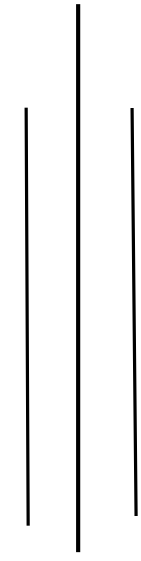
A NOTES ON

REAL TIME SYSTEM



REAL TIME SYSTEM

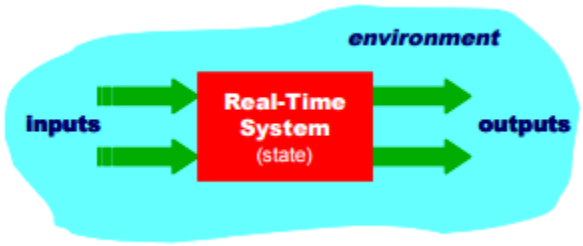
Loknath Regmi

Chapter-1

Introduction

Real Time System:

Any system in which the time at which output is produced is significant is called real time system. It is any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period. Generally, real-time systems are systems that maintains a continuous timely interaction with its environment as below.



There are two types of real-time systems.

1. Reactive real time system
2. Embedded real time system

- Reactive real-time system involves a system that has constant interaction with its environment for e.g. a pilot controlling an aircraft.
- An embedded real-time system is used to control specialized hardware that is installed within a larger system for e.g. a microprocessor that controls the fuel-to-air mixture for automobiles.

Real time is a level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process for example, to present visualizations of the weather as it constantly changes. Real-time is an adjective pertaining to computers or processes that operate in real time. Real time describes a human rather than a machine sense of time. Examples of real-time systems include

- Software for cruise missile
- Heads-up cockpit display
- Airline reservation system
- Industrial Process Control
- Banking ATM

Real-time systems can also be found in many industries;

- Defense systems
- Telecommunication systems
- Automotive control
- Signal processing systems

- Radar systems
- Automated manufacturing systems
- Air traffic control
- Satellite systems

Digital Control:

A digital control system model can be viewed from different perspectives including control algorithm, computer program, conversion between analog and digital domains, system performance etc. One of the most important aspects is the sampling process level. In continuous time control systems, all the system variables are continuous signals. Whether the system is linear or nonlinear, all variables are continuously present and therefore known (available) at all times. A typical continuous time control system is shown in Figure 1.

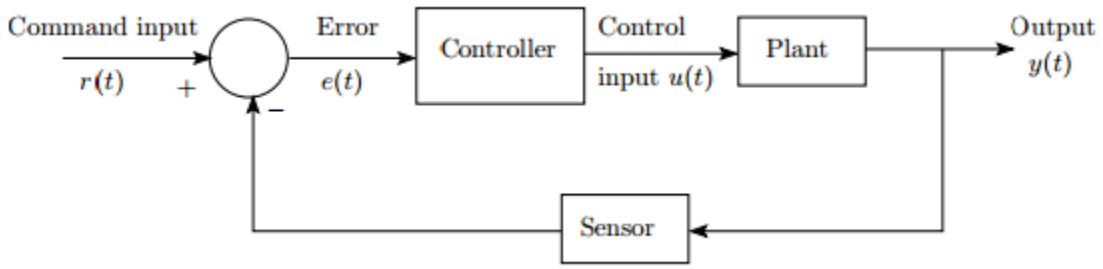


Fig: A typical closed loop continuous time control system

In a digital control system, the control algorithm is implemented in a digital computer. The error signal is discretized and fed to the computer by using an A/D (analog to digital) converter. The controller output is again a discrete signal which is applied to the plant after using a D/A (digital to analog) converter. General block diagram of a digital control system is shown in Figure 2.

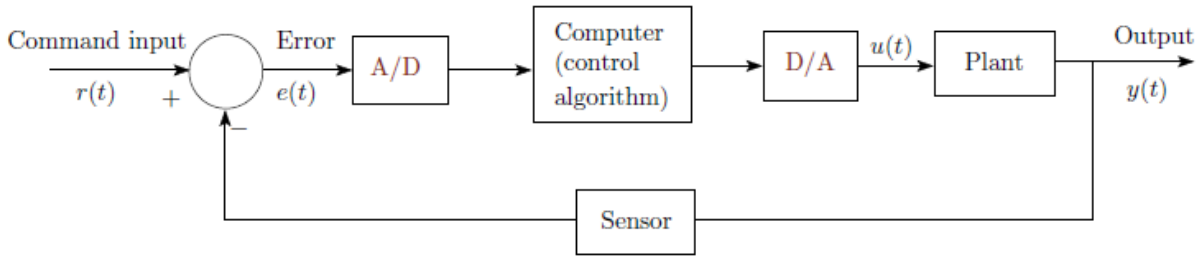


Figure 2: General block diagram of a digital control system

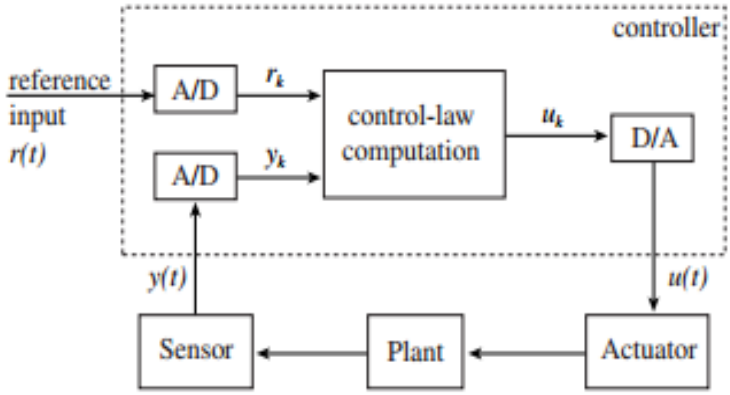
e(t) is sampled at intervals of T. In the context of control and communication, sampling is a process by which a continuous time signal is converted into a sequence of numbers at discrete time intervals. It is a fundamental property of digital control systems because of the discrete nature of operation of digital computer.

Sampled Data Systems:

Long before digital computers became cost-effective and widely used, analog i.e., continuous time and continuous-state controllers were in use, and their principles were well established. A common approach be developed to design the digital controller that use the system that converts the analog version of system into a digital i.e., discrete-time and discrete-state version and resultant system is called sampled data system.

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) called as control-law computation and corresponding controller is called digital controller.

It typically samples (i.e., reads) and digitizes the analog sensor readings periodically and carries out its control-law computation every period. The sequence of digital outputs thus produced is then converted back to an analog form needed to activate the actuators.



For example: Single-input single-output PID controller

PID means Proportional, Integral and Derivative. In this controller, 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 depends on three terms as

- a term that is proportional to e(t)
- a term preoperational to the integral of e(t)
- a term preoperational to the derivative of e(t)

In the sampled data version, the inputs to the control-law computation are the sampled values y_k and r_k, k=0,1,2,.., which analog-to-digital converters produce by sampling and digitizing y(t) and r(t) periodically every T units of time then

$$e_k=r_k-y_k$$

is the k-th sample value of e(t). There are many ways to discretize the derivative and integral of e(t).

For example, we can approximate derivative of e(t) for (k-1)T<=t<=kT by

$$(e_k - e_{k-1})/T$$

By using the right rectangular rule of numerical integration to transform a continuous integral into a discrete as

$$\int_a^b f(x)dx = f(b)(b-a)$$

Suppose α, β, υ are some constants and they are chosen at the design time. During the k-th sampling period, the RTS computes the output of the controller according to this expression.

$$u_k = \alpha e_k + \beta(e_k - e_{k-1})/T + \gamma \int_0^{kT} e(t)dt = \alpha e_k + \beta(e_k - e_{k-1})/T + \gamma \sum_{i=1}^{k} e_i T$$

$$u_{k-1} = \alpha e_{k-1} + \beta(e_{k-1} - e_{k-2})/T + \gamma \sum_{i=1}^{k-1} e_i T$$

Different discretization methods may lead to different expressions for u_k, but they all are simple to compute and requires 10-20 machine instructions. The corresponding controller algorithms can be derived as:

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

Make digital-to-analog conversion and output u

End do

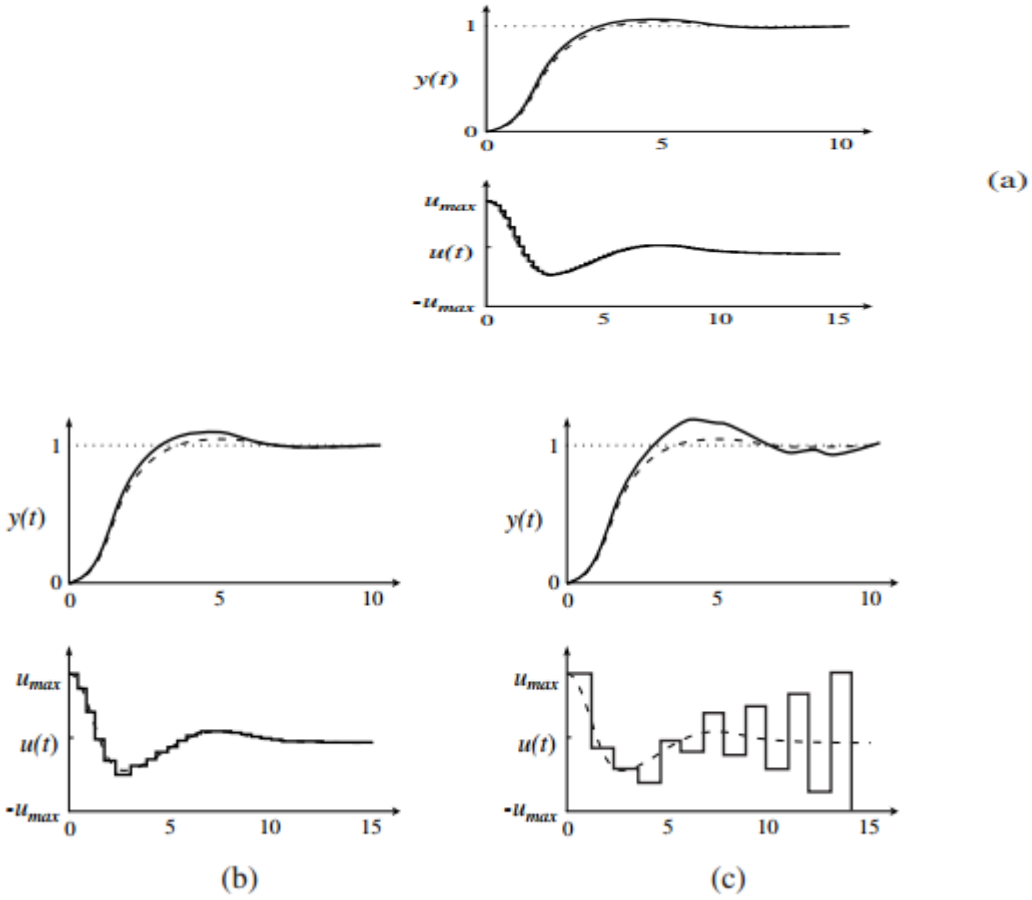
}

Selection of 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. Ideally, we want the sampled data version behave like the

analog version. This can be done by making T small. However, this will lead to more frequent control-law computation and higher processor-time demand.

The operator may issue a command at any time, say at t. The consequent change in the reference input is read and reacted to by the digital controller at the next sampling instant. This instant can be as late as (t +T) Thus, sampling introduces a delay in a system response. The operator will feel the system sluggish when the delay exceeds a tenth of a second. Therefore, the sampling period should be under this limit. The second factor is the dynamic behavior of the plant.



The dashed lines in Figure (a) give the output u (t) of the analog controller with respect to y(t) as a function of time where the sampling time be nearly in both analog and digital version. The solid lines in Figure (b) give the behavior of the digital version when the sampling period is increased by 2.5 times. Figure (c) give the behavior of the digital version when the sampling period is increased by 5 times. This shows that sampling time is the key characteristics to convert analog to digital form. It is recommended to use sampling frequency twice as highest possible frequency occurring in the system to be able to reproduce such signals (Nyquist-Kotelnikov theorem).

Multirate Systems:

A system that have more than one degree of freedom then it is called multi input multi output system i.e. a typical system with multiple input and multiple output. Its state is defined by multiple state variables (e.g., the rotation speed, temperature, etc) hence they are monitored by multiple

sensors and controlled by multiple actuators. The sampling periods required to achieve smooth responses from the perspective of different state variables may be different because different state variables may have different dynamics. When highest sampling period be used then this waste the processor time so multiple sampling rate be adopted to convert the digital version and corresponding system is called multirate system. A software based control be used in multirate system that decides the rate of sampling for different state variable and corresponding structure is called software controlled structure. A typical software controlled structure is given below.

Do the following in each 1/180-second cycle:
- Validate sensor data and select data source; in the presence of failures, reconfigure the system.
- Do the following 30-Hz avionics tasks, each once every six cycles:
  - keyboard input and mode selection
  - data normalization and coordinate transformation
  - tracking reference update
- Do the following 30-Hz computations, each once every six cycles:
  - control laws of the outer pitch-control loop
  - control laws of the outer roll-control loop
  - control laws of the outer yaw- and collective-control loop
- Do each of the following 90-Hz computations once every two cycles, using outputs produced by 30-Hz computations and avionics tasks as input:
  - control laws of the inner pitch-control loop
  - control laws of the inner roll- and collective-control loop
- Compute the control laws of the inner yaw-control loop, using outputs produced by 90-Hz control-law computations as input.
- Output commands.
- Carry out built-in-test.
- Wait until the beginning of the next cycle.

Figure shows the software structure of a flight controller. The plant is a helicopter. It has three velocity components; together, they are called "collective" in the figure. It also has three rotational (angular) velocities, referred to as roll, pitch, and yaw. The system uses three sampling rates: 180, 90, and 30 Hz. After initialization, the system executes a do loop at the rate of one iteration every 1/180 second. Specifically, at the start of each 1/180-second cycle, the controller first checks its own health and reconfigures itself if it detects any failure. It then does either one of the three avionics tasks or computes one of the 30-Hz control laws. We note that the pilot's command (i.e., keyboard input) is checked every 1/30 second. At this sampling rate, the pilot should not perceive the additional delay introduced by sampling. The movement of the aircraft along each of the coordinates is monitored and controlled by an inner and faster loop and an outer and slower loop. The output produced by the outer loop is the reference input to the inner loop. Each inner loop also uses the data produced by the avionics tasks.

Timing Characteristics:

Work load generated by each multivariate, multirate digital controller consists of a few periodic control-law computations. Their periods range from a few milliseconds to a few seconds. A control system may contain numerous digital controllers, each of which deals with some attribute of the plant. Together they demand tens or hundreds of control laws be computed periodically, some of them continuously and others only when requested by the operator or in reaction to some events. The control laws of each multirate controller may have harmonic periods. They typically use the data produced by each other as inputs and are said to be a rate group. In some systems, it is

necessary to keep this variation small so that the digital control outputs produced by the controller become available at time instants more regularly spaced in time.

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

For example, 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 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.

