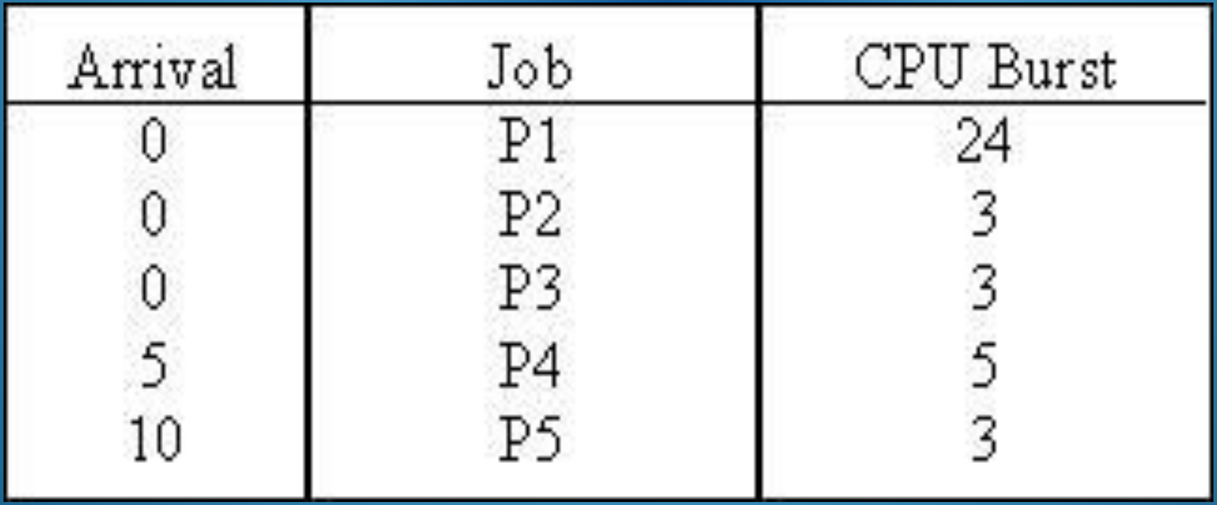
CS240 Operating Systems, Communications and Concurrency

<u>Process Scheduling Algorithms</u>

Non Preemptive Priority Scheduling using HRN
Preemptive Scheduling using RR
Analysis of RR with different quantum lengths
Multilevel Feedback Queues
Case Study – Traditional Unix Scheduler
and Fair Share Scheduler Concept

CS240 Operating Systems, Communications and Concurrency

Non Preemptive Highest Response Ratio Next (HRN) Scheduling:-
The Response Ratio of a job is calculated as follows:-
Response Ratio = (Waiting Time) / (Service Time)



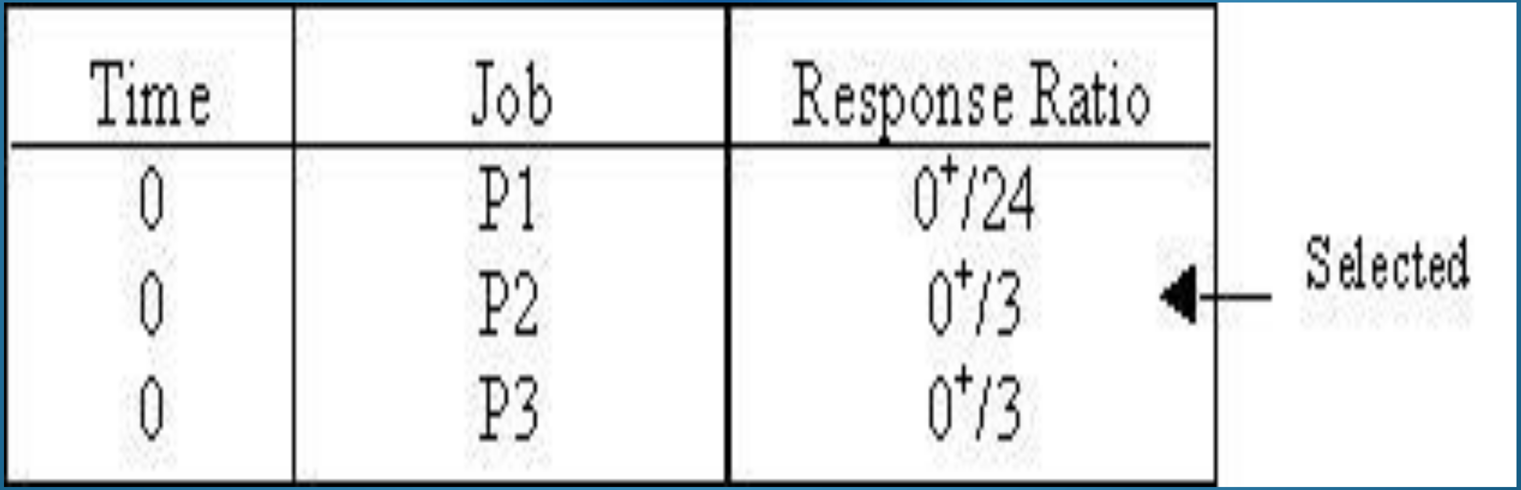
Arrival	Job	CPU Burst
0	P1	24
0	P2	3
0	P3	3
5	P4	5
10	P5	3

How would this workload work out with FCFS or SJF?

