UNIT 2
Clock-Driven Scheduling

1. Mention the assumptions and notations in a Clock-Driven Approach.
   The clock-driven approach to scheduling is applicable only when the system is by and large deterministic. For this reason, we assume a restricted periodic task model. The following are the restrictive assumptions-

   - There are *n* periodic tasks in the system. As long as the system stays in an operation mode, *n* is fixed.
   - The parameters of all periodic tasks are known a priori. In particular, variations in the interrelease times of jobs in any periodic task are negligibly small.
   - In other words, for all practical purposes, each job in T_i is released *pi* units of time after the previous job in T_i .
   - Each job $J_{i,k}$ is ready for execution at its release time $r_{i,k}$.

   We refer to a periodic task T_i with phase φ_i , period p_i , execution time e_i , and relative deadline D_i by the 4-tuple $(\varphi_i , p_i , e_i , D_i)$. For example, *(*1*,* 10*,* 3*,* 6*)* is a periodic task whose phase is 1, period is 10, execution time is 3, and relative deadline is 6.

2. Write about static, timer-driven scheduler.
   The operating system maintains a queue for aperiodic jobs. When an aperiodic job is released, it is placed in the queue without the attention of the scheduler. Whenever the processor is available for aperiodic jobs, the job at the head of this queue executes.

   Whenever the parameters of jobs with hard deadlines are known before the system begins to execute, a straightforward way to ensure that they meet their deadlines is to construct a *static schedule* of the jobs off-line.
   At *decision time* t_k, which is an instant when a scheduling decision is made

3. What is a cyclic schedule?
   We call a periodic static schedule a *cyclic schedule*. Again, this approach to scheduling hard real-time jobs is called the *clock-driven* or *time-driven* approach because each scheduling decision is made at a specific time, independent of events.

4. What are Frames and Major Cycles?
   The below figure shows a good structure of cyclic schedules. A restriction imposed by this structure is that scheduling decisions are made periodically, rather than at arbitrary times. The scheduling decision times partition the time line into intervals called *frames*.

   Every frame has length *f* ; *f* is the *frame size*. Because scheduling decisions are made only at the beginning of every frame, there is no preemption within each frame. The phase of each periodic task is a nonnegative integer multiple of the frame size.

5. What are the Frame Size Constraints, Hyperperiod?
   Ideally, we want the frames to be sufficiently long so that every job can start and complete its execution within a frame. In this way, no job will be preempted. We can meet this objective if we make the frame size *f* larger than the execution time e_i of every task T_i . In other words,

   $$f \geq \max_{1 \leq i \leq n} (e_i) \qquad (5.1)$$

   To keep the length of the cyclic schedule as short as possible, the frame size *f* should be chosen so that it divides *H*, the length of the hyperperiod of the system. This condition is met when *f* divides the period *pi* of at least one task T_i , that is,

   $$\lfloor p_i/f \rfloor - p_i/f = 0 \qquad (5.2)$$

We let *F* denote this number and call a ***Hyperperiod*** that begins at the beginning of the *(kF* + 1*)*st frame, for any *k* = 0, 1*, . . . ,* a *major cycle*.

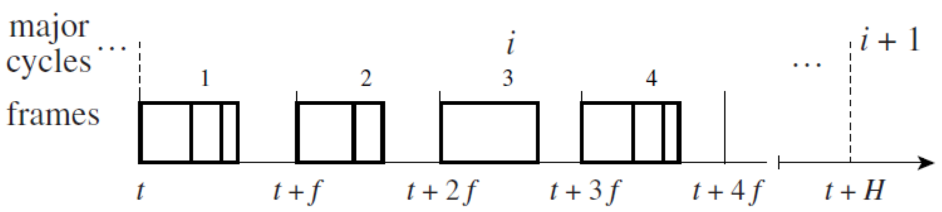

FIGURE 5–3 General structure of a cyclic schedule.

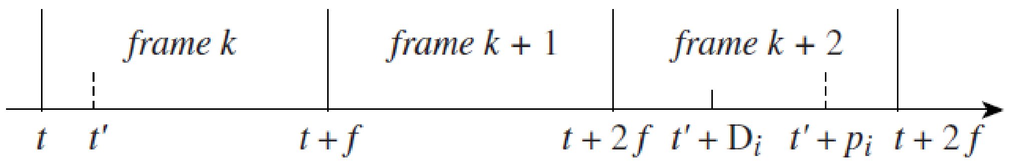

FIGURE 5–4 A constraint on the value of frame size.