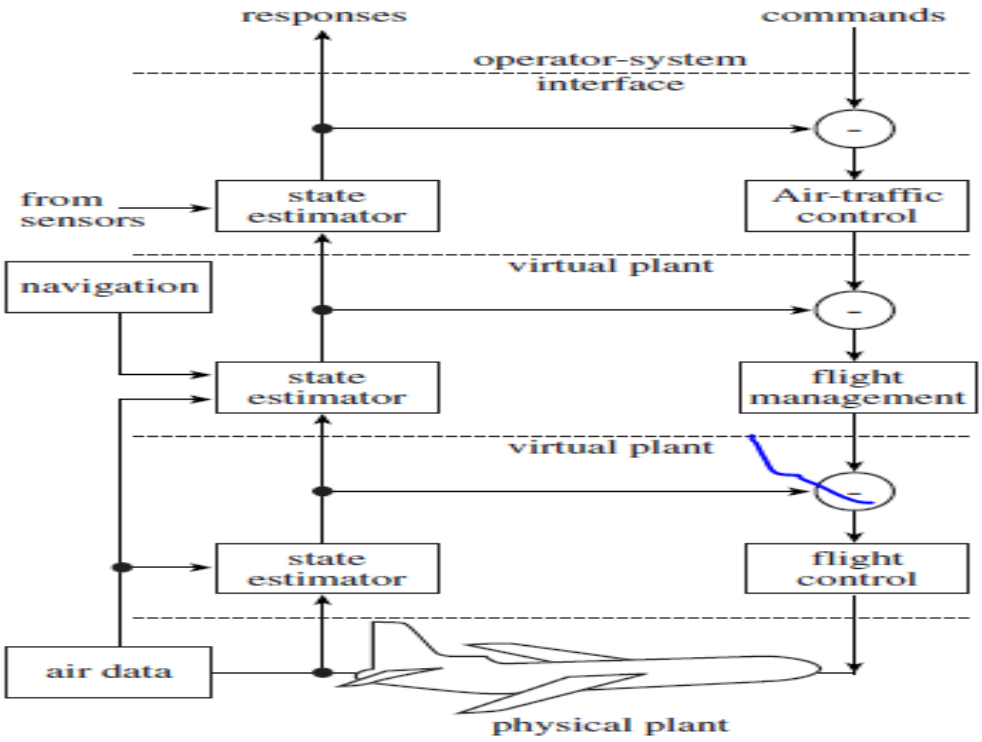


Figure shows a more complex example: the hierarchy of flight control, avionics, and air traffic control systems. The Air Traffic Control (ATC) system is at the highest level. It regulates the flow of flights to each destination airport. It does so by assigning to each aircraft an arrival time at each metering fix (or waypoint) en route to the destination: The aircraft is supposed to arrive at the metering fix at the assigned arrival 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, decent/accent rates, and so forth required to follow the chosen time-referenced flight path are the reference inputs to the flight controller at the lowest level of the control hierarchy.

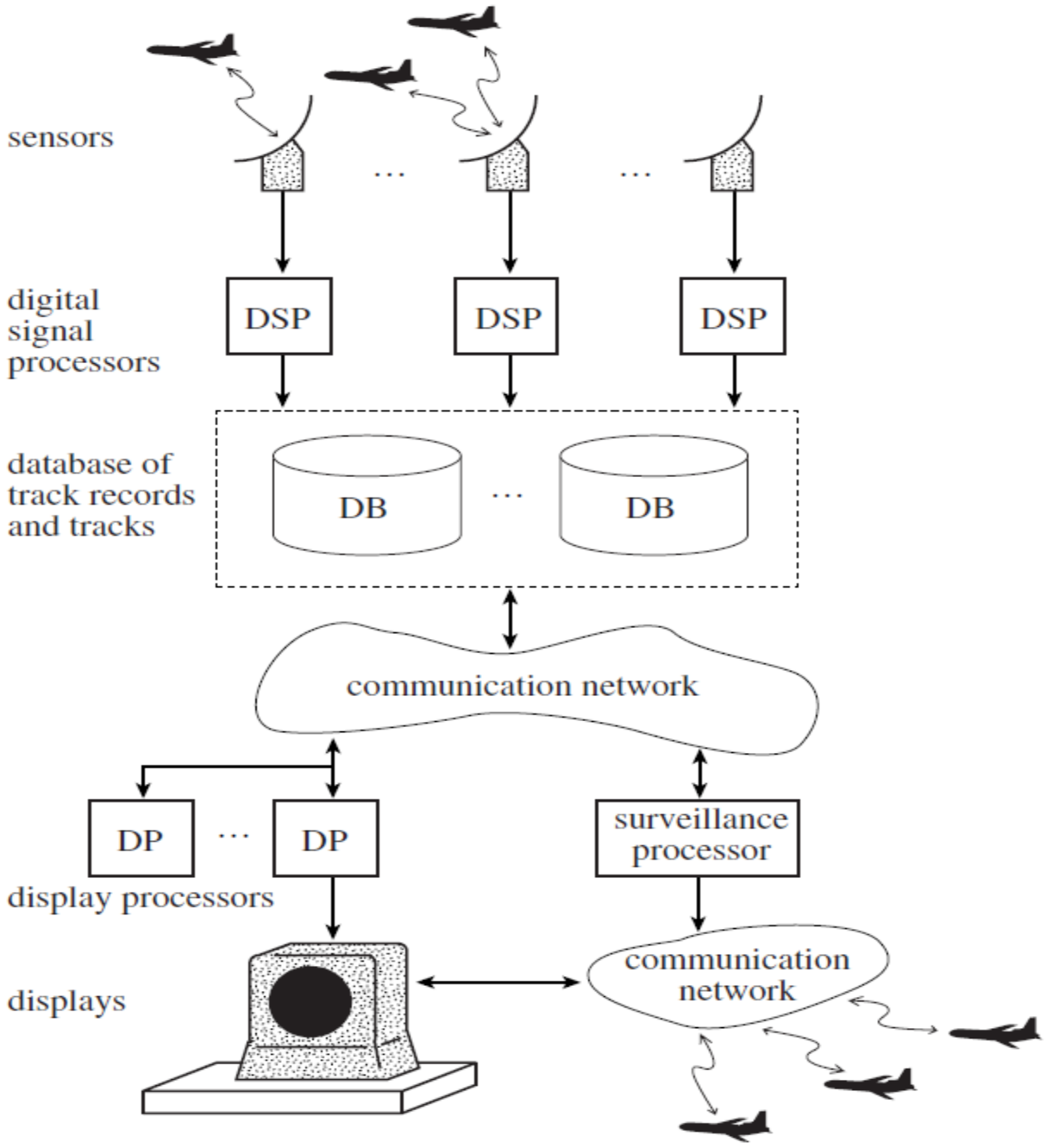
Guidance and Control:

While a digital controller deals with some dynamical behavior of the physical plant, a second level controller typically performs guidance and path planning functions to achieve a higher level goal. In particular, it tries to find one of the most desirable trajectories among all trajectories that meet the constraints of the system. The trajectory is most desirable because it optimizes some cost function(s). The algorithm(s) used for this purpose is the solution(s) of some constrained optimization problem(s).

As an example, we look again at a flight management system. The constraints that must be satisfied by the chosen flight path include the ones imposed by the characteristics of the aircraft, such as the maximum and minimum allowed cruise speeds and decent/accent rates, as well as constraints imposed by external factors, such as the ground track and altitude profile specified by the ATC system and weather conditions. A cost function is fuel consumption: A most desirable flight path is a most fuel efficient among all paths that meet all the constraints and will bring the aircraft to the next metering fix at the assigned arrival time. This problem is known as the constrained fixed-time, minimum-fuel problem. When the flight is late, the flight management system may try to bring the aircraft to the next metering fix in the shortest time. In that case, it will use an algorithm that solves the time-optimal problem.

Real-Time Command and Control:

The controller at the highest level of a control hierarchy is a command and control system. The controller at the highest level of a control hierarchy is a command and 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. 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. Fortunately, most of the timing requirements of a command and control system are less stringent. Whereas 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. In this respect, it resembles interactive, on-line transaction systems (e.g., a stock price quotation system) which are also sometimes called real-time systems.



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. Figure 1–5 shows a possible architecture. 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 (i.e., the air traffic controllers). Outputs from the ATC system include the assigned arrival times to metering fixes for individual aircraft. As stated earlier, 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 radar interrogates each aircraft periodically. When interrogated, an air-craft responds by sending to the ATC system its "state variables": identifier, position, altitude, heading, and so on. (In Figure 1–5, these variables are referred to collectively as a track record, and the current trajectory of the aircraft is a track.) 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).

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 application has some kind of real time requirements. Signal processing may include digital filtering, video/voice compression and decompression and radar signal processing.

Processing Bandwidth Demands:

The bandwidth demand of a signal processing depends on the sampling rate and number of outputs at a time. A real signal processing application computes one or more outputs in each sampling period. Each output x(k) is a weighted sum of n inputs y(i).

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

Where a(k,i) are the weights whose values are known and fixed. The weight value in signal processing system takes at least one addition or multiplication. This computation transforms the given representation of objects(voice, image or radar signal) in terms of inputs y(i) into another representation in terms of outputs x(k). Different set of weights a(k,i) give different kind of transformation.

Radar Signal Processing:

The Radar signal processing 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.

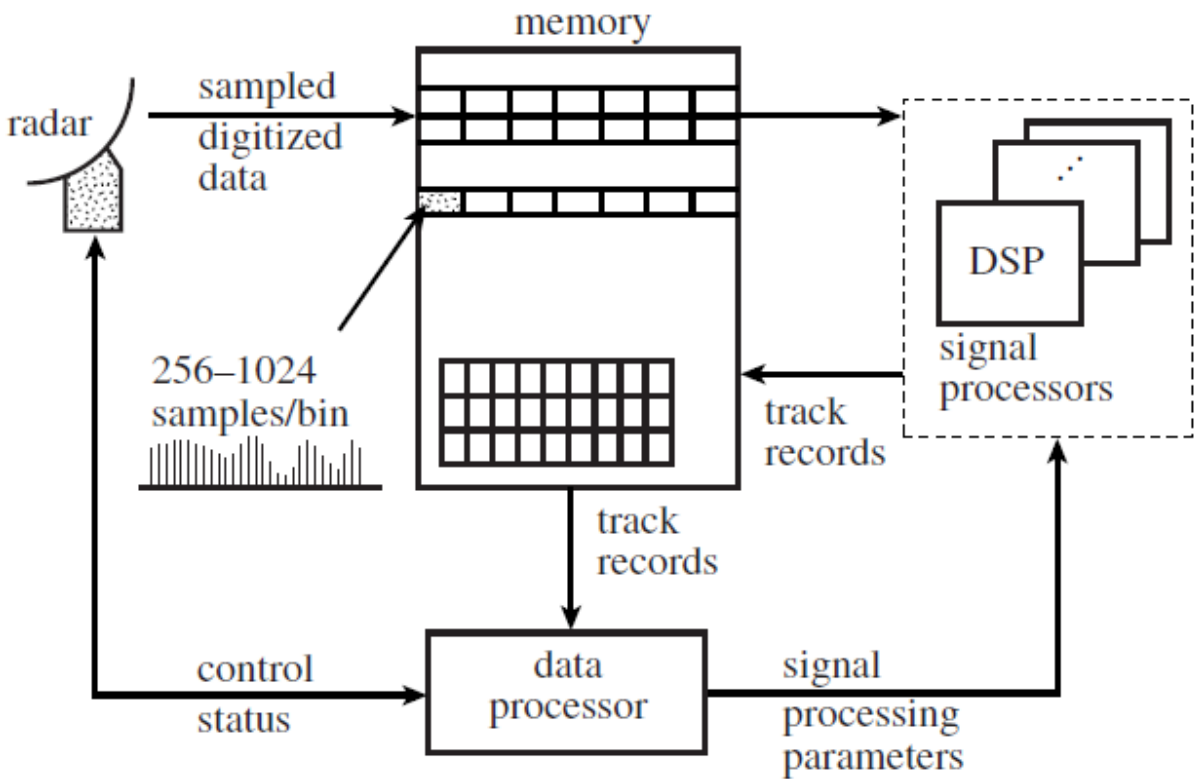


Fig. Radar Signal processing and Tracking System

Signal Processing:

To search for objects of interest in its coverage area, the radar 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 × 108 meters per second is the speed of light. The echo signal collected at this time should be stronger than when there is no reflected signal. If the object is moving, the frequency of the reflected signal is no longer equal to that of the transmitted pulse. The amount of frequency shift (called Doppler shift) is proportional to the velocity of the object. Therefore, by examining the strength and frequency spectrum of the echo signal, the system can determine whether there are objects in the direction pointed at by the antenna and if there are objects, what their positions and velocities are. The radar memory is divided into

small partitions called bin. The bin is equivalent to the small distance during which the samples are generated.

Tracking

Strong 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. Tracker assigns each measured value (i.e., the tuple of position and velocity contained in each of the track records generated in a scan) to a trajectory.

If the trajectory is an existing one, the measured value assigned to it gives the current position and velocity of the object moving along the trajectory. If the trajectory is new, the measured value gives the position and velocity of a possible new object. Tracking is carried out in two steps: gating and data association.

Gating:

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.

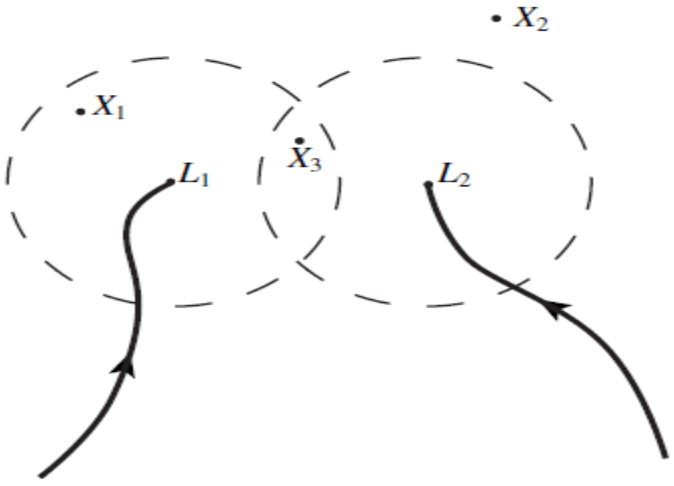


Fig. Gating Process

Above fig. illustrates this 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, $L1$ and $L2$. $X1$, $X2$, and $X3$ are the measured values given by three track records. $X1$ is

assigned to $L1$ because it is within distance G from $L1$. $X3$ is assigned to both $L1$ and $L2$ for the same reason. On the other hand, $X2$ is not assigned to any of the trajectories. It represents either a false return or a new object. Since it is not possible to distinguish between these two cases, the tracker hypothesizes that $X2$ 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.

Data Association:

The tracking process completes if, after gating, every measured value 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:

Two most common real-time applications are real-time databases and multimedia applications.

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 distinguishes these databases from non-real-time databases is the perishable nature of the data maintained by them. Specifically, 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.

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 stored position and heading will deviate more and more from the actual position and heading. In this sense, the quality of stored data degrades. This is why we say that real-time data are perishable. In contrast, an underlying assumption of non-real-time databases (e.g., a payroll database) is that in the absence of updates the data contained in them remain good (i.e., the database remains in some consistent state satisfying all the data integrity constraints of the database).

Absolute Temporal Consistency:

The age of a data object measures how up-to-date the information provided by the object is. The age of an image object at any time is the length of time since the instant of the last update, that is, when its value is made equal to that of the real-world object it represents. The age of a data object whose value is computed from the values of other objects is equal to the oldest of the ages of those objects.

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.

TABLE 1–1 Requirements of typical real-time databases

Applications	Size	Ave. Resp. Time	Max Resp. Time	Abs. Cons.	Rel. Cons.	Permanence
Air traffic control	20,000	0.50 ms	5.00 ms	3.00 sec.	6.00 sec.	12 hours
Aircraft mission	3,000	0.05 ms	1.00 ms	0.05 sec.	0.20 sec.	4 hours
Spacecraft control	5,000	0.05 ms	1.00 ms	0.20 sec.	1.00 sec.	25 years
Process control		0.80 ms	5.00 sec	1.00 sec.	2.00 sec	24 hours

Relative Temporal Consistency:

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. The column labeled "Rel. Cons." in Table 1–1gives typical values of this threshold.

