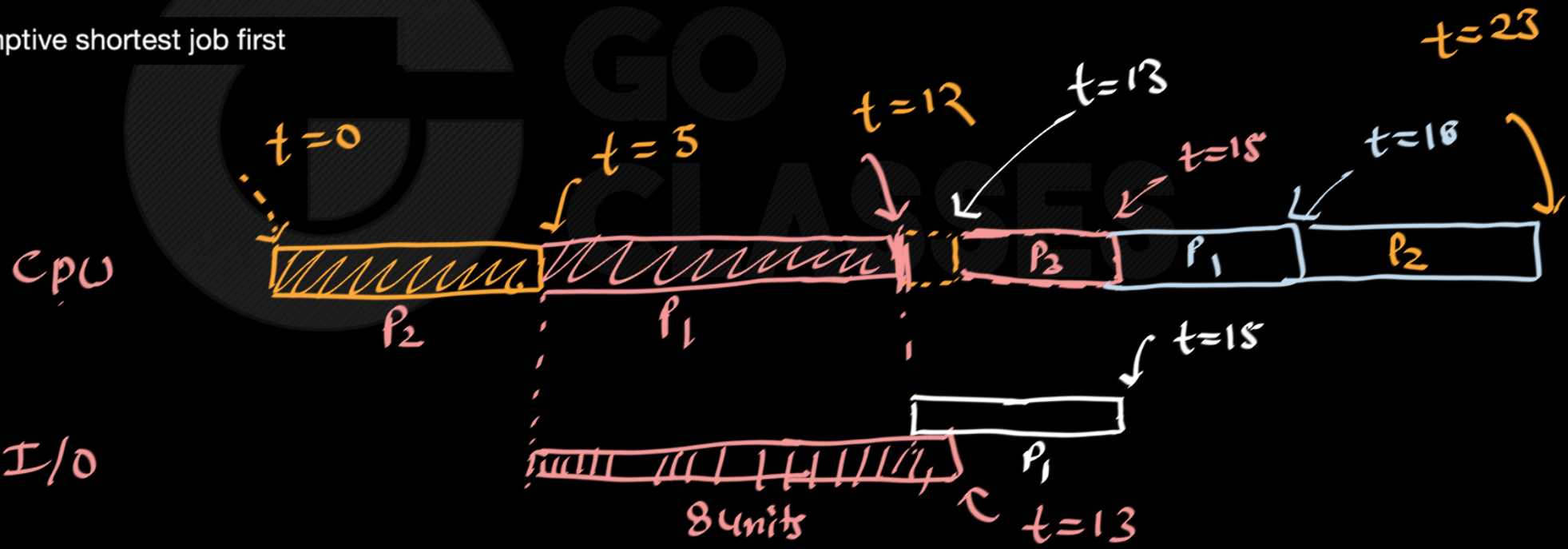
P1: CPU-7, IO-3, CPU-3
P2: CPU-5, IO-8, CPU-7

a. First-come/first-served
b. Shortest job first
c. Preemptive shortest job first
d. Round robin with a quantum of 2 ← H.W.

https://people.umass.edu/tongping/teaching/ece670/Midterm-Review.pdf

P1: CPU-7, IO-3, CPU-3
P2: CPU-5, IO-8, CPU-7 5

c. Preemptive shortest job first



a. First-come/first-served

Algorithm	P1 WT	P2 WT	Avg WT	P1 FT	P1 FT	Schd Len	CPU Ut
FCFS	2	7	4.5	15	27	27	81.48%

0					5					10					15					20					25					30					35
P1	R	R	R	R	R	R	R	w	w	w	r	r	R	R	R																			
P2	r	r	r	r	r	r	r	R	R	R	R	R	w	w	w	w	w	w	w	w	R	R	R	R	R	R	R							

b. Shortest job first

					5					10					15					20					25					30					35
P1	r	r	r	r	r	R	R	R	R	R	R	R	w	w	w	r	r	r	r	r	R	R	R											
P2	R	R	R	R	R	w	w	w	w	w	w	w	w	R	R	R	R	R	R	R														

c. Preemptive shortest job first

Algorithm	P1 WT	P2 WT	Avg WT	P1 FT	P1 FT	Schd Len	CPU Ut
PSJF	5	3	4	18	23	23	95.65%

					5					10					15					20					25					30					35
P1	r	r	r	r	r	R	R	R	R	R	R	R	w	w	w	R	R	R																
P2	R	R	R	R	R	w	w	w	w	w	w	w	w	R	R	r	r	r	R	R	R	R	R											

d. Round robin with a quantum of 2

					5					10					15					20					25					30					35
P1	R	R	r	r	R	R	r	r	R	R	r	R	w	w	w	R	R	R																
P2	r	r	R	R	r	r	R	R	r	r	R	w	w	w	w	w	w	w	w	R	R	R	R	R	R	R								

Algorithm	P1 WT	P2 WT	Avg WT	P1 FT	P1 FT	Schd Len	CPU Ut
RR 2	5	6	5.5	18	26	26	84.62%

Priority Scheduling

→ Static

→ Dynamic

FCFS

SJF

SRTF

RR

LJF

LRTF

Priority Scheduling

→ Static

→ Dynamic

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process **(Highest Response Ratio Next is one of the way to do it)**

https://www.andrew.cmu.edu/course/14-712-s20/applications/ln/14712-l6.pdf

Priority Scheduling

- Not all processes are equal, so rank them
- A priority number is associated with each process
- Run highest priority process first

Priority Scheduling

- Priorities can be **static** (i.e. they don't change) or **dynamic** (they may change during execution)
- Could be preemptive or nonpreemptive

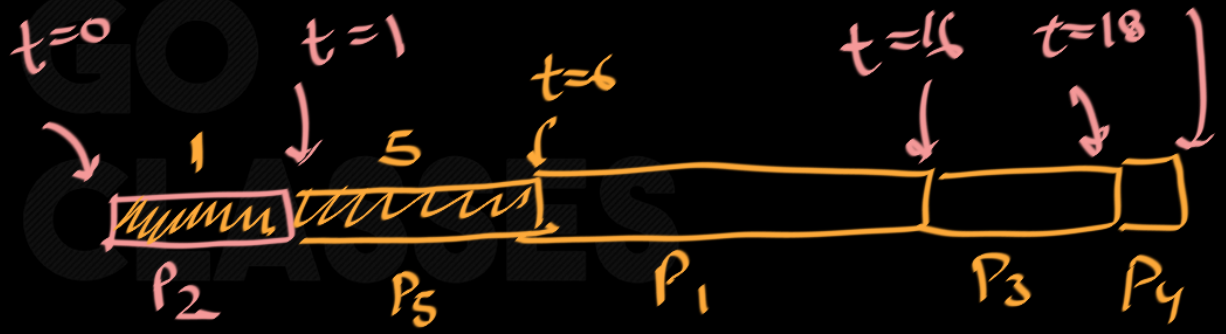
Non preemptive Priority Example

Draw Gnatt chart for following problem using Non preemptive Priority Scheduling

lower numbers are given higher priority

Arrival time = 0

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

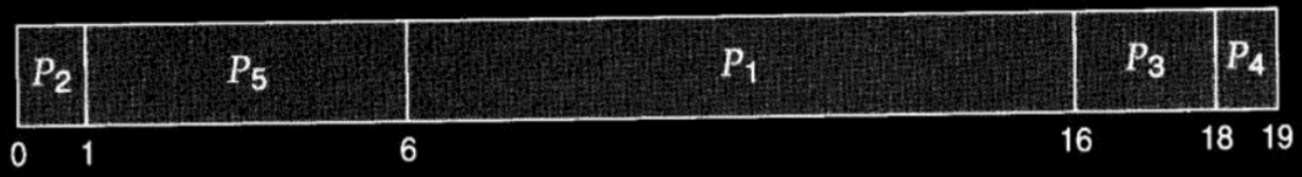


https://littleflowercollege.edu.in/upload/e_contents/files/e404245c60fe9b3488026680f8e384e0.pdf

Non preemptive Priority

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:



The average waiting time is 8.2 milliseconds.