Because the difference *t* _ − *t* is at least equal to the greatest common divisor *gcd(p_i , f)* of p_i and *f* , this condition is met if the following inequality holds:

$$2f - gcd(p_i, f) \leq D_i \qquad (5.3)$$

We refer to Eqs. (5.1), (5.2) and (5.3) as the *frame-size constraints*.

6. What are Job Slices?
Sometimes, the given parameters of some task systems cannot meet all three frame size constraints simultaneously. An example is the system T = {*(*4, 1*)*, *(*5, 2, 7*)*, *(*20, 5*)*}. For Eq. (5.1) to be true, we must have *f* ≥ 5, but to satisfy Eq. (5.3) we must have *f* ≤ 4. In this situation, we are forced to partition each job in a task that has a large execution time into slices (i.e., subjobs) with smaller execution times.

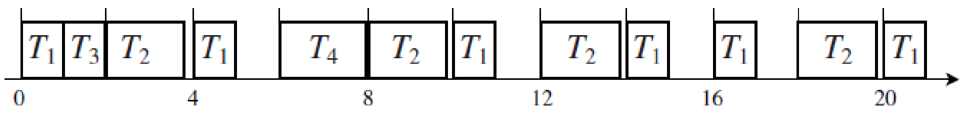

FIGURE 5–5 A cyclic schedule with frame size 2.

For T = {*(*4, 1*)*, *(*5, 2, 7*)*, *(*20, 5*)*}, we can divide each job in *(*20, 5*)* into a chain of three slices with execution times 1, 3, and 1. In other words, the task *(*20, 5*)* now consists of three subtasks *(*20, 1*)*, *(*20, 3*)* and *(*20, 1*)*.
The resultant system has five tasks for which we can choose the frame size 4. Figure 5–6 shows a cyclic schedule for these tasks. The three original tasks are called T_1, T_2 and T_3, respectively, and the three subtasks of T_3 are called $T_{3,1}$, $T_{3,2}$, and $T_{3,3}$.

7. What are cyclic executives?
We use the term *cyclic executive* in a more general sense to mean a table-driven cyclic scheduler for all types of jobs in a multithreaded system.

8. What is Slack Stealing and Slack time?
A natural way to improve the response times of aperiodic jobs is by executing the aperiodic jobs ahead of the periodic jobs whenever possible. This approach, called *slack stealing*, was originally proposed for priority-driven systems. Let the total amount of time allocated to all the slices scheduled in the frame *k* be x_k.

The *slack* (time) available in the frame is equal to $f - x_k$ at the beginning of the frame. When an aperiodic job executes ahead of slices of periodic tasks, it consumes the slack in the frame. After *y* units of slack time are used by aperiodic jobs, the available slack is reduced to $f - x_k - y$.

The cyclic executive can let aperiodic jobs execute in frame *k* as long as there is slack, that is, the available slack $f - x_k - y$ in the frame is larger than 0.

9. What is aperiodic bandwidth or processor bandwidth?

Let u_i denote the average utilization of the *i*th task; it is the average fraction of processor time required by all the jobs in the task. u_i is equal to $\lambda_i E[\beta_i]$. We call the sum U_A of u_i over all aperiodic tasks the *total average utilization of aperiodic tasks*; it is the average fraction of processor time required by all the aperiodic tasks in the system.

Let U be the total utilization of all the periodic tasks. $1-U$ is the fraction of time that is available for the execution of aperiodic jobs. We call it the *aperiodic* (*processor*) *bandwidth*.

If the total average utilization of the aperiodic jobs is larger than or equal to the aperiodic bandwidth of the system (i.e., $U_A \geq 1 - U$), the length of the aperiodic job queue and the average response time will grow without bound. Hence we consider here only the case where $U_A < 1 - U$.

