Unit – 1

Typical Real-Time Applications

1. What is a Digital Control ?

   Many real-time systems are embedded in sensors and actuators and function as digital controllers. Figure 1–1 shows such a system. The term plant in the block diagram refers to a controlled system, for example, an engine, a brake, an aircraft, a patient.

2. What is Control-law computation?

   The state of the plant is monitored by sensors and can be changed by actuators. The real-time system estimates from the sensor readings the current state of the plant and computes a control output based on the difference between the current state and the desired state (called reference input in the figure). We call this computation the *control-law computation* of the controller. The output thus generated activates the actuators, which bring the plant closer to the desired state.

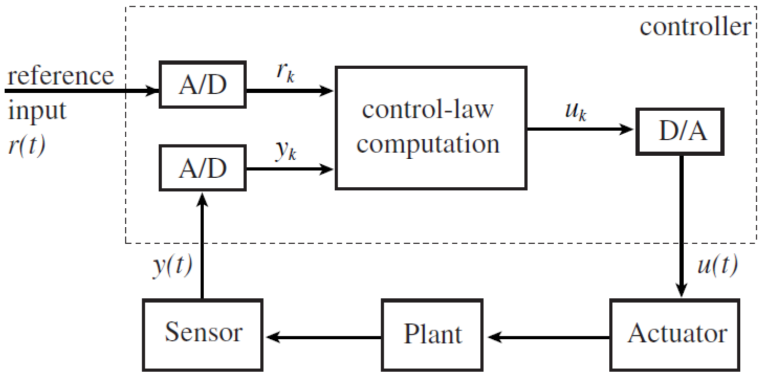


FIGURE 1–1 A digital controller.

3. What are Sampled data systems?

   A common approach to designing a digital controller is to start with an analog controller that has the desired behavior. The analog version is then transformed into a digital version. The resultant controller is a ***sampled data system.***

4. What is a feedback control loop?

   As an example, we consider an analog single-input/single-output PID (Proportional, Integral, and Derivative) controller. This simple kind of controller is commonly used in practice. The analog sensor reading *y(t)* gives the measured state of the plant at time *t*.

   Let *e(t)* = *r (t)* − *y(t)* denote the difference between the desired state *r (t)* and the measured state *y(t)* at time *t* . The output *u(t)* of the controller consists of three terms: a term that is proportional to *e(t)*, a term that is proportional to the integral of *e(t)* and a term that is proportional to the derivative of *e(t)*. The following incremental expression of the *k*th output u_k :

$$u_k = u_{k-2} + \alpha e_k + \beta e_{k-1} + \gamma e_{k-2} \qquad (1.1)$$

   α, *β*, and *γ* are proportional constants; they are chosen at design time.1 During the *k*th sampling period, the real-time system computes the output of the controller according to this expression.

   From Eq. (1.1), we can see that during any sampling period, the control output u_k depends on the current and past measured values y_i for $i \le k$. Such a system is called a ***(feedback) control loop*** or simply a ***loop***.

5. Give an example for a feedback loop.

```
set timer to interrupt periodically with period $T$;
at each timer interrupt, do
    do analog-to-digital conversion to get $y$;
    compute control output $u$;
    output $u$ and do digital-to-analog conversion;
end do;
```

6. List the factors affecting the selection of sampling period.
   The first is the perceived responsiveness of the overall system (i.e., the plant). Sampling introduces a delay in the system response.
   The second factor is the dynamic behavior of the plant.

7. What are multirate systems?
   Multirate Systems:
   A plant typically has more than one degree of freedom. Its state is defined by multiple state variables (e.g., the rotation speed, temperature, etc. of an engine or the tension and position of a video tape). Therefore, it is monitored by multiple sensors and controlled by multiple actuators.

8. Brief the Successive loop closure method.
   According to this method, the designer begins by selecting the sampling period of the controller that should have the fastest sampling rate among all the controllers. In this selection, the controller is assumed to be independent of the others in the system. After a digital version is designed, it is converted back into an analog form. The analog model is then integrated with the slower portion of the plant and is treated as a part of the plant.

   This step is then repeated for the controller that should have the fastest sampling rate among the controllers whose sampling periods remain to be selected. The iteration process continues until the slowest digital controller is designed. Each step uses the model obtained during the previous step as the plant.

9. What are the characteristics of a simple controller?
   1. Sensor data give accurate estimates of the state-variable values being monitored and controlled.
   2. The sensor data give the state of the plant.
   3. All the parameters representing the dynamics of the plant are known.

11. What is a deadbeat control?
    Deadbeat Control: A discrete-time control scheme that has no continuous-time equivalence is deadbeat control. In response to a (step) change in the reference input, a deadbeat controller brings the plant to the desired state by exerting on the plant a fixed number (say n) of control commands. The control-law computation of a deadbeat controller is also simple. The output produced by the controller during the kth sampling period is given by

$$u_k = \alpha \sum_{i=0}^{k} (r_i - y_i) + \sum_{i=0}^{k} \beta_i x_i$$

12. What is a Kalman Filter?
    Kalman Filter: Kalman filtering is a commonly used means to improve the accuracy of measurements and to estimate model parameters in the presence of noise and uncertainty. To illustrate, we consider a simple monitor system that takes a measured value y_k every sampling period k in order to estimate the value x_k of a state variable.

    The Kalman filter starts with the initial estimate $\tilde{x}_1 = y_1$ and computes a new estimate each sampling period. Specifically, for k > 1, the filter computes the estimate $\tilde{x}_k$ as follows:

$$\tilde{x}_k = \tilde{x}_{k-1} + K_k (y_k - \tilde{x}_{k-1}) \qquad (1.2a)$$

In this expression,

$$K_k = \frac{P_k}{\sigma_k{}^2 + P_k} \tag{1.2b}$$

$$P_k = E[(\tilde{x}_k - x)^2] = (1 - K_{k-1})P_{k-1} \tag{1.2c}$$

13. What are High Level Controls?
    Controllers in a complex monitor and control system are typically organized hierarchically. One or more digital controllers at the lowest level directly control the physical plant. Each output of a higher-level controller is a reference input of one or more lower-level controllers.

14. Give an example for Control Hierarchy.
    A patient care system may consist of microprocessor-based controllers that monitor and control the patient's blood pressure, respiration, glucose, and so forth. There may be a higher-level controller (e.g., an expert system) which interacts with the operator (a nurse or doctor) and chooses the desired values of these health indicators. While the computation done by each digital controller is simple and nearly deterministic, the computation of a highlevel controller is likely to be far more complex and variable. While the period of a lowlevel control-law computation ranges from milliseconds to seconds, the periods of high-level control-law computations may be minutes, even hours.

