UNIT – 3
Priority-Driven Scheduling of Periodic Tasks

1. What are fixed-priority versus dynamic-priority algorithms?

   Priority-driven algorithms differ from each other in how priorities are assigned to jobs. We classify algorithms for scheduling periodic tasks into two types: ***fixed priority*** and ***dynamic priority***.

   A ***fixed-priority*** **algorithm** assigns the same priority to all the jobs in each task. In other words, the priority of each periodic task is fixed relative to other tasks.

   In contrast, a ***dynamic-priority*** **algorithm** assigns different priorities to the individual jobs in each task. Hence the priority of the task with respect to that of the other tasks changes as jobs are released and completed. This is why this type of algorithm is said to be "dynamic."

2. Mention how RM algorithm works and Define rate.
   A well-known fixed-priority algorithm is the ***rate-monotonic*** **algorithm**. This algorithm assigns priorities to tasks based on their periods: the shorter the period, the higher the priority. The ***rate*** (of job releases) of a task is the inverse of its period. Hence, the higher its rate, the higher its priority.

3. Mention how DM algorithm works

   Another well-known fixed-priority algorithm is the ***deadline-monotonic algorithm***, called the **DM algorithm** hereafter. This algorithm assigns priorities to tasks according their relative deadlines: the shorter the relative deadline, the higher the priority.

4. What is Schedulable utilization?
   A criterion we will use to measure the performance of algorithms used to schedule periodic tasks is the schedulable utilization. The ***schedulable utilization*** of a scheduling algorithm is defined as follows: *A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if the total utilization of the tasks is equal to or less than the schedulable utilization of the algorithm*.

5. Give the schedulability test for an EDF algorithm.
   A test for the purpose of validating that the given application system can indeed meet all its hard deadlines when scheduled according to the chosen scheduling algorithm is called a ***schedulability test***. If a schedulability test is efficient, it can be used as an on-line acceptance test.

   Checking whether a set of periodic tasks meet all their deadlines is a special case of the validation problem that can be stated as follows: We are given

1. the period p_i , execution time e_i , and relative deadline D_i of every task T_i in a system **T** = $\{T_1, T_2, \ldots T_n\}$ of independent periodic tasks, and
2. a priority-driven algorithm used to schedule the tasks in **T** preemptively on one processor.

To determine whether the given system of n independent periodic tasks surely meets all the deadlines when scheduled according to the preemptive EDF algorithm on one processor, we check whether the inequality

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \leq 1$$

is satisfied. We call this inequality the ***schedulability condition*** of the EDF algorithm. If it is satisfied, the system is schedulable according to the EDF algorithm.

6. When is an RM algorithm optimal and What is a simply periodic system?
   While the ***RM algorithm*** is not optimal for tasks with arbitrary periods, it ***is optimal in the special case when the periodic tasks in the system are simply periodic and the deadlines of the tasks are no less than their respective periods.***

   A system of periodic tasks is ***simply periodic*** if for every pair of tasks T_i and T_k in the system and $p_i < p_k$, p_k is an integer multiple of p_i . An example of a simply periodic task system is the flight control system in which the shortest period is 1/180 seconds. The other two periods are two times and six times 1/180 seconds.

7. What is a Critical Instant?
   A ***critical instant*** of a task T_i is a time instant which is such that

1. the job in T_i released at the instant has the maximum response time of all jobs in T_i, if the response time of every job in T_i is equal to or less than the relative deadline D_i of T_i, and
2. the response time of the job released at the instant is greater than D_i if the response time of some jobs in *Ti* exceeds *Di* .

   We call the response time of a job in T_i released at a critical instant the ***maximum (possible) response time*** of the task and denote it by W_i .

$$W_{i,1} = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{W_{i,1} + \phi_i - \phi_k}{p_k} \right\rceil e_k - \phi_i$$

8. What is Time-Demand Analysis?
   To determine whether a task can meet all its deadlines, we first compute the total demand for processor time by a job released at a critical instant of the task and by all the higher-priority tasks as a function of time from the critical instant. We then check whether this demand can be met before the deadline of the job. For this reason, we name this test a ***time-demand analysis*.**

   To carry out the time-demand analysis on **T**, we consider one task at a time, starting from the task T_1 with the highest priority in order of decreasing priority. To determine whether a task T_i is schedulable after finding that all the tasks with higher priorities are schedulable, we focus on a job in T_i , supposing that the release time t_0 of the job is a critical instant of T_i . At time $t_0 + t$ for $t \geq 0$, the total (processor) time demand $w_i(t)$ of this job and all the higher-priority jobs released in $[t_0, t]$ is given by

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \qquad \text{for } 0 < t \leq p_i$$

   We call $w_i(t)$ the ***time-demand function*** of the task T_i .