Preemptive Priority

GATE CSE 2017 Set 2 | Question: 51

9,366 views

31

Consider the set of process with arrival time (in milliseonds), CPU burst time (in millisecods) and priority (0 is the highest priority) shown below. None of the process have I/O burst time

Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

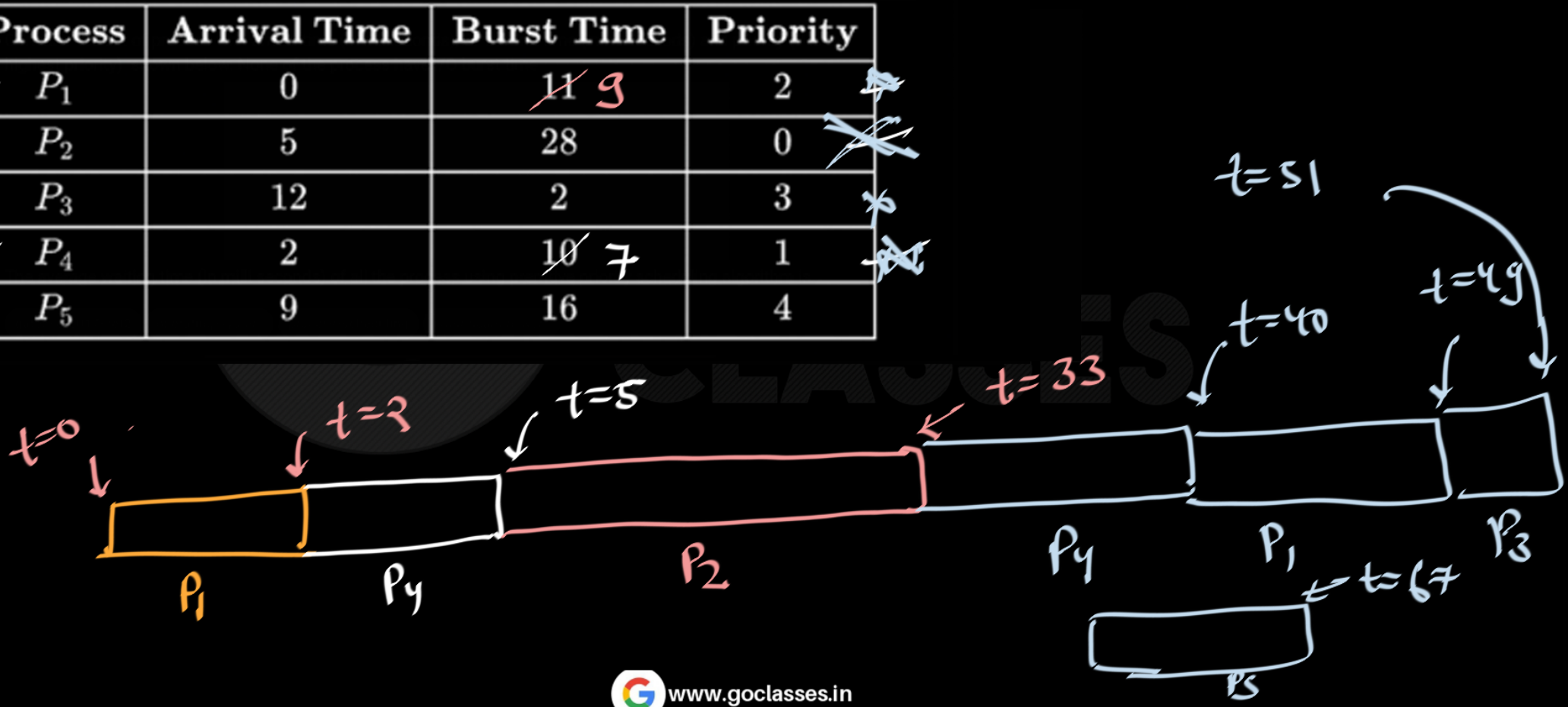
The average waiting time (in milli seconds) of all the process using premtive priority scheduling algorithm is _____

gatecse-2017-set2 operating-system process-scheduling numerical-answers

GATE CSE 2017 Set 2 | Question: 51

9,366 views

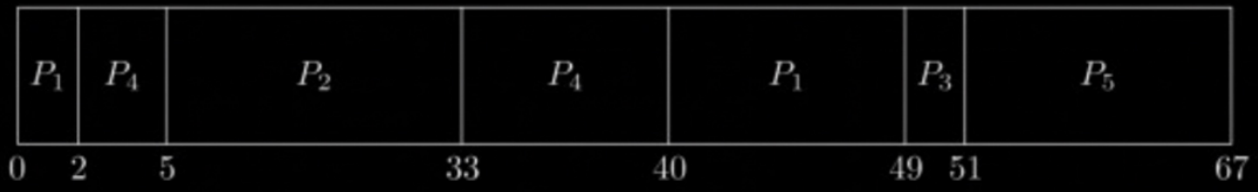
Process	Arrival Time	Burst Time	Priority
P_1	0	~~11~~ 9	2
P_2	5	28	0
P_3	12	2	3
P_4	2	~~10~~ 7	1
P_5	9	16	4



59

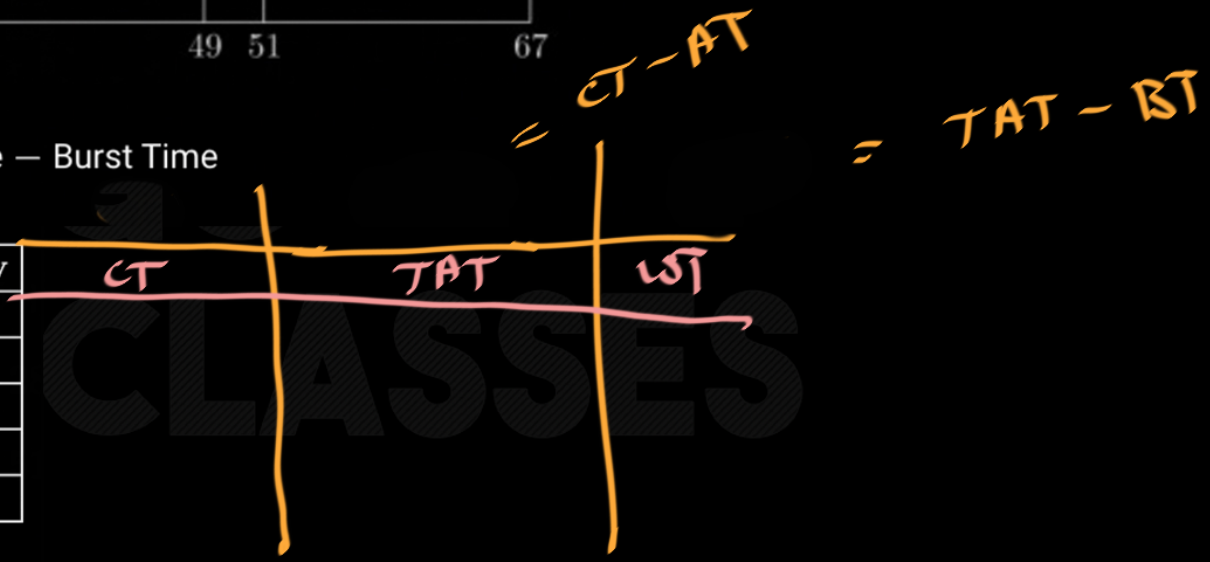
Best answer

Gantt Chart for above problem looks like :



Waiting Time = Completion time − Arrival time − Burst Time

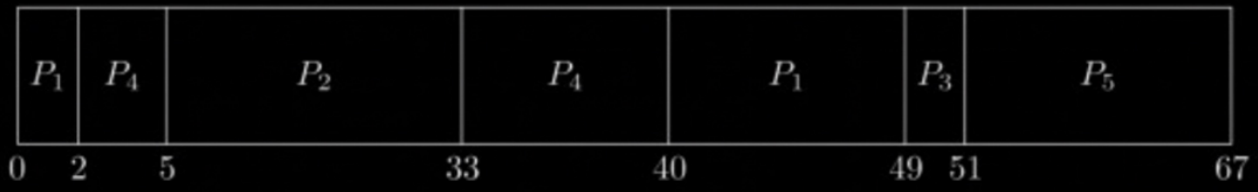
Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4



59

Best answer

Gantt Chart for above problem looks like :



Waiting Time = Completion time − Arrival time − Burst Time

Total Arrival Time = $\sum AT = 0 + 5 + 12 + 2 + 9 = 28$

Total Burst Time = $\sum BT = 11 + 28 + 2 + 10 + 16 = 67$

Total Completion Time = $\sum CT = 67 + 51 + 49 + 40 + 33 = 240$

Waiting time $= 240 - 28 - 67 = 145$

Average Waiting Time $= \frac{145}{5} = 29$ msec.