15. Write about Signal Processing.
    Most signal processing applications have some kind of real-time requirements. Examples are digital filtering, video and voice compressing/decompression, and radar signal processing.
    Typically, a real-time signal processing application computes in each sampling period one or more outputs. Each output x(k) is a weighted sum of n inputs y(i)'s:

$$x(k) = \sum_{i=1}^{n} a(k, i)y(i) \tag{1.3}$$

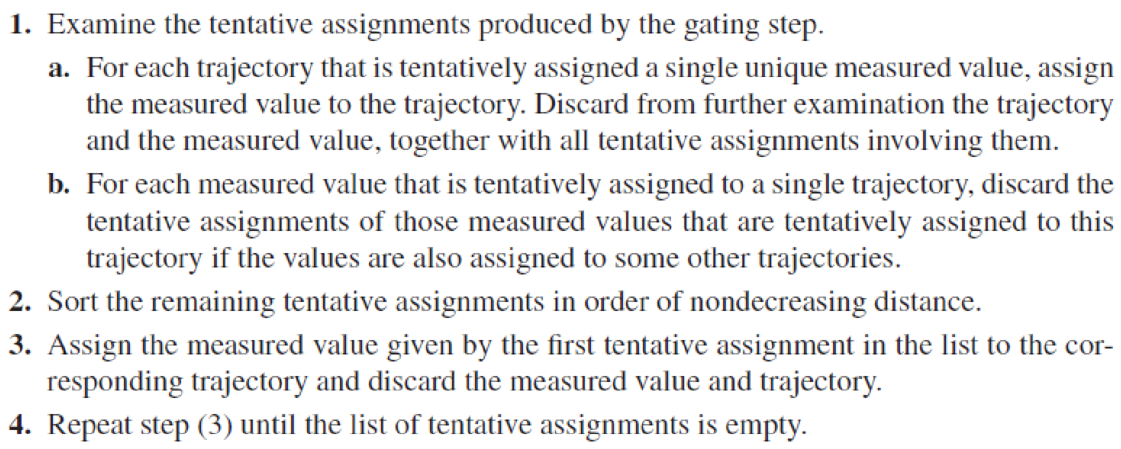
16. Write about Radar System.
    A signal processing application is typically a part of a larger system. As an example, Figure 1–6 shows a block diagram of a (passive) radar signal processing and tracking system.
    The system consists of an Input/Output (I/O) subsystem that samples and digitizes the echo signal from the radar and places the sampled values in a shared memory. An array of digital signal processors processes these sampled values. The data thus produced are analyzed by one or more data processors to control the radar and select parameters to be used by signal processors in the next cycle.

17. What is Tracking and What is a Tracker?
    Tracking. Strong noise and man-made interferences, including electronic counter measure (i.e., jamming), can lead the signal processing and detection process to wrong conclusions about the presence of objects. A track record on a nonexisting object is called a false return. An application that examines all the track records in order to sort out false returns from real ones and update the trajectories of detected objects is called a tracker.

18. What is Gating?
    Gating. Typically, tracking is carried out in two steps: gating and data association. Gating is the process of putting each measured value into one of two categories depending on whether it can or cannot be tentatively assigned to one or more established trajectories.

    The gating process tentatively assigns a measured value to an established trajectory if it is within a threshold distance G away from the predicted current position and velocity of the object moving along the trajectory. The threshold G is called the track gate. It is chosen so that the probability of a valid measured value falling in the region bounded by a sphere of radius G centered around a predicted value is a desired constant.

19. What is Data Association?
    There are many data association algorithms. One of the most intuitive is the nearest neighbor algorithm. This algorithm works as follows:



1. Examine the tentative assignments produced by the gating step.
   a. For each trajectory that is tentatively assigned a single unique measured value, assign the measured value to the trajectory. Discard from further examination the trajectory and the measured value, together with all tentative assignments involving them.
   b. For each measured value that is tentatively assigned to a single trajectory, discard the tentative assignments of those measured values that are tentatively assigned to this trajectory if the values are also assigned to some other trajectories.
2. Sort the remaining tentative assignments in order of nondecreasing distance.
3. Assign the measured value given by the first tentative assignment in the list to the corresponding trajectory and discard the measured value and trajectory.
4. Repeat step (3) until the list of tentative assignments is empty.

20. What are Real-Time Databases?
    The term real-time database systems refers to a diverse spectrum of information systems, ranging from stock price quotation systems, to track records databases, to real-time file systems.
    What distinguish these databases from nonrealtime databases is the perishable nature of the data maintained by them.

21. What are Image objects?
    A real-time database contains data objects, called image objects, that represent real-world objects. The attributes of an image object are those of the represented real world object.

22. Why real-time data are perishable?
    For example, an air traffic control database contains image objects that represent aircraft in the coverage area. The attributes of such an image object include the position and heading of the aircraft. The values of these attributes are updated periodically based on the measured values of the actual position and heading provided by the radar system.
    Without this update, the quality of stored data degrades. This is why we say that real-time data are perishable. In contrast, an underlying assumption of nonrealtime databases is that in the absence of updates the data contained in them remain good.

23. What is Absolute Temporal Consistency?
    A set of data objects is said to be absolutely (temporally) consistent if the maximum age of the objects in the set is no greater than a certain threshold.

24. What is Relative Temporal Consistency?
25. A set of data objects is said to be relatively consistent if the maximum difference in ages of the objects in the set is no greater than the relative consistency threshold used by the application.

26. Why the normal Concurrency control mechanisms are not suitable for Real-time Databases?
    Concurrency control mechanisms, such as two-phase locking, have traditionally been used to ensure the serializability of read and update transactions and maintain data integrity of nonreal-time databases. These mechanisms often make it more difficult for updates to complete in time. Late updates may cause the data to become temporally inconsistent which is not appropriate for real time systems. For this reason, several weaker consistency models have been proposed.

27. What are the Concurrency control mechanisms for Real-time Databases?
Kuo and Mok proposed the use of similarity as a correctness criterion for real-time data. Intuitively, we say that two values of a data object are similar if the difference between the values is within an acceptable threshold from the perspective of every transaction that may read the object.
Two views of a transaction are similar if every read operation gets similar values of every data object read by the transaction.
Two database states are similar if, in the states, the corresponding values of every data object are similar.
Two schedules of a set of transactions are similar if, for any initial state,
(1) the transactions transform similar database states to similar final database states
(2) every transaction in the set has similar views in both schedules.

28. What are Multimedia Applications?
A multimedia application may process, store, transmit, and display any number of video streams, audio streams, images, graphics, and text. A video stream is a sequence of data frames which encodes a video. An audio stream encodes a voice, sound, or music. Without compression, the storage space and transmission bandwidth required by a video are enormous. Therefore, a video stream, as well as the associated audio stream, is invariably compressed as soon as it is captured.

29. Write in short on MPEG Compression/Decompression?
A video compression standard is MPEG-2. The standard makes use of three techniques. They are motion compensation for reducing temporal redundancy, discrete cosine transform for reducing spatial redundancy, and entropy encoding for reducing the number of bits required to encode all the information. Depending on the application, the compressed bit rate ranges from 1.5 Mbits/sec to 35Mbits/sec. As you will see from the description below, the achievable compression ratio depends on the content of the video.

30. Write about Motion Estimation?
The first step of compression is motion analysis and estimation. Because consecutive video frames are not independent, significant compression can be achieved by exploiting interframe dependency. This is the rationale behind the motion estimation step.
The motion-compensation techniques used in this step assume that most small pieces of the image in the current frame can be obtained either by translating in space corresponding small pieces of the image in some previous frame or by interpolating some small pieces in some previous and subsequent frames.

For this reason, each image is divided into 16×16-pixel square pieces; they are called major blocks. The luminance component of each major block consists of four 8×8 pixel blocks. Each of the chrominance components has only a quarter of this resolution. Hence, each chrominance component of a major block is an 8×8 pixel block.

Only frames $1 + \alpha k$, for $k = 0, 1, 2, \ldots$ are encoded independently of other frames, where α is an application-specified integer constant. These frames are called I-frames (i.e., intra-coded frames). The coder treats each I-frame as a still image, and the decoder can decompress each compressed I-frame independently of other frames. Consequently, I-frames are points for random access of the video. The smaller the constant α, the more random accessible is the video and the poorer the compression ratio.

A good compromise is $\alpha = 9$. The coder generates a motion vector based on which the decoder can identify the best matching I-frame major block. Such a P-frame major block is said to be predictively coded. Some P-frame major blocks may be images of newly visible objects and, hence, cannot be obtained from any major block in the previous I-frame. The coder represents them in the same way as I-frame major blocks. A B-frame is a bidirectionally predicted frame: It is predicted from both the previous I-frame (or P-frame) and the subsequent P-frame (or I-frame).