CS240 Operating Systems, Communications and Concurrency

Using HRN we get

At time 0

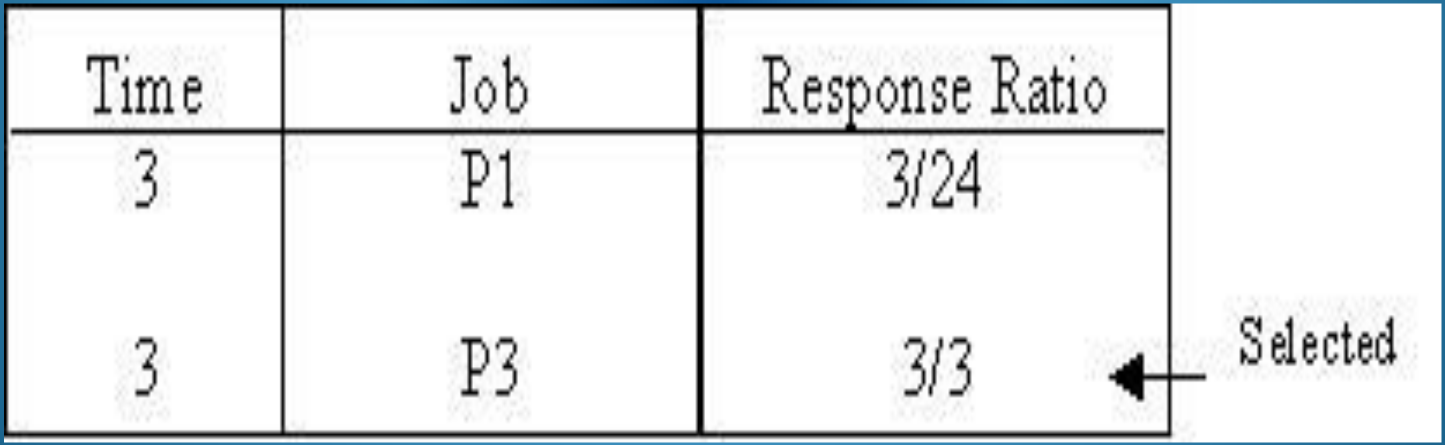


Time	Job	Response Ratio	
0	P1	$0^+/24$	
0	P2	$0^+/3$	← Selected
0	P3	$0^+/3$	

CS240 Operating Systems, Communications and Concurrency

Using HRN we get

At time 3

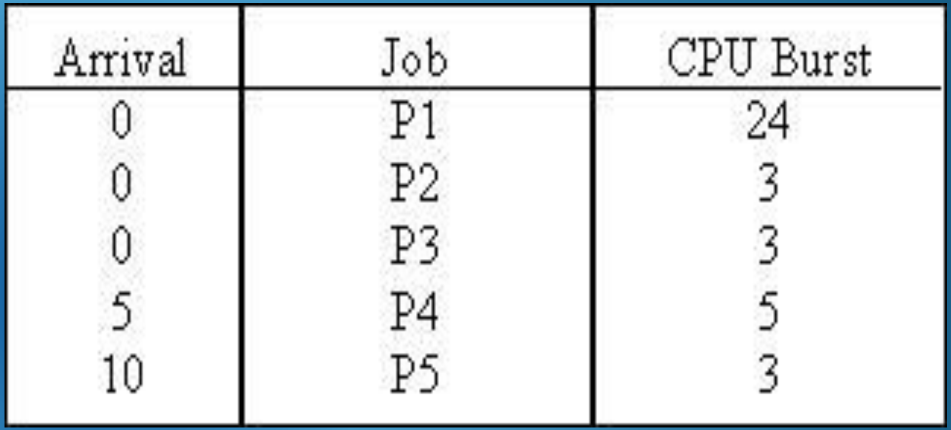


Time	Job	Response Ratio	
3	P1	3/24	
3	P3	3/3	← Selected

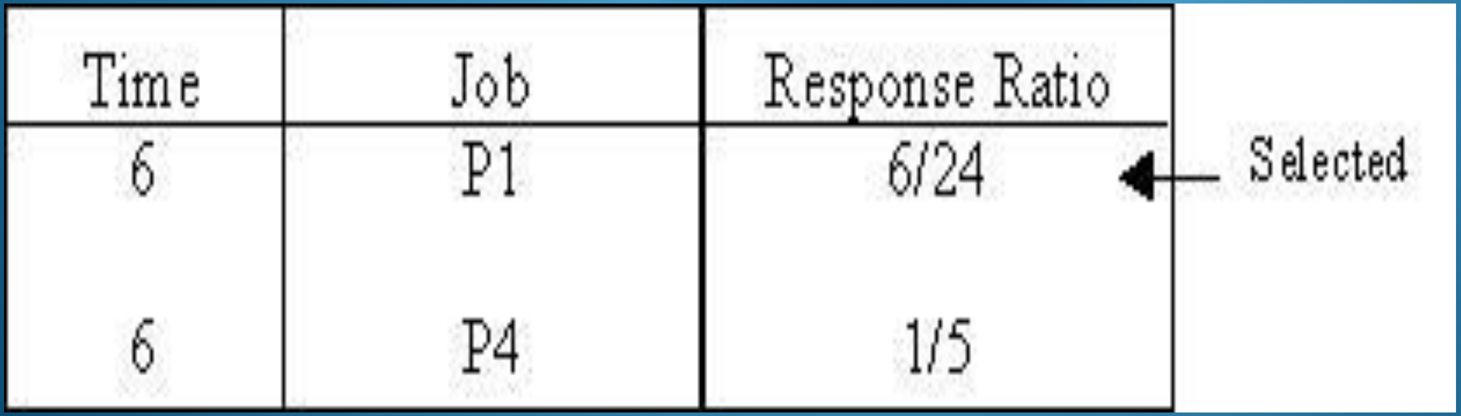
CS240 Operating Systems, Communications and Concurrency