GO CLASSES

if there is one Process that can never get a chance.

is Starvation Possible in Priority Scheduling (that we studied) ?

Yes

- may be higher priority Processes keep comming in the ready queue, lower priority Processes will never get a chance

Question

2. **Scheduling.** Suppose a processor uses a prioritized round robin scheduling policy. New processes are assigned an initial quantum of length q. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice its current quantum. If a process blocks before its quantum expires, its new quantum is reset to q. For the purposes of this question, assume that every process requires a finite total amount of CPU time.

ever process will terminate

   (a) (5) Suppose the scheduler gives higher priority to processes that have larger quanta. Is starvation possible in this system? Why or why not?

← "q"

   (b) (5) Suppose instead that the scheduler gives higher priority to processes that have smaller quanta. Is starvation possible in this system? Why or why not?

https://es-static.fbk.eu/people/murphy/classes/456/2002/finalSoln.pdf

Question

2. **Scheduling.** Suppose a processor uses a prioritized round robin scheduling policy. New processes are assigned an initial quantum of length q. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice its current quantum. If a process blocks before its quantum expires, its new quantum is reset to q. For the purposes of this question, assume that every process requires a finite total amount of CPU time.

ever process will terminate

(a) (5) Suppose the scheduler gives higher priority to processes that have larger quanta. Is starvation possible in this system? Why or why not?

Some process with lowest priority ⇒ quantam = q

GO CLASSES

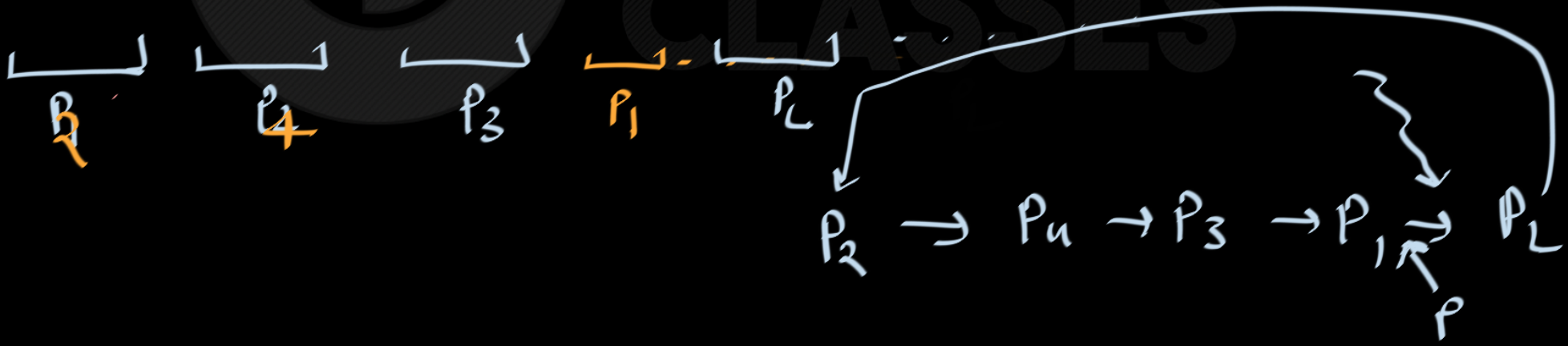
lowest Priority

P_L ← 9 quantum

assume

P_L will never get a chance.

P



lowest Priority

P_L ← loop queuing

assume ↙

P_L will never get a chance.

P

Process = q ← new processes are always highest priority Processes

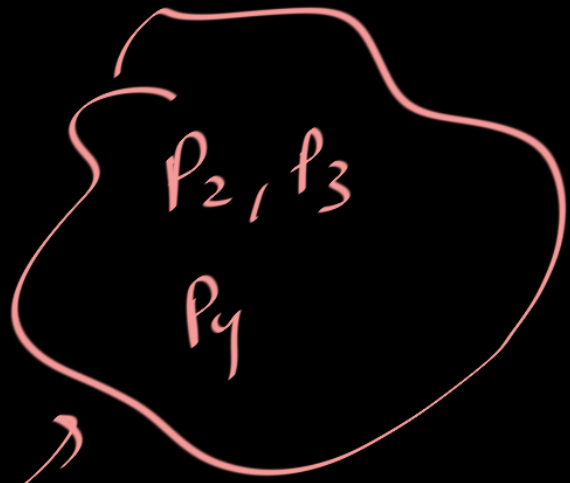
P_2 P_4 P_3 P_1

2q 4q

← will never get a chance if processes keep on com

GO CLASSES

RR



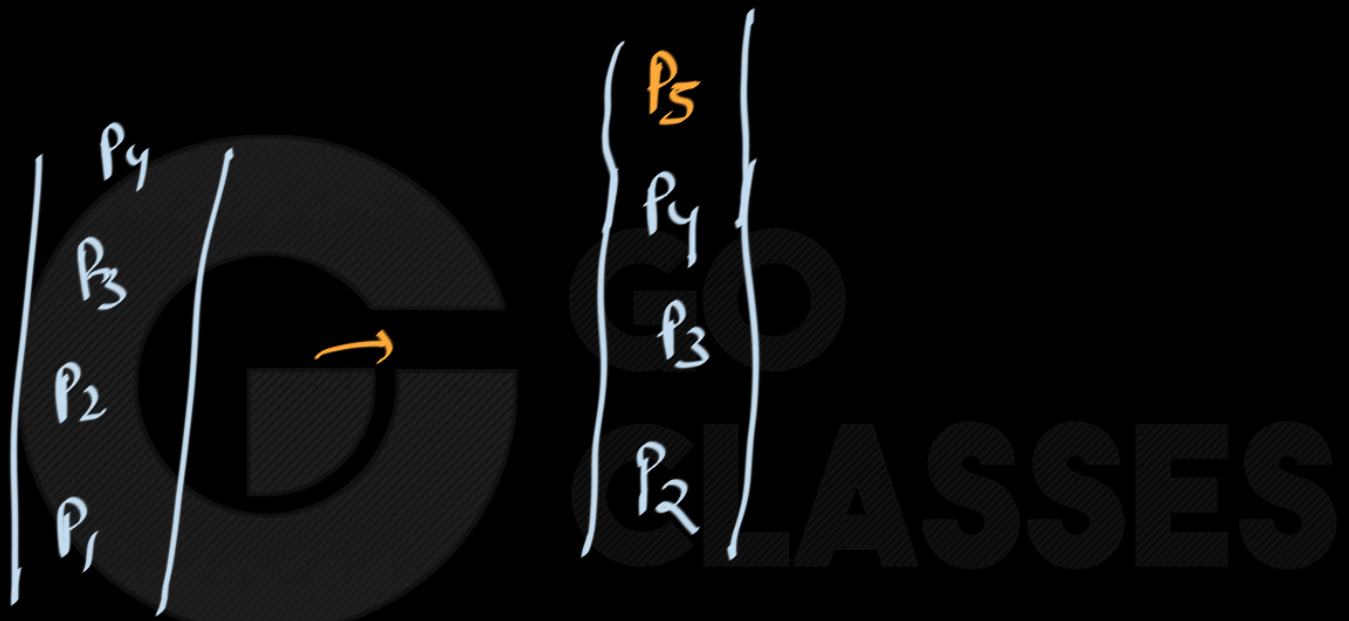
P5 come Ready queue

P1

(when P1 is running)