9. What is Worst case simulation method?
   Instead of carrying out a time-demand analysis, we can also determine whether a system of independent preemptable tasks is schedulable by simply simulating this condition and observing whether the system is then schedulable. It suffices for us to construct only the initial segment of length equal to the largest period of the tasks. If there is no missed deadline in this segment, then all tasks are feasibly scheduled by the given algorithm.

This schedule allows us to draw the same conclusion about the maximum response times of the four tasks. We refer to this alternative as the ***worst-case simulation method***. It is easy to see that the time complexity of this method is also $O(nq_n,1)$, where $q_n,1$ is the ratio p_n/p_1.

For now, it appears that the more conceptually complex time-demand analysis method has no advantage over the simulation method. If we want to know the maximum possible response time W_i of each task T_i , we must solve the below equation in an iterative manner, starting from an initial guess $t^{(1)}$ of W_i . During the lth iteration for $l \geq 1$, we compute the value $t^{(l+1)}$ on the according to

$$t^{(l+1)} = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t^{(l)}}{p_k} \right\rceil e_k$$

We terminate the iteration either when $t^{(l+1)}$ is equal to $t^{(l)}$.

10. What is a Busy Intervals?
    We will use the term level-π_i busy interval. A ***level-π_i busy interval*** $(t_0, t]$ begins at an instant t_0 when
    (1) all jobs in $\mathbf{T}_i$ released before the instant have completed and
    (2) a job in $\mathbf{T}_i$ is released.

    The interval ends at the first instant t after t_0 when all the jobs in $\mathbf{T}_i$ released since t_0 are complete.

11. List out the General Time-Demand Analysis Method.
    Test one task at a time starting from the highest priority task T_1 in order of decreasing priority. For the purpose of determining whether a task T_i is schedulable, assume that all the tasks are in phase and the first level-π_i busy interval begins at time 0. While testing whether all the jobs in T_i can meet their deadlines (i.e., whether Ti is schedulable), consider the subset $\mathbf{T}_i$ of tasks with priorities π_i or higher.

    (i) If the first job of every task in $\mathbf{T}_i$ completes by the end of the first period of the task, check whether the first job $J_{i,1}$ in T_i meets its deadline. T_i is schedulable if $J_{i,1}$ completes in time. Otherwise, T_i is not schedulable.
    (ii) If the first job of some task in $\mathbf{T}_i$ does not complete by the end of the first period of the task, do the following:
    (a) Compute the length of the in phase level-π_i busy interval by solving the equation $t = \sum_{k=1}^{i} \lceil \frac{t}{p_k} \rceil e_k$ iteratively, starting from $t^{(1)} = \sum_{k=1}^{i} e_k$ until $t^{(l+1)} = t^{(l)}$ for some $l \geq 1$. The solution $t^{(l)}$ is the length of the level-π_i busy interval.
    (b) Compute the maximum response times of all $\lceil t^{(l)}/p_i \rceil$ jobs of T_i in the in-phase level-π_i busy interval in the manner described below and determine whether they complete in time.
    T_i is schedulable if all these jobs complete in time; otherwise T_i is not schedulable.

12. When is a system difficult to schedule?
    We say that "**a system is *difficult to schedule*" if** it is schedulable according to the RM algorithm, but it fully utilizes the processor for some interval of time so that any increase in the execution time or decrease in the period of some task will make the system unschedulable.

13. Define Adjacent Period Ratio?
    The ratio of the larger period p_k to the smaller period p_i , that is, $q_{k,i} = p_k/p_i$ is called *adjacent period ratio.*

14. What are peak frames and normal frames?
    Specifically, ***in the multiframe task model***, each (multiframe) task T_i is characterized by a 4-tuple $(p_i, \xi_i , e_i^p, e_i^n)$. In the 4-tuple, p_i is the period of the task and has the same meaning as the period of a periodic task. Jobs in T_i have either one of two possible maximum execution times: e_i^p and e_i^n, where $e_i^p \geq e_i^n$.

The former is its *peak execution time*, and the latter is its ***normal execution time*.** Each period which begins at the release time of a job with the peak execution time is called a ***peak frame***, and the other periods are called ***normal frames*.** Each peak frame is followed by $\xi_i - 1$ normal frames, which in turn are followed by a peak frame and so on.

15. What is blocking and priority inversion?
    A (ready) job J_i is ***blocked*** when it is prevented from executing by a lower-priority job: The lower-priority job executes while J_i waits. We say that a ***priority inversion*** occurs whenever a lower-priority job executes while some ready higher-priority job waits.