Using HRN we get

At time 6



Arrival	Job	CPU Burst
0	P1	24
0	P2	3
0	P3	3
5	P4	5
10	P5	3

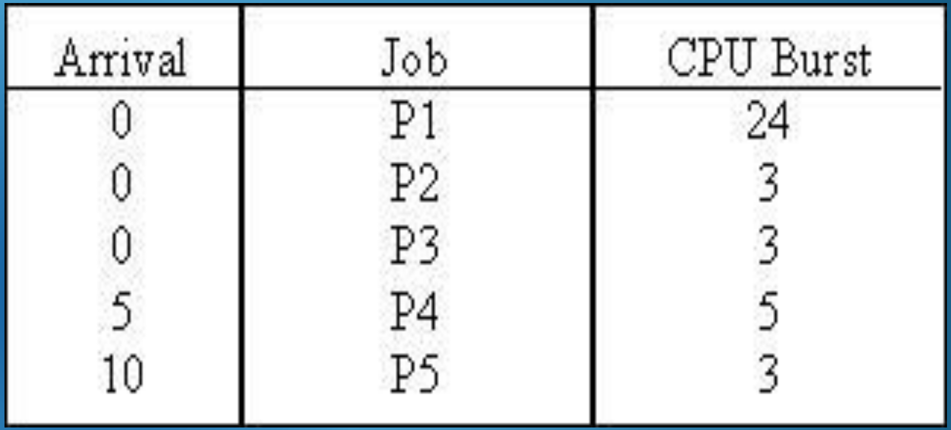


Time	Job	Response Ratio	
6	P1	6/24	← Selected
6	P4	1/5	

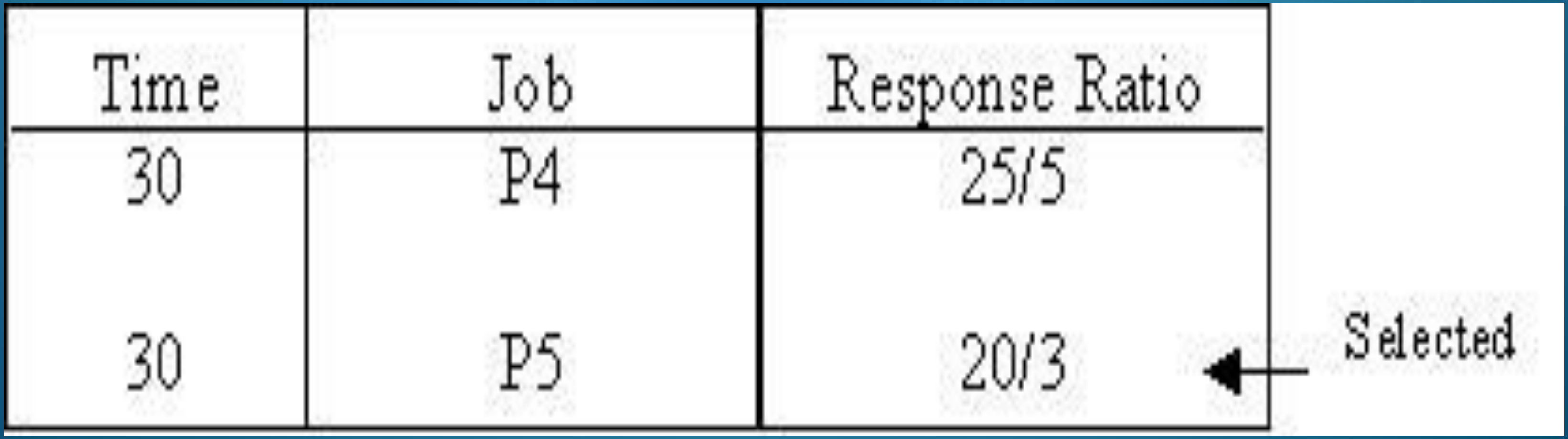
CS240 Operating Systems, Communications and Concurrency

