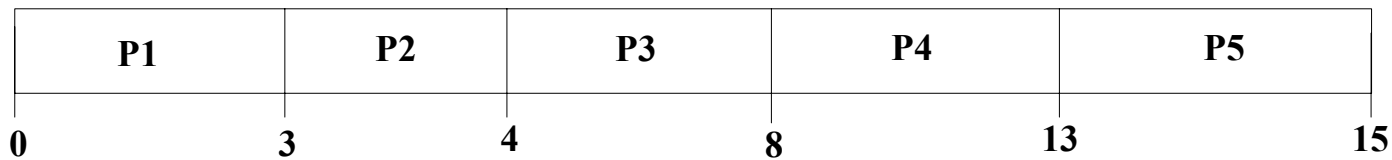


Class Notes Sept 13 2006.

Consider the following example of processes in the ready queue. We assume that all processes entered the ready queue at time 0, in the order given.

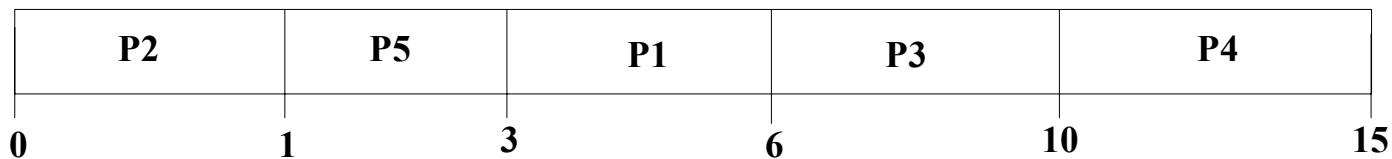
Process	Burst
P1	3
P2	1
P3	4
P4	5
P5	2

Let's use a GANTT chart to compute the average wait time and the average turnaround time for the First Come First Served algorithm.



Process	Burst	Wait	Turnaround
P1	3	0	3
P2	1	3	4
P3	4	4	8
P4	5	8	13
P5	2	13	15
Total		30	43
Average		6	8.6

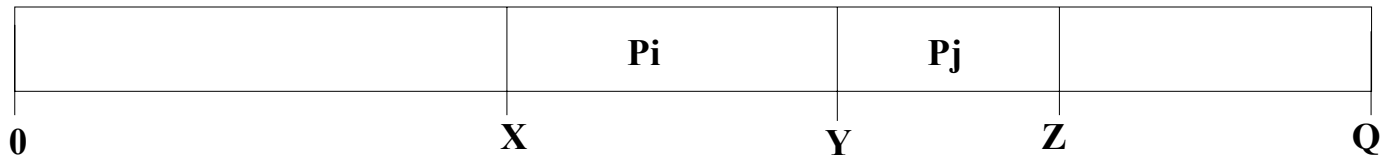
Now let's try Shortest Job First. We rearrange the queue in ascending order by burst time



Process	Burst	Wait	Turnaround
P1	3	3	6
P2	1	0	1
P3	4	6	10
P4	5	10	15
P5	2	1	3
Total		20	35
Average		5	7

Shortest Job First reduces the average wait time and the average turn around time in this case. We can prove that SJF always gives the minimal wait time. Suppose we have the GANTT chart of a schedule that is NOT in SJF order. Suppose P_i and P_j are two consecutive processes in this schedule. If it WERE in SJF order, the burst time of P_i , B_i , would be less than or equal to the burst time of P_j , B_j . $B_i \leq B_j$. Since this schedule is NOT in SJF order, there must be at least one pair of consecutive processes where $B_i > B_j$.

Examine the GANTT chart of this pair of processes:



If we switch P_i and P_j , the number X will not change, nor will anything to the left of it. Similarly the number Z will not change, nor will anything to the right of it. The only number that will change will be Y . $Y = B_i + X$. The new Y , Y' is $Y' = B_j + X$. Since $B_i > B_j$, $Y' < Y$. Therefore, we have reduced the total wait time by $Y - Y'$, which is a positive number.

SJF reduces wait time, but how can we figure out the burst time? In a real-time system, you may have timed every logic path and may know the burst times of each. In this case, you may know ahead of time what the burst time will be. In a more typical OS, there isn't really any way to tell. You can guess. The most common way of guessing is to use the exponential average. Let τ_0 be the initial guess, which is determined by averaging burst times over a long period of time (months). Let τ_n will be our guess for the n th burst, and t_n will be the actual time for the n th burst. We can average the two together using a probability $0 \leq p \leq 1$, like this.

$$\tau_{n+1} = t_n p + \tau_n (1 - p)$$

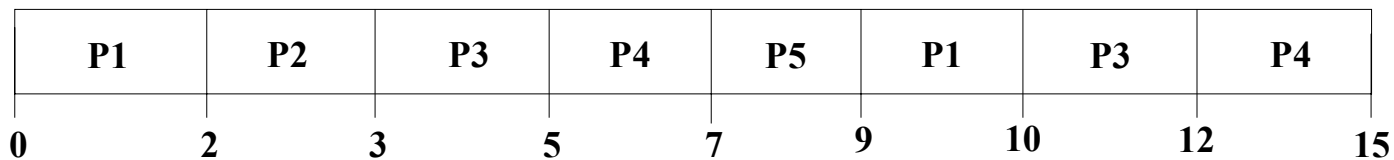
The probability p gives the relative importance of τ_n and t_n . If we substitute this out for the τ_n and end up with a long equation involving just t_n and τ_0 , we will have a term containing $(1-p)^{n+1}$, thus the term *exponential average*. This is never used in practice.

In practice, priority and round-robin are used. Priority gives a list of queues numbered 0-255 (usually) OS's differ in whether 0 is highest priority or 255. The OS searches the queues from high to low priority and schedules the processes from a particular queue in FCFS order. If queue 25 always has a process, then queue 26 processes will starve. It's possible to move processes from one queue to another to alleviate starvation.

Sometimes processes are scheduled in Round-Robin fashion. This means I run a process for a fixed amount of time. If it gets done, fine. If it doesn't, it goes back into the ready queue and the next process in FCFS order is scheduled.

Round Robin can reduce response time below SJF, but usually increases wait and turnaround time.

In round robin, a process is run for a fixed time called a quantum. A timer interrupt is used to enforce the quantum. Let's do a GANTT chart for the above algorithm, assuming a quantum of 2.



Process	Burst	Wait	Resp.	Turn.
P1	3	7	0	10
P2	1	2	2	3
P3	4	8	3	12
P4	5	10	5	15
P5	2	7	7	9
Total		34	17	49
Average		6.8	3.4	9.8

Round robin increases wait and turnaround time over FCFS, but has a much better response time than SJF.