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. (As an example, we consider a small 100 × 100-pixel, 30-frames/second color video. If uncompressed, the video requires a transmission bandwidth of 2.7 Mbits per second when the value of each component at each pixel is encoded with 3 bits.) Therefore, a video stream, as well as the associated audio stream, is invariably compressed as soon as it is captured.

MPEG Compression/ Decompression:

This compression standard makes use of 3 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.

Motion Estimation:

In this step analysis and estimation is done. Video frames are not independent and hence significance amount of compression can be done. For this each image is divided into 16*16 pixel square pieces called major 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 I frames. I(Intra coded) frames are the points for random access of the video. More the value of α , more random accessible the video is and poor the compression ratio. A good compromise is $\alpha=9$. The frame between consecutive I-frames are called P and B frames. When α is 9 frames produced are I,B,B,P,B,B,P,B,B,I,B,B,P… For every $k \geq 0$ frame $1+9k+3$ is P(Predictive coded)-frame. P frame is generated by prediction from previous I frame. B (bidirectionally predicted) frame is predicted from both I and P frames.

Chapter - 2

Hard versus Soft Real Time Systems

Jobs and Task:

- A job is a unit of work that is scheduled and executed by a system
  – E.g. computation of a control-law, computation of an FFT on sensor data, transmission of a data packet, retrieval of a file
- A task is a set of related jobs which jointly provide some function

– E.g. the set of jobs that constitute the "maintain constant altitude" task, keeping an airplane flying at a constant altitude

Processors:

A job executes or is executed by the operating system. The system that consists the operating system is called processor. The job be executed on a processor and may depend on some resources i.e. A processor, P is an active component on which jobs scheduled. For Examples

- Threads scheduled on a CPU
- Data scheduled on a transmission link
- Read/write requests scheduled to a disk
- Transactions scheduled on a database server

Each processor has a speed attribute which determines the rate of progress a job makes toward completion that is represent as instructions-per-second for a CPU, bandwidth of a network, etc.

Resources:

- A resource, R, is a passive entity upon which jobs may depend. For example memory, sequence numbers, mutexes, database locks, etc.
- Resources have different types and sizes, but do not have a speed attribute
- Resources are usually reusable, and are not consumed by use

Use of Resources:

If the system contains ρ ("rho") types of resource, this means:

- There are ρ different types of serially reusable resources
- There are one or more units of each type of resource, only one job can use each unit at once (mutually exclusive access)

- A job must obtain a unit of a needed resource, use it, then release it

A resource is plentiful if no job is ever prevented from executing by the unavailability of units of the resource at this condition Jobs never block when attempting to obtain a unit of a plentiful resource. We typically omit such resources from our discussion, since they don't impact performance or correctness.

Execution Time:

This is the amount of time required to complete the execution of a job when it executes alone and has all the resources it needs. Say A job J_i will execute for time e_i . The value of depends upon complexity of the job and speed of the processor on which it is scheduled. This may changes due to variety of reasons:

- Conditional branches
- Cache memories and/or pipelines
- Compression (e.g. MPEG video frames)

The execution time falls under the intervals$[e_i^-, e_i^+]$. This time is in sufficient for execution and valid for hard real time job but not for each e_i but it is the safe bound for execution time.

Release and Response Time:

Release time is the timing instant on which a job becomes available for execution. A job can be scheduled and executed at any time at or after its release time. The execution of job starts only when processor provided its resource and dependency conditions are met.

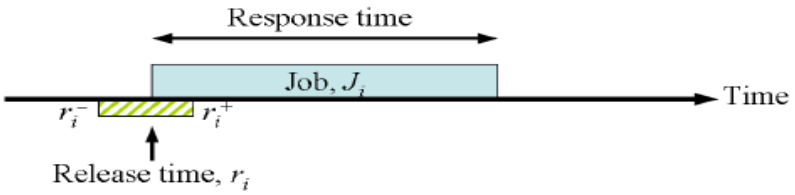


Fig: Release vs. Response time

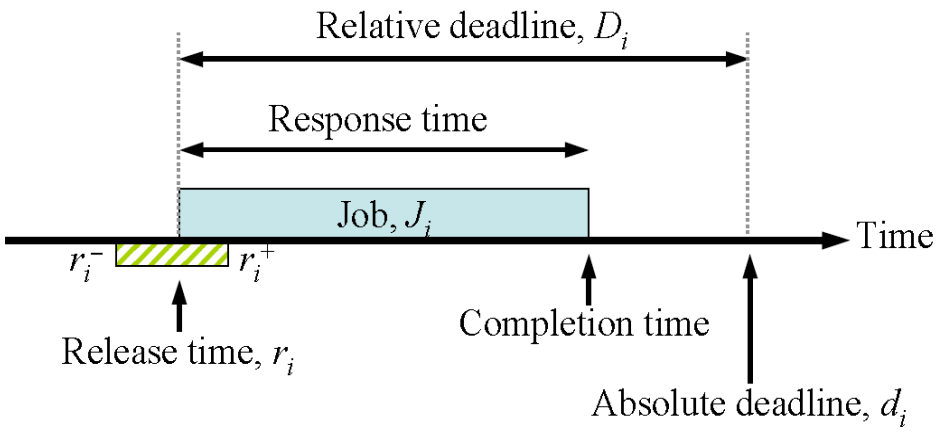
Response time is a timing interval on a job be completely processed. It is length of time from the release time of the job to the time instant when it completes. It is not the same with execution time, since a job may not execute continually.

Deadlines and Timing Constraints:

The instant at which a job completes execution is called completion time. The ***deadline*** of a job is the instant of time by which its execution is required to be completed. Deadline time is divided into the two categories.

Relative deadline – the maximum allowable job response time

Absolute deadline – the instant of time by which a job is required to be completed. It often called simply as deadline**.** The absolute deadline is the sum of the relative deadlines and release time.
I.e. *absolute deadline = release time + relative deadline*



From figure, absolute deadline is represented by the interval for a job Ji is the interval (ri, di].

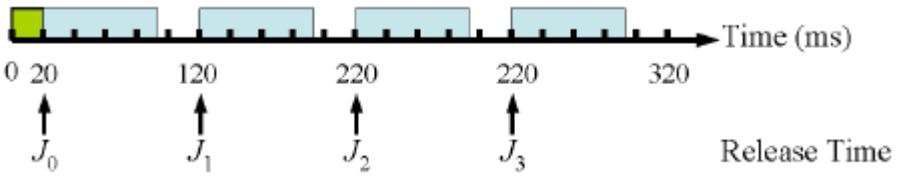
A constraint imposed on the timing behavior of a job a *timing constraint*. Deadlines are the example of timing constraints i.e. a timing constraint of a job can be specified in terms of its release time and relative or absolute deadlines.

For example:

A system to monitor and control a heating furnace and requires 20ms to initialize when turned on. After initialization, every 100 ms, and the system:

- Samples and reads the temperature sensor
- Computes the control-law for the furnace to process temperature readings, determine the correct flow rates of fuel, air and coolant
- Adjusts flow rates to match computed values

The periodic computations can be stated in terms of release times of the jobs computing the control-law: *J0, J1, …, Jk, …* as shown in fig below.



The release time of J_k is computed as below.

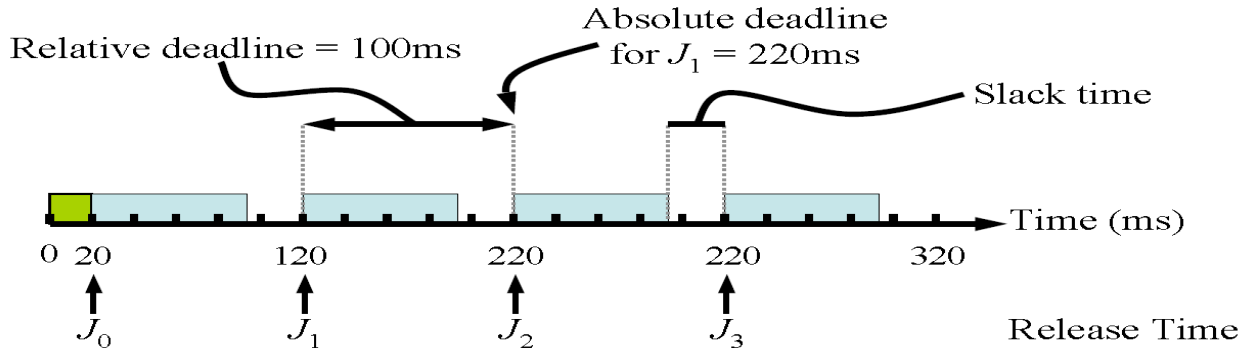
Release time = 20 + (k × 100) ms

Suppose each job must complete before the release of the next job then

- relative deadline of Jk' s is 100 ms

- absolute deadline of Jk' s is 20 + ((k + 1) × 100) ms

Alternatively, each control-law computation may be required to finish sooner i.e. the relative deadline is smaller than the time between jobs, allowing some slack time for other jobs. The difference between the completion time and the earliest possible completion time is called the slack time.



Hard and soft timing constraints:

The timing constraint are classified as hard and soft based on the functional criticality of jobs, usefulness of late results, and deterministic or probabilistic nature of the constraints.

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. E.g.

- a late command to stop a train may cause a collision
- a bomb dropped too late may hit a civilian population instead of the intended military target.

A timing constraint or deadline is *soft* if a few misses of deadlines do no serious harm but only the system's overall performance becomes poorer. The system performance becomes poorer when more and more jobs with soft deadlines complete late so that late completion of a job is undesirable.

Hard Timing Constraints and Temporal Quality-of-Service Guarantees:

The timing constraint of a job is hard then it is called as hard real-time job. The validation is required that meets the system timing constraints to demonstrate a real time system by using a provably correct, efficient procedure or exhaustive simulation and testing.

In case of soft timing constraint is impose on the job then it is called soft job and there is no system validation is required to demonstrate the real time system but timing constraint must meet the some statistical constraints i.e., a timing constraint specified in terms of statistical averages.

The temporal quality of service measures the systems in terms of different parameters like response time, jitter etc. when the system requires the validation of these parameter to guarantee and satisfaction of real time system over the define timing constraint then these timing constraint are called hard. Some real time system are operated to give the best quality of service but there is no care of violation of timing constraint slightly and no needs of validation then these timing constrains are called soft.

Hence the validation of temporal quality of system must be needed in case of hard real time system and there is no guarantee of best quality service whereas quality of service must garneted in soft real time system. Timing constraints can be expressed in many ways:

– Deterministic: it a constraint expressed in terms of numeric value.

- e.g. the relative deadline of every control-law computation is 50 ms; the response time of at most 1 out of 5 consecutive control-law computations exceeds 50ms

– Probabilistic: it is a constraint expressed in terms of probability.

- e.g. the probability of the response time exceeding 50 ms is less than 0.2

– In terms of some usefulness function

- e.g. the usefulness of every control-law computation is at least 0.8

Hard Real-Time versus Soft Real-Time:

Depending on the characteristics of the application, i.e., on factors outside the computer system there are two types of real time system.

1. Hard Real -Time system
2. Soft Real -Time system

Hard Real-Time System:

If a job must never misses its deadline then the system is called hard real-time system. For a hard real-time system, every deadline must be hit. In a real hard real-time system, if the system fails to hit the deadline even once the system is said to have failed.

A hard real-time system is also known as an immediate real-time system. It is a hardware or software that must operate within the confines of a stringent deadline. The application is considered to have failed if it does not complete its function within the given allocated time span. Some examples of hard real-time systems are medical application like pacemakers, aircraft control systems and anti-lock brakes. Some characteristics of hard real time system are:

- The hard real-time system is called guaranteed services.

- Response time is hard
- Often safety critical
- Size of data files are small and medium
- Peak load performance is predictable
- Use the autonomous error detection
- Have short-term data integrity

Soft Real -Time System:

If some deadlines can be missed occasionally acceptably with low probability then the system is called soft real time system. In a soft real time system, even if the system fails to meet the deadline one or more than once, the system is still not considered to have failed. For example, streaming audio-video.

- The soft real-time system is called best effort service.
- Response time is soft
- Non-critical safety
- Size of data files are large
- Peak load performance is degradable
- Use user assisted error detection
- Have long-term data integrity

Chapter-3

Reference model for real-time systems

Reference Model of Real-Time Systems:

The reference model of the real time system is developed to steady about the timing behavior of the system over the stetted constraints. A reference model is characterized by:

- A workload model that describes the applications supported by the system
- A resource model that describes the system resources available to the applications
- Algorithms that define how the application system uses the resources at all times

Processor and Resource:

Sometimes some elements of the system model are termed as processors and sometimes as resources, depending on how they are used by the model. For example, in a distributed system a computation job may invoke a server on a remote processor.

- If it is observed that how the response time of this job is affected by the job which is scheduled on its local processor then remote server serve as a resource.
- Remote server can be modelled as a processor.

There is no fixed rules to guide us in deciding whether to model something as a processor or as a resource, or to guide us in many other modelling choices but it deepens on the design criteria. A good model can give us better insight into the real-time problem that are considered during modelling. A bad model can confuse us and lead to a poor design and implementation. For example, a model transactions that query and update a database as jobs; these jobs execute on a database server so database server acts as the processor. If the database server uses a locking mechanism to ensure data integrity, then a transaction also needs the locks on the data objects it reads or writes in order to proceed. The locks on the data objects available on database server are considered as resources.

Real-time Workload Parameters:

Workload specifies the no of jobs assign to a processor. Workload on processors consists of jobs, each of which is a unit of work to be allocated processor time and other resources. The real time work load parameter are:

1. Number of tasks or jobs in the system:

In many embedded systems the number of tasks is fixed for each operational mode, and these numbers are known in advance. In case of some other systems the number of tasks may change as the system executes task but the number of tasks with hard timing constraints is known at all times. When the satisfaction of timing constraints is to be guaranteed, the admission and deletion of hard real-time tasks is usually done under the control of the run-time system.

2. Run- time system:

The run-time system must maintain information on all existing hard real-time tasks, including the number of such tasks, and all their real-time constraints and resource requirements.

Temporal Parameters of Job:

Each job J_i is characterized by its temporal parameters, interconnection parameters and functional parameters and tell us its timing constraints and behaviour. Its interconnection parameters tell us how it depends on other jobs and how other jobs depend on it and its functional parameters specify the intrinsic properties of the job. The temporal parameter of job are:

- Release time r_i
- Absolute deadline d_i
- Relative deadline D_i
- Feasible interval $(r_i, d_i]$

Where d_i and D_i are usually derived from the timing requirements of J_i, other jobs in the same task as J_i, and the overall system.

Periodic Task Model:

The periodic task model is a well-known deterministic workload model and best suited for hard real time system. When a model consists number of task τi has its period Ti and task τi is composed of sequence of jobs where Ti is minimal inter-arrival time between consecutive jobs and task computation time is the maximum computation time among all jobs of τi then this model is called periodic task model and corresponding task is called periodic task.

With its various extensions, the model characterizes accurately many traditional hard real-time applications, such as digital control, real-time monitoring, and constant bit-rate voice/video transmission. The jobs of a given task are repeated at regular and are modeled as periodic, with period p. The accuracy of model decreases with increasing jitter. Suppose a task T_i is a series of periodic Jobs J_{ij} which may be described with the following parameters:

- **p_i** - *period*, minimum inter-release interval between jobs in Task T_i.
- **e_i** - maximum execution time for jobs in task T_i.
- **r_{ij}** - release time of the j^{th} Job in Task i (J_{ij} in T_i).
- **φ_i** - Phase of Task T_i, equal to r_{i1}, i.e. it is the release time of the first job in the task T_i.
- **H** – *Hyper period* = Least Common Multiple of p_i for all i: H = l cm (pi), for all i. The number of jobs in a hyper period is equal to the sum of (H/p_i) over all i.
- **u_i** - utilization of Task T_i and is equal to e_i/p_i.
- **U** - Total utilization = Sum over all u_i.
- **d_i** - absolute deadline
- **D_i** - relative deadline
- **(r_i, d_i]** - feasible interval

Sporadic jobs:

Most real-time systems have to respond to external events which occur randomly. When such an event occurs the system executes a set of jobs in response. The release times of those 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 times.

