Real-Time Systems

Real-Time Systems: A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period

- A real-time system is one in which the correctness of the computations not only depends on their logical correctness, but also on the time at which the result is produced.
- Failure to respond is as bad as the wrong response.
- A real-time system changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped.
- Real-Time systems are becoming pervasive. Many embedded systems are referred to as real-time systems.
- Telecommunications, flight control and electronic engines, Networked Multimedia Systems, Command Control Systems etc are some of the popular real-time system applications where as computer simulation, user interface and Internet video are categorized as non-real time applications.

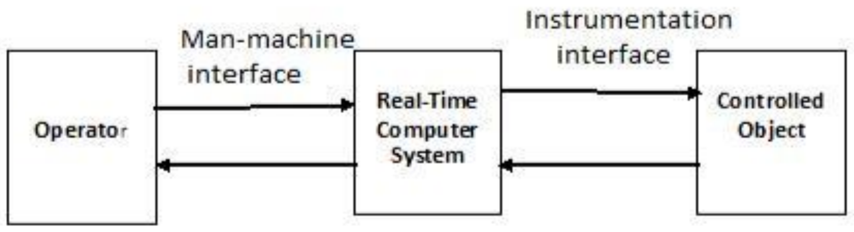
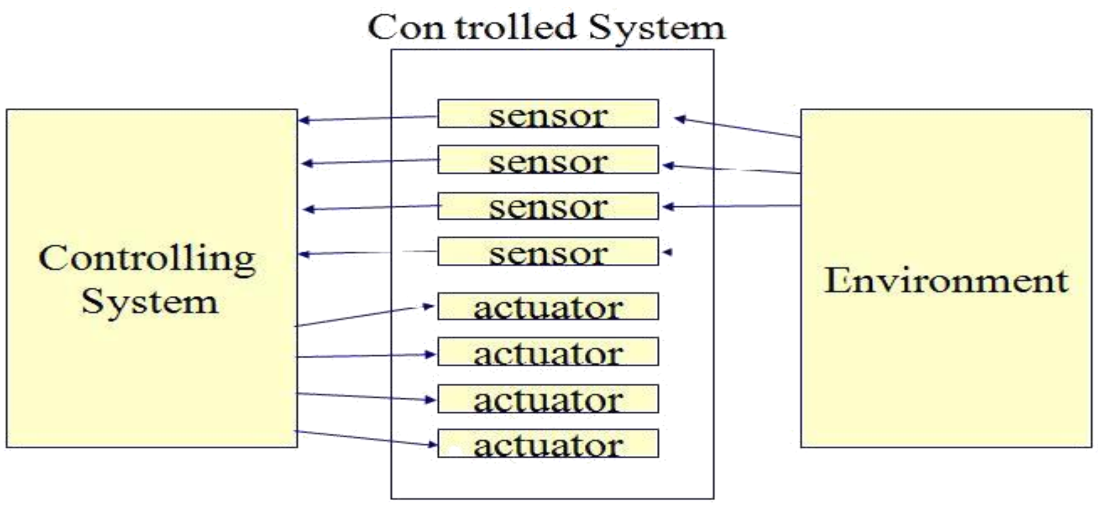


Fig1: Real-Time System

- Real-time systems often are comprised of a *controlling* system, *controlled* system and *environment*.

  Controlling system: acquires information about environment using *sensors* and controls the environment with *actuators*.



- *Timing constraints* derived from *physical* impact of controlling systems activities. Hard and soft constraints.

  Periodic Tasks : Time-driven recurring at regular intervals.

  Aperiodic Tasks : event-driven

Needs of the Real-Time Systems:

- Fast context switches?
  - should be fast anyway(Must be fast response time)
- Small size?
  - should be small anyway(Portable)
- Quick response to external triggers?
  - not necessarily quick but predictable
- Multitasking?
  - often used, but not necessarily
- "Low Level" programming interfaces?
  - might be needed as with other embedded systems
- High processor utilisation?
  - desirable in any system (avoid oversized system)

Typical Real-Time Applications:

A real-time system is required to complete its work and deliver its services on a timely basis. Examples of real-time systems include digital control, command and control, signal processing, and telecommunication systems.

Digital Control: Many real-time systems are digital control, systems, they are embedded in sensors and actuators and function as digital controllers (shown in fig). The term plant in the block diagram refers to a controlled system, for example, an engine, a brake, an aircraft, a patient. The state of the plant is monitored by sensors and can be changed by actuators.

- The real-time (computing) 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.

A Simple Example:

Consider an analog single-input/single-outputPID (Proportional, Integral, and Derivative) controller. This simple kind of controller is com-monly used in practice. The analog sensor reading *y(t)* gives the measured state of the plant at time *t* .