16. What is Self-Suspension?
    Self-blocking or ***self-suspension*** *of a job* occurs when the job is suspended and waits until such an operation completes before its execution can continue. While it waits, the operating system removes it from the ready queue and places it in a blocked queue. We assume that the maximum amount of time each external operation takes to complete and, hence, the maximum duration of each self-suspension, is known.

    A job may self-suspend after its execution has begun, and the amounts of time for which jobs in a task self-suspend may differ. As a consequence, the task no longer behaves as a periodic task. In particular, it may demand more time in some interval than a periodic task.

17. What are the different types of Mappings?
    In Fixed priority systems, we map the assigned priorities onto the priority grid so that all the assigned priorities equal to or higher than π_1 are mapped to π_1 and all assigned priorities in the range $(\pi_{k-1}, \pi_k]$ are mapped to the system priority π_k for $1 < k \leq \_s$.

    A natural mapping is the **uniform mapping**. According to this method, the priority grid is uniformly distributed in the range of the assigned priorities. Specifically, let Q denote the integer

$$\lfloor \Omega_n / \Omega_s \rfloor. \ \pi_k = kQ \text{ for } k = 1, 2, \ldots, \Omega_s - 1 \text{ and } \pi_{\Omega_s} \text{ is equal to } \Omega_n.$$

    Hence, the highest Q assigned priorities 1, 2, . . . , Q are mapped to $\pi_1 = Q$, the next Q highest assigned priorities are mapped to $\pi_2 = 2Q$, and so on.

    A better method is the ***constant ratio mapping***. This method keeps the ratios of $(\pi_{i-1}+1)/\pi_i$, for $i = 2, 3, \ldots, \_s$ as equal as possible. Consequently, there are more system priority levels at the higher-priority end of the assigned priority range than at the lower-priority end.

18. Define Grid ratio.
    The **grid ratio *g*** is the minimum of the ratios $D_{i,\min}/D_{i,\max}$ among all Ω_s distinct relative deadline grid points. A sufficient schedulability condition for the on-time completion of all jobs is that the total density of all periodic tasks be no greater than g.

19. What is Tick Scheduling?
    An important assumption underlying all the schedulability tests described above is that the scheduler is event-driven. This assumption is sometimes not valid. A way to implement the scheduler is to make it time-driven. By this we mean that the execution of the scheduler is triggered by a timer which is set to expire periodically. Scheduling decisions that are made at these time instants are called ***clock interrupts***. This method is called ***tick scheduling*** **or time-based scheduling**.

20. List the factors considered in tick scheduling for schedulability analysis.
    Tick scheduling introduces two additional factors that must be accounted for in schedulability analysis.
    First, the fact that a job is ready may not be noticed and acted upon by the scheduler until the next clock interrupt. The delayed action of the scheduler may delay the completion of the job.

Second, a ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue.

21. What is – Subtask, Canonical form and Interference Block

    Let $n(i)$ denote the number of segments in each job of a periodic task T_i . These segments have different priorities. In a fixed-priority system, the corresponding segments of all the jobs in the same task have the same priority. It is convenient to think of each such task T_i as composed of $n(i)$ as ***subtasks*** $T_{i,1}, T_{i,2}, \ldots, T_{i,n(i)}$. A job in $T_{i,k}$ is the kth segment of a job in T_i .

    A task is in ***canonical form*** if every later subtask has a higher priority than its immediate predecessor subtask. A task that is not in canonical form can be transformed into canonical form.

    An ***interference block*** of a subtask $T_{i,k}$ is a chain of one or more contiguous subtasks $T_{l,x}, T_{l,x+1}, \ldots, T_{l,y}$ in another task T_l , for some $l != i$ and $y \geq x$, that have the following properties.

    (1) All of these subtasks have equal or higher priorities than the priority $\pi_{i,k}$ of $T_{i,k}$;
    (2) either $T_{l,x}$ has no predecessor or the priorities of its immediate predecessor are lower than $\pi_{i,k}$; and
    (3) either $T_{l,y}$ has no successor or the priority of its immediate successor is lower than $\pi_{i,k}$.

22. How to convert a task into Canonical form?

    A task that is not in canonical form can be transformed into canonical form in the following manner. Starting with the last subtask, we examine the subtasks of the task in turn in reverse precedence order. We lower the priority of each immediate predecessor to the priority of the immediate successor if the given priority of the immediate predecessor is higher; otherwise, we leave the priority of the immediate predecessor unchanged. We repeat this process until the priority of the first subtask is thus determined. If in this process the priorities of adjacent subtasks become the same, we combine these subtasks into one subtask and decrement the number of subtasks accordingly.