Using HRN we get

At time 30



Arrival	Job	CPU Burst
0	P1	24
0	P2	3
0	P3	3
5	P4	5
10	P5	3

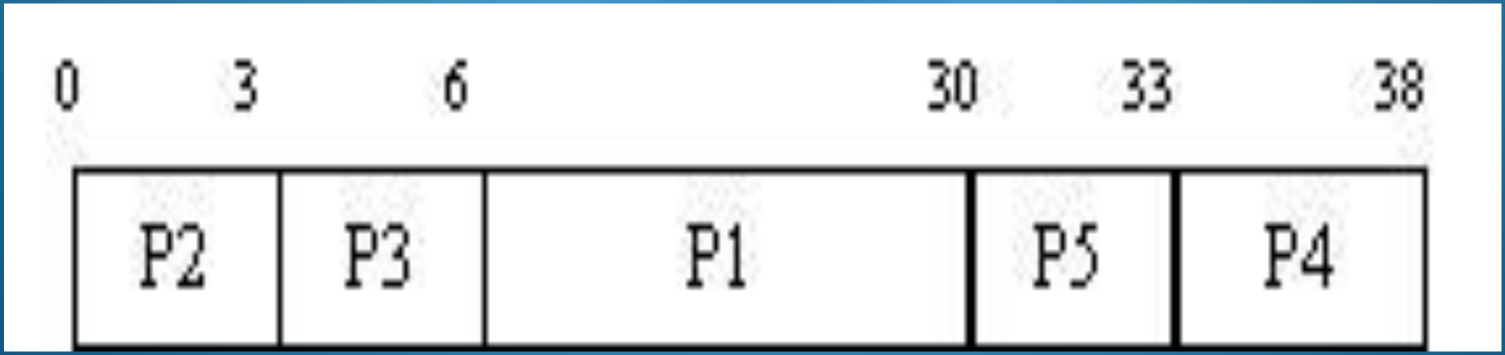


Time	Job	Response Ratio	
30	P4	25/5	
30	P5	20/3	← Selected

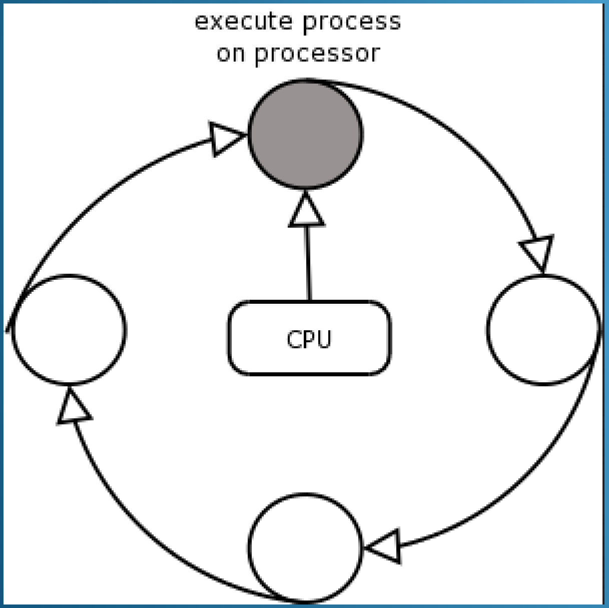
CS240 Operating Systems, Communications and Concurrency

And finally the last remaining job P4 is scheduled using HRN.

Arrival	Job	CPU Burst
0	P1	24
0	P2	3
0	P3	3
5	P4	5
10	P5	3



CS240 Operating Systems, Communications and Concurrency



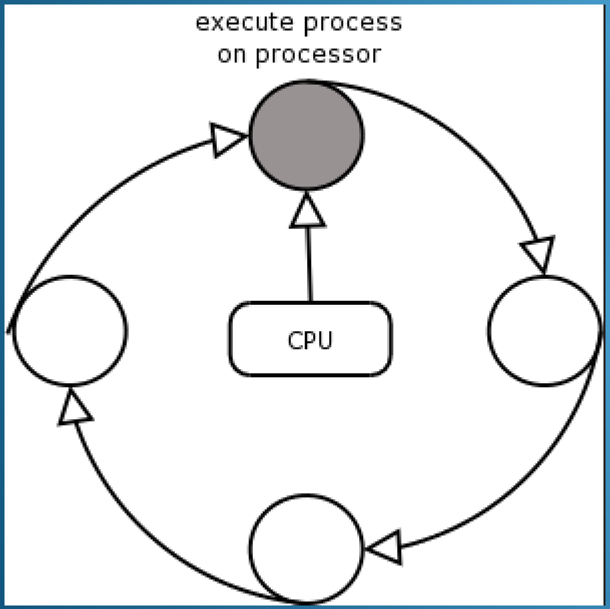
Preemptive Scheduling

A common type of algorithm used as the basis for scheduling in multitasking systems is known as Round Robin.

Round Robin is chosen because it offers good response time to all processes, which is important for achieving satisfactory interactivity performance in multitasking systems.

CS240 Operating Systems, Communications and Concurrency

Preemptive Scheduling



Round Robin is a preemptive version of FCFS.

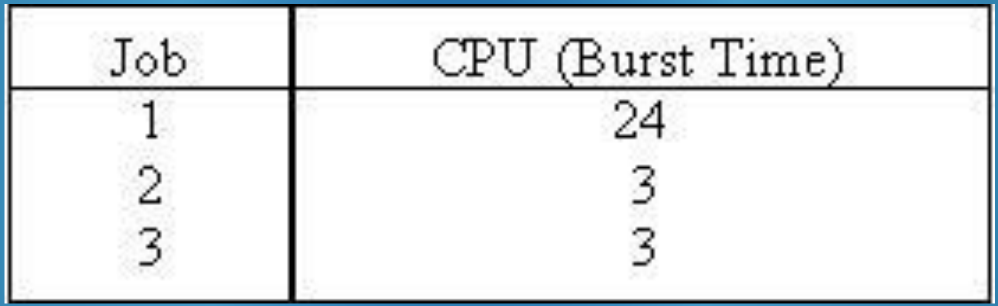
Each process executes in turn until its quantum expires forcing a task context switch.

The quantum can be varied, to give best results for a particular workload.