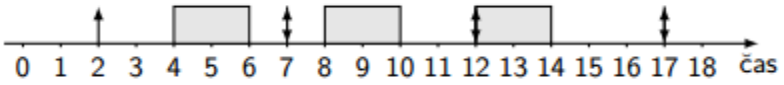
If the tasks containing jobs that are released at random time instants and have hard deadlines then they are called sporadic task. Sporadic tasks are treated as hard real-time tasks. To ensure that their deadlines are met is the primary concern whereas minimizing their response times is of secondary importance. For example,

- An autopilot is required to respond to a pilot's command to disengage the autopilot and switch to manual control within a specified time.
- A fault tolerant system may be required to detect a fault and recover from it in time to prevent disaster

When the task or job have no any deadlines or soft deadline then it is called aperiodic task or job. For example,

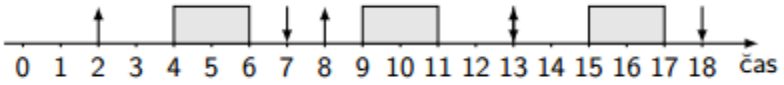
- An operator adjusts the sensitivity of a radar system. The radar must continue to operate and in the near future change its sensitivity.

For example: Periodic task τi with ri = 2, Ti = 5, ei = 2, Di = 5 can be executed like this (continues until infinity).



Legend: ↑ = job release time , ↓ = deadline.

According to Liu, this task can execute, for example, like this:



The rest of the world calls this sporadic task.

Release Time of Sporadic Job:

- The release times of sporadic and aperiodic jobs are random variables.
- The system model gives the probability distribution A(*x*) of the release time of such a job.
- A(*x*) gives us the probability that the release time of a job is at or earlier than *x*, or in the case of inter release time, that it is less than or equal to *x*.

Arrival Time:

Rather than speaking of release times for aperiodic jobs, sometimes it is termed as arrival time (or inter arrival time) commonly used in queuing theory. An aperiodic job arrives when it is released. A(x) is the arrival time distribution or inter arrival time distribution where A (x) is probability of release time of job is at or earlier than x.

Characterization of Execution Time:

The execution time e_i of job J_i is in the range $[e_i^-, e_i^+]$ where e_i^- is the minimum execution time and e_i^+ is the maximum execution time of job J_i. The e_i^- and e_i^+ of every hard real-time job J_i must be known even if there is unknown value of e_i. The purpose of determining e_i^- *and* e_i^+ to determine the deadline as well as execution time e_i cannot exceed than e_i^+.

Maximum Execution Time:

- For the purpose of determining whether each job can always complete by its deadline, it suffices to know its maximum execution time.
- In most deterministic models used to characterize hard real-time applications, the term execution time ei of each job Ji specifically means its maximum execution time.

- However we don't mean that the actual execution time is fixed and known, only that it never exceeds our ei (which may actually be ei+).

Precedence Constraint:

Data flow and control dependencies between the jobs can constrain the order in which the jobs can be executed such job or task have two main types of dependencies: Mutual exclusion and Precedence constraints. When Job Ji can start only after another job Jk finishes in a task model then such constraint is called precedence constraint. If jobs can execute in any order, they are said to be independent. Similarly when a tasks or job executed without any dependency on other tasks are called independent task or job. For Example: Consider an information server some of the precedence constraint are.

- Before a query is processed and the requested information retrieved, its authorization to access the information must first be checked.
- The retrieval job cannot begin execution before the authentication job completes.
- The communication job that forwards the information to the requester cannot begin until the retrieval job completes.

Similarly, in a radar surveillance system the signal processing task is the producer or track records which the tracker task is the consumer then

- Each tracker job processes the track records produced by a signal processing job.
- The tracker job is precedence constrained.

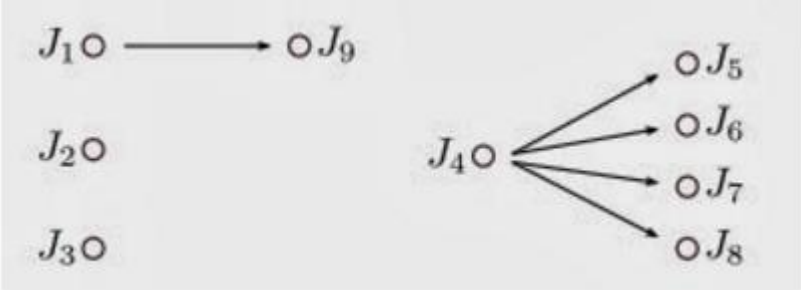
.Precedence Graph and Task graph:

Precedence relation on a set of jobs is a relation that determines precedence constrains among individual jobs. It is denoted by a partial order relation (<). A job Ji is a predecessor of another job jk (and jk is a successor of ji) if jk cannot begin execution until the execution of ji completes. This is represented as ji < jk then

- Ji is an immediate predecessor of jk (and jk is 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 precedence graph is a directed graph which represents the precedence constraints among a set of jobs J where each vertex represents a job in J. There is a directed edge from vertex Ji to vertex Jk when the job Ji is an immediate predecessor of job Jk. For example, A system contains nine non-preemptable jobs named Ji, for i = 1, 2, ..., 9. J_1 is the immediate predecessor of J9, and J4 is the immediate predecessor of J5, J6, J7, and J8. There are no other precedence constraints. For all the jobs, Ji has a higher priority than Jk if i < k. then the precedence graph can be constructed as,

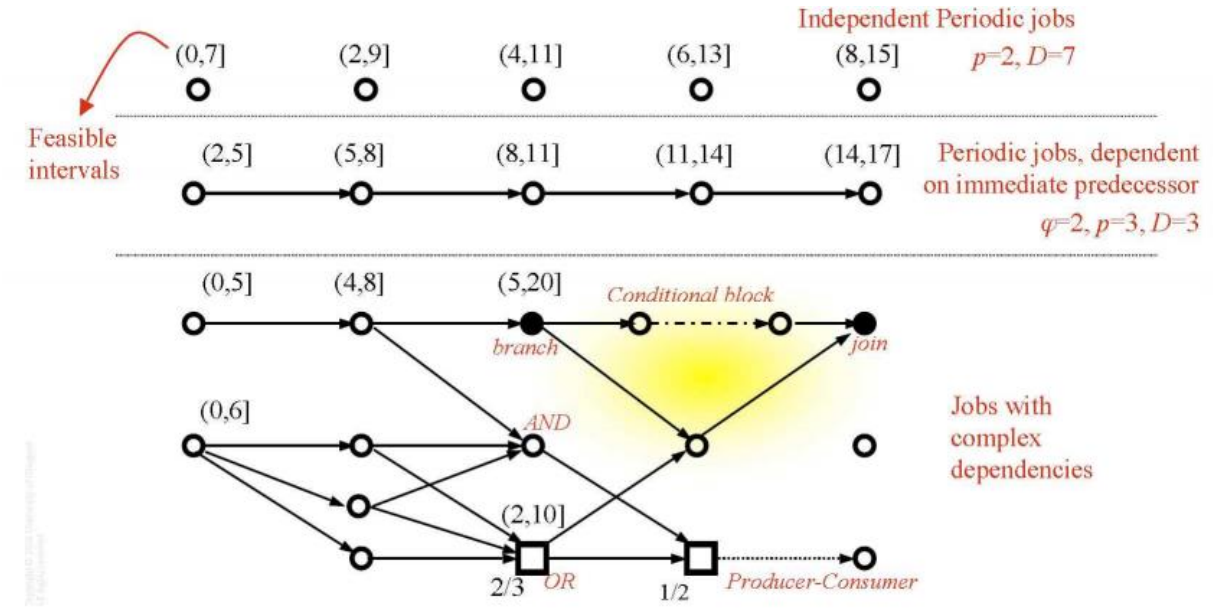


Task Graph:

A task graph, which gives us a general way to describe an application system. It is an extended form of precedence graph. As in a precedence graph, vertices represent jobs. They are shown as circles and squares (the distinction between them will come later). The numbers in the bracket above each job gives us its feasible interval and 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. Normally a job must wait for the completion of all immediate predecessors is an AND constraint represented Unfilled circle in the task graph .An OR constraint indicates that a job may begin after its release time if only some of the immediate predecessors have completed and represented by unfilled squares in the task graph . Conditional branches and joins are represented by filled in circles. A dotted edge represent a pair of producer/consumer jobs. For example:

The system shown in the sample task graph includes two periodic tasks. The task whose jobs are represented by the vertices in the top row has a phase 0, period 2, and relative deadline 7. The jobs in it are independent since there are no edges to or from these jobs. In other words, the jobs released in later periods are ready for execution as soon as they are released, even though some job released earlier is not yet complete. This is common with periodic tasks. The vertices in the

second row represent jobs in a periodic task with phase 2, period 3, and relative deadline 3. The jobs in it are dependent. The first job is the immediate predecessor of the second job, the second job is the immediate predecessor of the third job, etc. The precedence graph of the jobs in this task is a chain. A subgraph's being a chain indicates that for every pair of jobs Ji and
J k in the subgraph, either Ji < Jk or Jk < Ji . Hence the jobs must be executed in serial order.



Data Dependency:

Data dependency cannot be captured by a precedence graph. In many real-time systems jobs communicate via shared data hence data of one job is dependent with other and called as data dependency. Often the designer chooses not to synchronize producer and consumer jobs such that consumer requires the data at any time instead the producer places the data in a shared address space.

In this case the precedence graph will show the producer and consumer jobs as independent since they are apparently not constrained to run in turn.

In a task graph, data dependencies are represented explicitly by data dependency edges among jobs. There is a data dependency edge from the 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.

In multiple processor systems the volume of data to be transferred can be used to make decisions about scheduling of jobs on processors.

Sometimes the scheduler may not be able to schedule data dependent jobs independently. To ensure data integrity some locking mechanism must be used to ensure that only one job can access the shared data at a time. This leads to resource contention, which may also constrain the way jobs execute. However this constraint is imposed by scheduling and resource control algorithms. It is not a precedence constraint because it is not an intrinsic constraint on the execution order of jobs.

Functional Parameters:

While scheduling and resource control decisions are made independently of most functional characteristics of jobs, there are several functional properties that do affect these decisions. The workload model must explicitly describe these properties using functional parameters:

- Preemptivity
- Criticality
- Optional execution
- Laxity type

Preemptivity of Jobs:

Execution 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, the less urgent job can resume its 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 can later be resumed from the point of suspension.

A job is non-preemptable if it must be executed from start to completion without interruption. This constraint may be imposed because its execution, if suspended, must be executed again from the beginning. Sometimes a job may be preemptable everywhere except for a small portion which is constrained to be non-preemptable.

An example is an interrupt handling job. An interrupt handling job usually begins by saving the state of the processor. This small portion of the job is non-preemptable since suspending the execution may cause serious errors in the data structures shared by the jobs.

During preemption the system must first save the state of the preempted job at the time of preemption so that it can resume the job from that state. Then the system must prepare the execution environment for the preempting job before starting the job. These actions are called a context switch. The amount of time required to accomplish a context switch is called a context-switch time. The terms context switch and context-switch time are used to mean the overhead work done during preemption, and the time required to accomplish this work.

For example, in the case of CPU jobs, the state of the preempted job includes the contents of the CPU registers. After saving the contents of the registers in memory and before the preempting job can start, the operating system must load the new register values, clear pipelines, perhaps clear the caches, etc.

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 a job is with respect to other jobs. It also define by the term priority and weight. The more important a job, the higher its priority or the larger its weight. During an overload when it is not possible to schedule all the jobs to meet their deadlines, it may make sense to sacrifice the less critical jobs, so that the more critical jobs meet their deadlines. For this reason, some scheduling algorithms try to optimize weighted performance measures, taking into account the importance of jobs.

Optional Executions:

It defines the identification of jobs (or portion of jobs) that are either optional or mandatory.

Laxity type or Laxity function:

Laxity can be used to indicate the relative importance of a time constraint, for example hard versus soft constraints. May be supplemented with a utility function (for soft constraints) that gives the usefulness of a result versus its degree of tardiness.

Resource Parameters of Jobs:

A job require a processor and some resources throughout its execution. The resource parameters of each job give us the type of processor and the units of each resource type required by the job and the time intervals during its execution when the resources are required. These parameters are needed to support resource management decisions.

The resource parameters of jobs give us a partial view of the processors and resources from the perspective of the applications. Sometimes it need to describe the characteristics of processors and resources independent of the application. For this there parameters of resources.

A resource parameter is preemptivity. A resource is non-preemptable if each unit of the resource is constrained to be used serially. Once a unit of a non-preemptable resource is allocated to a job, other jobs needing the unit must wait until the job completes its use. If jobs can use every unit of a resource in an interleaved way, the resource is preemptable. A lock on a data object is an example of a non-preemptable resource. This does not mean that the job is non-preemptable on others resources or on the processor. The transaction can be preempted on the processor by other transactions not waiting for the locks.

Resource Graph:

A resource graph describes the configuration of resources. There is a vertex Ri for every processor or resource Ri in the system. The attributes of the vertex are the parameters of the resource.

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 resource graphs represent the relationship among resources. There are 2 types of edges in resource graphs.

1. Is-a-part-of edge:

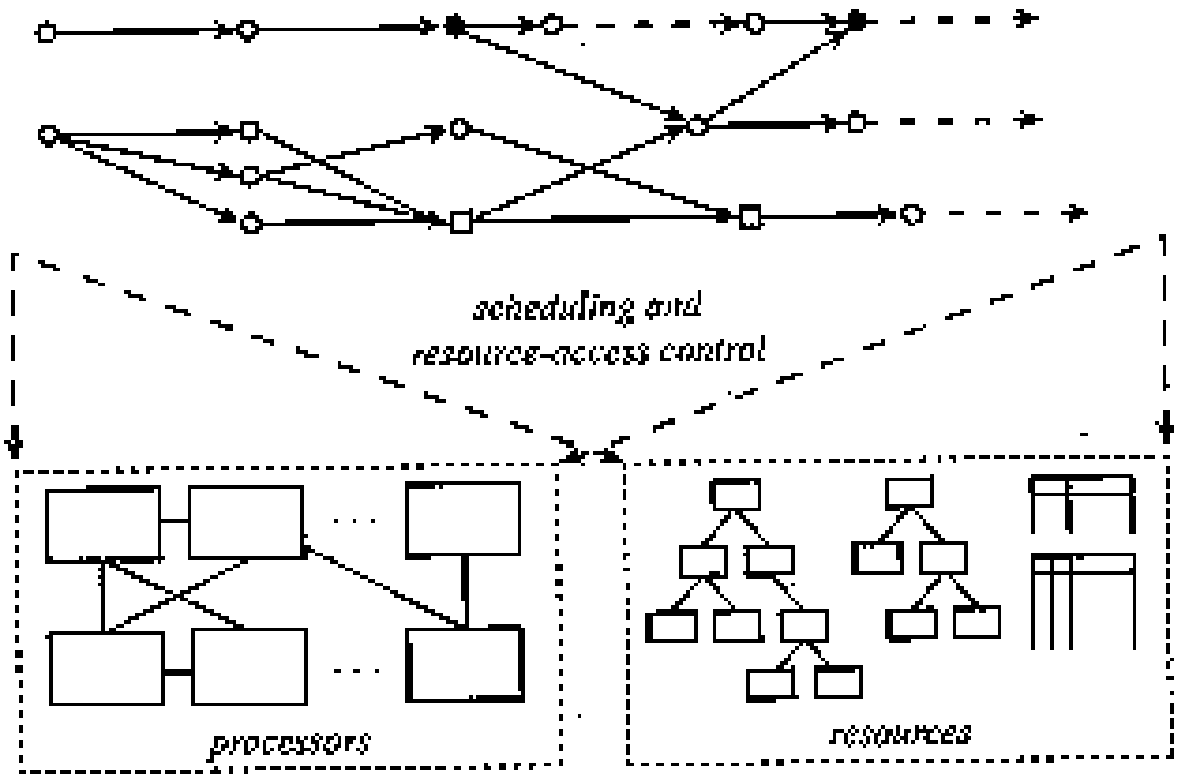
An edge from vertex Ri to vertex Rk can mean that Rk is a component of Ri is called is-a-part-of edge e.g. a memory is part of a computer and so is a monitor. The subgraph containing all the is-a-part-of edges is a forest. The root of each tree represents a major component, with subcomponents represented by vertices. e.g. the resource graph of a system containing 2 computers consists of 2 trees. The root of each tree represents a computer with children of this vertex including CPUs etc.