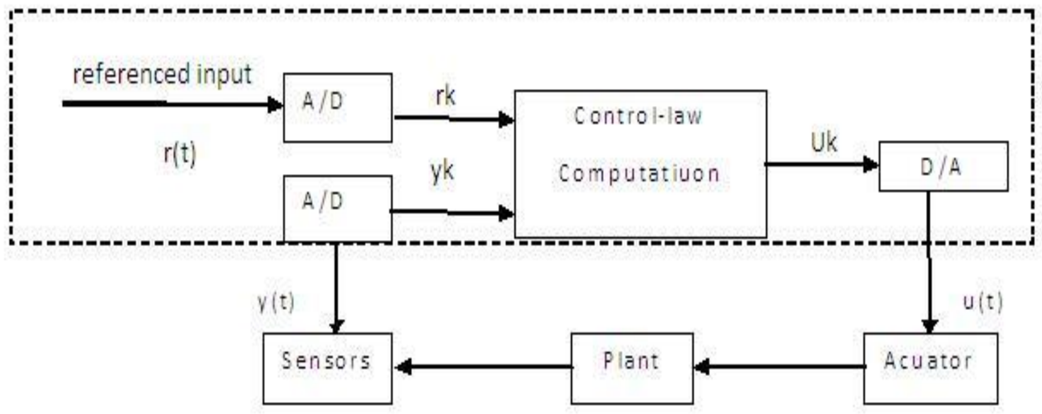


Fig : A Digital Controller

`

- 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.
- During any sampling period (say the *k*th), the control output u_k depends on the current and past measured values y_i for $i \leq k$. The future measured values y_i ’s for $i>k$ in turn depend on u_k . Such a system is called a *(feedback) control loop* or simply a *loop*. We can implement it as an infinite timed 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;
```

Sampling Period: The length T of time between any two consecutive instants at which y(t) and r(t) are sampled is called the sampling period. T is a key design choice. The behavior of the resultant digital controller critically depends on this parameter.

- We consider two factors. The first is the perceived responsiveness of the overall system. The second factor is the dynamic behavior of the plant.

High-level Control: 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. With few exceptions, one or more of the higher-level controllers interfaces with the operator.

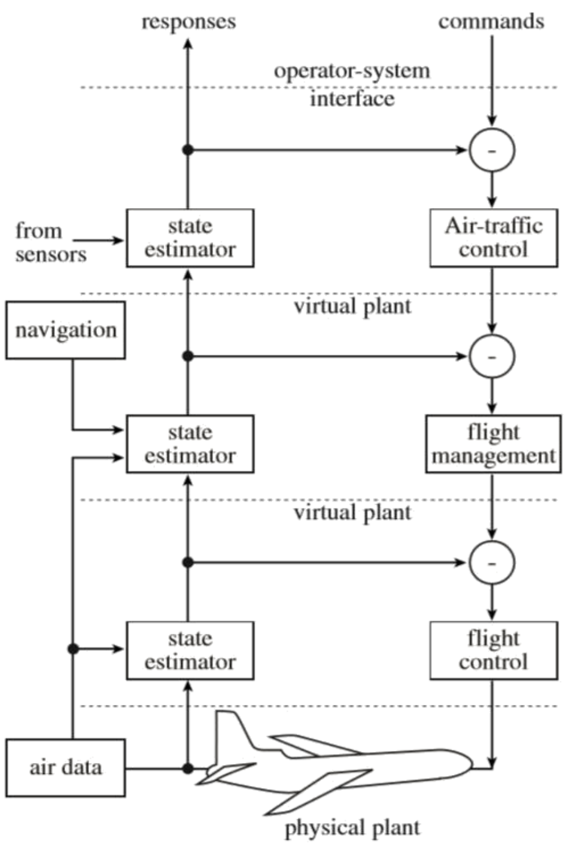
Examples of Control Hierarchy

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.

- The computation done by each digital controller is simple and nearly deterministic; the computation of a high-level controller is likely to be far more complex and variable.
- While the period of a low-level control-law computation ranges from milliseconds to seconds, the periods of high-level control-law computations may be minutes, even hours.

Flight Control System:

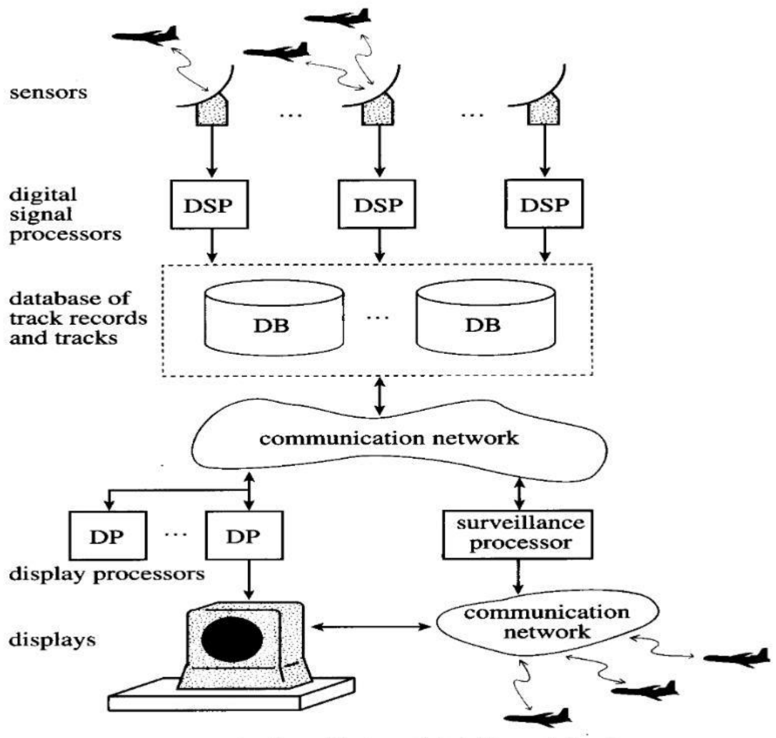
Below figure shows hierarchy of flight control, flight management and air traffic control systems.



- The Air Traffic Control (ATC) system is at the highest level.
- It regulates the flow of flights to each destination airport.
- It does so by assigning to each aircraft an arrival time at each metering fix (known geographical point, adjacent points are 40-60 miles apart) in the route to destination.
- The aircraft is supposed to arrive at the metering fix at the assigned time.
- At any time while in flight, the assigned arrival time to the next metering fix is a reference input to the on-board flight management system.
- The flight management system chooses a time-referenced flight path that brings the aircraft to the next metering fix at the assigned arrival time.
- The cruise speed, turn radius, descend/ascend rates and so for required to the chosen time-referenced flight path are the reference inputs to the flight controller at the lowest level of hierarchy.

- The controller at the highest level of a control hierarchy is a command and control system.
- An Air Traffic Control (ATC) system is an excellent example.
- The ATC system monitors the aircraft in its coverage area and the environment (e.g, weather condition) and generates and presents the information needed by the operators.
- Outputs from the ATC system include the assigned arrival times to metering fixes for individual aircraft.

- These outputs are reference inputs to on-board flight management systems. Thus, the ATC system indirectly controls the embedded components in low levels of the control hierarchy.
- In addition, the ATC system provides voice and telemetry links to on-board avionics.
- Thus it supports the communication among the operators at both levels (i.e., the pilots and air traffic controllers).
- The ATC system gathers information on the “state” of each aircraft via one or more active radars. Such a radar interrogates each aircraft periodically. When interrogated, an aircraft responds by sending to the ATC system its “state variables”: identifier, position, altitude, heading, and so on.
- The ATC system processes messages from aircraft and stores the state information thus obtained in a database.
- This information is picked up and processed by display processors. At the same time, a surveillance system continuously analyzes the scenario and alerts the operators whenever it detects any potential hazard (e.g., a possible collision).
- Again, the rates at which human interfaces (e.g., keyboards and displays) operate must be at least 10 Hz. The other response times can be considerably larger.
- For example, the allowed response time from radar inputs is one to two seconds, and the period of weather updates is in the order of ten seconds. And we can see that a command and control system bears little resemblance to low-level controllers.



An architecture of air traffic control system.

- In contrast to a low-level controller whose workload is either purely or mostly periodic, a command and control system also computes and communicates in response to sporadic events and operators' commands.
- Furthermore, it may process image and speech, query and update databases, simulate various scenarios, and the like. The resource and processing time demands of these tasks can be large and varied.
- A low-level control system typically runs on one computer or a few computers connected by a small network or dedicated links, a command and control system is often a large distributed system containing tens and hundreds of computers and many different kinds of networks.

SIGNAL PROCESSING:

Anything that carries information is called as signals. It is a real or complex value function of one or more variables. For example: temperature is one dimensional signal, image is 2-dimensional.Processing means operating in some fashion on a signal to extract some useful information. The signal is processed by system which can be electronic, mechanical or a program.