CS240 Operating Systems, Communications and Concurrency

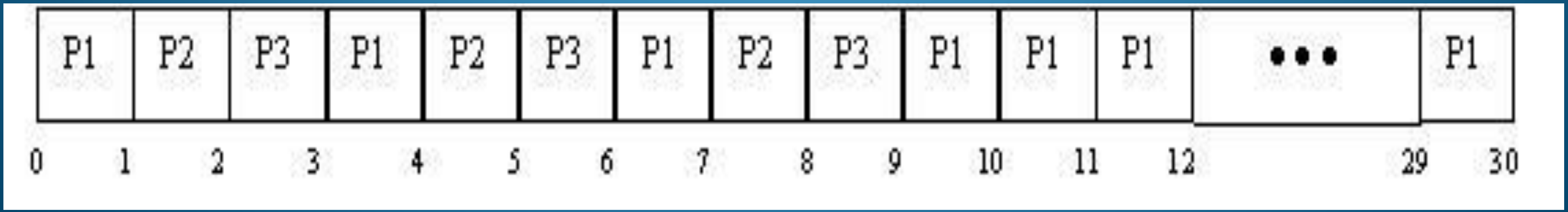
Preemptive Scheduling

Example



Job	CPU (Burst Time)
1	24
2	3
3	3

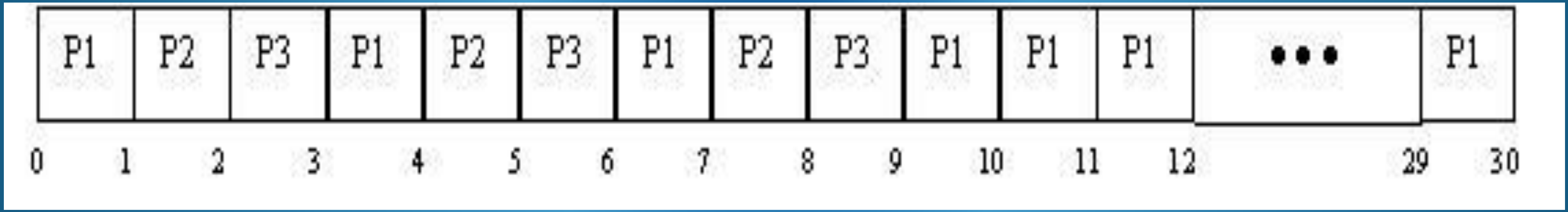
Using Round Robin (RR) with a quantum of 1 we get the following order of execution:-

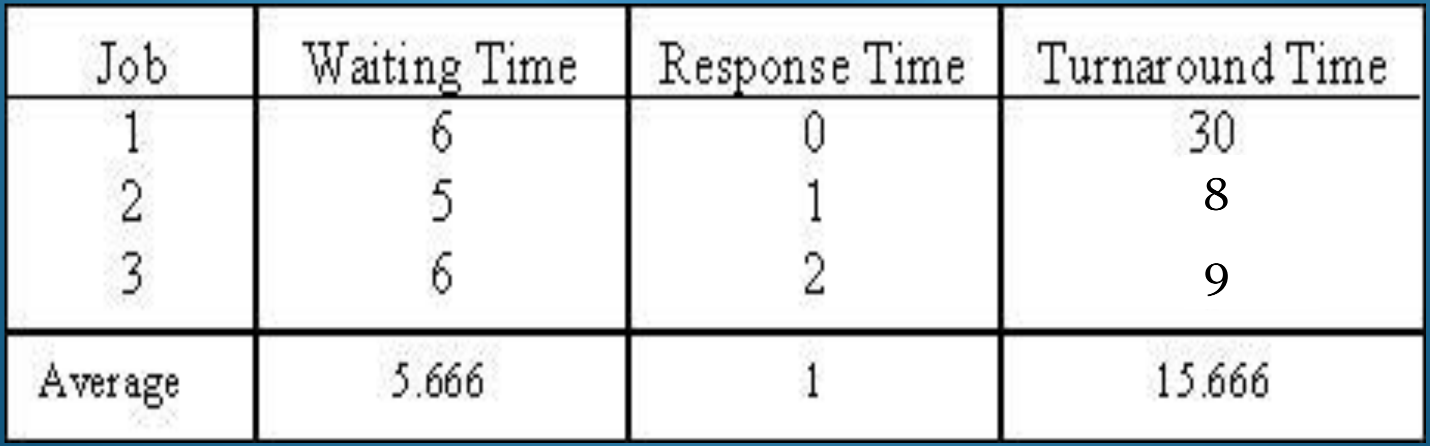


CS240 Operating Systems, Communications and Concurrency

Preemptive Scheduling

Analysing Performance



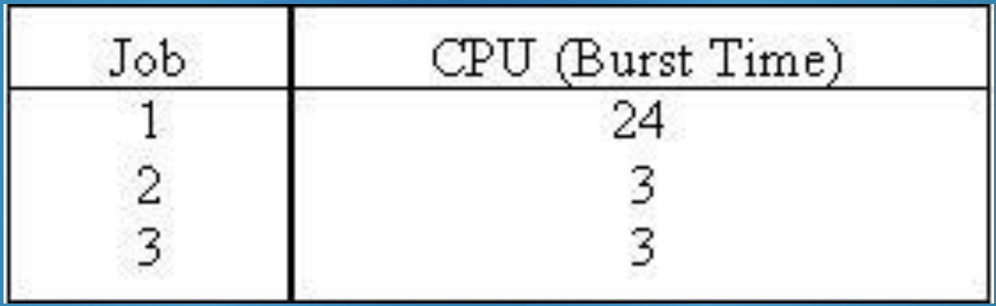


Job	Waiting Time	Response Time	Turnaround Time
1	6	0	30
2	5	1	8
3	6	2	9
Average	5.666	1	15.666