For each B-frame major block, an interpolation of the best matching major blocks in the I-frame and P-frame is first computed. The B-frame major block is represented by the difference between it and this interpolation. Again, the coder generates the motion vectors that the decoder will need to identify the best matching I-frame and P-frame major blocks. Whereas some P-frame major blocks are encoded independently, none of the B-frame major blocks are.

31. Write about Discrete Cosine Transform and Encoding?
In the second step of MPEG Compression, a cosine transform is performed on each of the 8×8 pixel blocks produced by the coder after motion estimation. We let x(i, j), for i, j = 1, 2, . . . , 8, denote the elements of an 8×8 transform matrix obtained from transforming the originalmatrix that gives the 8×8 values of a pixel block. The transform matrix usually has more zeros than the original matrix.

By quantizing the x(i, j)'s to create more zeros, encoding the entries in the transform matrix as 2-tuples (run length, value), and using a combination of variable-length and fixed-length codes to further reduce the bit rate, significant compression is achieved.

32. Write about Decompression?
During decompression, the decoder first produces a close approximation of the original matrix (i.e., an 8 × 8 pixel block) by performing an inverse transform on each stored transform matrix. It then reconstruct the images in all the frames from the major blocks in I-frames and difference blocks in P- and B-frames.

33. List some of the Real-Time Characteristics?
To a great extent, the timing requirements of a multimedia application follow from the required video and audio quality. A video of standard television quality consists of 30 frames per second. High-definition television uses 60 frames per second to give the picture even less flicker. Much lower frame rates (e.g., 10–20) are tolerable for other applications, such as teleconferencing.

The quality of an audio component depends on the sampling rate and granularity used to digitize the audio signal. The total bit rate of an audio ranges from 16 Kbits per second for telephone speech quality to 128 Kbits per second for CD quality. Some loss of audio data is unavoidable.

The quality of speech is usually tolerable when the loss rate is under one percent. Another dimension of quality of a multimedia application is lip synchronization.

For batch applications, a system can often provide the desired quality by trading between real-time performance and space usage. For example, we want to present the audio to the user without pauses. This can clearly be achieved if there is little or no jitter (i.e., variation) in the delay suffered by audio data packets as they are transmitted over the network.

Hard versus Soft Real-Time Systems

1. What are Jobs?
   Each unit of work that is scheduled and executed by the system is a *job* and a set of related jobs which jointly provide some system function a *task*. Hence, the computation of a control law is a job.

   Eg: So is the computation of a FFT (Fast Fourier Transform) of sensor data, or the transmission of a data packet, or the retrieval of a file, and so on. We call them a control-law computation, a FFT computation, a packet transmission.

2. What are Processors?
   Every job executes on some resource. These resources are called servers in queuing theory literature and, sometimes, active resources in real-time systems literature. All these resources are referred to as *processors* except occasionally when we want to be specific about what they are.

3. What is Release Time?
   The ***release time*** of a job is the instant of time at which the job becomes available for execution. The job can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met.

   We say that ***jobs have no release time*** if all the jobs are released when the system begins execution.

4. What is a Deadline?
   The ***deadline*** of a job is the instant of time by which its execution is required to be completed. Suppose that in the previous example, each control-law computation job must complete by the release time of the subsequent job. Then, their deadlines are 120 msec, 220 msec, and so on, respectively.

   A job has ***no deadline*** if its deadline is at infinity.

5. What is Response Time?
   It is more natural to state the timing requirement of a job in terms of its ***response time***, that is, the length of time from the release time of the job to the instant when it completes.

6. What is Relative Deadline?
   The maximum allowable response time of a job is its ***relative deadline***. Hence the relative deadline of every control-law computation job mentioned above is 100 or 50 msec.

7. What is Absolute Deadline?
   The deadline of a job, sometimes called its ***absolute deadline***, is equal to its release time plus its relative deadline.

8. What is a Timing Constraint?
   A constraint imposed on the timing behavior of a job is a ***timing constraint***. In its simplest form, a timing constraint of a job can be specified in terms of its release time and relative or absolute deadlines, as illustrated by the above example.

9. What is a Hard Timing Constraint?
   A ***timing constraint or deadline is hard*** if the failure to meet it is considered to be a fatal fault. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences.

10. What is a Soft Timing Constraint?
    The late completion of a job that has a ***soft deadline*** is undesirable. However, a few misses of soft deadlines do no serious harm; only the system's overall performance becomes poorer and poorer when more and more jobs with soft deadlines complete late.

11. What is Tardiness of a job?

The ***tardiness*** of a job measures how late it completes respective to its deadline. Its tardiness is zero if the job completes at or before its deadline; otherwise, if the job is late, its tardiness is equal to the difference between its ***completion time*** and its deadline.

12. The usefulness of a result produced by a soft real-time job decreases gradually as the tardiness of the job increases, but the usefulness of a result produced by a hard real-time job falls off abruptly and may even become negative when the tardiness of the job becomes larger than zero.

13. The deadline of a job is softer if the usefulness of its result decreases at a slower rate. There is often no natural choice of usefulness functions.

14. What is Validation?

    By ***validation***, we mean a demonstration by a provably correct, efficient procedure or by exhaustive simulation and testing.

    If no validation is required, or only a demonstration that the job meet some *statistical constraint* suffices, then the timing constraint of the job is soft.

    This way to differentiate between hard and soft timing constraints is compatible with the distinction between *guaranteed* and *best-effort* services.

15. When will be the timing constraints soft and hard?

    If the user wants the temporal quality of the service provided by a task guaranteed and the satisfaction of the timing constraints defining the temporal quality validated, then the timing constraints are hard.

    If the user demands the best quality of service the system can provide but allows the system to deliver qualities below what is defined by the timing constraints, then the timing constraints are soft.

16. Mention the reasons for Requiring Timing Guarantees.

    Many embedded systems are hard real-time systems. Deadlines of jobs in an embedded system are typically derived from the required responsiveness of the sensors and actuators monitored and controlled by it.

    As an example (Eg 1), we consider an "**automatically controlled train**". It cannot stop instantaneously. When the signal is red (stop), its braking action must be activated a certain distance away from the signal post at which the train must stop. This braking distance depends on the speed of the train and the safe value of deceleration.

    From the speed and safe deceleration of the train, the controller can compute the time for the train to travel the braking distance. This time in turn imposes a constraint on the response time of the jobs which sense and process the stop signal and activate the brake. No one would question that this timing constraint should be hard and that its satisfaction must be guaranteed.

    Eg 2: Similarly, each control-law computation job of a flight controller must be completed in time so that its command can be issued in time. Otherwise, the plane controlled by it may become oscillatory or even unstable and uncontrollable. For this reason, we want the timely completion of all control-law computations guaranteed.

    Jobs in some nonembedded systems may also have hard deadlines. An example is a critical information system that must be highly available: The system must never be down for more than a minute. Because of this requirement, reconfiguration and recovery of database servers and network connections in the system must complete within a few seconds or tens of seconds, and this relative deadline is hard.

    In general, if safety or property loss is involved, the designer/builder of the system has the burden of proof that bad things will never happen. Whenever it is not possible to prove that a few timing constraint violations will not jeopardize the safety of users or availability of some critical infrastructure, we take the safe approach and insist on the satisfaction of all timing constraints.

17. How are the Hard and Soft Timing constraints specified?
    A hard timing constraint to be specified in any terms. Examples are

1. deterministic constraints (e.g., the relative deadline of every control-law computation is 50 msec or the response time of at most one out of five consecutive control-law computations exceeds 50 msec);

2. probabilistic constraints, that is, constraints defined in terms of tails of some probability distributions (e.g., the probability of the response time exceeding 50 milliseconds is less than 0.2);

3. constraints in terms of some usefulness function (e.g., the usefulness of every control law computation is 0.8 or more).