2. Accessibility edges:

Some edges in resource graphs represent connectivity between components. These edges are called accessibility edges. For example 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 of the other computer.

Each accessibility edge may have several parameters. For example, a parameter of an accessibility edge from a processor Pi to another Pk is the cost of sending a unit of data from a job executing on Pi to a job executing on Pk.

Scheduling Hierarchy:



The figure 'model of a real-time system' shows the three elements of our model of real-time systems. The application system is represented by

- a task graph which gives the processor time and resource requirements of jobs, their timing constraints and dependencies
- A resource graph describing the resources available to execute the application system, their attributes and rules governing their use
- And between these graphs are the scheduling and resource access-control algorithms used by the operating system

Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols. The scheduler is a module that implements these algorithms. The scheduler assigns processors to jobs, or equivalently, assigns jobs to processors. A schedule is an assignment by the scheduler of all the jobs in the system on the available processors. For proper execution of job, a schedule must be valid. A valid schedule is the schedule satisfies the following conditions:

- ✓ Every processor is assigned to at most one job at any time.

- ✓ Every job is assigned at most one processor at any time.
- ✓ No job is scheduled before its release time.
- ✓ Depending on the scheduling algorithms used, the total amount of processor time assigned to every job is equal to its maximum or actual execution time.
- ✓ All the precedence and resource usage constraints are satisfied.
- ✓ Schedule must be feasible and scheduling algorithms must be optimal.

A **valid schedule** is a **feasible schedule** if every job completes by its deadline and in general meets its timing constraints. 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. If an optimal algorithm cannot find a feasible schedule, we can conclude that a given set of jobs cannot feasibly be scheduled by any algorithm.

Chapter- 4

Commonly Used Approaches to Real-Time Scheduling

Three commonly used approaches to real-time scheduling are:

- Clock-driven
- Weighted round-robin
- Priority-driven

The weighted round-robin approach is mainly used for scheduling real-time traffic in high-speed switched networks. It is not ideal for scheduling jobs on CPUs.

Clock-Driven Approach:

Clock-driven scheduling is also called as time-driven scheduling. When scheduling is *clock-driven*, decisions are made at specific time instants on what jobs should execute when. Typically in clock-driven scheduling system, all the parameters of hard real-time jobs are fixed and known.

A schedule of the jobs is computed off-line and is stored for use at run-time. The scheduler schedules the jobs according to this schedule at each scheduling decision time. Hence scheduling overhead at run-time is minimized. Scheduling decisions are usually made at regularly spaced time instants.

One way to implement this is to use a hardware timer set to expire periodically which causes an interrupt which invokes the scheduler. When the system is initialized, the scheduler selects and schedules the jobs 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 repeats these actions.

Round-Robin Approach:

The round-robin approach is commonly used for scheduling time-shared applications. When jobs are scheduled in a round-robin system, every job joins a first-in-first-out (FIFO) queue when it becomes ready for execution. The job at the head of the queue executes for at most one-time slice. If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue to wait for its next turn.

When there are *n* ready jobs in the queue, each job gets one time slice in *n* that is in every *round*. The length of the time slice is relatively short (typically tens of milliseconds) so the execution of each jobs begins almost immediately after it becomes ready.

Generally, each job gets 1/*n*th share of the processor when there are *n* jobs ready for execution. This is why the round-robin algorithm is also known as the processor-sharing algorithm.

Weighted Round-Robin Approach:

The *weighted round-robin algorithm* is used for scheduling real-time traffic in high-speed switched networks. In this approach, different jobs may be given different *weights* rather than giving an equal shares of the processor for ready jobs.

In weighted round robin each job *Ji* is assigned a weight *Wi* where each job will receive *Wi* consecutive time slices each round, and the duration of a round is equals to the sum of the weights of all the ready jobs for execution. We can speed up or slow down the progress of each job by adjusting the weights of jobs.

When fraction of time of processor allocated to a job then a round-robin scheduler delays the completion of every job. If round-robin scheduling is used to schedule precedence constrained jobs then response time of set of jobs becomes very large. For this reason, the weighted round-robin approach is unsuitable for scheduling such jobs.

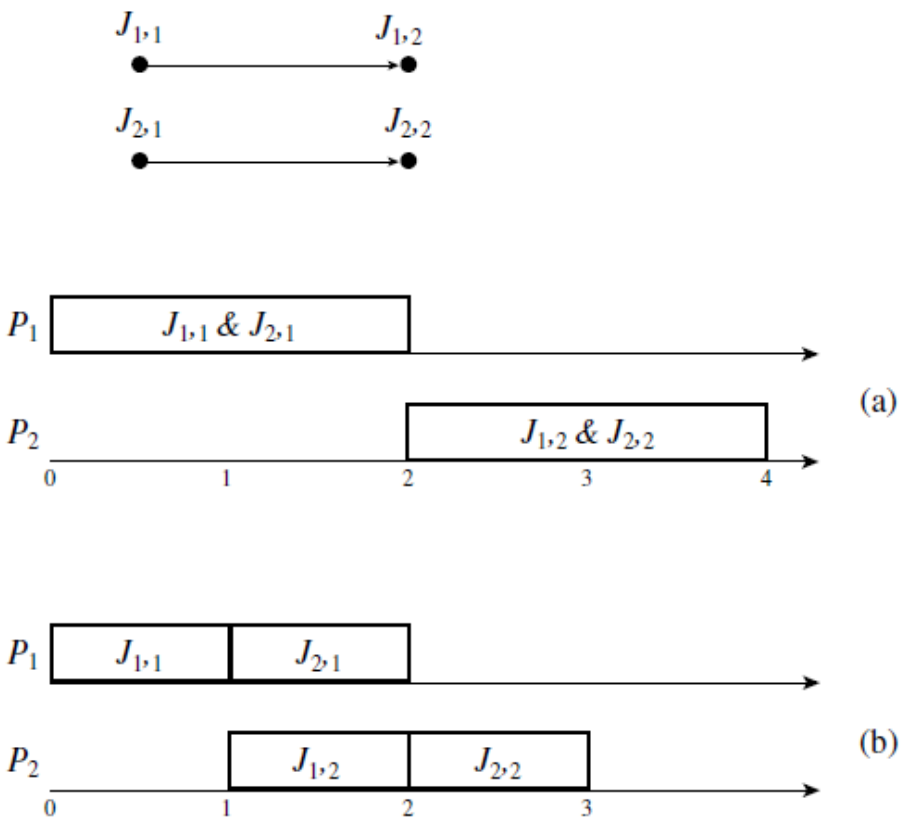
For example consider two sets of jobs $\mathbf{J_1} = \{J_{1,1}, J_{1,2}\}$ and $\mathbf{J_2} = \{J_{2,1}, J_{2,2}\}$

- ✓ The release times of all jobs are 0
- ✓ The execution times of all jobs are 1
- ✓ $J_{1,1}$ and $J_{2,1}$ execute on processor P_1
- ✓ $J_{1,2}$ and $J_{2,2}$ execute on processor P_2
- ✓ Suppose that $J_{1,1}$ is the predecessor of $J_{1,2}$
- ✓ Suppose that $J_{2,1}$ is the predecessor of $J_{2,2}$

Figure (a) shows 'weighted round-robin scheduling' that both sets of jobs complete approximately at time 4. If the jobs are scheduled in a weighted round-robin manner one after the other, one of the chains can complete at time 2 and the other at time 3 shown in figure (b).

Suppose that the result of the first job in each set is piped to the second job in the set then second job be executed latter after each one or a few time slices of the former complete. Then it is better to schedule the jobs on a round-robin basis, because both sets can complete a few time slices after time 2.

In a switched network a downstream switch can begin to transmit an earlier portion of the message as soon as it receives the portion. It does not have to wait for the arrival of the rest of the message. The weighted round-robin approach does not require a sorted priority queue, only a round-robin queue. This is a distinct advantage for scheduling message transmissions in ultrahigh-speed networks since fast priority queues are very expensive



Priority- Driven Approach:

The term priority-driven algorithms refer to a class of scheduling algorithms that never leave any resource idle intentionally. A resource becomes idles only when job does not require the resource for execution. It is a event driven approach for job scheduling and scheduling decision are made only when release and completion of job occur. Commonly used terms for this approach are greedy scheduling, list scheduling, and work-conserving scheduling.

A priority-driven algorithm is said to be greedy because it tries to make locally optimal decisions. The resource becomes idle when resources are not locally optimal.

The term list scheduling is also used because any priority-driven algorithm can be implemented by assigning priorities to jobs. In this approach, Jobs ready for execution are placed in one or more queues ordered by the priorities of the jobs. At any scheduling decision time, the jobs with the highest priorities are scheduled and executed on the available processors. Hence a priority-driven scheduling algorithm is defined largely by the list of priorities it assigns to jobs.

It is also called as work conserving scheduling since when a processor or resource is available and some job can use it to make progress, a priority-driven algorithm never makes the job in wait i.e. after completion of a job another enters into execution

Examples include:

Most scheduling algorithms used in non-real-time systems are priority-driven they are

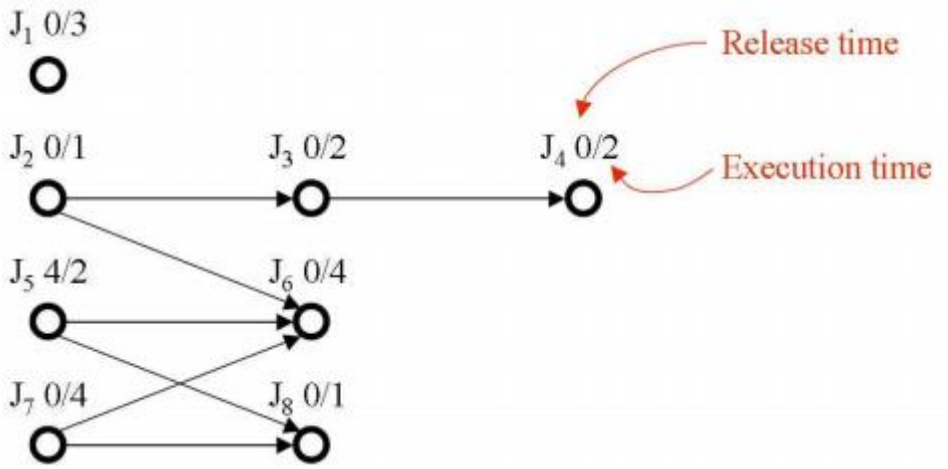
- ✓ FIFO (first-in-first-out) and LIFO (last-in-first-out) algorithms which assign priorities to jobs based on their release times.
- ✓ SETF (shortest-execution-time-first) and LETF (longest-execution-time-first) algorithms which assign priorities based on job execution times.

Real-time priority scheduling assigns priorities based on deadline or some other timing constraint they are:

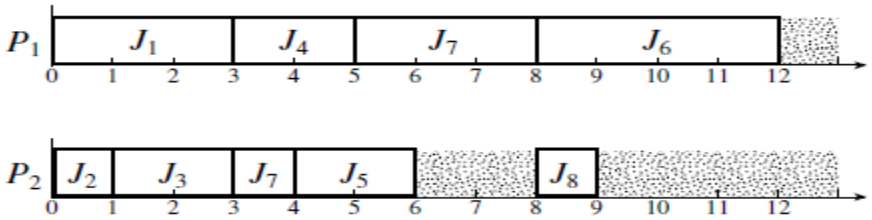
- ✓ Earliest deadline first
- ✓ Least slack time first
- ✓ Etc.

Consider a task graph with following condition,

- – Jobs $J_1, J_2, \ldots, J_8$, where J_i had higher priority than J_k if $i < k$.
- – Jobs are scheduled on two processors P1 and P2
- – Jobs communicate via shared memory, so communication cost is negligible –
- – The schedulers keep one common priority queue of ready jobs



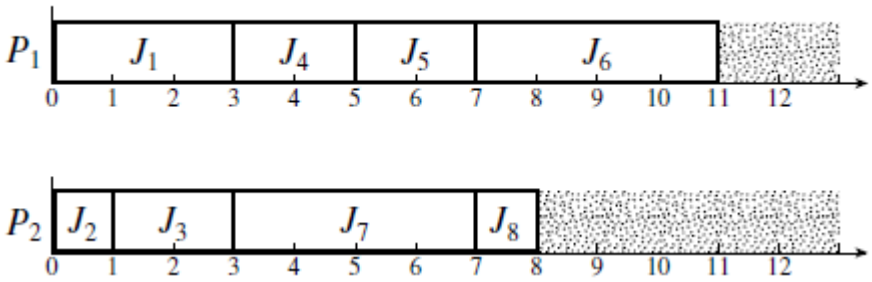
When the jobs are scheduled in preemptive priority driven approach the jobs are scheduled as



Compiled By: Loknath Regmi

- At time 0, jobs J1, J2, and J7 are ready for execution.
- They are the only jobs in the priority queue at this time.
- Since J1 and J2 have higher priorities than J7 they are ahead of J7 in the queue and hence are scheduled.
- At time 1, J2 completes and hence J3 becomes ready. J3 is placed in the priority queue ahead of J7 and is scheduled on P2, the processor freed by J2.
- At time 3, both J1 and J3 complete. J5 is still not released. J4 and J7 are scheduled.
- At time 4, J5 is released. Now there are three ready jobs. J7 has the lowest priority among them so it is preempted and J4 and J5 have the processors.
- At time 5, J4 completes. J7 resumes on processor P1.
- At time 6, J5 completes. Because J7 is not yet completed, both J6 and J8 are not yet ready for execution. Thus processor P2 becomes idle.
- J7 finally completes at time 8. J6 and J8 can now be scheduled on the processors.

When the jobs are scheduled in preemptive priority driven approach the jobs are scheduled as



- Before time 4 this schedule is the same as before.
- However at time 4 when J5 is released, both processors are busy. J5 has to wait until J4 completes at time 5 before it can begin execution.
- It turns out that for this system, postponement of the higher priority job benefits the set of jobs as a whole.
- The entire set completes one time unit earlier according to the non-preemptive schedule.

When is preemptive scheduling better than non-preemptive scheduling and vice versa?

Dynamic vs. Static Systems:

If jobs are scheduled on multiple processors, and a job can be dispatched from the priority run queue to any of the processors then such system is called dynamic system that means A job migrates if it starts execution on one processor and is resumed on a different processor.

If jobs are partitioned into subsystems, and each subsystem is bound statically to a processor then it is called static system.