CS240 Operating Systems, Communications and Concurrency

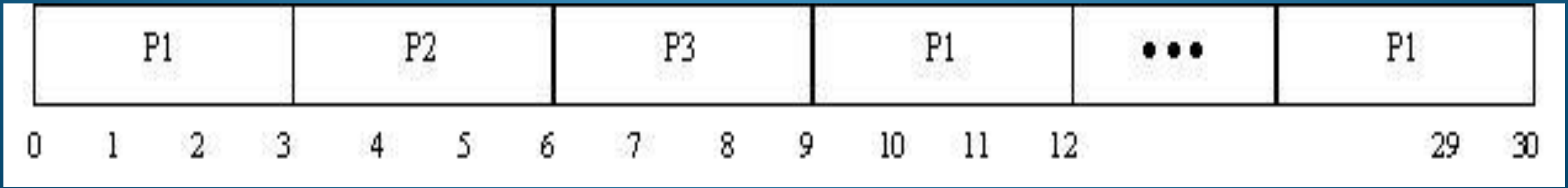
Preemptive Scheduling

Example



Job	CPU (Burst Time)
1	24
2	3
3	3

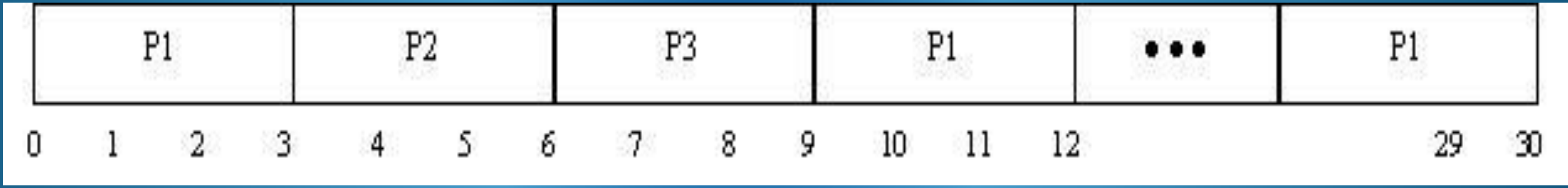
Using Round Robin (RR) with a quantum of 3 we get the following order of execution:-



CS240 Operating Systems, Communications and Concurrency

Preemptive Scheduling

Analysing Performance

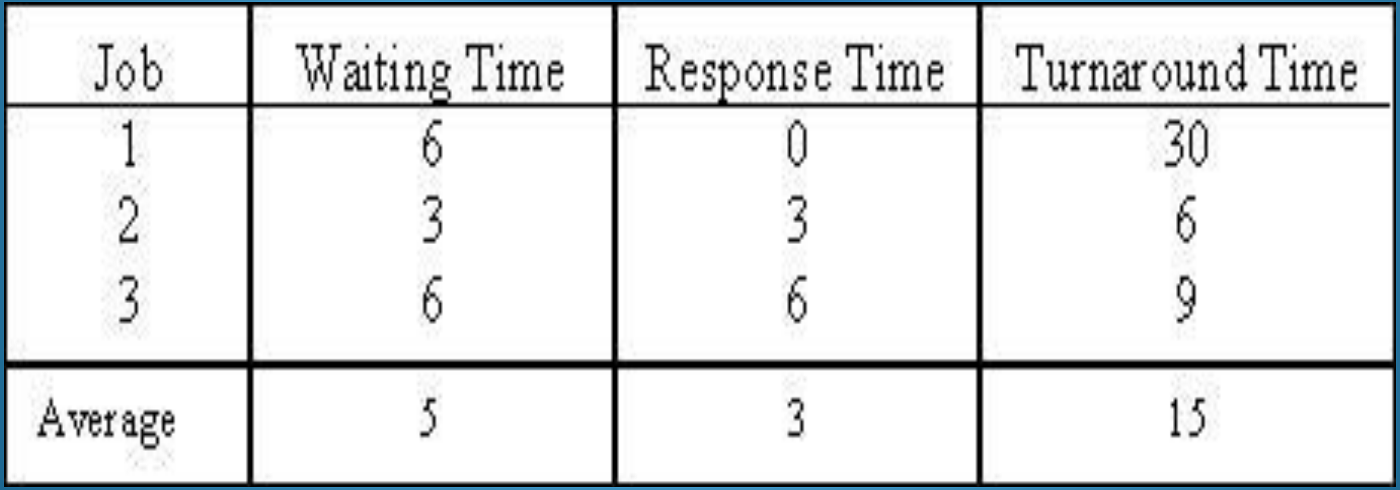


Job	Waiting Time	Response Time	Turnaround Time
1	6	0	30
2	3	3	6
3	6	6	9
Average	5	3	15

CS240 Operating Systems, Communications and Concurrency

Preemptive Scheduling

Analysing Performance



Job	Waiting Time	Response Time	Turnaround Time
1	6	0	30
2	3	3	6
3	6	6	9
Average	5	3	15

Matching the quantum to the average length of the CPU burst has improved some performance criteria.

CS240 Operating Systems, Communications and Concurrency

Preemptive Scheduling

Note that the round robin algorithm incurs a greater number of task switches than non preemptive algorithms.

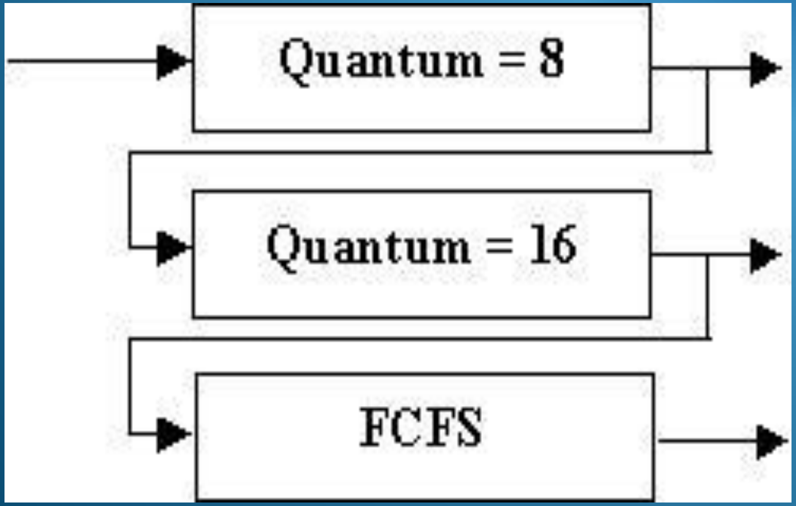
Each task switch takes a certain amount of time for the CPU to change the process environment.