When the jobs in all aperiodic tasks are scheduled on the FIFO basis, we can estimate the average response time W (also known as waiting time) of any aperiodic job by the following expression:

$$W = \sum_{i=1}^{n_a} \frac{\lambda_i E[\beta_i]}{\lambda(1-U)} + \frac{W_0}{(1-U)^2[1-U_A/(1-U)]} \qquad (5.4a)$$

where W_0 is given by

$$W_0 = \sum_{i=1}^{n_a} \frac{\lambda_i E[\beta_i{}^2]}{2} \qquad (5.4b)$$

Figure 5–9 shows the behavior of the average queueing time. The average queueing time is inversely proportional to the square of the aperiodic bandwidth. It remains small for a large range of U_A but increases rapidly and approaches infinity when *UA* approaches *(*1 − *U)*.

10. What is an Acceptance Test?

A common way to deal with this situation is to have the scheduler perform an acceptance test when each sporadic job is released. During an *acceptance test*, the scheduler checks whether the newly released sporadic job can be feasibly scheduled with all the jobs in the system at the time. Here, by *a job in the system*, we mean either a periodic job or a sporadic job.

The job can complete in time only if the *current* (*total*) *amount of slack time* $\sigma_c(t, l)$ in frames $t, t+1, \ldots l$ is equal to or greater than its execution time e. Therefore, the scheduler should reject S if $e > \sigma_c(t, l)$.

11. When will a cyclic EDF algorithm be optimal and when it won't be optimal?

The cyclic EDF algorithm is optimal in the following sense: As long as the given string of sporadic jobs is schedulable by any algorithm in this class, the EDF algorithm can always find a feasible schedule of the jobs.

However, the cyclic EDF algorithm is not optimal when compared with algorithms that perform acceptance tests at arbitrary times. If we choose to use an interrupt-driven scheduler which does an acceptance test upon the release of each sporadic job, we should be able to do better.

The advantage of the interrupt-driven alternative is outweighed by a serious shortcoming: It increases the danger for periodic-job slices to complete late. The cyclic EDF algorithm for scheduling sporadic jobs is an on-line algorithm.

12. What are the reasons for Frame Overruns and how to handle them?

A frame overrun can occur for many reasons. For example, when the execution time of a job is input data dependent, it can become unexpectedly large for some rare combination of input values which is not taken into account in the precomputed schedule.

There are many ways to handle a frame overrun. Which one is the most appropriate depends on the application and the reason for the overrun.

A way to handle overruns is to simply abort the overrun job at the beginning of the next frame and log the premature termination of the job. Such a fault can then be handled by some recovery mechanism later when necessary. This way seems attractive for applications where late results are no longer useful.

Another way to handle an overrun is to continue to execute the offending job. The start of the next frame and the execution of jobs scheduled in the next frame are then delayed.

13. What are Mode Changes?
The number *n* of periodic tasks in the system and their parameters remain constant as long as the system stays in the same (operation) mode. During a *mode change*, the system is reconfigured.

Some periodic tasks are deleted from the system because they will not execute in the new mode. When the mode change completes, the new set of periodic tasks are scheduled and executed.

14. Write about Aperiodic Mode Change.
A reasonable way to schedule a mode-change job that has a soft deadline is to treat it just like an ordinary aperiodic job. Once the job begins to execute, however, it may modify the old schedule in order to speed up the mode change.

A periodic task that will not execute in the new mode can be deleted and its memory space and processor time are freed as soon as the current job in the task completes. During mode change, the scheduler continues to use the old schedule table. Before the periodic task server begins to execute a periodic job, however, it checks whether the corresponding task is marked and returns immediately if the task is marked.

In this way, the schedule of the periodic tasks that execute in both modes remain unchanged during mode change, but the time allocated to the deleted task can be used to execute the mode-change job. Once the new schedule table and code of the new tasks are in memory, the scheduler can switch to use the new table.

15. Write about Sporadic Mode Change.
A sporadic mode change has to be completed by a hard deadline. There are two possible approaches to scheduling this job. We can treat it like an ordinary sporadic job and schedule it.

This approach can be used only if the application can handle the rejection of the job. Specifically, if the mode-change job is not schedulable and is therefore rejected, the application can either postpone the mode change or take some alternate action.