   In practice, hard timing constraints are rarely specified in the latter two ways.

18. What are Soft Real-time Systems?
    A system in which jobs have soft deadlines is a *soft real-time system*. The developer of a soft real-time system is rarely required to prove rigorously that the system surely meet its realtime performance objective.

    Examples of such systems include on-line transaction systems and telephone switches, as well as electronic games.

    An occasional missed deadline or aborted execution is usually considered tolerable; it may be more important for such a system to have a small average response time and high throughput.

19. Give an example of a Soft Real-time system with critical timing requirements.
    A system may have critical timing requirements but is nevertheless considered to be a soft real-time system.

    An example is a stock price quotation system. It should update the price of each stock each time the price changes. Here, a late update may be highly undesirable, because the usefulness of a late price quotation decreases rapidly with time. However, in a volatile market when prices fluctuate at unusually high rates, we expect that the system cannot keep up with every price change but does its best according to some criteria. Occasional late or missed updates are tolerated as a trade-off for other factors, such as cost and availability of the system and the number of users the system can serve.

    The timing requirements of soft real-time systems are often specified in probabilistic terms.

20. Give an example of Soft Real-time systems.
    Take a telephone network for example. In response to our dialing a telephone number, a sequence of jobs executes in turn, each routes the control signal from one switch to another in order to set up a connection through the network on our behalf. We expect that our call will be put through in a short time. To ensure that this expectation is met most of the time by the network, a timing constraint may be imposed on this sequence of jobs as a design objective.

    Example of multimedia systems that provide the user with services of “guaranteed” quality. For example, a frame of a movie must be delivered every thirtieth of a second, and the difference in the times when each video frame is displayed and when the accompanied speech is presented should be no more than 80 msec.

    The timing constraints of multimedia systems are guaranteed on a statistical basis. Moreover, users of such systems rarely demand any proof that the system indeed honor its guarantees. The quality-of-service guarantee is soft, the validation requirement is soft, and the timing constraints defining the quality are soft.

Unit – 1
A Reference Model of Real-Time Systems

1. List the Characteristics of the Reference model of Real-time systems.
   Let us assume a reference model of real-time systems. According to this model, each system is characterized by three elements:
   a. a workload model that describes the applications supported by the system
   b. a resource model that describes the system resources available to the applications
   c. algorithms that define how the application system uses the resources at all times.

2. What are Processors?
   We divide all the system resources into two major types: processors and resources. Again, processors are often called servers and active resources; computers, transmission links, disks, and database server are examples of processors. They carry out machine instructions, move data from one place to another, retrieve files, process queries, and so on.

   Every job must have one or more processors in order to execute. Sometimes, we will need to distinguish the types of processors. Two processors are of the same type if they are functionally identical and can be used interchangeably. Hence two transmission links with the same transmission rate between a pair of sender and receiver are processors of the same type. Processors that are functionally different cannot be used interchangeably, are of different types.

3. What are Resources?
   By resources, we will specifically mean passive resources. Examples of resources are memory, sequence numbers, mutexes, and database locks. We will use the letter R to denote resources. The resources in the examples mentioned above are reusable, because they are not consumed during use.

4. What is a Plentiful resource?
   To prevent our model from being cluttered by irrelevant details, we typically omit the resources that are plentiful. A resource is *plentiful* if no job is ever prevented from execution by the lack of this resource. A resource that can be shared by an infinite number of jobs need not be explicitly modelled and hence never appears in our model.

5. How does the no. of tasks in a system affects its performance?
   Many parameters of hard real-time jobs and tasks are known at all times; The number of tasks (or jobs) in the system is one such parameter. In many embedded systems, the number of tasks is fixed as long as the system remains in an operation mode. The number of tasks may change when the system operation mode changes, and the number of tasks in the new mode is also known. Moreover, these numbers are known a priori before the system begins execution. The number of tasks may change as tasks are added and deleted while the system executes.

6. What is Release-time jitter?
   In many systems, we do not know exactly when each job will be released. In other words, we do not know the actual release time ri of each job Ji ; only that ri is in a range [ri−, ri+]. ri can be as early as the earliest release time ri− and as late as the latest release time ri+. Indeed, some models assume that only the range of ri is known and call this range the jitter in ri, or release-time jitter. Sometimes, the jitter is negligibly small compared with the values of other temporal parameters.

7. What are Aperiodic and Sporadic jobs?
   Almost every real-time system is required to respond to external events which occur at random instants of time. When such an event occurs, the system executes a set of jobs in response. The release times of these jobs are not known until the event triggering them occurs. These jobs are called sporadic jobs or aperiodic jobs because they are released at random time instants.

For example, the pilot may disengage the autopilot system at any time. When this occurs, the autopilot system changes from cruise mode to standby mode. The jobs that execute to accomplish this mode change are sporadic jobs.

8. What is Interrelease time?
   The length of the time interval between the release times of two consecutive jobs in the stream).

9. What is Arrival time?
   Sometimes use the term arrival times (or interarrival time) which is commonly used in queueing theory. An aperiodic job arrives when it is released. A(x) is the arrival time distribution (or interarrival time distribution).

10. What is Execution time?
    Another temporal parameter of a job, Ji , is its execution time, ei . ei is the amount of time required to complete the execution of Ji when it executes alone and has all the resources it requires. Hence, the value of this parameter depends mainly on the complexity of the job and the speed of the processor used to execute the job, not on how the job is scheduled.

    The actual amount of time required by a job to complete its execution may vary for many reasons. As examples, a computation may contain conditional branches, and these conditional branches may take different amounts of time to complete.

    For the purpose of determining whether each job can always complete by its deadline, knowing the maximum execution time of each job often suffices. For this reason, in most deterministic models the term execution time ei of each job Ji specifically means its maximum execution time.

11. Mention the reasons for following the Deterministic approach.
    There are two good reasons for the common use of the deterministic approach.
    Many hard real-time systems are safety-critical. These systems are typically designed and implemented in a such a way that the variations in job execution times are kept as small as possible.
    The other reason for the common use of the deterministic approach is that the hard realtime portion of the system is often small. The timing requirements of the rest of the system are soft.

12. Write about the Periodic Task Model.
    The periodic task model is a well-known deterministic workload model. The model characterizes accurately many traditional hard real-time applications. Many scheduling algorithms based on this model have good performance and well-understood behavior.

13. What is a Period task?
    In the periodic task model, each computation that is executed repeatedly at regular or semiregular time intervals in order to provide a function of the system on a continuing basis is modeled as a period task. Specifically, each periodic task, denoted by Ti , is a sequence of jobs.

14. What is Period?
    The period pi of the periodic task Ti is the minimum length of all time intervals between release times of consecutive jobs in Ti. Its execution time is the maximum execution time of all the jobs in it. With a slight abuse of the notation, we use ei to denote the execution time of the periodic task Ti , as well as that of all the jobs in it.

15. Why is Periodic task an Inaccurate model?
    At all times, the period and execution time of every periodic task in the system are known. The accuracy of the periodic task model decreases with increasing jitter in release times and variations in execution times. So, a periodic task is an inaccurate model of the transmission of a variable bit-rate video, because of the large variation in the execution times of jobs.

    Note: We call the tasks in the system T_1, T_2, . . . , T_n. When it is necessary to refer to the individual jobs in a task T_i , we call them $J_{i,1}$, $J_{i,2}$ and so on, $J_{i,k}$ being the kth job in T_i. When we want to talk about properties of individual jobs but are not interested in the tasks to which they belong, we also call the jobs J_1, J_2, and so on.

16. What is Phase of a task?
    The release time ri,1 of the first job Ji,1 in each task Ti is called the phase of Ti. For the sake of convenience, we use ϕi to denote the phase of Ti , that is, ϕi = ri,1. In general, different tasks may have different phases. Some tasks are in phase, meaning that they have the same phase.