GO CLASSES

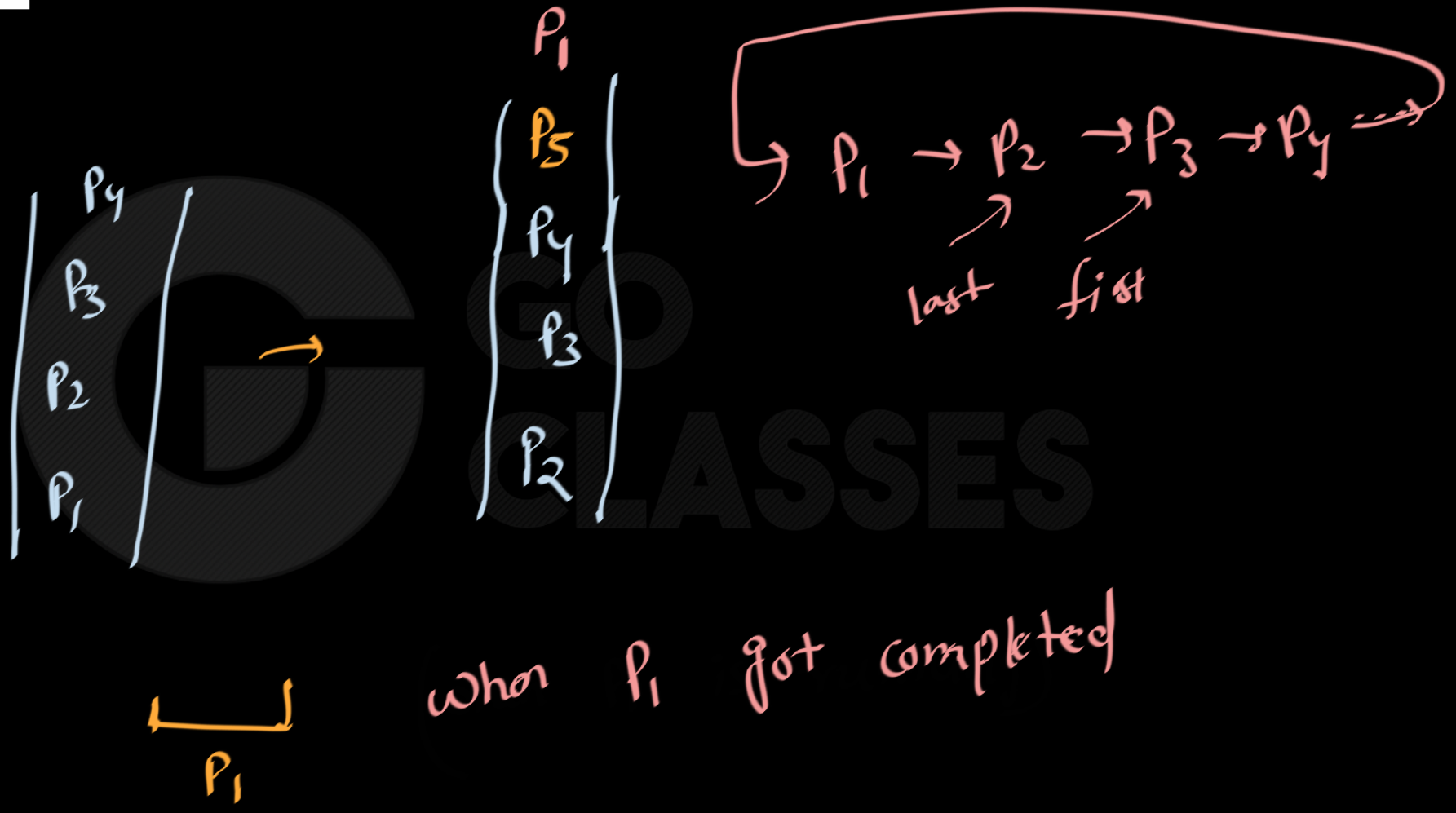
RR



(when P_1 is running)

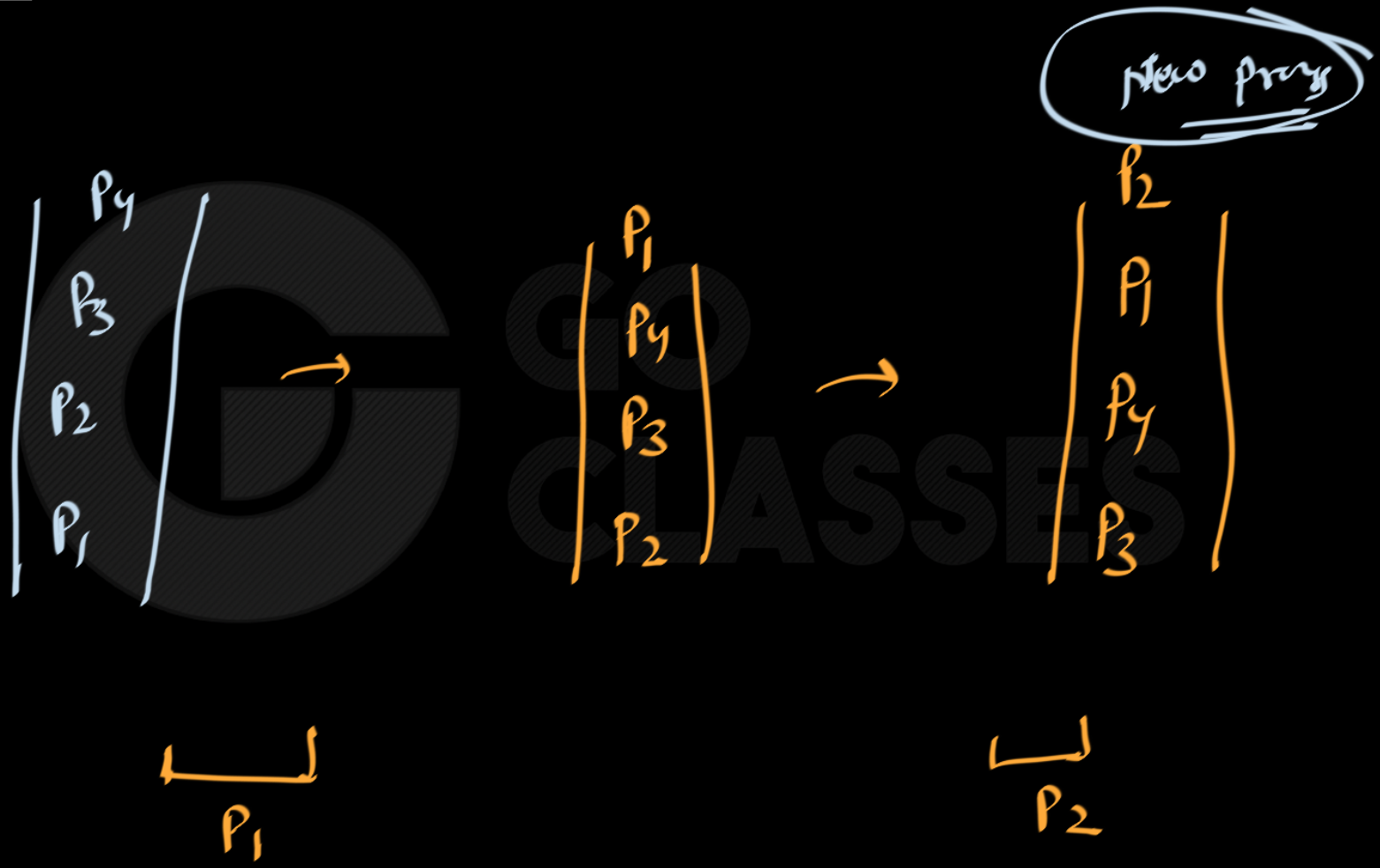
GO CLASSES

RR



GO CLASSES

RR



2. **Scheduling.** Suppose a processor uses a prioritized round robin scheduling policy. New processes are assigned an initial quantum of length q. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice its current quantum. If a process blocks before its quantum expires, its new quantum is reset to q. For the purposes of this question, assume that every process requires a finite total amount of CPU time.

