Rate Monotonic Scheduling Algorithm

Definition :

Rate monotonic scheduling is a priority algorithm that belongs to the static priority scheduling category of Real Time Operating Systems. It is preemptive in nature. The priority is decided according to the cycle time of the processes that are involved. If the process has a small job duration, then it has the highest priority. Thus, if a process with the highest priority starts execution, it will preempt the other running processes.

A set of processes can be scheduled only if they satisfy the following equation.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

Where n is the number of processes in the process set, C_i is the computation time of the process, T_i is the Time period for the process to run and U is the processor utilization.

Example: An example to understand the working of Rate monotonic scheduling algorithm.

processes	Execution time (c)	Time period (T)
P1	3	20
P2	2	5
P3	2	10

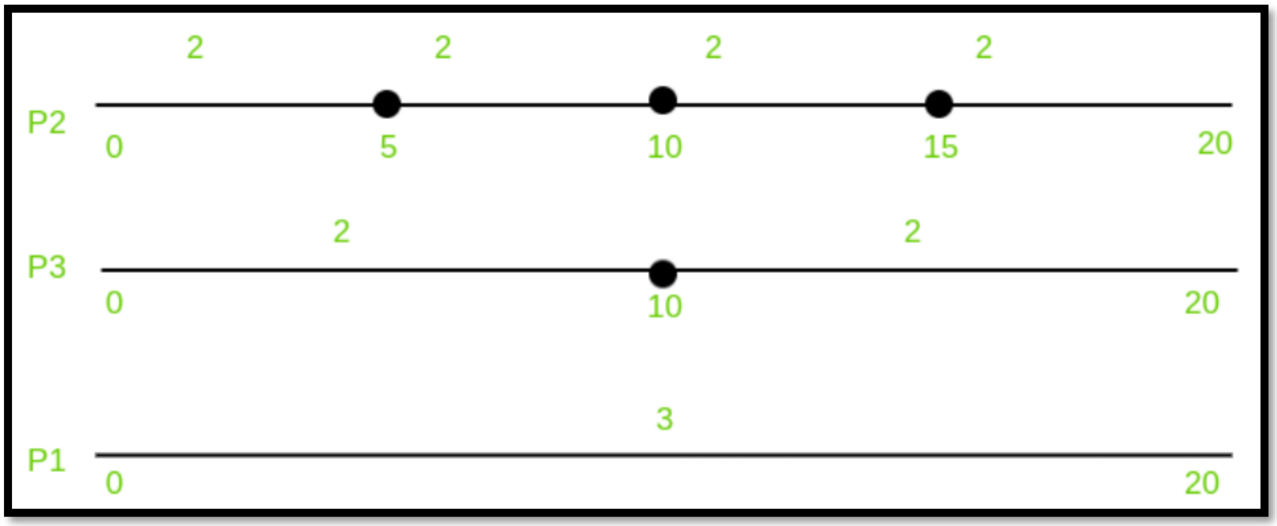
n(2^1/n - 1) = 3 (2^1/3 - 1) = 0.7977

U = 3/20 + 2/5 + 2/10 = 0.75

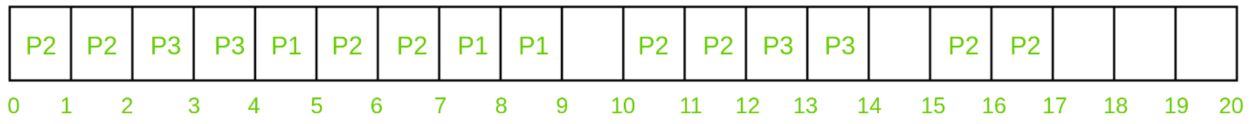
Scheduling time: For calculating the Scheduling time of the algorithm we must take the LCM of the Time period of all the processes. LCM (20, 5, 10) of the above example is 20. Thus, we can schedule it by 20-time units.

Priority: As discussed above, the priority will be the highest for the process which has the least running time period. Thus, P2 will have the highest priority, and after that P3 and lastly P1.

Representation and flow are as follows:



The above figure says that Process P2 will execute two times for every 5-time units, Process P3 will execute two times for every 10-time units and Process P1 will execute three times in 20 time units. This has to be kept in mind for understanding the entire execution of the algorithm below.



Process P2 will run first for 2-time units because it has the highest priority. After completing its two units, P3 will get the chance and thus it will run for 2-time units. As we know that process P2 will run 2 times in the interval of 5-time units and process P3 will run 2 times in the interval of 10-time units, they have fulfilled the criteria and thus now process P1 which has the least priority will get the chance and it will run for 1 time. And here the interval of five-time units has completed. Because of its priority P2 will preempt P1 and thus will run 2 times. As P3 has completed its

2-time units for its interval of 10-time units, P1 will get chance and it will run for the remaining 2 times, completing its execution which was thrice in 20-time units. Now 9-10 interval remains idle as no process needs it. At 10-time units, process P2 will run for 2 times completing its criteria for the third interval (10-15). Process P3 will now run for two times completing its execution. Interval 14-15 will again remain idle for the same reason mentioned above. At 15-time unit, process P2 will execute for two times completing its execution. This is how the rate monotonic scheduling works.