17. What is a Hyperperiod?
    We use H to denote the least common multiple of pi for i = 1, 2, . . . n. A time interval of length H is called a hyperperiod of the periodic tasks.

    The (maximum) number N of jobs in each hyperperiod is equal to ∑i=1 to n H/pi . The length of a hyperperiod of three periodic tasks with periods 3, 4, and 10 is 60. The total number N of jobs in the hyperperiod is 41.

18. What is Utilization and Total Utilization of a task?
    We call the ratio ui = ei /pi the utilization of the task Ti . ui is equal to the fraction of time a truly periodic task with period pi and execution time ei keeps a processor busy. It is an upper bound to the utilization of any task modeled by Ti. The total utilization U of all the tasks in the system is the sum of the utilizations of the individual tasks in it.

    So, if the execution times of the three periodic tasks are 1, 1, and 3, and their periods are 3, 4, and 10, respectively, then their utilizations are 0.33, 0.25 and 0.3. The total utilization of the tasks is 0.88; these tasks can keep a processor busy at most 88 percent of the time.

19. What are Aperiodic and Sporadic Tasks? Give examples.
    In the periodic task model, the workload generated in response to these unexpected events is captured by aperiodic and sporadic tasks. Each aperiodic or sporadic task is a stream of aperiodic or sporadic jobs, respectively. The interarrival times between consecutive jobs in such a task may vary widely and, in particular, can be arbitrarily small. The jobs in each task model the work done by the system in response to events of the same type.

    For example, the jobs that execute to change the detection threshold of the radar system are in one task; the jobs that change the operation mode of the autopilot are in one task; and the jobs that process sporadic data messages are in one task, and so on. Specifically, the jobs in each aperiodic task are similar in the sense that they have the same statistical behavior and the same timing requirement.

20. Mention the features of Aperiodic tasks.
    The execution times of jobs in each aperiodic (or sporadic) task are identically distributed random variables, each distributed according to the probability distribution. We say that a task is Aperiodic if the jobs in it have either soft deadlines or no deadlines. The task to adjust radar's sensitivity is an example. On the other hand, a late response is annoying but tolerable.

21. Mention the features of Sporadic tasks.
    When a transient fault occurs, a fault-tolerant system may be required to detect the fault and recover from it in time. The jobs that execute in response to these events have hard deadlines. Task containing jobs that are released at random time instants and have hard deadlines are Sporadic tasks. We treat them as hard real-time tasks. Our primary concern is to ensure that their deadlines are always met; minimizing their response times is of secondary importance.

22. When do jobs have Precedence constraints?
    Data and control dependencies among jobs may constrain the order in which they can execute. In classical scheduling theory, the jobs are said to have precedence constraints if they are constrained to execute in some order. Otherwise, if the jobs can execute in any order, they are said to be independent.

For example, in a radar surveillance system, the signal-processing task is the producer of track records, while the tracker task is the consumer. In particular, each tracker job pro cesses the track records generated by a signal-processing job. The designer may choose to synchronize the tasks so that the execution of the kth tracker job does not begin until the kth signal-processing job completes. The tracker job is precedence constrained.

23. Write about Precedence Graph
    We use a partial-order relation <, called a precedence relation, over the set of jobs to specify the precedence constraints among jobs. A job Ji is a predecessor of another job Jk (and Jk a successor of Ji) if Jk cannot begin execution until the execution of Ji completes. A shorthand notation to state this fact is Ji < Jk . Ji is an immediate predecessor of Jk (and Jk is an immediate successor of Ji) if Ji < Jk and there is no other job Jj such that Ji < Jj < Jk. Two jobs Ji and Jk are independent when neither Ji < Jk nor Jk < Ji .

    A job with predecessors is ready for execution when the time is at or after its release time and all of its predecessors are completed. A classical way to represent the precedence constraints among jobs in a set J is by a directed graph G = (J,<). Each vertex in this graph represents a job in J. We will call each vertex by the name of the job represented by it. There is a directed edge from the vertex Ji to the vertex Jk when the job Ji is an immediate predecessor of the job Jk . This graph is called a precedence graph.

24. What is a Task Graph?
    A task graph, which gives us a general way to describe the application system, is an extended precedence graph. As in a precedence graph, the vertices in a task graph represent jobs. They are shown as circles and squares in this figure.

    The numbers in the bracket above each job gives its feasible interval. The edges in the graph represent dependencies among jobs. If all the edges are precedence edges, representing precedence constraints, then the graph is a precedence graph.

    A subgraph's being a chain indicates that for every pair of jobs Ji and Jk in the subgraph, either $J_i < J_k$ or $J_i > J_k$. Hence the jobs must be executed in serial order.

    The type(s) of dependency represented by an edge is given by the type(s) of the edge. The types of an edge connecting two vertices and other parameters of the edge are interconnection parameters of the jobs represented by the vertices.

25. What is Data Dependency? Give example.
    As an example, in an avionics system, the navigation job updates the location of the airplane periodically. These data are placed in a shared space. Whenever the flight management job needs navigation data, it reads the most current data produced by the navigation job. There is no precedence constraint between the navigation job and the flight management job.

    In a task graph, data dependencies among jobs are represented explicitly by data dependency edges among jobs. There is a data-dependency edge from a vertex J_i to vertex J_k in the task graph if the job J_k consumes data generated by J_i or the job J_i sends messages to J_k . A parameter of an edge from J_i to J_k is the volume of data from J_i to J_k .

26. What is Temporal Dependency? Give example.
    We call the difference in the completion times of two jobs the temporal distance between them. Jobs are said to have a temporal distance constraint if their temporal distance must be no more than some finite value. Jobs with temporal distance constraints may or may not have deadlines.

    As an example, we consider the display of video frames and the accompanying audio when the video is that of a person speaking. To have lip synchronization, the time between the display of each frame and the generation of the corresponding audio segment must be no more than 160 msec. In a task graph, temporal distance constraints among jobs are represented by temporal dependency edges.

27. What are AND/OR Precedence Constraints?

A job with more than one immediate predecessor must wait until all its immediate predecessors have been completed before its execution can begin. Whenever it is necessary to be specific, we call such jobs AND jobs and dependencies among them AND precedence constraints. AND jobs are represented by unfilled circles in the task graph in Figure 3–1.

An example is the job labeled J in this figure. All three of its immediate predecessors must be completed before J can begin execution. An AND job such as J may be the transmission of a message to a user. Its immediate predecessors are the jobs that set up a connection for the message, encrypt the message to safeguard privacy, and check the user's account for the requested quality of the delivery service. These predecessors may execute in any order.

In contrast, an OR job is one which can begin execution at or after its release time provided one or some of its immediate predecessors has been completed.

28. What is in-type of a job?
In the task graph, the in-type of job (i.e., the vertex representing the job) tells us whether all its immediate predecessors must complete before its execution can begin. By default, the value of this job parameter is AND. It can have the value OR, if only one of its immediate predecessors must be completed, or k-out-of-l, if only k out l of its immediate predecessor must be completed before its execution can begin.

29. What are Conditional Branches?
All the immediate successors of a job must be executed; an outgoing edge from every vertex expresses an AND constraint. This convention makes it inconvenient for us to represent conditional execution of jobs.

This system can easily be modeled by a task graph that has edges expressing OR constraints. Only one of all the immediate successors of a job whose outgoing edges express OR constraints is to be executed. Such a job is called a branch job. In a meaningful task graph, there is a join job associated with each branch job.

30. What is out-type of a job?
Similar to the parameter in-type, the job parameter out-type tells whether all the job's immediate successors are to be executed. The default value of the out-type parameter of every job is AND, that is, all its immediate successors are to be executed. On the other hand, the out-type of a branch job is OR, because only one of its immediate successors must be executed.