   (a) (5) Suppose the scheduler gives higher priority to processes that have larger quanta. Is starvation possible in this system? Why or why not?

   No, starvation is not possible. Because we assume that a process will terminate, the worst that can happen is that a CPU bound process will continue to execute until it completes. When it finishes, one of the lower priority processes will execute. Because the I/O bound

   (b) (5) Suppose instead that the scheduler gives higher priority to processes that have smaller quanta. Is starvation possible in this system? Why or why not?

   Yes, starvation is possible. Suppose a CPU bound process runs on the processor, uses its entire quantum, and has its quantum doubled. Suppose a steady stream of I/O bound processes enter the system. Since they will always have a lower quantum and will be selected for execution before the process with the doubled quantum, they will starve the original process.

Question

H.W.

Assume three processes with estimated CPU bursts:
p1: 7 time units
p2: 1 time units
p3: 16 time units

a. Assume that all three processes are ready for execution. Explain and calculate the benefit of using short-process-next scheduling over just FIFO scheduling (first p1, then p2, then p3).

b. Now, assume that p1 and p3 are ready for execution, but p2 becomes ready only after 2 time units. Explain what is necessary to still achieve an optimal execution order that minimizes the average turnaround time. What is the resulting execution schedule?

https://student.cs.uwaterloo.ca/~cs350/common/354w03midsol.pdf

a. Assume that all three processes are ready for execution. Explain and calculate the benefit of using short-process-next scheduling over just FIFO scheduling (first p1, then p2, then p3).

The benefit of shortest-process-next scheduling is lower turnaround times. Under shortest process next scheduling, the turnaround times of the process are 1 time unit (p2), 8 time units (p1) and 24 time units (p3), for an average turnaround time of 11 time units. Under FIFO scheduling, the turnaround times are 7 (p1), 8 (p2), and 24 (p3), for an average of 13 time units.

b. Now, assume that p1 and p3 are ready for execution, but p2 becomes ready only after 2 time units. Explain what is necessary to still achieve an optimal execution order that minimizes the average turnaround time. What is the resulting execution schedule?

Preemption is needed. With preemption (SRT), p2 has a turnaround time of 1 (no waiting time), p1 has a turnaround time of 8 (1 time unit waiting), and p3 has a turnaround time of 24 (8 time units of waiting), as was the case which shortest-process-next scheduling in part (a).

The execution schedule is:

- *p1 runs for 2 time units*
- *p2 runs for 1 time unit*
- *p1 runs for 5 time units*
- *p3 runs for 16 time units*

GATE CSE 2015 Set 3 | Question: 1

31

The maximum number of processes that can be in $Ready$ state for a computer system with n CPUs is :

A. n
B. n^2
C. 2^n
D. Independent of n

gatecse-2015-set3 operating-system process-scheduling easy

https://gateoverflow.in/8390/gate-cse-2015-set-3-question-1

GATE CSE 2015 Set 3 | Question: 1

31

The maximum number of processes that can be in *Ready* state for a computer system with n CPUs is :

A. n
B. n^2
C. 2^n
D. Independent of n

gatecse-2015-set3 | operating-system | process-scheduling | easy

Running state with n CPUs

$\Rightarrow$ n

We can run n jobs at a time

https://gateoverflow.in/8390/gate-cse-2015-set-3-question-1

GATE CSE 1998 | Question: 2.17, UGCNET-Dec2012-III: 43

76

Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

A. $q \leq \frac{t-ns}{n-1}$
B. $q \geq \frac{t-ns}{n-1}$
C. $q \leq \frac{t-ns}{n+1}$
D. $q \geq \frac{t-ns}{n+1}$

$q > t = 3$
10

gate1998 | operating-system | process-scheduling | normal | ugcnetcse-dec2012-paper3