Static system provides the poor performance in comparison with dynamic system in terms of overall response time of job but it is possible to validate static systems, whereas this is not always true for dynamic systems. Hence most hard real time systems are static.

Effective Release Times and Deadlines:

Sometimes the release time of a job may be later than that of its successors, or its deadline may be earlier than that specified for its predecessors then concept of effective release time or effective deadline comes to play.

- ✓ Effective release time

If a job has no predecessors then its effective release time is equals to its release time. When it has predecessors then its effective release time is the maximum of its release time and the effective release times of its predecessors

- ✓ Effective deadline

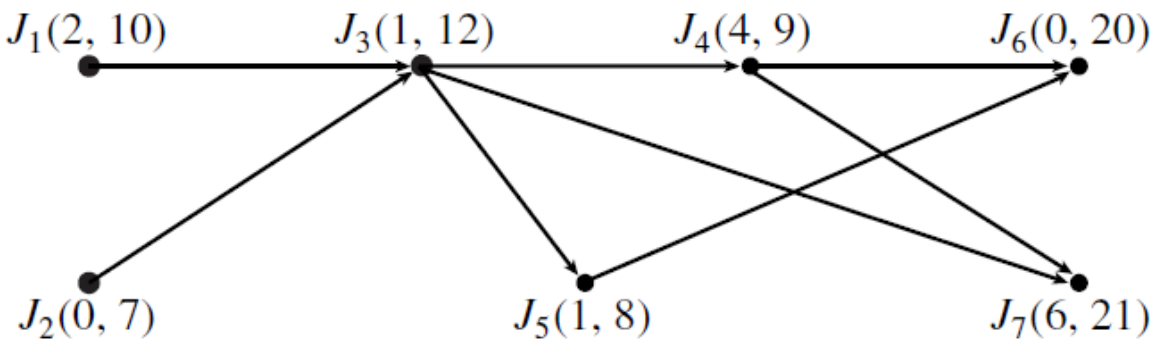
If a job has no successors then its effective deadline is equals to its deadline. When It if has successors then its effective deadline is the minimum of its deadline and the effective deadline of its successors.

To generate more accurate real time system on multiprocessor environment, effective release time and deadlines must be considered but there no unnecessary on single processor with preemptable jobs. It is feasible to schedule any set of jobs according to their actual release times and deadline when feasible to schedule according to effective release times and deadlines. Schedule use effective release times and deadlines as if all jobs independent then ignore the all precedence constraints.

Consider the following example whose task graph is given in the following figure.

Effective Release time

- The numbers in brackets next to each job are its given release time and deadline.
- Because J_1 and J_2 have no predecessors, their effective release times are their given release times, 2 and 0 respectively.
- The given release time of J_3 is 1, but the latest effective release time of its predecessors is 2 (that of J_1) so its effective release time is 2.
- The effective release times of J_4, J_5, J_6, J_7 are 4, 2, 4, 6 respectively.



Effective deadlines

- ***J₆*** and ***J₇*** have no successors so their effective deadlines are their given deadlines, 20 and 21 respectively.
- Since the effective deadlines of the successors of ***J₄*** and ***J₅*** are later than the given deadlines of ***J₄*** and ***J₅***, the effective deadlines of ***J₄*** and ***J₅*** are equal to their given deadlines, 9 and 8 respectively.
- However the given deadline of ***J₃*** (12) is larger than the minimum value (8) of its successors, so the effective deadline of ***J₃*** is 8.
- Similarly the effective deadlines of ***J₁*** and ***J₂*** are 8 and 7 respectively.

Priority Scheduling Based on Deadline:

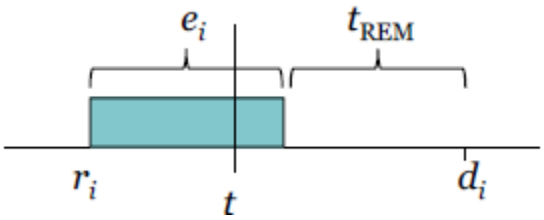
Priority scheduling based on deadline are:

1. Earliest deadline first (EDF)
2. Least slack time first (LST)

Earliest deadline first (EDF):

It is a type of the priority scheduling algorithms assign priority to jobs based on deadline. In this approach, earlier the deadline gets higher the priority. Simply, it just requires knowledge of deadlines.

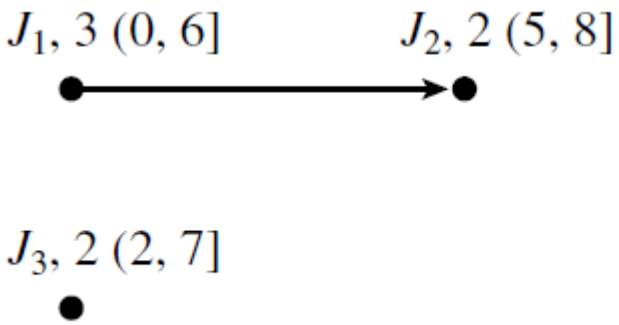
Least Slack Time first (LST):



Suppose a job J i has deadline d i, execution time e_i, and was released at time r_i then at time $t < d_i$:

- Remaining execution time $t_{rem} = e_i - (t - r_i)$
- Slack time $t_{slack} = d_i - t - t_{rem}$

In this approach, priority to jobs be assigned based on slack time .The smaller the slack time jobs gets higher the priority then next higher slack time and so on. It is more complex for implementation and requires knowledge of execution times and deadlines properly. Knowing the actual execution time is often difficult a priori, since it depends on the data, need to use worst case estimate**.** For example



Job J1 of example is released at time 0 and has its deadline at time 6 and execution time 3. Hence its slack is 3 at time 0. The job starts to execute at time 0. As long as it executes its slack remains 3 because at any time before its completion its slack is 6 - t - (3 – t).
Suppose J1 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. The LST algorithm assigns priorities to jobs based on their slacks. The smaller the slack, the higher the priority.

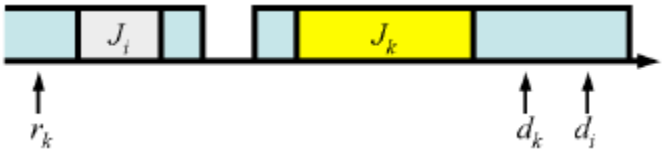
Optimality of EDF and LST:

These algorithms are optimal only when they always produce a feasible schedule if one exists.
It is constraints on a single processor as long as preemption is allowed and jobs do not contend for resources.

Optimality of EDF (Proof):

Theorem: When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if J has feasible schedules

Proof:
To show the optimality of EDF, we have to require to show any feasible schedule can be transformed into another an EDF schedule



If J i is scheduled to execute before J k, but J i's deadline is later than J k's such that there exist two conditions:

- The release time of J k is after the J i completes that means they're already in EDF order.
- The release time of J k is before the end of the interval in which J i executes.

This is always possible to swap J i and J k since J i's deadline is later than J k's such that

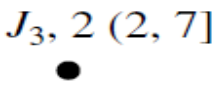
We can move any jobs following idle periods forward into the idle period then

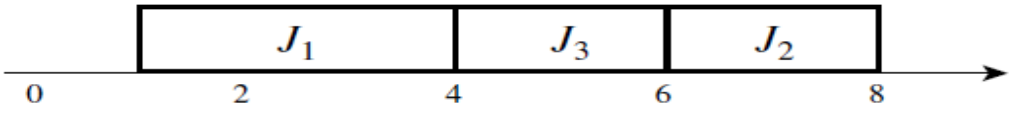
The result is an EDF schedule.

When deadline of J i's is earlier than J k's there is no possibility to generate the other feasible schedule and EDF failed to produce a feasible schedule. Hence the optimality of EDF is verified.

Latest-Release-Time (LRT) Algorithm:

The Latest-Release-Time algorithm treats release times as deadlines and deadlines as release times and schedules jobs backwards, starting from the latest deadline of all jobs, in a priority-driven manner, to the current time. The 'priorities' are based on the later the release time, the higher the 'priority'. Because it may leave the processor idle when there are jobs awaiting execution, the LRT algorithm is not a priority-driven algorithm. For example





In the following example, the number next to the job is the execution time and the feasible interval follows it.

The latest deadline is 8, so time starts at 8 and goes back to 0. At time 8, J2 is "ready" and is scheduled. At time 7, J3 is also "ready" but because J2 has a later release time, it has a higher priority, so J2 is scheduled from 7 to 6.

When J2 "completes" at time 6, J1 is "ready" however J3 has a higher priority so is scheduled from 6 to 4.

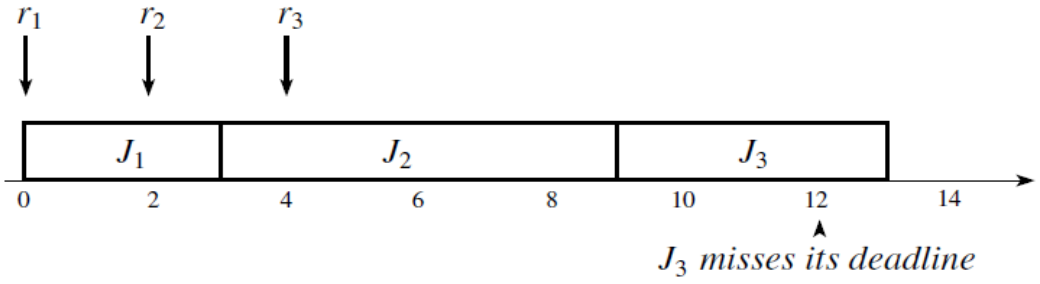
Finally J1 is scheduled from 4 to 1.

The following corollary states that the LRT algorithm is also optimal under the same conditions that the EDF algorithm is optimal. Its proof follows straightforwardly from the proof of EDF.

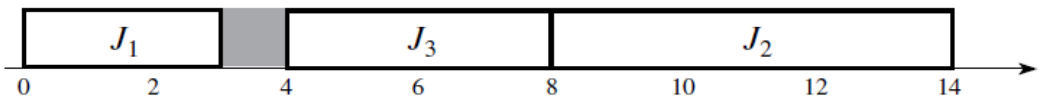
"When preemption is allowed and jobs do not contend for resources, the LRT algorithm can produce a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if feasible schedules of J exist".

Non-optimality of EDF and LST:

EDF and LST algorithms are not optimal if preemption is not allowed or there is more thhan one processor .Consider the following 3 independent non-preemptable jobs, J_1, J_2, J_3, with release times 0, 2, 4 and execution times 3, 6, 4, and deadlines 10, 14, 12 respectively.



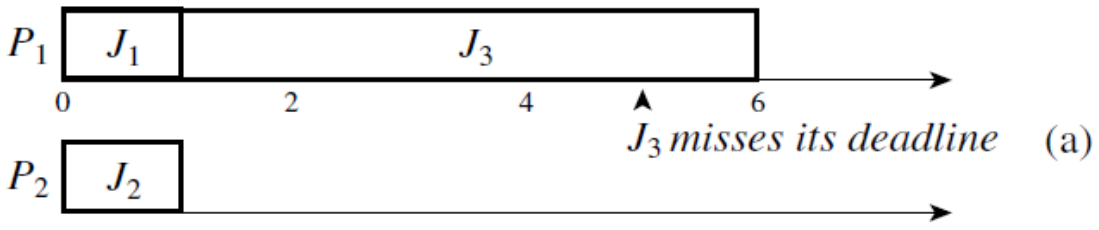
(a)

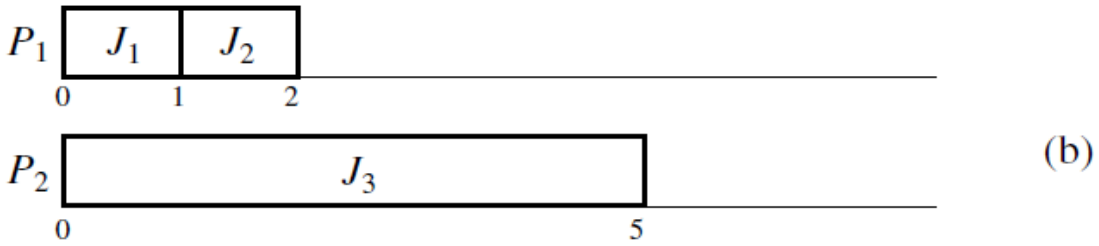


(b)

Both EDF and LST would produce the infeasible schedule shown in fig (a) Whereas a feasible schedule is possible fig (b) but left the idle period.

Consider we have two processors and three jobs J_1, J_2, J_3, with execution times 1, 1, 5 and deadlines 1, 2, 5 respectively. All with release time 0.





Then EDF gives the infeasible schedule (a) whereas LST gives a feasible schedule (b) but in general LST is also non-optimal for multiprocessors.

Hence both EDF and LST are not optimal in case of jobs non preemptive constrain and multiprocessor environment.

Validating Timing Constraints in Priority-Driven Systems:

Compared with the clock-driven approach, the priority-driven scheduling approach has many advantages. They are

- Easy to implement
- Often have simple priority assignment rules
- If the priority assignment rules are simple, the run-time overheads of maintaining priority queues can be small
- Does not require information about deadlines and release times in advance so it is suited to applications with varying time and resource requirements

Despite its merits, the priority-driven approach has not been widely used in hard real-time systems, especially safety-critical systems. The main reason for this is that the timing behaviour of a priority-driven system is not deterministic when job parameters vary. Thus it is difficult to validate

that the deadlines of all jobs scheduled in a priority-driven manner indeed meet their deadlines when job parameters vary.

Validation Problem:

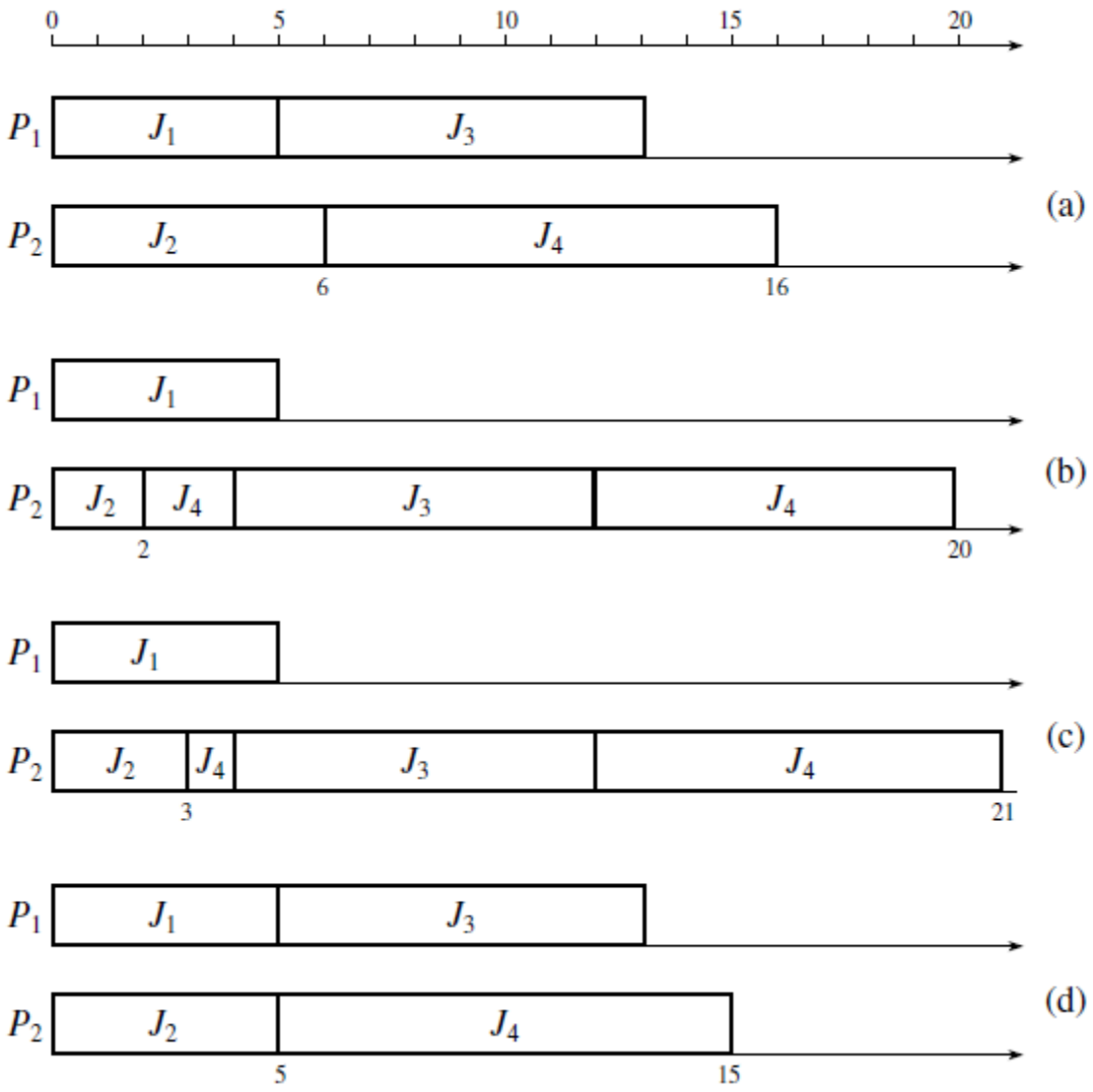
Validation problem stated that for given a set of jobs, the set of resources available to the jobs, and the scheduling (and resource access-control) algorithm to allocate processors and resources to jobs, determine whether all the jobs meet their deadlines. This is a very difficult problem to solve if there is no all the information about all the jobs available in advance.

