Lecture 7

–Round Robin Scheduling Algorithm

Round Robin (RR) is a machine scheduling algorithm that gives each task a fixed amount of time (called a time quantum or time slice) to execute. If the task doesn't finish in that time, it goes to the back of the queue, and the next task gets its turn.
Round Robin scheduling is preemptive, meaning a running task can be interrupted by another task and sent to the ready queue even if it has not completed its entire execution on the machine. It is a preemptive version of the First Come First Serve (FCFS) scheduling algorithm.

How Round Robin Algorithm Works

- All tasks are placed in a queue.
- The first task gets the machine for a time quantum.
- If the task finishes in that time, it leaves the queue.
- If not, it goes to the back of the queue to wait for another turn.
- This continues in a circular manner until all tasks are completed.

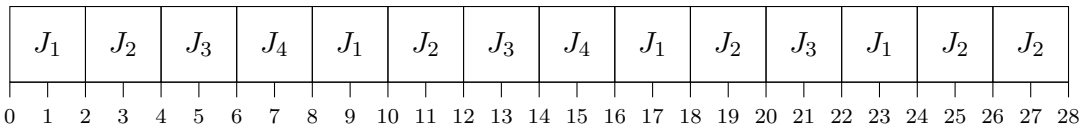
Examples of Round Robin Scheduling Algorithm

Example 1:
Let us consider a system that has four jobs which have arrived at the same time in the order J_1, J_2, J_3 and J_4. The processing time in milliseconds of each job is given by the following table Let

J_i	p_i
J_1	8
J_2	10
J_3	6
J_4	4

us consider time quantum of 2ms and perform RR scheduling on this. We will draw Gantt chart and find the average of the waiting time, completion time and Flowtime.

Gantt Chart with time quantum of 2ms



J_i	C_i	W_i	F_i
J_1	24	6+6+4=16	24
J_2	28	2+6+6+4=18	28
J_3	22	4+6+6=16	22
J_4	16	6+6=12	16
Average	22.5	15.5	22.5

Example 2:
Consider the following table of arrival time and processing time for three jobs J_1, J_2 and J_3 and given time quantum = 2, calculate the average of the waiting time, completion time and Flowtime.

J_i	r_i	p_i
J_1	0	5
J_2	4	2
J_3	5	4

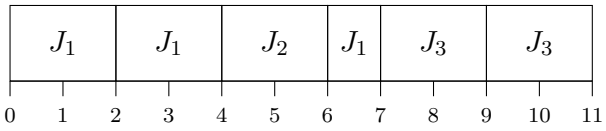
Step-by-Step Execution:

1. Time 0-2 (J_1 Executes):
   - J_1 starts execution as it arrives at 0 ms.
   - Runs for 2 ms; remaining processing time = 5 - 2 = 3 ms
   - Ready Queue: $[J_1]$
2. Time 2-4 (J_1 Executes Again):
   - J_1 continues execution since no other job has arrived yet.
   - Runs for 2 ms; remaining processing time = 3 - 2 = 1 ms.
   - J_2 arrive at 4 ms.
   - Ready Queue: $[J_2, J_1]$.
3. Time 4-6 (J_2 Executes):
   - J_2 starts execution as it arrives at 4 ms.
   - Runs for 2 ms; remaining processing time = 2 - 2 = 0 ms.
   - J_3 arrive at 5ms.

- Ready Queue: $[J_1, J_3]$.

4. Time 6-7 (J_1 Executes):
   - J_1 starts execution.
   - Runs for 1 ms; remaining processing time = 1 - 1 = 0 ms.
   - Ready Queue: $[J_3]$.
5. Time 7-9 (J_3 Executes):
   - J_3 starts execution.
   - Remaining processing time = 4 - 2 = 2 ms.
   - Ready Queue: $[J_3]$.
6. Time 9-11 (J_3 Executes Again):
   - J_3 resumes execution and runs for 2 ms and complete its execution
   - Remaining processing time = 2 - 2 = 0 ms.
   - Ready Queue: [ ].