https://gateoverflow.in/1690/gate-cse-1998-question-2-17-ugcnet-dec2012-iii-43

GATE CSE 1998 | Question: 2.17, UGCNET-Dec2012-III: 43

76

Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

A. $q \leq \frac{t-ns}{n-1}$
B. $q \geq \frac{t-ns}{n-1}$
C. $q \leq \frac{t-ns}{n+1}$
D. $q \geq \frac{t-ns}{n+1}$

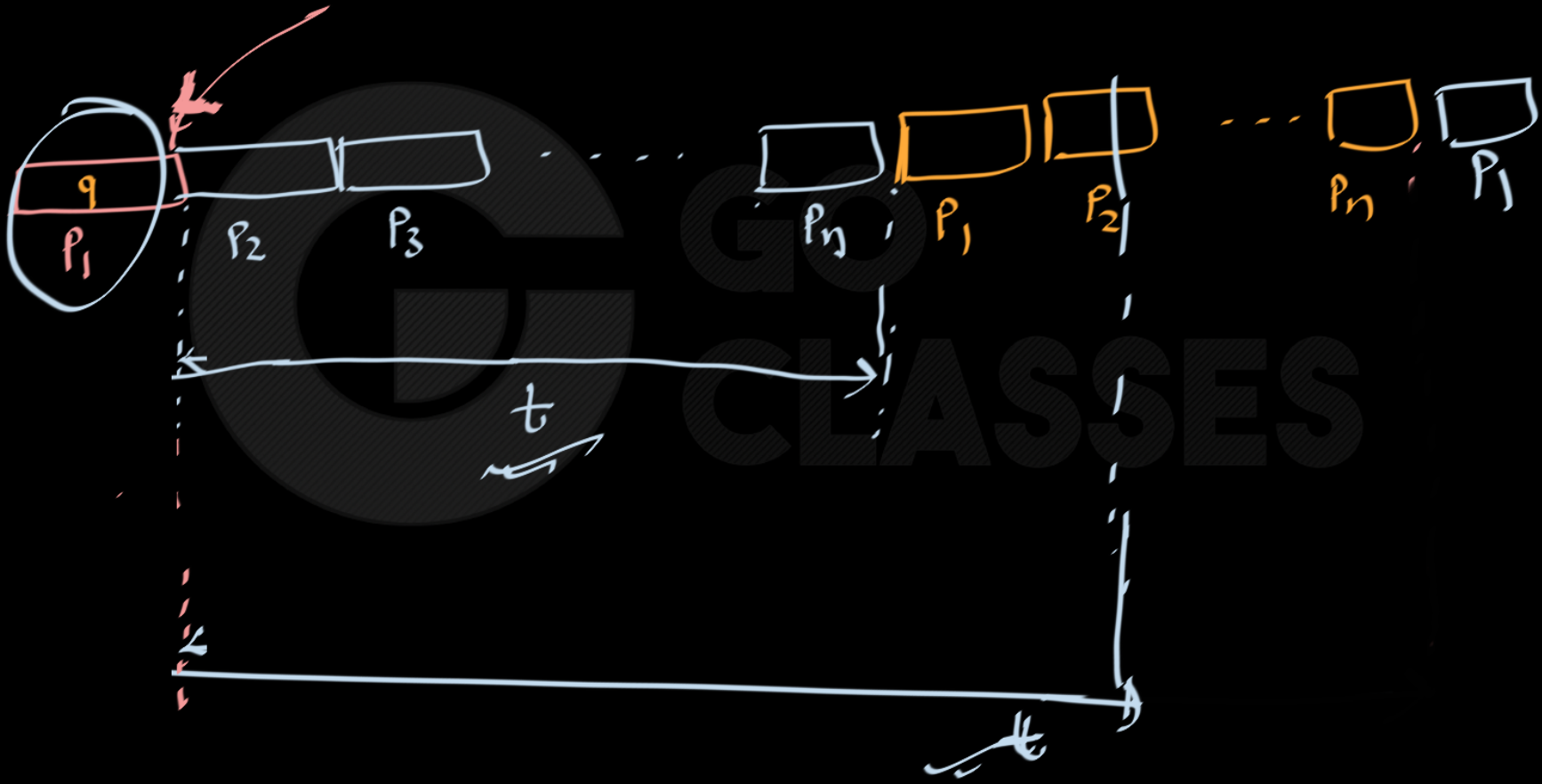
$t \geq (n-1)\,q + ns \Rightarrow$

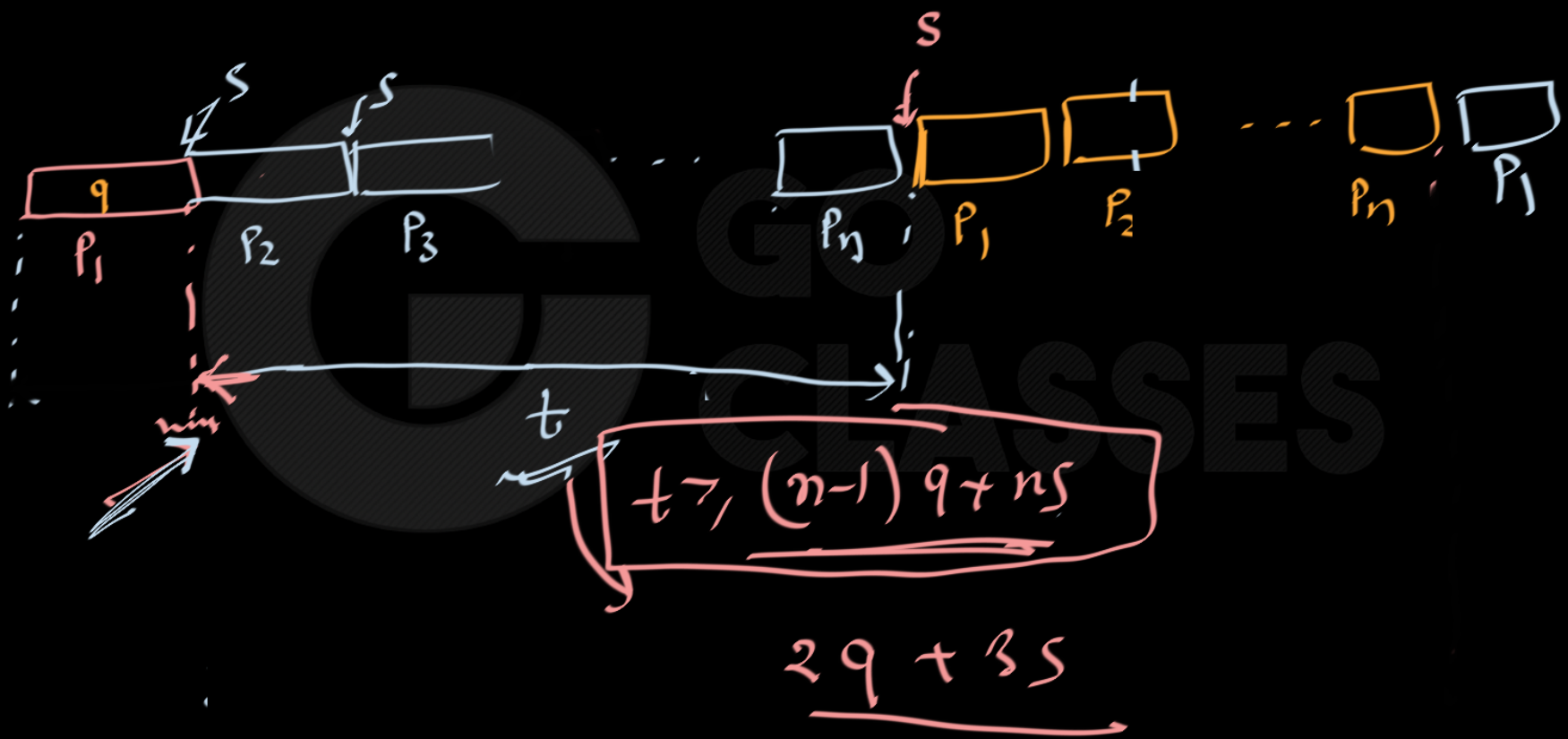
$q \leq \frac{t-ns}{n-1}$

[A]

gate1998 | operating-system | process-scheduling | normal | ugcnetcse-dec2012-paper3

https://gateoverflow.in/1690/gate-cse-1998-question-2-17-ugcnet-dec2012-iii-43


q
P_1
P_2
P_3
P_n
P_1
P_2
P_n
P_1
t
t


S
S
S
q
P1
P2
P3
Pn
P1
P2
Pn
P1
t
t = (n-1) q + ns
2q + 3s

Answer: (A)

98

Each process runs for q period and if there are n process: $p_1, p_2, p_3, \ldots, p_n$.

Then p_1's turn comes again when it has completed time quanta for remaining process p_2 to p_n, i.e, it would take at most $(n-1)q$ time.

Best answer

So,, each process in round robin gets its turn after $(n-1)q$ time when we don't consider overheads but if we consider overheads then it would be $ns + (n-1)q$

So, we have $ns + (n-1)q \leq t$

GATE CSE 2003 | Question: 77

83

A uni-processor computer system only has two processes, both of which alternate 10 ms CPU bursts with 90 ms I/O bursts. Both the processes were created at nearly the same time. The I/O of both processes can proceed in parallel. Which of the following scheduling strategies will result in the *least* CPU utilization (over a long period of time) for this system?

A. First come first served scheduling
B. Shortest remaining time first scheduling
C. Static priority scheduling with different priorities for the two processes
D. Round robin scheduling with a time quantum of 5 ms

gatecse-2003 operating-system process-scheduling normal

CPU ± I/O
10 90

https://gateoverflow.in/963/gate-cse-2003-question-77

GO CLASSES

A :

CPU	I/O	CPU	I/O	CPU	I/O
10	90	10	90	10	90

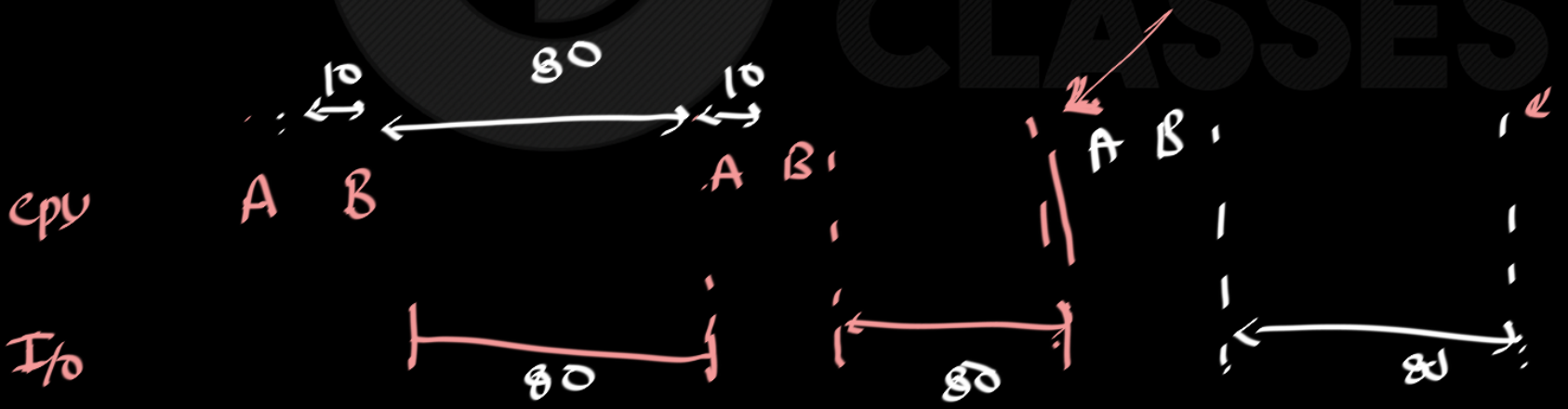
B :

CPU	I/O	CPU	I/O	CPU	I/O
10	90	10	90	10	90

A. First come first served scheduling
B. Shortest remaining time first scheduling
C. Static priority scheduling with different priorities for the two processes
D. Round robin scheduling with a time quantum of 5 ms