Most signal processing applications have some kind of real-time requirements. We focus here on those whose response times must be under a few milliseconds to a few seconds. Examples are digital filtering, video and voice compressing/decompression, and radar signal processing.

Processing Bandwidth Demands:

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

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

In the simplest case, the weights, *a(k,i)*'s, are known and fixed. In essence, this computation transforms the given representation of an object (e.g., a voice, an image or a radar signal) in terms of the inputs, *y(i)*'s, into another representation in terms of the outputs, *x(k)*'s. Different sets of weights, *a(k,i)*'s, give different kinds of transforms. This expression tells us that the time required to produce an output is *O(n)*.

Radar System

- A signal processing application is typically a part of a larger 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, which not only interface with the display system, but also generate commands to control the radar and select parameters to be used by signal processors in the next cycle of data collection and analysis.

Radar Signal Processing:

- To search for objects of interest in its coverage area, theradar scans the area by pointing its antenna in one direction at a time.
- During the time the antenna dwells in a direction, it first sends a short radio frequency pulse. It then collects and examines the echo signal returning to the antenna.
- The echo signal consists solely of background noise if the transmitted pulse does not hit any object.
- On the other hand, if there is a reflective object (e.g., an airplane or storm cloud) at a distance *x* meters from the antenna, the echo signal reflected by the object returns to the antenna at approximately $2x/c$ seconds after the transmitted pulse, where $c = 3 \times 10^8$ meters per second is the speed of light.

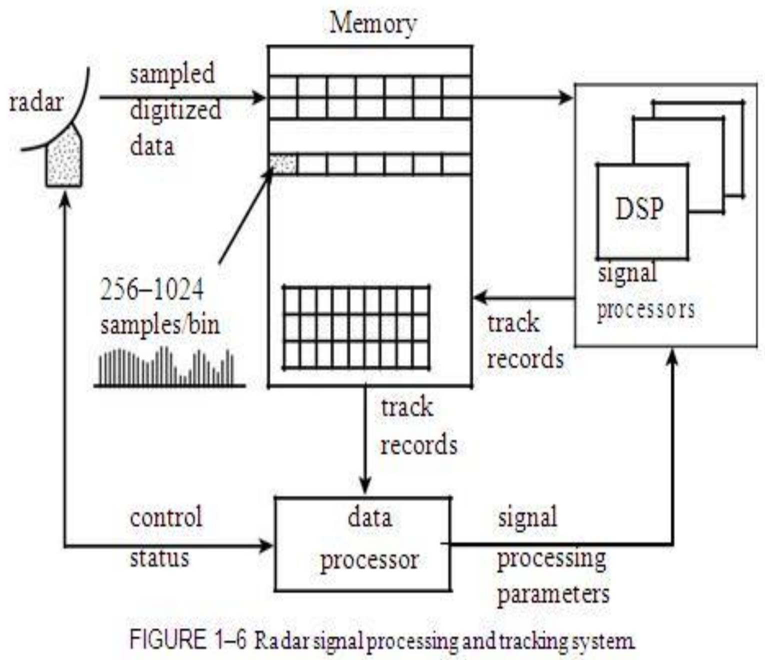


FIGURE 1–6 Radar signal processing and tracking system

Tracking:

- Noise and man-made interferences, including electronic countermeasure (i.e., jamming), can lead the signal processing and detection process to wrong conclusions about the presence of objects.
- A track record on a non existing 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*.

➢ 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.

Process:

- At the start, the tracker computes the predicted position (and velocity) of the object on each established trajectory.
- In this example, there are two established trajectories, L_1 and L_2. We also call the predicted positions of the objects on these tracks L_1 and L_2. X_1, X_2, and X_3 are the measured values given by three track records.
- x_1 is assigned to L_1 because it is within distance *G* from L_1. X_3 is assigned to both L_1 and $L_2$for the same reason.
- On the other hand, $X_2$is not assigned to any of the trajectories.
- Itrepresents either a false return or a new object. Since it is not possible to distinguish between these two cases, the tracker hypothesizes that X_2 is the position of a new object.
- Subsequent radar data will allow the tracker to either validate or invalidate this hypothesis. In the latter case, the tracker will discard this trajectory from further consideration.

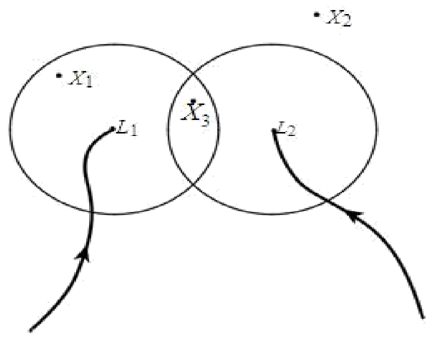


FIGURE 1–7 Gating process.

➢ Data Association:

- The tracking process completes if, after gating, every measuredvalue is assigned to at most one trajectory and every trajectory is assigned at most one measured value.
- This is likely to be case when (1) the radar signal is strong and interference is low (and hence false returns are few) and (2) the density of objects is low. Under adverse conditions, the assignment produced by gating may be ambiguous, that is, some measured value is assigned to more than one trajectory or a trajectory is assigned more than one measured value.
- The data association step is then carried out to complete the assignments and resolve ambiguities.

Other Real-Time Applications:

➢ **Real-time databases:**

- Transactions must complete by deadlines.
- **Main dilemma:** Transaction scheduling algorithms and real-time scheduling algorithms often have conflicting goals.
- Data may be subject to **absolute** and **relative temporal consistency** requirements.
- Overall goal: reliable responses

➢ **Multimedia:**

- Want to process audio and video frames at steady rates.
- TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
- Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec

Hard versus Soft Real-Time Systems:

➢ A real-time system can be further divided into soft and hard real-time system on the basis of severity of meeting its deadlines, timing constraints (response time , release time)

- A hard real-time system cannot afford missing even a single deadline. That is, it has to meet all the deadlines to be branded as a hard one.
- Soft real-time system takes the middle path between a non-real-time system and a hard real-time system. That is, it can allow occasional miss.

➢ Real-time systems can be **predictable** and **deterministic.**

- A predictable real-time system is one whose behavior is always within an acceptable range.
- A deterministic system is a special case of a predictable system. Not only is the timing behavior within a certain range, but that timing behavior can be predictable.