16. Write about INF algorithm.
The iterative algorithm enables us to find a feasible cyclic schedule if one exists. The algorithm is called the *iterative network-flow algorithm*, or the *INF algorithm* for short. Its key assumptions are that tasks can be preempted at any time and are independent.
Before applying the INF algorithm on the given system of periodic tasks, we find all the possible frame sizes of the system: These frame sizes met the constraints of Eqs. (5.2) and (5.3) but not necessarily satisfy Eq. (5.1).
The INF algorithm iteratively tries to find a feasible cyclic schedule of the system for a possible frame size at a time, starting from the largest possible frame size in order of decreasing frame size. A feasible schedule thus found tells us how to decompose some tasks into subtasks if their decomposition is necessary.

If the algorithm fails to find a feasible schedule after all the possible frame sizes have been tried, the given tasks do not have a feasible cyclic schedule that satisfies the frame size constraints even when tasks can be decomposed into subtasks.

17. What is Network-Flow Graph?
The algorithm used during each iteration is based on the well-known network-flow formulation of the preemptive scheduling problem. In the description of this formulation, it is more convenient to ignore the tasks to which the jobs belong and name the jobs to be scheduled in a major cycle of *F* frames $J_1, J_2, \dots, J_N$.

(……….The constraints on when the jobs can be scheduled are represented by the *network-flow graph* of the system. This graph contains the following vertices and edges; the capacity of an edge is a nonnegative number associated with the edge.
Key terms:

1. There is a *job vertex* J_i representing each job J_i, for $i = 1, 2, \dots, N$.
2. There is a *frame vertex* named *j* representing each frame *j* in the major cycle, for $j = 1, 2, \dots, F$.
3. There are two special vertices named *source* and *sink*.

4. There is a directed edge *($J_{i,j}$)* from a job vertex *Ji* to a frame vertex *j* if the job J_i can be scheduled in the frame *j* , and the *capacity* of the edge is the frame size *f* .
5. There is a directed edge from the *source* vertex to every job vertex J_i , and the capacity of this edge is the execution time e_i of the job.
6. There is a directed edge from every frame vertex to the *sink*, and the capacity of this edge is *f* ……….)

18. What is flow of an edge and flow of a network-flow graph?
A *flow of an edge* is a nonnegative number that satisfies the following constraints:
a. It is no greater than the capacity of the edge
b. with the exception of the *source* and *sink*,
The sum of the flows of all the edges into every vertex is equal to the sum of the flows of all the edges out of the vertex.
A *flow of a network-flow graph*, or simply a flow, is the sum of the flows of all the edges from the *source*; it should equal to the sum of the flows of all the edges into the *sink*.

19. Maximum Flow and Feasible Preemptive Schedule:
The maximum flow of a network-flow graph defined above is at most equal to the sum of the execution times of all the jobs to be scheduled in a major cycle. The set of flows of edges from job vertices to frame vertices that gives this maximum flow represents a feasible preemptive schedule of the jobs in the frames.

Specifically, *the flow of an edge ($J_{i,j}$) from a job vertex J_i to a frame vertex j gives the amount of time in frame j allocated to job J_i* .

The total amount of time allocated to a job J_i is represented by the total flow out of all the edges from the job vertex J_i . Since this amount is equal to the flow into J_i , this amount is e_i . Since the flow of the only edge out of every frame vertex is at most *f* , the total amount of time in every frame allocated to all jobs is at most *f* .

20. List the advantages of clock-driven scheduling
Advantages:
   - When the workload is mostly periodic and the schedule is cyclic, timing constraints can be checked and enforced at each frame boundary.
   - Context switching and communication overhead can be kept low by choosing as large a frame size as possible.
   - Systems based on the clock-driven scheduling paradigm are typically *time-triggered. In a time-triggered system,* interrupts in response to external events are queued and polled periodically.
   - Systems based on the clock-driven scheduling paradigm are typically *time-triggered. In a time-triggered system,* interrupts in response to external events are queued and polled periodically.

21. List the disadvantages of clock-driven scheduling
Disadvantages:
   - The most obvious one is that a system based on this approach is brittle: It is relatively difficult to modify and maintain.
   - The release times of all jobs must be fixed. In contrast, priority-driven algorithms do not require fixed release times.
   - In a clock-driven system, all combinations of periodic tasks that might execute at the same time must be known a priori so a schedule for the combination can be precomputed.
   - The pure clock-driven approach is not suitable for many systems that contain both hard and soft real-time applications.