A : CPU 10 | I/O 90 | CPU 10 | I/O 90 | CPU 10 | I/O 90

B : CPU 10 | I/O 90 | CPU 10 | I/O 90 | CPU 10 | I/O 90



$\frac{40}{200} = 20\%$

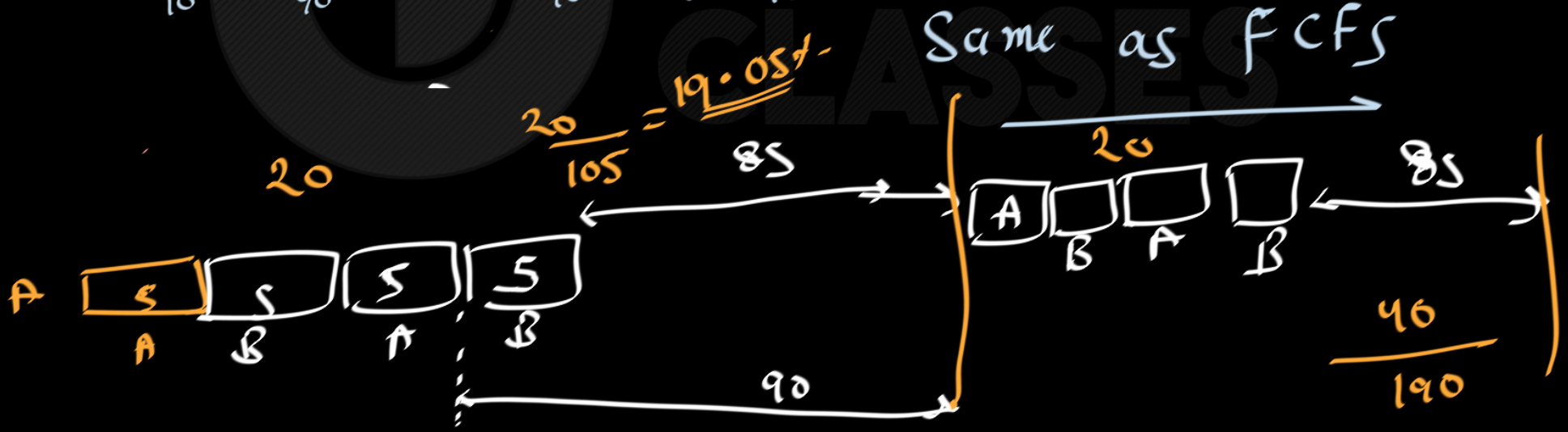
A:

CPU	I/O	CPU	I/O	CPU	I/O
10	90	10	90	10	90

B:

CPU	I/O	CPU	I/O	CPU	I/O
10	90	10	90	10	90

A. First come first served scheduling
B. Shortest remaining time first scheduling
C. Static priority scheduling with different priorities for the two processes
D. Round robin scheduling with a time quantum of 5 ms



155

Best answer

CPU utilization = CPU burst time/Total time.

FCFS:
from $0 - 10$: process 1
from $10 - 20$: process 2
from $100 - 110$: process 1
from $110 - 120$: process 2
....
So, in every 100 ms, CPU is utilized for 20 ms, CPU utilization $= 20\%$

SRTF:
Same as FCFS as CPU burst time is same for all the processes

Static priority scheduling:
Suppose process 1 is having higher priority. Now, the scheduling will be same as FCFS. If process 2 is having higher priority, then the scheduling will be as FCFS with process 1 and process 2 interchanged. So, CPU utilization remains at 20%

Round Robin:
Time quantum given as 5 ms.
from $0 - 5$: process 1
from $5 - 10$: process 2
from $10 - 15$: process 1
from $15 - 20$: process 2
from $105 - 110$: process 1
from $110 - 115$: process 2
...
So, in 105 ms, 20 ms of CPU burst is there. So, utilization $= 20/105 = 19.05\%$

19.05 is less than 20, so answer is (**D**).

(Round robin with time quantum 10ms would have made the CPU utilization same for all the schedules)

GATE CSE 2006 | Question: 06, ISRO2009-14

37

Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

A. 1
B. 2
C. 3
D. 4

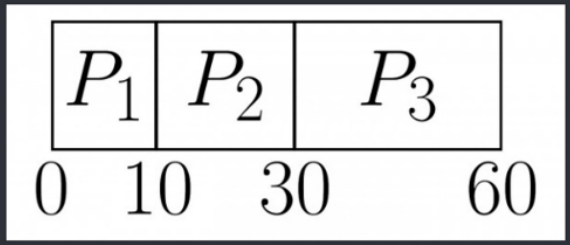
gatecse-2006 operating-system process-scheduling normal isro2009

https://gateoverflow.in/885/gate-cse-2006-question-06-isro2009-14

60

Best answer

Processes execute as per the following Gantt chart



So, here only 2 switching possible (when we did not consider the starting and ending switching)

Now here might be confusion that at $t = 2$ p_1 is preempted and scheduler checks that the newly arrived process has shorter job time or not, and does not find one, so it should not be consider as context switching (same happened at $t = 6$).

GATE CSE 2006 | Question: 65

53

Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

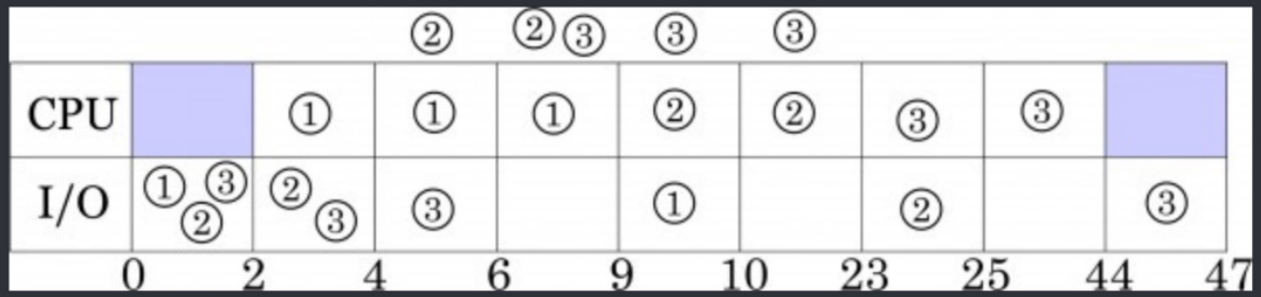
A. 0%
B. 10.6%
C. 30.0%
D. 89.4%

gatecse-2006 | operating-system | process-scheduling | normal

https://gateoverflow.in/1843/gate-cse-2006-question-65

81

Best answer



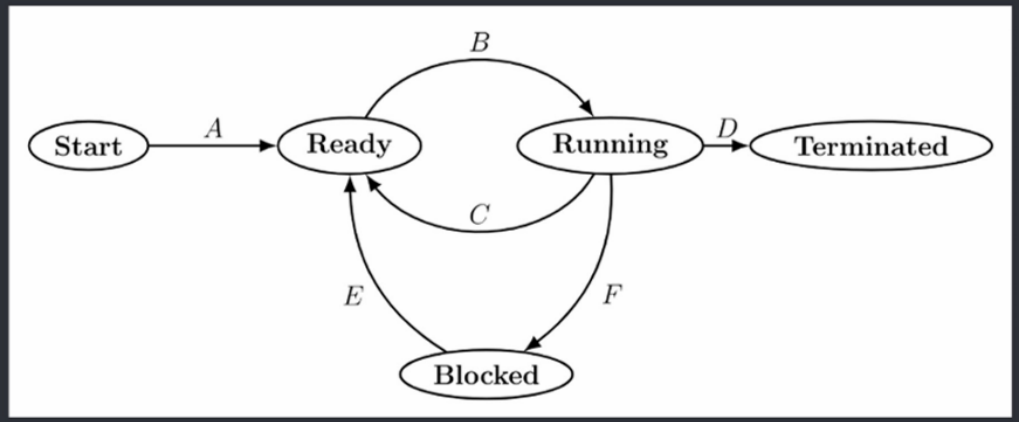
$$\text{CPU Idle time} = \frac{2+3}{47} \times 100 = 10.6383\%$$

Answer is **option (B).**

GATE CSE 2009 | Question: 32

31

In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state:



Now consider the following statements:

I. If a process makes a transition D, it would result in another process making transition A immediately.
II. A process P_2 in blocked state can make transition E while another process P_1 is in running state.
III. The OS uses preemptive scheduling.
IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?