Consider a simple system of four independent jobs scheduled on two processors in a priority-driven manner with following characteristics.

- There is a common priority queue and the priority order of jobs is J_1, J_2, J_3, J_4, with J_1 being of highest priority.
- It is a dynamic system.
- Jobs may be preempted but never migrated to another processor.
- The release times, deadlines and execution times or the jobs are provided in the table.

	r_i	d_i	$[e_i^-, e_i^+]$
J_1	0	10	5
J_2	0	10	[2, 6]
J_3	4	15	8
J_4	0	20	10

Suppose we schedule the jobs according to their priorities and try out J_2 with its maximum execution time 6 and also with its minimum execution time 2 shown in fig (a) and (b).



J4 is concerned, the worst-case schedule is (c) when execution time of J2 is 3 and J4 completes at 21 missing the deadline. The best-case schedule for J_4 is (d) when J_2 has execution time 5 and J_4 completes at time 15.

This is known as a scheduling anomaly, an unexpected timing behaviour of priority-driven systems and difficult to validate the system in case of parity driven scheduling approach.

Off line Vs Online Scheduling:

The scheduling that makes use of pre-computed schedule of all hard real time jobs i.e schedule is computed at offline before the system begins to execute and the computation is based on the knowledge of release time , processor time as well as resource requirement of all jobs for all time is called off-line scheduling. The example of offline scheduling is clock driven scheduling.

When the operation mode of the system changes, new schedule specifying when each job in new mode is also pre-computed and stored for use. The major dis advantage of offline schedule is inflexibility and only useful for deterministic system because shows deterministic timing behavior and optimize the utilization of resources up to 100%.

When the scheduler makes each scheduling decision without knowledge about the jobs that will be released in future and parameter of each job known to scheduler only after release of job then it is called online scheduling. The example of priority driven scheduling.

Online scheduling is suitable for a system whose future workload is unpredictable and there is one processor, optimal online algorithms exists.

Chapter- 5

Clock Driven Scheduling

Assumptions and Notation for Clock-Driven Scheduling:

Assumptions:

To represent the clock driven scheduling, following assumptions and notations are used. The assumptions are:

1. The clock driven scheduling is applicable to deterministic system.
2. . There is a constant number *n periodic tasks* in the system.
3. A restricted periodic task model can be used such that
    - Variations in inter-release times of jobs are negligibly small.
    - The parameters of all periodic tasks are known a priori.
    - Each job in T_i is released p_i units of time after the previous job in T_i.
4. Each job $J_{i,k}$ is ready for execution at its release time $r_{i,k}$ such that
    - There are aperiodic jobs released at unexpected time instants.
    - Aperiodic jobs are placed in special queue.
    - New jobs are added to the queue without need to notify scheduler.
    - When processor is available aperiodic jobs are scheduled.
    - There are no sporadic jobs (this assumption will be relaxed later).
5. There are no sporadic jobs.

Notations:

The notation of clock driven scheduling consists of four different parameters that is used to refer the job. The combination of four parameters is called tuple. The standard notation is $T_i = (\varphi_i, p_i, e_i, D_i)$ where T_i refers a periodic task with phase φ_i, period p_i, execution time e_i, and relative deadline D_i.

– Default phase of T_i is $\varphi_i = 0$, default relative deadline is the period $D_i = p_i$.

– Omit elements of the tuple that have default values.

Example:

i) $T_1 = (1, 10, 3, 6) \Rightarrow \varphi_1 = 1 , p_1 = 10, e_1 = 3 , D_1 = 6$

$J_{1,1}$ released at 1, deadline 7 and $J_{1,2}$ released at 11, deadline 17.

ii) $T_2 = (10, 3, 6) \Rightarrow \varphi_2 = 0 , p_2 = 10 , e_2 = 3 , D_2 = 6$

$J_{2,1}$ released at 0, deadline 6 and $J_{2,2}$ released at 10 and so on…. deadline 16

iii) $T3 = (10, 3) \Rightarrow \varphi 3 = 0, p3 = 10, e3 = 3, D3 = 10.$

$J_{3,1}$ released at 0, deadline 10 and $J_{3,2}$ released at 10, deadline 20.

Static, Clock-Driven Scheduler:

When the parameter of job with hard deadline are known before the system begins the execution then the static schedules of jobs can be developed at offline and processor time allocated to a jobs is equal to its maximum execution time. The scheduler dispatches the jobs according to the static schedule and repeats the jobs in each hyper periods. The static schedule guarantees that each job completes by its deadline and no job overrun can occur.

Example:

Four independent periodic tasks: T1 = (4, 1), T2 = (5, 1.8), T3 = (20, 1), T4 = (20, 2)

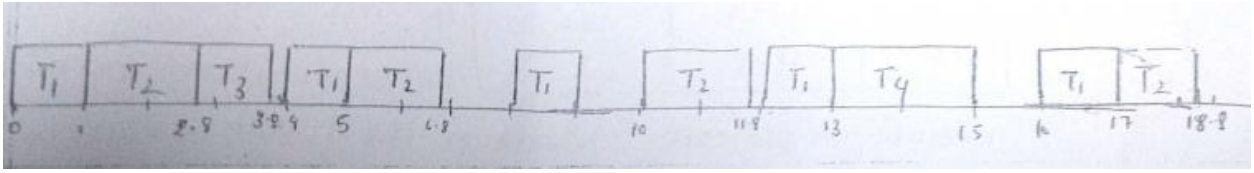
Utilization = 1/4 + 1.8/5 + 1/20 + 2/20 = 0.76

Hyperperiod = LCM (4, 5, 20, 20) = 20

According to this specification, schedules be generated as:

Time	Release Job	Running Job
0	$J_{11}, J_{21}, J_{31}, J_{41}$	$J_{11}(T_1)$
1	J_{21}, J_{31}, J_{41}	$J_{21}(T_2)$
2.8	J_{31}, J_{41}	$J_{31}(T_3)$
3.8	J_{41}	----------
4	J_{12}, J_{41}	$J_{12}(T_1)$
5	J_{13}, J_{41}	$J_{22}(T_2)$
6.8	J_{41}	----------
8	J_{23}, J_{41}	$J_{13}(T_1)$
9	J_{41}	----------
10	J_{23}, J_{41}	$J_{23}(T_2)$
11.8	J_{41}	----------
12	J_{14}, J_{41}	$J_{14}(T_1)$
13	J_{41}	$J_{41}(T_4)$
15	J_{24}	----------
16	J_{15}, J_{24}	$J_{15}(T_1)$
17	J_{24}	$J_{24}(T_2)$

Corresponding static schedule be constructed as:



In this schedule T_1 starts its execution at 0 time and repeats after each periods. Some intervals are not used by periodic task called as slack time. This adds the advantage since other a periodic jobs can be executed here.

Types of Clock Driven Schedules:

There are two types of clock driven schedules.

1. Table Driven Scheduling
2. Cyclic Schedule

Table Driven Scheduling:

Table driven schedulers usually pre-compute which task would run when and store this schedule in a table at the time the system is designed. Rather than automatic computation of schedule by the scheduler, the application programmer can be given the freedom to select his own schedule for the set of tasks in the application and store the schedule in a table (called schedule table) to be used by the scheduler at the run time. An example of a schedule table is shown as following fig. There is difficult to implement the scheduling since the table becomes very large in case of large and complex system.

Tasks	Start time in millisecond
T1	0
T2	3
T3	10
T4	12
T5	17

Fig: Time Driven Scheduling

Cyclic Schedule:

Cyclic schedules are very popular and extensively used in industry. Cyclic schedules are simple, efficient and are easy to program. An example application where cyclic schedule is used, is a temperature controller. A temperature controller periodically samples the temperature of a room and maintains it at a preset value. Such temperature controllers are embedded in typical computer-controlled air conditioners.

Tasks	Frame Number
T3	F1
T1	F2
T3	F3
T4	F2

Fig. Example schedule table for cyclic scheduler

A cyclic scheduler repeats a pre-computed schedule. The pre-computed schedule needs to be stored only for one major cycle.

General Structure of Cyclic Schedules:

The arbitrary driven cyclic schedules are flexible but they are inefficient. They depends on accurate time interrupt based on execution time of task and they also have high scheduling overloads. Hence the clock driven scheduling are implemented by using structural approach rather than tabular approach.

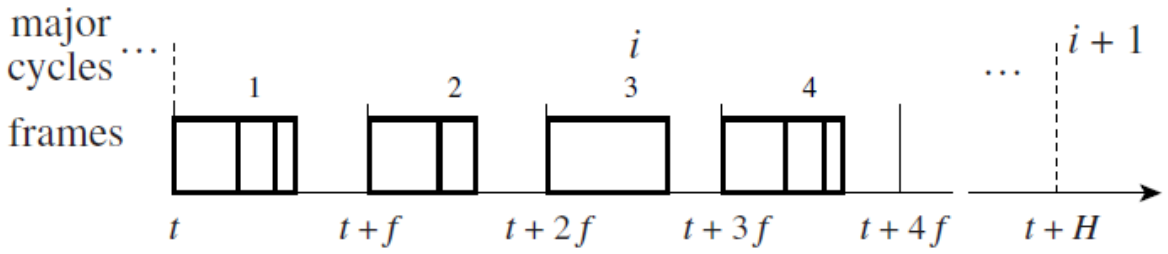


Fig: General Structure of a cyclic schedule

The total scheduling time are divided into number of time intervals called frames. Every frame has length f called frame size. Scheduling decision are made only at the beginning of frame and there is no preemption with in each frame. The phase of the each periodic task is a positive integer multiple of frame size. The first job of every task is released at beginning of frame and $\varphi = k \cdot f$. This approach provides two major benefits:

- Scheduler can easily check for overruns and missed deadlines at the end of each frame.
- Can use a periodic clock interrupt, rather than programmable timer?

Frame Size Constraints:

This constraint explains how to choose frame length in cyclic scheduling. To avoid preemption, want jobs to start and complete execution within a single frame:

$$f \geq \max(e1, e2, \ldots, en) \qquad \ldots\ldots\ldots\ldots\ldots \qquad \text{(Eq.1)}$$

To minimize the number of entries in the cyclic schedule, the hyper-period should be an integer multiple of the frame size (⇒ f divides evenly into the period of at least one task)

$$\exists\ i : \text{mod}(pi, f) = 0 \quad \text{.....(Eq.2)}$$

To allow scheduler to check that jobs complete by their deadline, should be at least one frame boundary between release time of a job and its deadline:

$$2*f - \gcd(pi, f) \leq Di \text{ for } i = 1, 2, ..., n \quad \text{..... (Eq.3)}$$

All 3 constraints should be satisfied.

Frame Size Constraints – Example:

Given tasks are T1 = (4, 1.0), T2 = (5, 1.8) T3 = (20, 1.0), T4 = (20, 2.0).

Hyper-period H = lcm (4, 5, 20, 20) = 20

Constraints: Eq.1 ⇒ $f \geq \max(1, 1.8, 1, 2) \geq 2$

Eq.2 ⇒ $f \in \{ 2, 4, 5, 10, 20 \}$

Eq.3 ⇒ $2f - \gcd(4, f) \leq 4$ (T1)

$2f - \gcd(5, f) \leq 5$ (T2)

$2f - \gcd(20, f) \leq 20$ (T3, T4)

These all constraints are satisfied by f = 2 only so required frame size = 2.

Job Slices:

Sometimes, a system cannot meet all three frame size constraints simultaneously. At this situation we can often solve by partitioning a job with large execution time into slices (sub-jobs) with shorter execution times/deadlines then these slices are called job slices.

Consider a system with three independent task

$T1 = (4, 1), T2 = (5, 2, 7), T3 = (20, 5)$

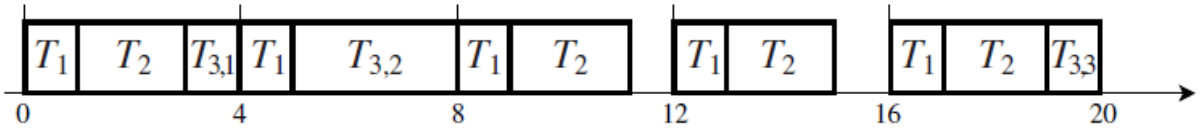
Condition 1 ➔ $f \geq 5$

Condition 2 ➔ 2, 4, 5, 10, 20

Condition 3 ➔ $2f - \gcd(Pi, f) \leq 4, \leq 7, \leq 20$

From condition 1 we must have $f \geq 5$ but to satisfy condition 3 we must have $f \leq 4$.

In above example, we can divide each job in (20, 5) into a chain of three slices with execution time 1, 3 and 1 respectively as (20, 1), (20, 3), (20, 1). The time schedule be generated as,



Sometimes need to partition jobs more slices than required by frame size constraints to yield a feasible schedule. To construct a cyclic schedule, we need to make 3 kinds of design constraints:

- ✓ Chose a frame size based on constraints
- ✓ Partition jobs into jobs
- ✓ Places slices in frames

Slack Stealing:

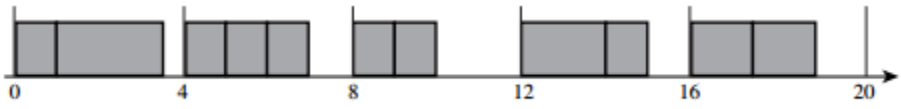
A natural way to improve the response times of aperiodic jobs is by executing the aperiodic jobs ahead of the periodic jobs whenever possible. This approach, called slack stealing. Every periodic jobs slice must be scheduled in a frame that ends no later than its deadline. When aperiodic job executes ahead of slice of periodic task then it consumes the slack in the frame. It reduce the response time of aperiodic jobs but requires accurate timer.