Hard real-time systems: A Real-Time System in which all deadlines are hard.Hard Real-Time System guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the to finish any request made of it.

- The requirement that all hard timing constraints must be validated invariably places many re-strictions on the design and implementation of hard real-time applications as well as on the architectures of hardware and system software used to support them.
- An overrun in response time leads to potential loss of life and big financial damage.
- Many of these systems are considered to be safety critical. Sometimes they are “only” mission critical, with the mission being very expensive.
- In a hard real-time system, the peak-load performance must be predictable and should not violate the predefined deadlines.
- Hard real-time systems are often safety critical even load is very high.
- Hard real-time systems have small data files and real-time databases.
- In general there is a cost function associated with the system.
- Strict about each task and its deadline.
- The preemption period (the delay it can handle max) should be very less(micro second)
- For example: Rocket launching, Nuclear Power Plant control, Flight Control System.

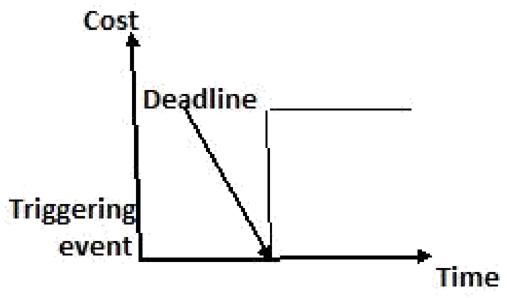


Fig : Hard Real-Time System

Some 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, 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.
- Jobs in some non-embedded 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.
- In recent years, this observation motivated a variety of approaches to soften hard dead-lines. Examples are to allow a few missed deadlines or premature terminations as long as they occur in some acceptable way.

More on Hard Timing Constraints:

There may be no advantage in completing a job with a hard deadline early. As long as the job completes by its deadline, its response time is not important. In fact, it is often advantageous, sometimes even essential, to keep the response times of a stream of jobs small. Hard and soft timing constraints allow a hard timing constraint to be specified in any terms. Examples are

- **Hard**: failure to meet constraint is a fatal fault. Validation system always meets timing constraints.
    - ✓ Deterministic constraints
    - ✓ Probabilistic constraints
    - ✓ Constraints in terms of some usefulness function.
- **Soft**: late completion is undesirable but generally not fatal. No validation or only demonstration job meets some statistical constraint. Occasional missed deadlines or aborted execution is usually considered tolerable. Often specified in probabilistic terms

<u>Soft real-time systems:</u> A real-time system in which some deadlines are soft. A Soft Real-Time Systems,in which jobs have soft deadlines is. The developer of a soft real-time system is rarely required to prove rigorously that the system surely meet its real-time performance objective. Examples of such systems include on-line transaction systems and telephone switches, as well as electronic games.

- In a soft real-time system, a degraded operation in a rarely occurring peak load can be tolerated. A hard real-time system must remain synchronous with the state of the environment in all cases.
- Soft real-time systems will slow down their response time if the load is very high.
- Deadline overruns are tolerable, but not desired.
- There are no catastrophic consequences of missing one or more deadlines.
- There is a cost associated to overrunning, but this cost may be abstract.
- The task and its deadline is manageable but we should met the condition most of the all the time.
- The preemption period for this can be more (around milisecond).
- For example : Washing machine , Stock price quotation System Mobile phone, digital cameras and orchestra playing robots.

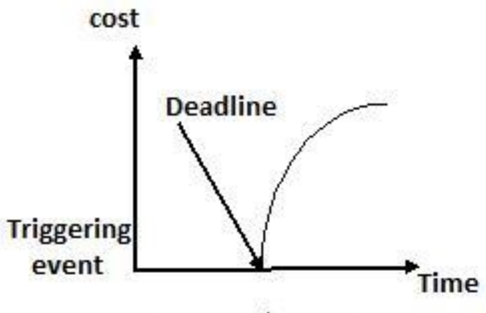


Fig: Soft Real-Time System

A Reference Model of Real-Time Systems:

- Modeling the system to focus on timing properties and resource requirements. Composed of three elements:
    - ➢ Workload model - describes applications supported by system
        - Temporal parameters
        - Functional parameters
    - ➢ Resource model - describes system resources available to applications
        - Modeling resources (Processors and Resources)
        - Resource parameters
    - ➢ Algorithms - defines how application uses resources at all times.
        - Scheduling Hierarchy

➢ Task (T_i): Set of related jobs jointly provide function.

➢ Job (J_{ij}): Unit of work, scheduled and executed by system. characterized by the following parameters:

i. Temporal parameters: timing constraints and behavior
ii. Functional parameters: intrinsic properties of the job.
iii. Interconnection parameters: how it depends on other jobs and how other jobs depend on it.
iv. Resource parameters: resource requirements.

➢ Processors and Resources: we can divide the system resources into two major types: processors and resources. 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 and make progress toward completion

- Resources can be divided into passive and active:
    - Active resources = *Processors* (P_i): they execute jobs.
        - Every job must have one or more processors
        - Same type if functionally identical and used interchangeably.
    - Passive resource = *Resource* (R_i):
        - Job may require Resources in addition to processor.
        - Reusable resources are not consumed

Task Temporal Parameters:

- Hard real-time: Number and parameters of tasks are known at all time. for Job J_i:

Release time (r_i) : is the time at which the job becomes ready for execution.