Too many task switches (i.e. quantum too small) means a greater proportion of CPU time is spent doing task switches instead of useful work. If the quantum is too large than RR degenerates to FCFS with poor response times.

CS240 Operating Systems, Communications and Concurrency

Other Scheduling Techniques

When choosing the quantum for round robin scheduling, it is difficult to suit all jobs. If there is a wide deviation in average CPU burst times then the multilevel queue approach, with feedback may be adopted. This adaptive approach reduces task switching overhead in the long term.



A process is initially submitted to the top level highest priority queue where it benefits from good response time. If its behaviour over time shows that it is computationally demanding then it can be demoted to a lower priority queue where it will incur less task switches due to more suitable CPU quantum length but the queue may receive less attention from the processor .

CS240 Operating Systems, Communications and Concurrency

Priority Scheduling

Round Robin doesn’t allow the user to tell the system which tasks are more important than others. In a priority scheduling system, processes are assigned a numeric value which indicates their scheduling priority. A user can indicate the priority to assign to a task over others.

Processes with the highest priority are always chosen first by the scheduler.

It is an easy scheme to implement but requires a means of calculating priorities so that lower priority tasks do not starve.

CS240 Operating Systems, Communications and Concurrency

Case Study – The Traditional Unix Scheduler

Designed to support a time sharing multitasking interactive environment on a single processor for a single user.

Not originally designed for real time process requirements(scheduling tasks within time constraints), multiuser or for symmetric multiprocessing.

Modern Unix implementations since about 2003 has been revamped to cater for these requirements.

CS240 Operating Systems, Communications and Concurrency

Case Study – The Traditional Unix Scheduler

Provides good response time for interactive user tasks while ensuring low priority background tasks don't starve and also that high priority system tasks can be done quickly.

Multilevel feedback approach

Priority queues which are serviced using round robin.

Priority is a value between 0 and 127. The lower the number, the higher the priority.

Priorities 0 to 49 were for Kernel processes, and priorities from 50 to 127 for user processes.

CS240 Operating Systems, Communications and Concurrency

Case Study – The Traditional Unix Scheduler

The priority of all processes system wide was calculated at one second intervals from their execution history and a base priority level used to place processes in bands of priority.

$$Pj(i) = Basej + CPUj(i)/2 + nicej$$

i represents the ith interval of interest, j represents the process id.

User processes are preempted after 1 second if still using the CPU and the highest priority task is chosen next.

CS240 Operating Systems, Communications and Concurrency

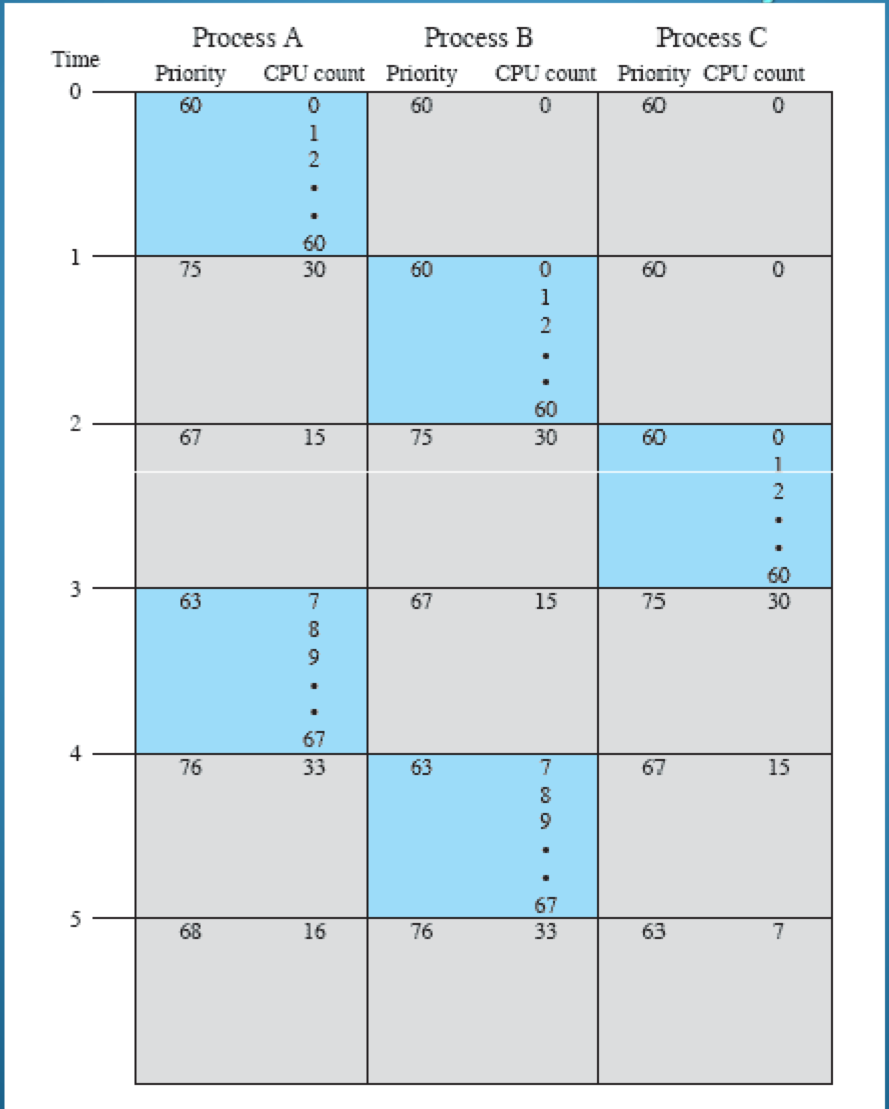
Case Study – The Traditional Unix Scheduler

Recent CPU usage reduced a process's scheduling priority by increasing its priority value.