Conditions: The analysis of Rate monotonic scheduling assumes few properties that every process should possess. They are:

1. Processes involved should not share the resources with other processes.
2. Deadlines must be similar to the time periods. Deadlines are deterministic.
3. Process running with highest priority that needs to run, will preempt all the other processes.
4. Priorities must be assigned to all the processes according to the protocol of Rate monotonic scheduling.

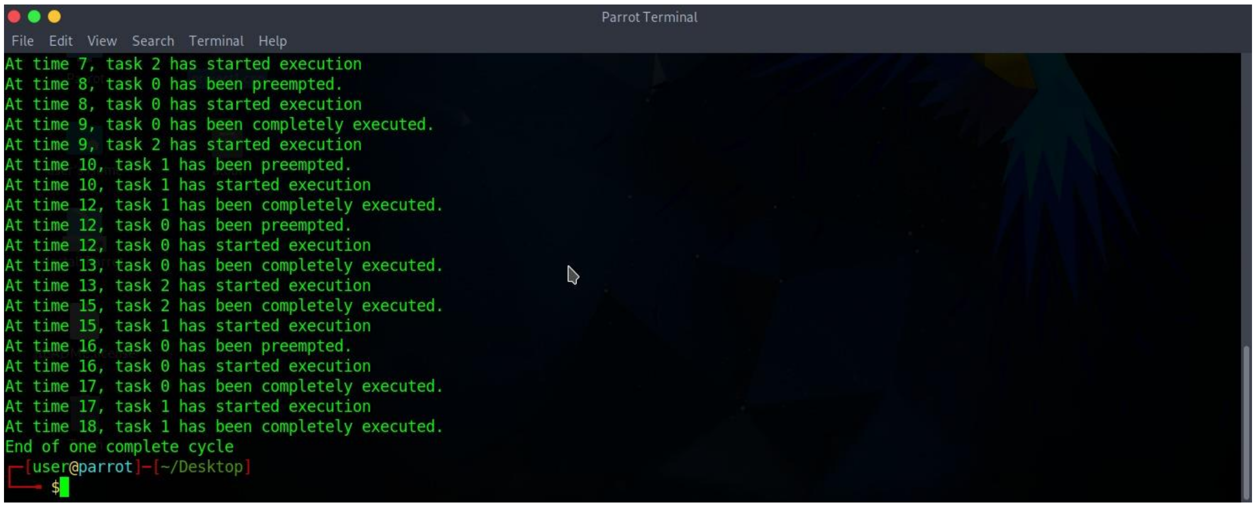
Advantages:

1. It is easy to implement.
2. If any static priority assignment algorithm can meet the deadlines, then rate monotonic scheduling can also do the same. It is optimal.
3. It consists of a calculated copy of the time periods, unlike other time-sharing algorithms such as round robin which neglects the scheduling needs of the processes.

Disadvantages:

1. It is very difficult to support aperiodic and sporadic tasks under RMA.
2. RMA is not optimal when the task period and deadline differ.

Output:



```
At time 7, task 2 has started execution
At time 8, task 0 has been preempted.
At time 8, task 0 has started execution
At time 9, task 0 has been completely executed.
At time 9, task 2 has started execution
At time 10, task 1 has been preempted.
At time 10, task 1 has started execution
At time 12, task 1 has been completely executed.
At time 12, task 0 has been preempted.
At time 12, task 0 has started execution
At time 13, task 0 has been completely executed.
At time 13, task 2 has started execution
At time 15, task 2 has been completely executed.
At time 15, task 1 has started execution
At time 16, task 0 has been preempted.
At time 16, task 0 has started execution
At time 17, task 0 has been completely executed.
At time 17, task 1 has started execution
At time 18, task 1 has been completely executed.
End of one complete cycle
┌─[user@parrot]─[~/Desktop]
└──╼ $
```

Conclusion

In conclusion, Rate Monotonic Scheduling (RMS) provides a straightforward and efficient approach to scheduling tasks in real-time systems with fixed priorities and deadlines. Its optimality for periodic tasks makes it a valuable tool for ensuring timely execution and meeting critical deadlines. However, its assumptions of task independence and static priorities might limit its applicability in more complex and dynamic environments. Thus, while RMS offers predictability and simplicity, it's essential to consider its suitability alongside the specific requirements and constraints of the system at hand.