Absolute deadline(d_i): is the time at which the job should be completed.

Relative deadline(D_i): is the time length between the arrival time and the absolute deadline.

Start time (sj): is the time at which the job starts its execution.

Finishing time (fj): is the time at which the job finishes its execution.

Execution time (e_i): May know range [e^-, e^+]. Most deterministic models use e^+.

- **Periodic Task Model:** In periodic task, jobs are released at regular intervals. A periodic task is one which repeats itself after a fixed time interval. A periodic task is denoted by five tuples: Ti = < Φi, Pi, ei, Di >

  Where,

  Task T_i is a serious of periodic Jobs J_{ij}.

  φ_i - phase of Task T_i, equal to r_{i1}.

  p_i - period, minimum inter-release interval between jobs in Task T_i. Must be bounded from below.

  e_i - maximum execution time for jobs in task T_i.

  Di – is the relative deadline of the task.

  r_{ij} - release time of the j^{th} Job in Task i (J_{ij} in T_i).

  H –Hyperperiod = Least Common Multiple of p_i for all i:

  H = lcm(p_i), for all i.

  u_i - utilization of Task T_i.

  U - Total utilization = Sum over all u_i.

1. Aperiodic Tasks: In this type of task, jobs are released at arbitrary time intervals i.e. randomly. Aperiodic tasks have soft deadlines or no deadlines.

2. Sporadic Tasks: They are similar to aperiodic tasks i.e. they repeat at random instances. The only difference is that sporadic tasks have hard deadlines. A sporadic task is denoted by three tuples: Ti =(ei, gi, Di)

Where

ei – the execution time of the task.

gi – the minimum separation between the occurrence of two consecutive instances of the task. Di – the relative deadline of the task.

Jitter: Sometimes actual release time of a job is not known. Only know that ri is in a range [ri-, ri+]. This range is known as release time jitter. Here ri– is how early a job can be released and ri+ is how late a job can be released. Only range [ei-, ei+] of the execution time of a job is known. Here ei– is the minimum amount of time required by a job to complete its execution and ei+ the maximum amount of time required by a job to complete its execution.

Functional Parameters:

While scheduling and resource access control decision are being made certain functional parameters do affect the job.

- Preemptivity
  - Preemption: suspend job then dispatch different job to processor. Cost includes context switch overhead.

- Non-preemptable task - must be run from start to completion.

- Criticalness - positive integer indicating the relative importance of a job. Useful during overload.
- Optional Executions - jobs or portions of jobs may be declared optional. Useful during overload. In contrast, jobs and portions of jobs that are not optional are mandatory; they must be executed to completion.
- Laxity - Laxity type of a job indicates whether its timing constraints are soft or hard. Supplemented by a usefulness function. Useful during overload.

Resource Parameters: Job resource parameters indicate processor and resource requirements.

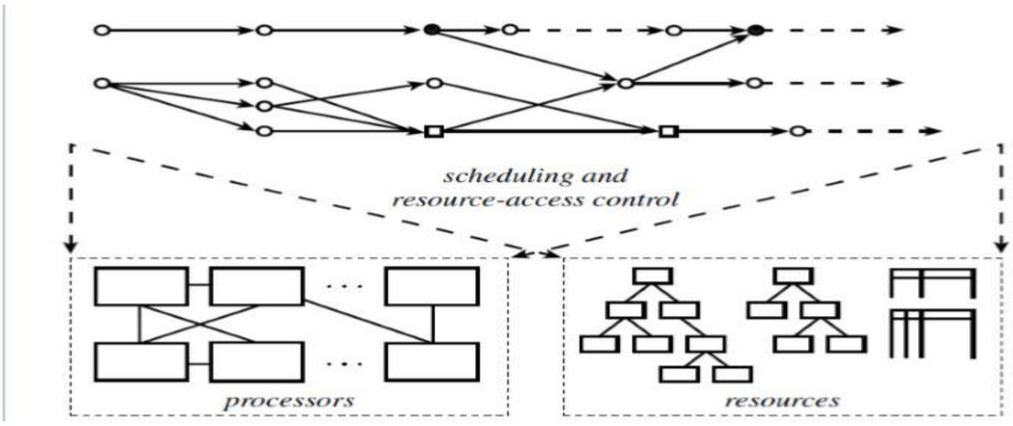
1. Preemptivity of resources
2. Resource Graph

Preemptivity of resources:

- A resource in non-preemptable if each unit of resource is constrained to be used serially.
- Once the unit of non-preemptable resource is allocated to a job, the other job needing the unit must wait until the job completes its use.
- If a job can use every resource in an interleaved manner, then the resource is called preemptable.

Resource Graph:

- Resource graph is used to describe the configuration of the resources.
- In a resource graph, there is a vertex for every processor or resource Ri in the system.
- The attributes of the vertex are the parameter of the resources.
- Resource type of a resource tell if the resource is processor or passive resource and number gives the available number of units.
- An edge from vertex Ri to Rk can mean Rk is component of Ri(memory is a part of computer).
- This edge is an is-a-part-of-edge.
- Some edges in resource graph represent the connectivity between the components.
- These edges are called accessibility edges.(Connection between two CPUs)



SCHEDULING ALGORITHMS:

- Scheduling algorithms are a governing part of real-time systems and there exists many different scheduling algorithms due to the varying needs and requirements of different real-time systems.
- The choice of algorithm is important in every real-time system and is greatly influenced by what kind of system the algorithm will serve.
- A scheduling algorithm can be seen as a rule set that tells the scheduler how to manage the real-time system, that is, how to queue tasks and give processor-time.

Fig: shows the three elements of our model of real-time systems together. The application system is represented by a task graph, exemplified by the graph on the top of the diagram. This graph gives the processor time and resource requirements of jobs, the timing constraints of each job, and the dependencies of jobs. The resource graph describes the amounts of the resources available to execute the application system, the attributes of the resources, and the rules governing their usage. Between them are the scheduling and resource access-control algorithms used by the operating system.

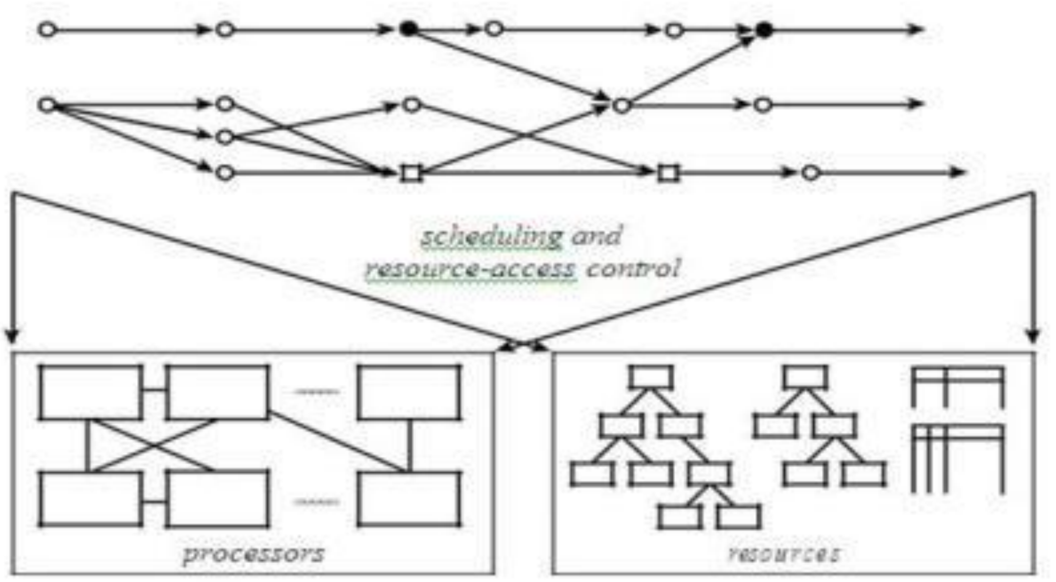


Fig: Model of Real-time systems

Scheduler and Schedules

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