To ensure processes are eventually rescheduled, the recorded CPU utilization of a process is decayed during each priority recalculation interval using the following formula:-

$$CPUj(i) = CPUj(i-1)/2$$

CS240 Operating Systems, Communications and Concurrency



Time	Process A		Process B		Process C	
	Priority	CPU count	Priority	CPU count	Priority	CPU count
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

In this example, the clock interrupts the system 60 times a second and updates the CPU usage of the currently running process.

How to deal with multiple users?

In multiuser multitasking systems each user may be running a collection of their own tasks.

It is possible that one user or application may be running significantly less processes than another.

Our scheduling algorithms so far focus only on achieving fair allocation among the total set of processes, not on achieving fair allocation of CPU time among different users or different applications.

CS240 Operating Systems, Communications and Concurrency

Dealing with multiple users - Fair-Share Scheduling

The scheme would require some alterations to the process priority calculation to assign a percentage of processor time to each group of processes. For example if there were k separate groups with different share weightings Wk per group.

Pj(i) = Basej + CPUj(i)/2 + GCPUk(i) / 4xWk

(where Wk is a weighting of CPU time assigned to group k such that 0 < Wk <=1 and the sum of all Wk = 1.)

And we could decay GCPU in the same way

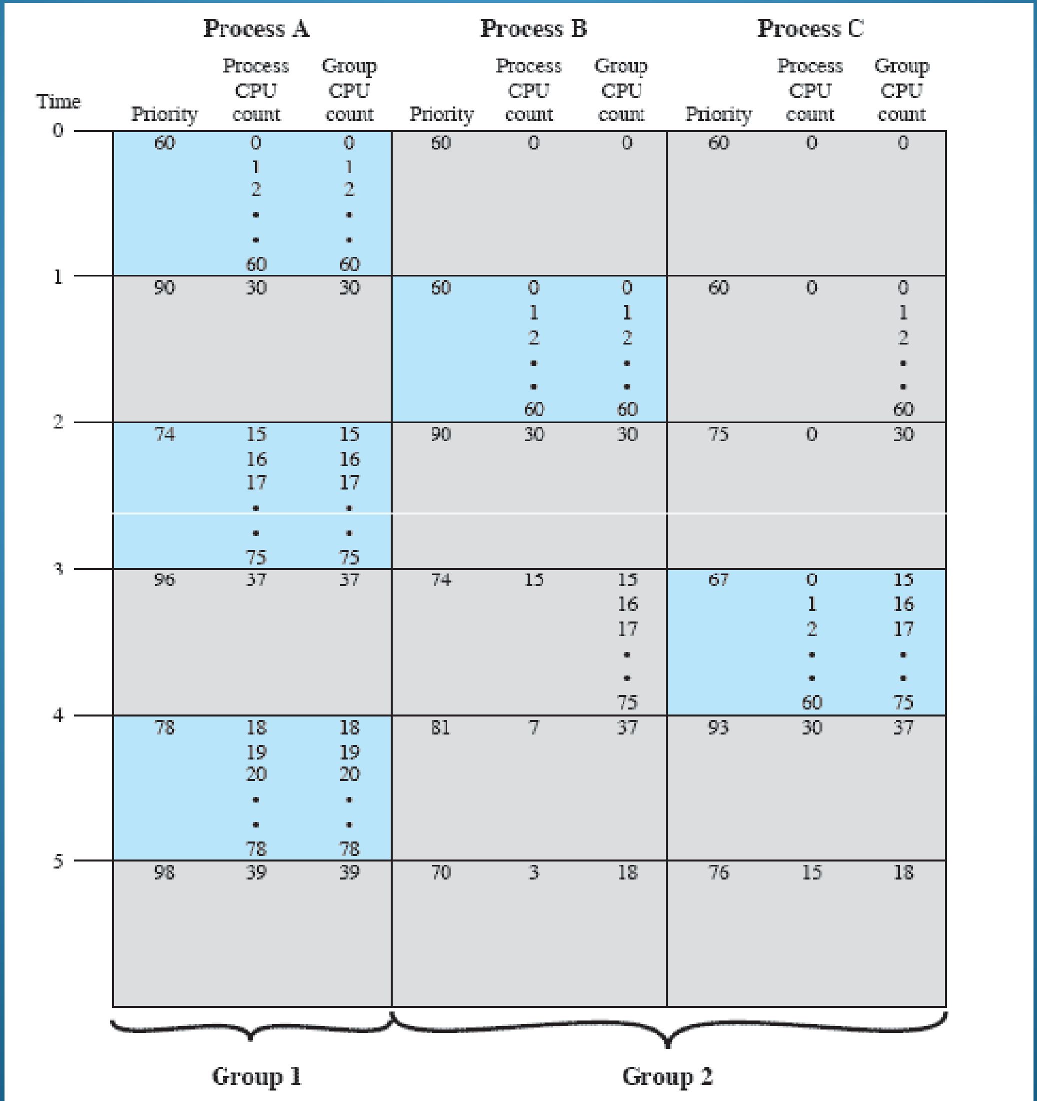
GCPUk(i) = GCPUk(i-1)/2

CS240 Operating Systems, Communications and Concurrency

Fair Share Scheduling

Process A in group one. Processes B and C in group 2.

Each group having a weighting of 0.5, i.e. equal time share.



	Group 1			Group 2					
	Process A			Process B			Process C		
Time	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count	Priority	Process CPU count	Group CPU count
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Pj(i) = Basej + CPUj(i)/2 + GCPUk(i) / 2