Gantt Chart with time quantum of 2ms



J_i	C_i	W_i	F_i
J_1	7	2	7
J_2	6	0	2
J_3	11	2	6
Average	8	1.33	5

–Earliest Due Date Scheduling Algorithm (EDD)

This algorithm is used to find an optimal schedule for the problem $n|1||L_{max}$, where

$$L_{max} = \max\{L_i = C_i - d_i | i = 1, \cdots, n\}$$

is the maximum lateness.
Assume that the jobs are numbered in non-decreasing order of their due dates d_i. To minimize the maximum lateness, it is quite natural to process the jobs in this order. This rule is known as EDD (Earliest Due Date).
EDD is non-preemptive algorithm and has assumptions about the tasks(jobs):

- tasks(jobs) have same arrival times
- tasks(jobs) are independent

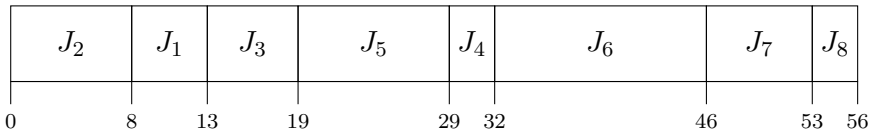
Example :
Let us consider a system that has eight jobs that have arrived at the same time. The processing time and the due date of each job are given in the table below. Using the EDD scheduling algorithm, calculate the average of the completion time, and lateness. Also, find the maximum lateness and the number of tardy jobs.

J_i	p_i	d_i
J_1	5	15
J_2	8	10
J_3	6	15
J_4	3	25
J_5	10	20
J_6	14	40
J_7	7	45
J_8	3	50

Solution:
Using the EDD algorithm, the optimal order to schedule these jobs is

$$2 \to 1 \to 3 \to 5 \to 4 \to 6 \to 7 \to 8$$

Gantt Chart



J_i	C_i	L_i
J_1	13	-2
J_2	8	-2
J_3	19	4
J_4	32	7
J_5	29	9
J_6	46	6
J_7	53	8
J_8	56	6
Average	32	4.5

The maximum lateness is 9 u.t.
The number of tardy jobs is 6 which are $J_3, J_4, J_5, J_6, J_7,$ and J_8.

–Moore's Algorithm

Moore's algorithm is a method for scheduling jobs to minimize the number of tardy jobs (n_T), meaning it is used to find a schedule for the problem $n|1||n_T$.

How does Moore's Algorithm work?

- Step 1: Sequence the jobs with EDD
- Step 2: Find the first tardy job in the current sequence, say job [L].If none exists, go to step 4.
- Step 3: Consider the jobs [1],[2],. . . ,[L]. Reject the job with the largest processing time. Return to step 2.
- Step 4: Form an optimal sequence by taking the current sequence and appending to it the rejected jobs. The jobs appended to the current sequence may be in any order.

Example :
Suppose you have the problem $6|1||n_T$ as below. Find the optimal scheduling.

J_i	1	2	3	4	5	6
d_i	15	6	9	23	20	30
p_i	10	3	4	8	10	6

Solution:
First, we will sequence the jobs with EDD and calculate the completion time for the jobs until the first tardy job (Steps 1 and 2). We stop since we find the first tardy job, which is J_1. The

The current sequence	2	3	1	5	4	6
d_i	6	9	15	20	23	30
p_i	3	4	10	10	8	6
C_i	3	7	**17**			

job which has the largest processing time in the subsequence [2 3 1] is J_1, so it will be rejected (according to step 3). Then return to step 2.

The new current sequence	2	3	5	4	6	rejected jobs
d_i	6	9	20	23	30	1
p_i	3	4	10	8	6	
C_i	3	7	17	**25**		

We stop since we find the first tardy job in the new current sequence, which is J_4. The job which has the largest processing time in the subsequence [2 3 5 4] is J_5, so it will be rejected (according to step 3). Then return to step 2.

The new current sequence	2	3	4	6	rejected jobs
d_i	6	9	23	30	1,5
p_i	3	4	8	6	
C_i	3	7	15	21	

We note that there is no tardy job, so we will go to step 4 and form the optimal sequences, which are

$$2 \to 3 \to 4 \to 6 \to 1 \to 5$$

or

$$2 \to 3 \to 4 \to 6 \to 5 \to 1$$