- Every job is scheduled in a time interval on a processor if the processor is assigned to the job, and hence the job executes on the processor, in the interval. The total amount of processor)time assigned to a job according to a schedule is the total length of all the time intervals during which the job is scheduled on some processor.
- By a schedule, we assignment of all the jobs in the system on the available processors produced by the scheduler. A scheduler is ***valid schedule*** , if it 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.

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.

4. All the precedence and resource usage constraints are satisfied.

- A valid schedule is a ***feasible schedule*** if every job completes by its deadline.
- A hard real-time scheduling algorithm is ***optimal*** if the algorithm (the scheduler) always produces a feasible schedule if the given set of jobs has feasible schedules.
- The ***lateness*** of a job is the difference between its completion time and its deadline. The lateness of a job which completes early is negative, while the lateness of a job which completes late is positive.
- 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***.
- *Miss rate* is the percentage of jobs that are executed but completed too late.
- *Loss rate* is the percentage of jobs that are discarded, that is, not executed at all.

2.1 Commonly used approaches to hard real-time scheduling:

There are there commonly used approaches to scheduling real-time systems:

1. Clock-driven
2. Weighted round-robin
3. Priority-driven

2.1.1 Clock-driven scheduling:

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.

- Primarily used for hard real-time systems where all properties of all jobs are known at design time, such that offline scheduling techniques can be used.
- These instants (parameters) are chosen a priori before the system begins execution.
- A schedule of the jobs is computed off-line and is stored for use at runtime; as a result, scheduling overhead at run-time can be minimized.
- Decisions about what jobs execute at what times are made at specific time instants
- Usually regularly spaced, implemented using a periodic timer interrupt
- Scheduler awakes after each interrupt, schedules the job to execute for the next period, then blocks itself until the next interrupt
- One way to implement a scheduler that makes scheduling decisions periodically 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

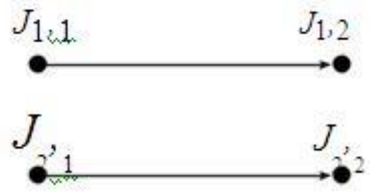
expiration of the timer. When the timer expires, the scheduler awakes and repeats these actions.

2.1.2 Weighted round-robin approach:

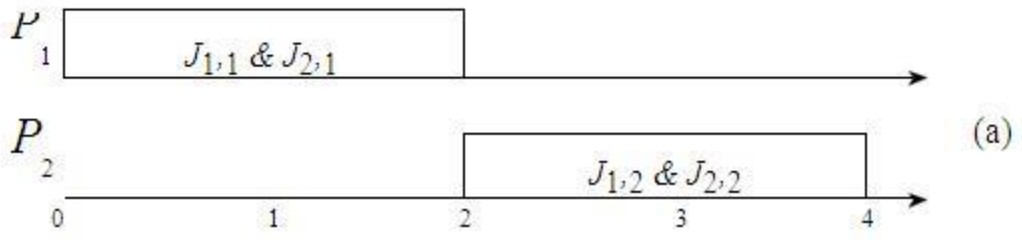
- Regular ***Round-robin*** scheduling is commonly used for scheduling time-shared applications
  - Every job joins a FIFO queue when it is ready for execution
  - When the scheduler runs, it schedules the job at the head of the queue to execute for at most one time slice (A time slice is the basic granule of time that is allocated to jobs.)
  - Sometimes called a quantum – typically order of tens of milliseconds.
  - If the job has not completed by the end of its quantum, it is preempted and placed at the end of the queue
  - When there are n ready jobs in the queue, each job gets one slice every n time slices (n time slices is called a round), i.e in every round.
  - Because the length of the time slice is relatively short, the execution of every job begins almost immediately after it becomes ready.
- **Weighted Round-robin**: Primarily 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.
  - The weight of a job refers to the fraction of processor time allocated to the job.
  - In *weighted round robin* each job *i* is assigned a weight *wi*; this job will receive *wi* consecutive time slices each round, and the duration of a round is : $\sum_{i=1}^{n} w_i$
  - Weight means the fraction of processor time allocated to a job.
    - Equivalent to regular round robin if all weights equal 1.
    - Simple to implement, since it doesn't require a sorted priority queue
  - By adjusting the weights of jobs, we can speed up or retard the progress of each job toward its completion.
  - Offers throughput guarantees
    - Each job makes a certain amount of progress each round.
  - By giving each job a fixed fraction of the processor time, a round-robin scheduler may delay the completion of every job.
    – A precedence constrained job may be assigned processor time, even while it waits for its predecessor to complete; a job can't take the time assigned to its successor to finish earlier
    – Not an issue for jobs that can incrementally consume output from their predecessor, since they execute concurrently in a pipelined fashion

- E.g. Jobs communicating using UNIX pipes
- E.g. Wormhole switching networks, where message transmission is carried out in a pipeline fashion and a downstream switch can begin to transmit an earlier portion of a message, without having to wait for the arrival of the later portion.

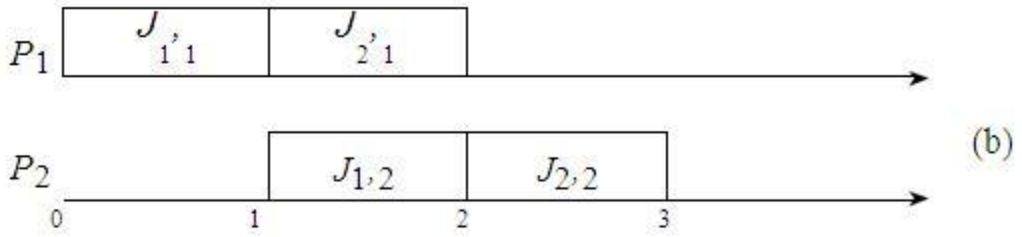
➢ As an example, we consider the two sets of jobs, J1 = {J1,1, J1,2} and J 2 = {J2,1, J2,2}, shown in Figure. The release times of all jobs are 0, and their execution times are 1. J1,1 and J2,1 execute on processor P1, and J1,2 and J2,2 execute on processor P2. Suppose that J1,1 is the predecessor of J1,2, and J2,1 is the predecessor of J2,2.

▪ 

➢ Figure(a), shows that both sets of jobs (i.e., the second jobs J1,2 and J2,2 in the sets) complete approximately at time 4 if the jobs are scheduled in a weighted round-robin manner. (We get this completion time when the length of the time slice is small compared with 1 and the jobs have the same weight.)