31. What is a Pipeline Relationship?
A dependency between a pair of producer-consumer jobs that are pipelined can theoretically be represented by a precedence graph. In this graph, the vertices are the granules of the producer and the consumer. Each granule of the consumer can begin execution when the previous granule of this job and the corresponding granule of the producer job have completed. In the task graph, we represent a pipeline relationship between jobs by a pipeline edge.

32. What are the Functional Parameters?
While scheduling and resource access-control decisions are made disregarding most functional characteristics of jobs, several functional properties do affect these decisions.

33. Write about Preemptivity, Nonpreemptivity of Jobs?
Executions of jobs can often be interleaved. The scheduler may suspend the execution of a less urgent job and give the processor to a more urgent job. Later when the more urgent job completes, the scheduler returns the processor to the less urgent job so the job can resume execution. This interruption of job execution is called preemption. A job is preemptable if its execution can be suspended at any time to allow the execution of other jobs and, later on, can be resumed from the point of suspension. A job is nonpreemptable if it must be executed from start to completion without interruption.

34. What is context switch?

In the case of CPU jobs, the state of the preempted job includes the contents of its program counter, processor status register, and registers containing temporary results. After saving the contents of these registers in memory and before the preempting job can start, the operating system must load the new processor status register, clear pipelines, and so on.These actions are collectively called a context switch. The amount of time required to accomplish a context switch is called a context-switch time. The fact that a job is nonpreemptable is treated as a constraint of the job.

35. What is Criticality of Jobs?
    In any system, jobs are not equally important. The importance (or criticality) of a job is a positive number that indicates how critical the job is with respect to other jobs; the more critical the job, the larger its importance.

    For example, in a flight control and management system, the job that controls the flight of the aircraft is more critical than the navigation job that determines the current position relative to the chosen course. The navigation job is more critical than the job that adjusts the course and cruise speed in order to minimize fuel consumption.

36. What are Optional Executions?
    It is often possible to structure an application so that some jobs or portions of jobs are optional. If an optional job or an optional portion of a job completes late or is not executed at all, the system performance may degrade, but nevertheless function satisfactorily. In contrast, jobs and portions of jobs that are not optional are mandatory; they must be executed to completion.

    Therefore, during a transient overload when it is not possible to complete all the jobs in time, we may choose to discard optional jobs so that the mandatory jobs can complete in time.

    Eg: We may consider the job that computes the correct evasive action and informs the operator of this action optional. Normally, we want the system to help the operator by choosing the correct action. However, in the presence of a failure and when the system is operating in a degraded mode, it is not possible to complete this computation in time. The collision avoidance system may still function satisfactorily if it skips this computation as long as it generates a warning and displays the course of the object about to collide with it in time.

37. What is Laxity Type and Laxity Function
    The laxity type of a job indicates whether its timing constraints are soft or hard. The laxity type of a job is sometimes supplemented by a usefulness function. This function gives the usefulness of the result produced by the job as a function of its tardiness. The usefulness of the result becomes zero or negative as soon as the job is tardy.

38. Write about Preemptivity of Resources.
    The resource parameters of jobs give us a partial view of the processors and resources from the perspective of the applications that execute on them. A resource parameter is preemptivity. A resource is nonpreemptable if each unit of the resource is constrained to be used serially. The resource parameters of each job give us the type of processor and the units of each resource type required.

    If jobs can use every unit of a resource in an interleaved fashion, the resource is preemptable. The lock on a data object in a database is an example of nonpreemptable resource. When a job modeling a transaction that reads and writes the data object has acquired the lock, other jobs that also require this lock at the time must wait. The lock is a nonpreemptable resource and, consequently, every transaction job is nonpreemptable in its use of the lock.

39. What is a Resource Graph?
    We can describe the configuration of the resources using a resource graph. In a resource graph, there is a vertex R_i for every processor or resource R_i in the system. The attributes of the vertex are the parameters of the resource. In particular, the resource type of a resource tells us whether the resource is a processor or a (passive) resource, and its number gives us the number of available units.

Edges in a resource graph represent the relationship among resources. Using different types of edges, we can describe different configurations of the underlying system. There are two types of edges in resource graphs. An edge from a vertex R_i to another vertex R_k can mean that R_k is a component of R_i . This edge is an is-a-part-of edge. Clearly, the subgraph containing all the is-a-part-of edges is a forest. The root of each tree represents a major component which contains subcomponents represented by vertices in the tree.

Some edges in resource graphs represent connectivity between components. These edges are called accessibility edges. If there is a connection between two CPUs in the two computers, then each CPU is accessible from the other computer, and there is an accessibility edge from each computer to the CPU on the other computer. A parameter of an accessibility edge from a processor Pi to another Pk is the cost for sending a unit of data from a job executing on Pi to a job executing on Pk .

40. What is a Scheduler and what are Schedules?
Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithmsand resource access-control protocols. The module which implements these algorithms is called the scheduler.

By a schedule, we mean an assignment of all the jobs in the system on the available processors produced by the scheduler.

41. What is Correctness of a schedule and what is a Valid schedule?
By correctness, we mean that the scheduler produces only valid schedules; a valid schedule satisfies the following conditions:
    1. Every processor is assigned to at most one job at any time.
    2. Every job is assigned at most one processor at any time.
    3. No job is scheduled before its release time.
    4. Depending on the scheduling algorithm(s) used, the total amount of processor time assigned to every job is equal to its maximum or actual execution time.
    5. All the precedence and resource usage constraints are satisfied.

42. What is Feasibility, Optimality, and Performance Measures
A valid schedule is a **feasible schedule** if every job completes by its deadline. We say that a set of jobs is schedulable according to a scheduling algorithm if when using the algorithm the scheduler always produces a feasible schedule. A hard real-time scheduling algorithm is optimal if (using) the algorithm (the scheduler) always produces a feasible schedule if the given set of jobs has feasible schedules.

In addition to the criterion based on feasibility, other commonly used performance measures include the maximum and average tardiness, lateness, and response time and the miss, loss, and invalid rates.

43. What is Lateness of a job?
The **lateness** of a job is the difference between its completion time and its deadline. Unlike the tardiness of a job which never has negative values, the lateness of a job which completes early is negative, while the lateness of a job which completes late is positive.

44. What is Response time of a job?
The **response time** of this job is the response time of the set of jobs as a whole and is often called the makespan of the schedule. This is a performance criterion commonly used to compare scheduling algorithms in classical scheduling.

45. What is Miss rate and Lost Rate?
For many soft real-time applications, it is acceptable to complete some jobs late or to discard late jobs. For such an application, suitable performance measures include the miss rate and loss rate. The former gives the percentage of jobs that are executed but completed too late, and the latter give the percentage of jobs that are discarded, that is, not executed at all.

When it is impossible to complete all the jobs on time, a scheduler may choose to discard some jobs. By doing so, the scheduler increases the loss rate but completes more jobs in time. Thus, it reduces the miss rate. Similarly, reducing the loss rate may lead to an increase in miss rate. A performance measure, invalid rate, is the sum of the miss and loss rates and gives the percentage of all jobs that do not produce a useful result.

46. Mention the reasons for existence of a Scheduling hierarchy.
A system typically has a hierarchy of schedulers. This scheduling hierarchy arises for two reasons-
First, some processors and resources used by the application system are not physical entities; they are logical resources. Logical resources must be scheduled on physical resources. A scheduler that schedules a logical resource may be different from the scheduler that schedules the application system using the resource.

Second, a job may model a server that executes on behalf of its client jobs. The time and resources allocated to the server job must in turn be allocated to its client jobs. Again, the algorithm used by the server to schedule its clients may be different from the algorithm used by the operating system to schedule the server with other servers.