A. I and II
B. I and III
C. II and III
D. II and IV

https://gateoverflow.in/1318/gate-cse-2009-question-32

63

Best answer

1. If a process makes a transition D, it would result in another process making transition A immediately. - This is false. It is not said anywhere that one process terminates, another process immediately come into Ready state. It depends on availability of process to run & Long term Scheduler.
2. A process $P2$ in blocked state can make transition E while another process $P2$ is in running state. - This is correct. There is no dependency between running process & Process getting out of blocked state.
3. The OS uses preemptive scheduling. :- This is true because we got transition C from Running to Ready.
4. The OS uses non-preemptive scheduling.Well as previous statement is true, this becomes false.

So answer is (**C**) II and III .

GATE CSE 2013 | Question: 10

55

A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is **TRUE** if the processes have no I/O operations and all arrive at time zero?

A. This algorithm is equivalent to the first-come-first-serve algorithm.
B. This algorithm is equivalent to the round-robin algorithm.
C. This algorithm is equivalent to the shortest-job-first algorithm.
D. This algorithm is equivalent to the shortest-remaining-time-first algorithm.

gatecse-2013 | operating-system | process-scheduling | normal

https://gateoverflow.in/1419/gate-cse-2013-question-10

57

(B) Because here the quanta for round robin is T units, after a process is scheduled it gets executed for T time units and waiting time becomes least and it again gets chance when every other process has completed T time units.

GATE CSE 2015 Set 1 | Question: 46

91

Consider a uniprocessor system executing three tasks T_1, T_2 and T_3 each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of $3, 7$ and 20 milliseconds, respectively. The priority of each task is the inverse of its period, and the available tasks are scheduled in order of priority, which is the highest priority task scheduled first. Each instance of T_1, T_2 and T_3 requires an execution time of $1, 2$ and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1^{st} millisecond and task preemptions are allowed, the first instance of T_3 completes its execution at the end of_____milliseconds.

gatecse-2015-set1 | operating-system | process-scheduling | normal | numerical-answers

https://gateoverflow.in/8330/gate-cse-2015-set-1-question-46

Answer is **12**

105

T_1, T_2 and T_3 have infinite instances, meaning infinite burst times. Here, problem say Run " T_1 for 1 ms", "T_2 for 2 ms", and "T_3 for 4 ms". i.e., every task is run in parts. Now for timing purpose we consider t for the **end** of cycle number t.

Best answer

- $T_1 : 0, 3, 6, 9, 12, \ldots \infty$ (T_1 repeats every 3 ms)
- $T_2 : 0, 7, 14, 21, \ldots \infty$ (T_2 repeats every 7 ms)
- $T_3 : 0, 20, 40, 60, \ldots \infty$ (T_3 repeats every 20 ms)

1. Priority of $T_1 = \frac{1}{3}$
2. Priority of $T_2 = \frac{1}{7}$
3. Priority of $T_3 = \frac{1}{20}$

Gantt Chart

0	T_1 1	T_2 2	T_2 3	T_1 4	T_3 5	T_3 6	T_1 7	T_2 8	T_2 9	T_1 10	T_3 11	T_3 12	...	.....

At $t = 0$, No process is available
At $t = 2$, T_2 runs because it has higher priority than T_3 and no instance of T_1 present

At $t = 4$, We have T_1 arrive again and T_3 waiting but T_1 runs because it has higher priority

At $t = 5$, T_3 runs because no instance of T_1 or T_2 is present
At $t = 11$, T_3 runs because no instance of T_1 or T_2 is present
At $t = 12$, T_3 continue run because no instance of T_1 or T_2 is present and first instance of T_3 completes

GATE CSE 2019 | Question: 41

33

Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below:

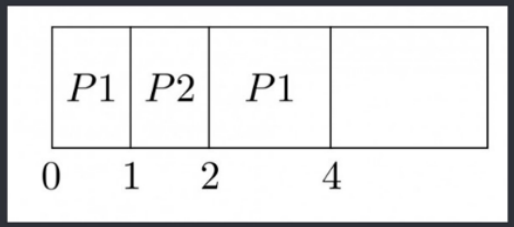
Process	$P1$	$P2$	$P3$	$P4$
Arrival Time	0	1	3	4
CPU burst time	3	1	3	Z

These processes are run on a single processor using preemptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is _____

gatecse-2019 | numerical-answers | operating-system | process-scheduling | 2-marks

https://gateoverflow.in/302807/gate-cse-2019-question-41

39



Best answer

Till $t = 4$, the waiting time of $P1 = 1$ and $P2 = 0$ and $P3 = 1$ but $P3$ has not started yet.

Case 1 :

Note that if $P4$ burst time is less than $P3$ then $P4$ will complete and after that $P3$ will complete. Therefore Waiting time of $P4$ should be 0. And total waiting time of $P3 = 1+$ (Burst time of $P4$) because until $P4$ completes $P3$ does not get a chance.

Then average waiting time $= \frac{1+0+(1+x)+0}{4} = 1$

$\frac{2+x}{4} = 1 \Rightarrow x = 2.$

Case 2 :

Note that if $P4$ burst time is greater than $P3$ then $P4$ will complete after $P3$ will complete. Therefore, Waiting time of $P3$ remains the same. And total waiting time of $P4 =$ (Burst time of $P3$) because until $P3$ completes $P4$ does not get a chance.

Then average waiting time $= \frac{1+0+1+3}{4} = 1$

$\frac{5}{4} \neq 1 \Rightarrow$ This case is invalid.

Correct Answer: 2

GATE IT 2007 | Question: 26

55

Consider n jobs $J_1, J_2 \ldots J_n$ such that job J_i has execution time t_i and a non-negative integer weight w_i. The weighted mean completion time of the jobs is defined to be $\frac{\sum_{i=1}^{n} w_i T_i}{\sum_{i=1}^{n} w_i}$, where T_i is the completion time of job J_i. Assuming that there is only one processor available, in what order must the jobs be executed in order to minimize the weighted mean completion time of the jobs?

A. Non-decreasing order of t_i
B. Non-increasing order of w_i
C. Non-increasing order of $w_i t_i$
D. Non-increasing order of w_i / t_i

gateit-2007 | operating-system | process-scheduling | normal

https://gateoverflow.in/3459/gate-it-2007-question-26

126

Best answer

Lets take an example:

Process	Weight	Execution time
P_1	1	3
P_2	2	5
P_3	3	2
$p4$	4	4

For option **1** non decreasing t_i

$$= (3 \times 2 + 1 \times 5 + 4 \times 9 + 2 \times 14)/10 = (6 + 5 + 36 + 28)/10 = 7.5$$

For option **2** non increasing w_i

$$= (4 \times 4 + 3 \times 6 + 2 \times 11 + 1 \times 14)/10 = (16 + 18 + 22 + 14)/10 = 7$$

For option **3** non increasing $w_i t_i$

$$= (16 + 2 \times 9 + 3 \times 11 + 1 \times 14)/10 = (16 + 18 + 33 + 14)/10 = 8.1$$

For option **4** non increasing w_i/t_i

$$= (3 \times 2 + 4 \times 6 + 2 \times 11 + 1 \times 14)/10 = (6 + 10 + 22 + 14)/10 = 6.6$$

Minimum weighted mean obtained from non increasing w_i/t_i **(option D)**

The solution above is a classical example of greedy algorithm - that is at every point we choose the best available option and this leads to a global optimal solution. In this problem, we require to minimize the weighted mean completion time and the denominator in it is independent of the order of execution of the jobs. So, we just need to focus on the numerator and try to reduce it. Numerator here is a factor of the job weight and its completion time and since both are multiplied, our greedy solution must be

- to execute the shorter jobs first (so that remaining jobs have smaller completion time) and
- to execute highest weighted jobs first (so that it is multiplied by smaller completion time)

So, combining both we can use w_i/t_i to determine the execution order of processes - which must then be executed in non-increasing order.

answered Jun 6, 2016 • edited Apr 15, 2019 by **Pooja Khatri**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful


khushtak

share this