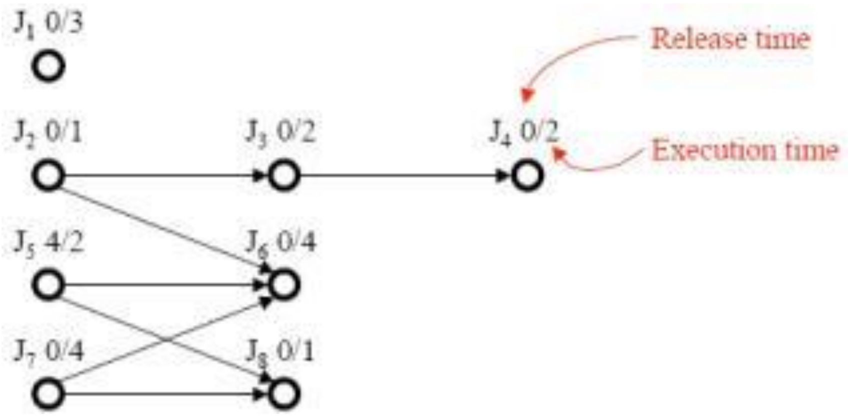
➢ In contrast, the schedule in Figure(b) shows that if the jobs on each processor are executed one after the other, one of the chains can complete at time 2, while the other can complete at time 3.



➢ On the other hand, suppose that the result of the first job in each set is piped to the second job in the set. The latter can execute after each one or a few time slices of the former complete. Then it is better to schedule the jobs on the round-robin basis because both sets can complete a few time slices after time 2.

2.1.3 Priority-Driven approach:

- The term priority-driven algorithms refer 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.
- Assign priorities to jobs, based on some algorithm
- Make scheduling decisions based on the priorities, when events such as releases and job completions occur
  - – Priority scheduling algorithms are *event-driven*
  - – Jobs are placed in one or more queues; at each event, the ready job with the highest priority is executed
  - – The assignment of jobs to priority queues, along with rules such a whether preemption is allowed, completely defines a priority scheduling algorithm.
- Priority-driven algorithms make locally optimal decisions about which job to run
  - – Locally optimal scheduling decisions are often not globally optimal
  - – Priority-driven algorithms never intentionally leave any resource idle
    - • Leaving a resource idle is not locally optimal
- Consider the following task:

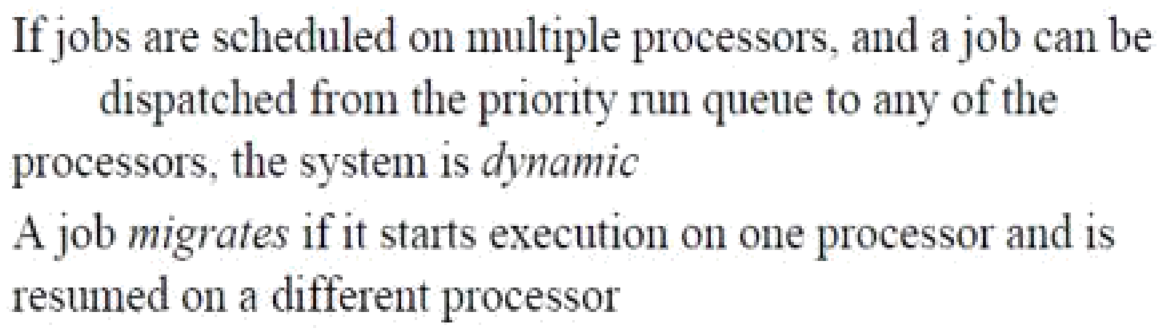


  - – Jobs *J*1, *J*2, …, *J*8, where *Ji* had higher priority than *Jk* if *i* < *k*
  - – Jobs are scheduled on two processors *P*1 and *P*2
  - – Jobs communicate via shared memory, so communication cost is negligible
  - – The schedulers keep one common priority queue of ready jobs
  - – All jobs are preemptable; scheduling decisions are made whenever some job becomes ready for execution or a job completes

This scheduling can also be achieved by static system.

Let us assign J1,J2,J3 and J4 to P1 and remaining to P2.

Jobs on P1 are completed by time 8 and the jobs on P2 by time 11.


If jobs are scheduled on multiple processors, and a job can be dispatched from the priority run queue to any of the processors, the system is *dynamic*

A job *migrates* if it starts execution on one processor and is resumed on a different processor

2.2 Clock-Driven Scheduling:

2.2.2 Scheduling sporadic tasks:

- Sporadic jobs have hard deadlines.
- Their minimum release times and maximum execution times are unknown.
- It is impossible to guarantee a priori that all sporadic jobs can complete in time. <u>Acceptance Test:</u>

- 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.
- A job in the system, we mean either **a periodic job**, for which time has already been allocated in the precomputed cyclic schedule, or **a sporadic job** which has been scheduled but not yet completed.
- According to the existing schedule, If there is a sufficient amount of time in the frames before its deadline to complete the newly released sporadic job without causing any job in the system to complete too late, the scheduler accepts and schedules the job. Otherwise, the scheduler rejects the new sporadic job.
- By rejecting a sporadic job that cannot be scheduled to complete in time immediately after the job is released, the scheduler gives the application system as much time as there is to take any necessary recovery action.
- Example: **A quality control system.**
    - A sporadic job that activates a robotic arm is released when a defective part is detected.
    - The arm, when activated, removes the part from the conveyor belt. This job must complete before the part moves beyond the reach of the arm.
    - When the job cannot be scheduled to complete in time, it is better for the system to have this information as soon as possible.

  - System can slow down the belt, stop the belt, or alert an operator to manually remove the part.
  - Otherwise, if the sporadic job were scheduled but completed too late, its lateness would not be detected until its deadline. By the time the system attempts a recovery action, the defective part may already have been packed for shipment.
- We assume that the maximum execution time of each sporadic job becomes known upon its release.
- It is impossible for the scheduler to determine which sporadic jobs to admit and which to reject unless this information is available.
- Therefore, the scheduler must maintain information on the maximum execution times of all types of sporadic jobs that the system may execute in response to the events it is required to handle.
- We also assume that all sporadic jobs are preemptable. Therefore, each sporadic job can execute in more than one frame if no frame has a sufficient amount of time to accommodate the entire job.
- Conceptually, it is quite simple to do an acceptance test.
- Let us suppose that at the beginning of frame t , an acceptance test is done on a sporadic job $S(d, e)$, with deadline d and (maximum) execution time e.
- Suppose that the deadline d of S is in frame l+1 (i.e., frame l ends before d but frame l+1 ends after d) and $l \geq t$.
- Clearly, the job must be scheduled in the lth or earlier frames.
- 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 $e > \sigma c(t, l)$.
- The scheduler accepts the new job $S(d, e)$ only if $e \leq \sigma c(t, l)$ and no sporadic jobs in system are adversely affected.