47. Describe the Interaction among Schedulers or Give an example for Hierarchy between Schedulers.
As an example of servers, we consider an application system containing periodic tasks and aperiodic jobs on one processor. All the aperiodic jobs are placed in a queue when they are released. There is a poller. Together with the periodic tasks, the poller is scheduled to execute periodically. When the poller executes, it checks the aperiodic job queue. If there are aperiodic jobs waiting, it chooses an aperiodic job from the queue and executes the job. Hence, the aperiodic jobs are the clients of the poller. We again have two levels of scheduling. In the lower level the scheduler provided by the operating system schedules the poller and the periodic tasks. In the higher level, the poller schedules its clients.

Unit - 1
Commonly Used Approaches to Real-Time Scheduling

1. What is Clock-driven Approach?
   As the name implies, when scheduling is *clock-driven* (also called *time-driven*), decisions on what jobs execute at what times are made at specific time instants. These instants are chosen a priori before the system begins execution.

   Typically, in a system that uses clock-driven scheduling, all the parameters of hard real-time jobs are fixed and known. In this way, scheduling overhead during run-time can be minimized. A frequently adopted choice is to make scheduling decisions at regularly spaced time instants.

   One way to implement such scheduler is to use a hardware timer. The timer is set to expire periodically without the intervention of the scheduler. When the system is initialized, the scheduler selects and schedules the job(s) that will execute until the next scheduling decision time and then blocks itself waiting for the expiration of the timer. When the timer expires, the scheduler awakes and repeats these actions.

2. What is Weighted Round-Robin approach?
   The *weighted round-robin algorithm* has been used for scheduling real-time traffic in high-speed switched networks. It builds on the basic round-robin scheme. Rather than giving all the ready jobs equal shares of the processor, different jobs may be given different *weights*. Here, the weight of a job refers to the fraction of processor time allocated to the job. A job with weight *wt* gets *wt* time slices every round.

   By giving each job a fraction of the processor, a round-robin scheduler delays the completion of every job. If it is used to schedule precedence constrained jobs, the response time of a chain of jobs can be unduly large. For this reason, the weighted round-robin approach is not suitable for scheduling such jobs. On the other hand, a successor job may be able to incrementally consume what is produced by a predecessor. In this case, weighted round-robin scheduling is a reasonable approach, since a job and its successors can execute concurrently in a pipelined fashion.

3. What is Priority-driven approach?
   The term *priority-driven algorithms* refers to a large class of scheduling *algorithms that never leave any resource idle intentionally*. Scheduling decisions are made when events such as releases and completions of jobs occur. Hence, priority-driven algorithms are *event-driven*.

4. What are the different used for Priority-driven approach?
   Other commonly used names for this approach are *greedy scheduling*, *list scheduling* and *work-conserving scheduling*. A priority-driven algorithm is *greedy* because it tries to make locally optimal decisions. The term *list scheduling* is also descriptive because any priority-driven algorithm can be implemented by assigning priorities to jobs. Jobs ready for execution are placed in one or more queues ordered by the priorities of the jobs. A priority-driven scheduling algorithm is defined to a great extent by the list of priorities it assigns to jobs.

5. Give examples of Priority-driven approach.
   Examples include the FIFO (First-In-First-Out) and LIFO (Last-In-First-Out) algorithms, which assign priorities to jobs according their release times, and the SETF (Shortest-Execution-Time-First) and LETF (Longest-Execution-Time-First) algorithms.

6. What is makespan of an algorithm?
   Makespan is the response time of the job that completes last among all jobs. In the special case when jobs have the same release time, preemptive scheduling is better when the cost of preemption is ignored. Specifically, in a multiprocessor system, the minimum makespan achievable by an optimal preemptive algorithm is shorter than the makespan achievable by an optimal nonpreemptive algorithm.

   When there are two processors, the minimum makespan achievable by nonpreemptive algorithms is never more than 4/3 times the minimum makespan achievable by preemptive algorithms when the cost of preemption is negligible.

7. What is a Dynamic system?
   Jobs that are ready for execution are placed in a priority queue common to all processors. When a processor is available, the job at the head of the queue executes on the processor. We will refer to such a multiprocessor system as a dynamic system, because jobs are dynamically dispatched to processors. Here, we allow each preempted job to resume on any processor and hence, jobs are migratable. We say that a job **migrates** if it starts execution on a processor, is preempted, and later resumes on a different processor.

8. What is a Static system?
   Another approach to scheduling in multiprocessor and distributed systems is to partition the jobs in the system into subsystems and assign and bind the subsystems statically to the processors. Jobs are moved among processors only when the system must be reconfigured, that is, when the operation mode of the system changes or some processor fails. Such a system is called a *static system*, because the system is *statically configured*.

   "Most hard real-time systems built today are static. So, in such case, we need to use Clock Driven and Weighted Round Robin scheduling. If we have proper validating techniques, then we can use an online algorithm like Priority driven scheduling to schedule the jobs of a real time task."

9. What are Derived constraints?
   The given release times and deadlines of jobs are sometimes inconsistent with the precedence constraints of the jobs. Therefore, rather than working with the given release times and deadlines, we first derive a set of effective release times and deadlines from these timing constraints, together with the given precedence constraints. The derived timing constraints are consistent with the precedence constraints.

10. How to compute Derived constraints?
    When there is only one processor, we can compute the derived constraints according to the following rules:

    Effective Release Time: The effective release time of a job without predecessors is equal to its given release time. The effective release time of a job with predecessors is equal to the maximum value among its given release time and the effective release times of all of its predecessors.

    Effective Deadline: The effective deadline of a job without a successor is equal to its given deadline. The effective deadline of a job with successors is equal to the minimum value among its given deadline and the effective deadlines of all of its successors.

    The effective release times of all the jobs can be computed in one pass through the precedence graph in $O(n^2)$ time where n is the number of jobs. Similarly, the effective deadlines can be computed in $O(n^2)$ time.

11. What is Effective Release Time?
    Effective Release Time: The effective release time of a job without predecessors is equal to its given release time. The effective release time of a job with predecessors is equal to the maximum value among its given release time and the effective release times of all of its predecessors.

12. What is Effective Deadline?
    Effective Deadline: The effective deadline of a job without a successor is equal to its given deadline. The effective deadline of a job with successors is equal to the minimum value among its given deadline and the effective deadlines of all of its successors.

    More accurately, the effective deadline of a job should be as early as the deadline of each of its successors minus the execution time of the successor. The effective release time of a job is that of its predecessor plus the execution time of the predecessor. The more accurate calculation is unnecessary, however, when there is only one processor.

13. Give the statement for EDF algorithm.
    A way to assign priorities to jobs is on the basis of their deadlines. In particular, the earlier the deadline, the higher the priority. The priority-driven scheduling algorithm based on this priority assignment is called the Earliest-Deadline-First (EDF) algorithm. This algorithm is important because it is optimal when used to schedule jobs on a processor as long as preemption is allowed and jobs do not contend for resources. This fact is stated formally below.

14. Write about the LRT algorithm.
We may want to postpone the execution of hard real-time jobs for some reason. So, we sometimes also use the *latest release time (LRT)* algorithm (or reverse EDF algorithm). This algorithm treats release times as deadlines and deadlines as release times and schedules the jobs backwards, starting from the latest deadline of all jobs, in "priority-driven" manner, to the current time. In particular, the "priorities" are based on the release times of jobs: the later the release time, the higher the "priority." Because it may leave the processor idle when there are jobs ready for execution, the LRT algorithm is not a priority-driven algorithm.

15. Write about the LST algorithm.
Another algorithm that is optimal for scheduling preemptive jobs on one processor is the *Least-Slack-Time-First* (*LST*) algorithm (also called the *Minimum-Laxity-First* (*MLF*) algorithm). At any time *t* , the *slack* (or *laxity*) of a job with deadline at *d* is equal to *d* − *t* minus the time required to complete the remaining portion of the job. Take the job J_1 in Figure 4–5 as an example. It is released at time 0, its deadline is 6, and its execution time is 3. Hence, its slack is equal to 3 at time 0. The job starts to execute at time 0. As long as it executes, its slack remains at 3, because at any time *t* before its completion, its slack is 6 − *t* − (3 − *t*). Now suppose that it is preempted at time 2 by J_3, which executes from time 2 to 4. During this interval, the slack of *J*1 decreases from 3 to 1. (At time 4, the remaining execution time of *J*1 is 1, so its slack is 6 − 4 − 1 = 1.)

The LST algorithm assigns priorities to jobs based on their slacks: the smaller the slack, the higher the priority.

16. What is Slack of ajob? Give example.
At any time t , the slack (or laxity) of a job with deadline at d is equal to d − t minus the time required to complete the remaining portion of the job.
Eg: Take a job J_1 as an example. It is released at time 0, its deadline is 6, and its execution time is 3. Hence, its slack is equal to 3 at time 0. The job starts to execute at time 0. As long as it executes, its slack remains at 3, because at any time t before its completion, its slack is 6 − t − (3 − t). Now suppose that it is preempted at time 2 by J3, which executes from time 2 to 4. During this interval, the slack of J1 decreases from 3 to 1. (At time 4, the remaining execution time of J1 is 1, so its slack is 6 − 4 − 1 = 1.)

17. Why validation is required? State the Validation problem.
The timing behavior of a priority-driven system is nondeterministic when job parameters vary. Consequently, it is difficult to validate that the deadlines of all jobs scheduled in a prioritydriven manner indeed meet their deadlines when the job parameters vary.

In general, this *validation problem* can be stated as follows: "Given a set of jobs, the set of resources available to the jobs, and the scheduling (and resource access-control) algorithm to allocate processors and resources to jobs, determine whether all the jobs meet their deadlines".

18. What is Scheduling Anomaly?
Scheduling anomaly *is* an unexpected timing behavior of priority-driven systems. When jobs are nonpreemptable, scheduling anomalies can occur even when there is only one processor. It makes the problem of validating a priority-driven system difficult whenever job parameters may vary.

19. What is maximal schedule and minimal schedule?
The schedule of J produced by the given scheduling algorithm when the execution time of every job has its maximum value is a *maximal schedule* of J. Similarly, the schedule of J produced by the given scheduling algorithm when the execution time of every job has its minimum value is the *minimal schedule*. When the execution time of every job has its actual value, the resultant schedule is the *actual schedule* of J.

20. When is a job predictable.
"The execution of J under the given priority-driven scheduling algorithm is predictable if the actual start time and actual completion time of every job according to the actual schedule are bounded by its start times and completion times according to the maximal and minimal schedules".

Let $s+(J_i)$ and $s-(J_i)$ be the start times of J_i according to the maximal schedule and minimal schedule of J, respectively. We say that *Ji* is ***start-time predictable*** if $s-(J_i) \le s(J_i) \le s+(J_i)$.

Let *$f^+(J_i)$* and *$f^-(J_i)$* be the completion times of J_i according to the maximal schedule and minimal schedule of J, respectively. We say that J_i is ***completion-time predictable*** if $f^-(J_i) \leq f(J_i) \leq f^+(J_i)$.

The execution of J_i is *predictable*, or simply J_i is ***predictable***, if J_i is both start-time and completion-time predictable.

21. What is a Validation algorithm?
A *Validation algorithm* allows us to determine whether all jobs in a system indeed meet their timing constraints despite scheduling anomalies. While there are mature validation algorithms and tools for static systems, good validation algorithms for dynamic, priority-driven systems are not yet available.

22. When is a Validation algorithm Correct, Good, Robust and Inaccurate?
Specifically, a *validation algorithm is correct* if it never declares that all timing constraints are met when some constraints may not be. The merits of (correct) validation algorithms are measured in terms of their complexity, robustness, and accuracy.

    A *validation algorithm is good* when it achieves a good balance in performance according to these conflicting figures of merit.

    A *validation algorithm is robust* if it remains correct even when some assumptions of its underlying workload model are not valid. The use of a robust validation algorithm significantly reduces the need for an accurate characterization of the applications. Efficiency and robustness can be achieved easily if we are not concerned with the accuracy of the validation test.

    A *validation algorithm is inaccurate* when it is overly pessimistic and declares tasks unable to meet their timing constraints except when system resources are unduly underutilized. A scheduler using an inaccurate validation algorithm for an acceptance test may reject too many new tasks which are in fact acceptable.

23. Write about off-line scheduling.
A static schedule is computed off-line before the system begins to execute, and the computation is based on the knowledge of the release times and processor-time/resource requirements of all the jobs for all times. When the operation mode of the system changes, the new schedule specifying when each job in the new mode executes is also precomputed and stored for use. So, the scheduling is done off-line, and the precomputed schedules are *off-line schedules*.

    An obvious disadvantage of off-line scheduling is inflexibility. This approach is possible only when the system is deterministic, meaning that the system provides some fixed set(s) of functions and that the release times and processor-time/resource demands of all its jobs are known and do not vary or vary only slightly.

24. Write about the On-Line Scheduling.
We say that scheduling is done *on-line*, or that we use an *on-line scheduling algorithm*, if the scheduler makes each scheduling decision without knowledge about the jobs that will be released in the future; the parameters of each job become known to the on-line scheduler only after the job is released. The priority-driven algorithms described earlier and in subsequent chapters are on-line algorithms. Such an acceptance test is on-line.

    On-line scheduling is the only option in a system whose future workload is unpredictable. An on-line scheduler can accommodate dynamic variations in user demands and resource availability. The price of the flexibility and adaptability is a reduced ability for the scheduler to make the best use of system resources.

25. Write about Overloaded systems.
The system is said to be *overloaded* when the jobs offered to the scheduler cannot be feasibly scheduled even by a clairvoyant scheduler. When the system is not overloaded, an optimal on-line scheduling algorithm is one that always produces a feasible schedule of all offered jobs.

    During an overload, some jobs must be discarded in order to allow other jobs to complete in time. A reasonable way to measure the performance of a scheduling algorithm during an overload is by the amount of work the scheduler can feasibly schedule according to the algorithm: the larger this amount, the better the algorithm. In general, all on-line scheduling algorithms perform rather poorly when the system gets overloaded.

26. What is the Competetive factor of an algorithm?

The *competitive factor of an algorithm* captures this aspect of performance. To define this performance measure, we say that the *value of a job* is equal to its execution time if the job completes by its deadline according to a given schedule and is equal to zero if the job fails to complete in time according to the schedule.

The *value of a schedule* of a sequence of jobs is equal to the sum of the values of all the jobs in the sequence according to the schedule. A scheduling algorithm is optimal if it always produces a schedule of the maximum possible value for every finite set of jobs.

An on-line algorithm has a competitive factor c if and only if the value of the schedule of any finite sequence of jobs produced by the algorithm is at least *c* times the value of the schedule of the jobs produced by an optimal clairvoyant algorithm.

27. What is the Value of a schedule?
The *value of a schedule* of a sequence of jobs is equal to the sum of the values of all the jobs in the sequence according to the schedule. A scheduling algorithm is optimal if it always produces a schedule of the maximum possible value for every finite set of jobs.

28. What is Load Ratio and Loading Factor?
The *load ratio* of the interval [*t, t*] : It is the ratio of the total execution time of all jobs whose feasible intervals are contained in the interval [*t, t*] to the length *t* − *t* of the interval. A system is said to have a *loading factor X* if the load ratio of the system is equal to or less than *X* for all intervals.