- More than one sporadic job may be waiting to be tested at the same time.
- A good way to order them is on the Earliest-Deadline-First (EDF) basis.
- Newly released sporadic jobs are placed in a waiting queue ordered in non decreasing order of their deadlines: the earlier the deadline, the earlier in the queue.
- The scheduler always tests the job at the head of the queue and removes the job from the waiting queue after scheduling it or rejecting it.

EDF Scheduling of the Accepted Jobs: Earliest deadline first (EDF) is dynamic priority scheduling algorithm for real time systems. Earliest deadline first selects a task according to its deadline such that a task with earliest deadline has higher priority than others

- The EDF algorithm is a good way to schedule accepted sporadic jobs.
- The scheduler maintains a queue of accepted sporadic jobs in non decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order.
- Whenever all the slices of periodic tasks scheduled in each frame are completed, the cyclic executive lets the jobs in the sporadic job queue execute in the order they appear in the queue.

An example of EDF is given below for task set of table.

Task	**Release time(ri)**	**Execution Time(Ci)**	**Deadline (Di)**	**Time Period(Ti)**
T1	0	1	4	4
T2	0	2	6	6
T3	0	3	8	8

U= 1/4 +2/6 +3/8 = 0.25 + 0.333 +0.375 = 0.95 = 95%
As processor utilization is less than 1 or 100% so task set is surely schedulable by EDF.

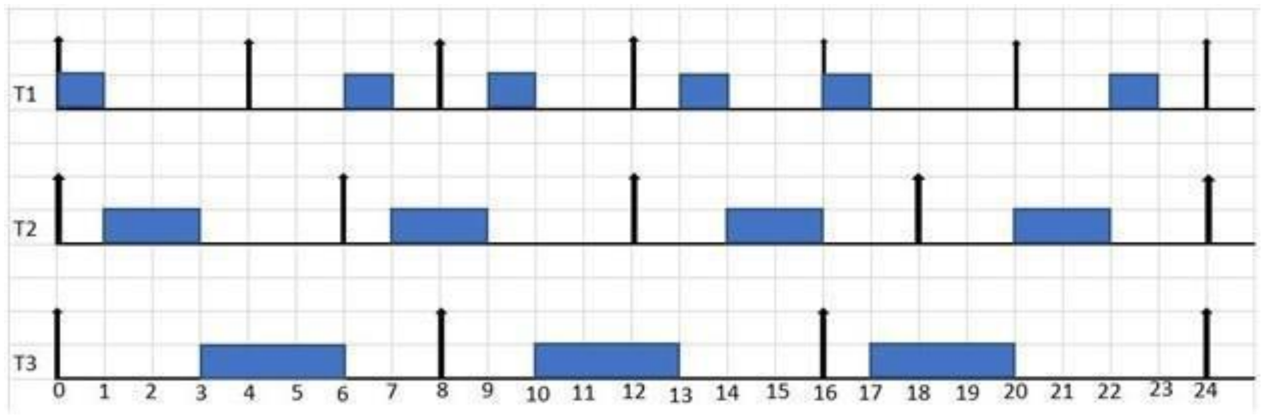


Figure 2. Earliest deadline first scheduling of task set in Table

At t=0 all the tasks are released, but priorities are decided according to their absolute deadlines so T1 has higher priority as its deadline is 4 earlier than T2 whose deadline is 6 and T3 whose deadline is 8, that's why it executes first.

At t=1 again absolute deadlines are compared and T2 has shorter deadline so it executes and after that T3 starts execution but at t=4 T1 comes in the system and deadlines are compared, at this instant both T1 and T3 has same deadlines so ties are broken randomly so we continue to execute T3.

At t=6 T2 is released, now deadline of T1 is earliest than T2 so it starts execution and after that T2 begins to execute. At t=8 again T1 and T2 have same deadlines i.e. t=16, so ties are broken randomly an T2 continues its execution and then T1 completes. Now at t=12 T1 and T2 come in the system simultaneously so by comparing absolute deadlines, T1 and T2 has same deadlines therefore ties broken randomly and we continue to execute T3.

At t=13 T1 begins it execution and ends at t=14. Now T2 is the only task in the system so it completes it execution.

At t=16 T1 and T2 are released together, priorities are decided according to absolute deadlines so T1 execute first as its deadline is t=20 and T3's deadline is t=24.After T1 completion T3 starts and reaches at t=17 where T2 comes in the system now by deadline comparison both have same deadline t=24 so ties broken randomly ant we T continue to execute T3.

At t=20 both T1 and T2 are in the system and both have same deadline t=24 so again ties broken randomly and T2 executes. After that T1 completes it execution. In the same way system continue to run without any problem by following EDF algorithm.

2.2.2 Algorithm for constructing static schedules:

- First consider the special case where the periodic tasks contain no non preemptable section. After presenting a polynomial time solution for this case, we then discuss how to take into account practical factors such as non preemptivity.
- **Scheduling Independent Preemptable Tasks**
    - A system of independent, preemptable periodic tasks whose relative deadlines are equal to or greater than their respective periods is schedulable if and only if the total utilization of the tasks is no greater than 1.
    - Because some tasks may have relative deadlines shorter than their periods.
    - The iterative algorithm described below enables us to find a feasible cyclic schedule if one exists. The algorithm is called the **iterative network-flow algorithm, or the INF** algorithm.
    - 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
    - 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.

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 J1, J2. JN
- 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 Non-negative number associated with the edge.

1. There is a job vertex Ji representing each job Ji, for i = 1, 2, . . . , N.
2. There is a frame vertex named j representing each frame j in the major cycle, for j = 1, 2,..F.
3. There are two special vertices named source and sink.
4. There is a directed edge (Ji , j) from a job vertex Ji to a frame vertex j if the job Ji 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 Ji , and the capacity of this
   edge is the execution time ei of the job.
6. There is a directed edge from every frame vertex to the sink, and the capacity of this edge is f

- A flow of an edge is a nonnegative number that satisfies the following constraints:
  (1) It is no greater than the capacity of the edge.
  (2) 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.

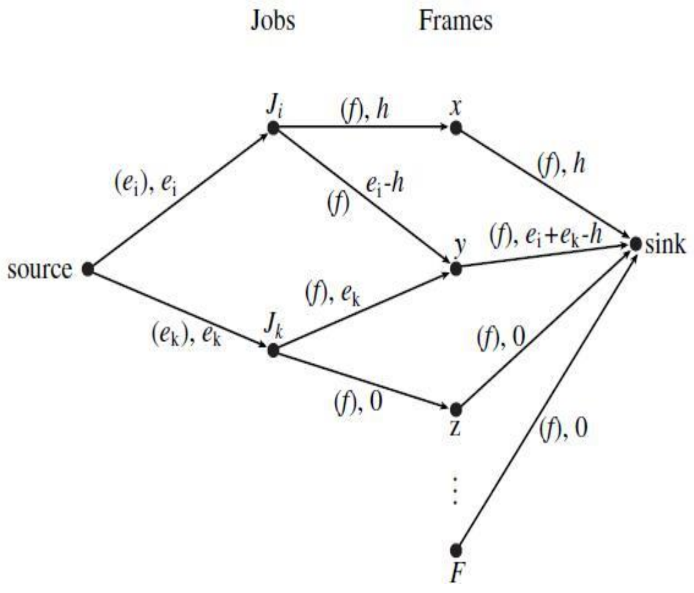


Fig: Part of network-flow graph

- For simplicity, only job vertices Ji and Jk are shown. The label “(capacity), flow” of the each edge gives its capacity and flow. This graph indicates that job Ji can be scheduled in frames x and y and the job Jk can be scheduled in frames y and z.
- 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. There are many algorithms for finding the maximum flows of network-flow sgraphs.