For example:

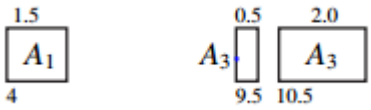
Consider a system with

- ✓ Three periodic tasks T1 (4, 1), T2 (5, 2), and T3 (8, 10, 1)
- ✓ Three aperiodic tasks A1 (4, 1.5), A2 (9.5, 0.5), and A3 (10.5, 2)

Now major cycle in the cyclic schedule of the periodic tasks is:



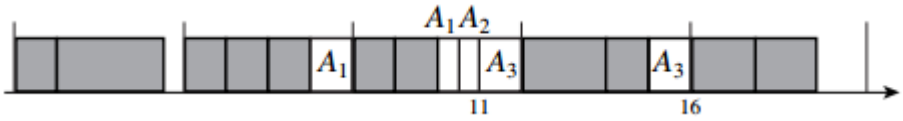
This shows that the slack time be available at 3 to 4, 10 to 12, 15 to 16 and 18 to 20 where aperiodic task can be executed. The aperiodic task for execution shown as below:



When the aperiodic jobs execute by use of the cyclic executive such that

- ✓ Job A1 can executed in idle period starting at 7 and preempted at 8. Similarly resumed at 10 and executed at 10.5.
- ✓ At time 10.5 A2 starts its execution and finished at 11. Similarly A3 starts its execution at time 11 and preempted at 12.
- ✓ Job A3 is resumed on slack starts at 15 and executed at 16.

Now the schedule be constructed as:



Here,

Response time of A1 = 6.5

Response time for A2 = 1.5

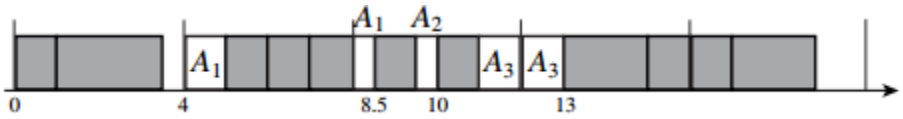
Response time for A3 = 5.5

Average response time = 4.5

If the cyclic executive does slack stealing such that,

- ✓ At time between 4 and 8 there is the slack of 1 such that A1 can be executed at time 4 and preempted at 5.
- ✓ At time between 8 to 12 there exists the slack of 2 such that A1 can resumed up to 8.5 and A2 can execute between 9.5 to 10. Similarly A3 can execute between 11 to12 and preempted.
- ✓ At time between 12 and 16 there exists the slack of 1 and task A3 can resumed on 12and executed at 13.

Now the schedule can be generated as:



Here,

Response time of A1 = 4.5

Response time for A2 = 0.5

Response time for A3 = 2.5

Average response time = 2.5

This shows that the average response time is improved if slack stealing approach is used.

Scheduling of Sporadic Jobs:

Generally sporadic job have hard deadlines such that their minimum and maximum release times are known in advance. It is impossible to guarantee a priori that all jobs can complete in time. Hence scheduling of sporadic jobs are done in two steps:

1. Acceptance test
2. EDF scheduling of accepted jobs

Acceptance Test:

A main problem is to determine whether all sporadic jobs can complete in time. 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. If there is sufficient amount of time in the frames before its deadlines to complete the newly released sporadic job without causing any job in the system to complete too late then scheduler accepts and schedule the job otherwise rejects the new sporadic jobs. That means

If total amount of slack time in frame >= its execution time and no adverse effect on sporadic job then

Accept the job.

Else

Reject the job.

A queue of sporadic job may be formed for testing at a same time on EDF basis.

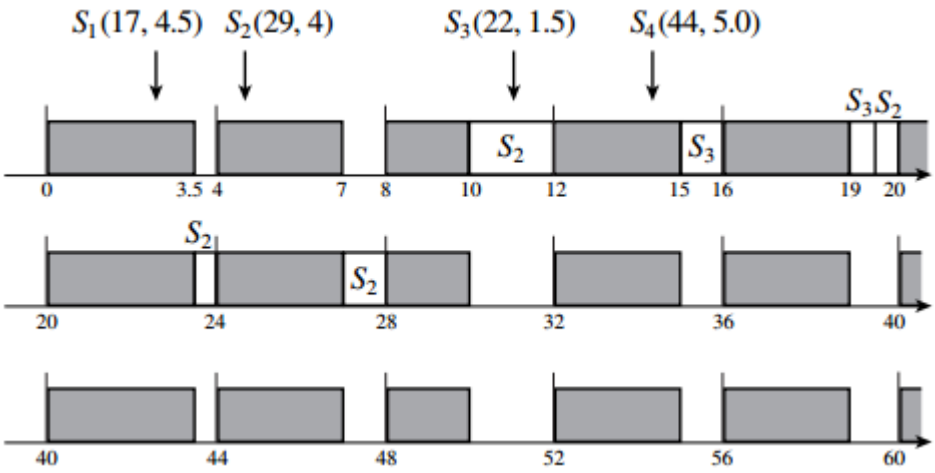
EDF Scheduling of the Accepted Jobs:

EDF is best suited method to schedule the accepted sporadic jobs. For this purpose, the schedule maintains a queue of accepted sporadic jobs in increasing order of their deadlines and inserts each newly accepted sporadic job into this queue in increasing order.

For example:

Frame size is 4, gray rectangles are periodic tasks S1 – S4 are sporadic tasks with parameters

(Di , Ci).



- ✓ S1 is released in time 3.
  - ➢ Must be scheduled in frames 2, 3 and 4.
  - ➢ Acceptance test – at the beginning of frame 2 and Slack time is 4 which is less that execution time such that job is rejected.
- ✓ S2 is released in time 5.
  - ➢ Must be scheduled in frames 3 through 7.
  - ➢ Acceptance test – at the beginning of frame 3 but Slack time is 5.5 such that job is accepted.
  - ➢ First part (2 units) executes in current frame.
- ✓ S3 is released in time 11.
  - ➢ Must be scheduled in frames 4 and 5. S3 runs ahead of S2.
  - ➢ Acceptance test – at the beginning of frame 4 and Slack time is 2 (enough for S3 and the part of S2) – job is accepted. First part (1 unit) S3 executes in current frame, followed by second part of S2.
- ✓ S4 is released in time 14

- Acceptance test – at beginning of frame 5 and Slack time is 4.5 (accounted for slack committed by S2 and S3) such that job is rejected.
- Remaining portion of S3 completes in current frame, followed by the part of S2.
- Remaining potions of S2 execute in the next two frames.

Practical Considerations:

Practical problems in clock driven scheduling falls into following three categories.

- **Handling overruns:**
  - ✓ Jobs are scheduled based on maximum execution time, but failures might cause overrun.
  - ✓ A robust system will handle this by either: 1) killing the job and starting an error recovery task; or 2) preempting the job and scheduling the remainder as an aperiodic job.
  - ✓ Depends on usefulness of late results, dependencies between jobs, etc.

- **Mode changes:**
  - ✓ A cyclic scheduler needs to know all parameters of real-time jobs a priori.
  - ✓ Switching between modes of operation implies reconfiguring the scheduler and bringing in the code/data for the new jobs.
  - ✓ This can take a long time: schedule the reconfiguration job as an aperiodic or sporadic task to ensure other deadlines met during mode change.
- **Multiple processors:**

  Can be handled, but off-line scheduling table generation more complex.

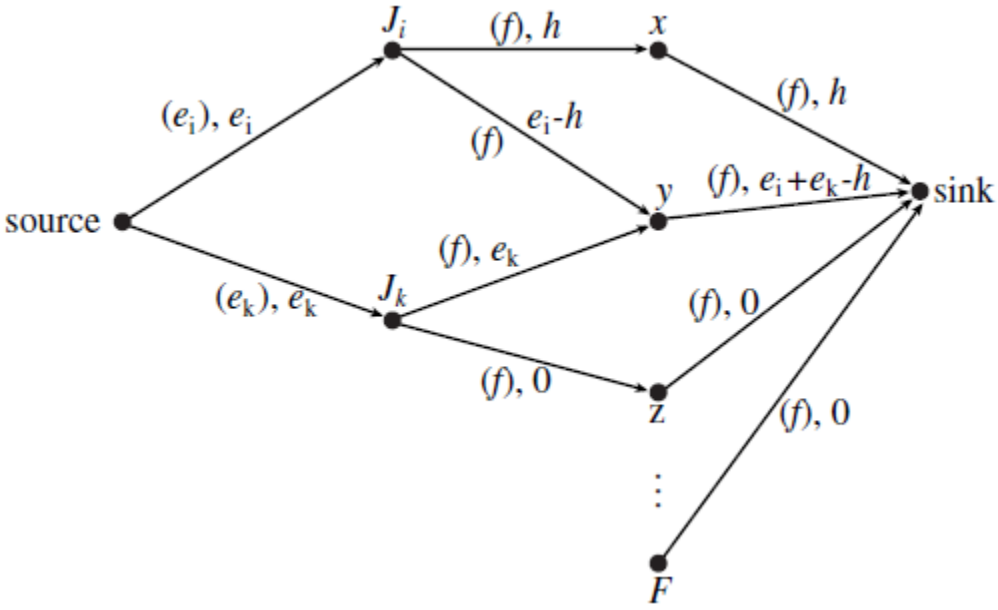
Algorithms for Constructing Static Schedules:

A system of independent preemptive task whose relative deadline are equal to or greater than their respective periods is schedulable if and only if the total utilization of the task is greater than 1. Similarly, some tasks may have relative deadlines shorter than their periods but there is no existence of feasible schedule due to frame size constraints. Therefore an algorithms is used to find the feasible schedule in cyclic scheduling is called iterative network flow (INF) algorithms.

The INF algorithms assumes that the task can be preempted at any time and are independent. In order to apply INF, find all possible frame size for given system. For example, consider three independent tasks T1(4, 1), T2(5, 2, 7) and T3 (20, 5). For this valid frame can be 4 or 2 but it cannot satisfy the constraints of frame size so there is impossible to construct the feasible schedule without using INF.The INF iteratively finds the feasible schedule for the possible frame size

starting from the largest frame. The feasible schedule found in the way is to decompose the task into subtasks.

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



The graph contains following vertices and edges.

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. There are two special vertices named source and sink.
3. 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 .
4. 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.
5. There is a directed edge from every frame vertex to the sink, and the capacity of this edge is f.

A flows of an edge is a non-negative number satisfying following conditions.

- ⇨ It is no greater than the capacity of edges.

⇨ The sum of flows of all edges into every vertex is equal to sum of all edges out of vertex.

Pros and Cons of Clock driven Scheduling:

Pros:

- Conceptual simplicity
    - Ability to consider complex dependencies, communication delays, and resource contention among jobs when constructing the static schedule, guaranteeing absence of deadlocks and unpredictable delays.
    - Entire schedule is captured in a static table.
    - Different operating modes can be represented by different tables.
    - No concurrency control or synchronization required.
    - If completion time jitter requirements exist, can be captured in the schedule.
- When workload is mostly periodic and the schedule is cyclic, timing constraints can be checked and enforced at each frame boundary.
- Choice of frame size can minimize context switching and communication overheads.
- Relatively easy to validate, test and certify.

Cons:

- Inflexible
    - Pre-compilation of knowledge into scheduling tables means that if anything changes materially, have to redo the table generation.
    - Best suited for systems which are rarely modified once built.
- Other disadvantages:
    - Release times of all jobs must be fixed.
    - All possible combinations of periodic tasks that can execute at the same time must be known a priori, so that the combined schedule can be pre-computed.

Chapter- 6

Priority-Driven Scheduling of Periodic Tasks:

Assumptions:

1. The priority driven scheduling of periodic tasks is perform on single processor.
2. A restricted periodic task is used. The restricted model means:
   - A fixed number independent periodic task exist.
   - Jobs comprising these tasks are ready for execution as soon as they are released.
   - Jobs can be preempted at any time.
   - Jobs never suspend themselves.
   - New task only admitted after an acceptance test.
   - The period of task is the minimum inter release time of jobs.
   - There are no aperiodic or sporadic task.
   - Scheduling decisions are made immediately upon the job release and complete.
   - Algorithms are event driven, not clock driven.
   - The content switched overload are negligibly small.

Fixed – Priority Vs. Dynamic – Priority Algorithms:

- A priority – driven scheduler is an on-line scheduler. It does not pre-compute a schedule of tasks, instead if assigns priority to jobs after they are released and places the jobs in a ready job queue in priority order.
- When preemption is allowed at any time, a scheduling decision is made whenever a job is released or completes. At each scheduling decision time, the scheduler update the ready job and then schedules and executes the job a head of head of the queue.
- Priority driven algorithms differ from each other in how priorities are assigned to jobs
  - Fixed priority
  - Dynamic priority
- A fixed priority algorithms assigns the same priorities to all the jobs in each tasks. The priority of each periodic task is fixed relative to other tasks.
- A dynamic priority algorithms assigns different priorities to the individual jobs in each task. Hence the priority of the task with respect to that of other tasks changes as jobs are released and completed.

Rate – Monotonic (RM) algorithms:

A well-known fixed priority algorithms is the rate monotonic algorithms. The algorithm assigns priorities to task based on their periods as shorter period have higher priority. The rate (of job release) of a task is inverse of periods that means higher rate gets higher priority.

For example: consider a system contains three tasks T1 (4, 1), T2 (5, 2) and T3 (20, 5) then

Rate for task T1 = ¼

Rate for task T2 = 1/5

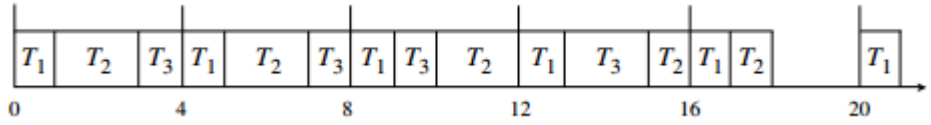
Rate for task T3 = 1/ 20

So the priority be assign as T1>T2>T3

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21, J31	J11
1	J21, J31	J21
3	J31	J31
4	J12, J31	J12
5	J22, J31	J22
7	J31	J31
8	J13, J31	J13
9	J31	J31
10	J23, J31	J23
12	J14, J31	J14
13	J31	J31
15	J24	J24
16	J15, J24	J15
17	J24	J24

The required schedule be constructed as:



Deadline Monotonic Algorithms:

This algorithms assigns priorities to tasks according to their relative deadlines as the shorter deadline gets higher priority. When the relative deadlines are arbitrary, DM algorithms performs better in the sense that it can sometimes produce the feasible schedule if RM algorithms fails that means RM algorithms always fails if DM algorithms fails.

For example: consider three tasks

T1 (50, 50, 25, 100)

T2 (0, 62.5, 10, 20)

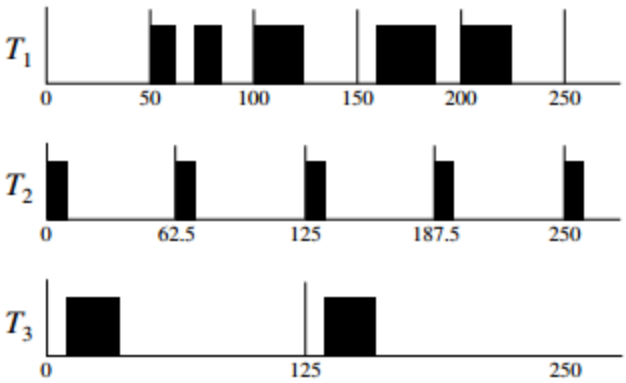
T3 (0, 125, 25, 50)

According to DM priority be assigned as T2> T3 >T1

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J21, J31	J21
10	J31	J31
35	----------	----------
50	J11	J11
62.5	J22, J11	J22
72.5	J11	J11
85	----------	----------
100	J12	J12
125	J23, J32	J23
135	J32	J32
150	J32, J13	J32
160	J13	J13
185	----------	----------
187.5	J24	J24
197.5	----------	----------
200	J14	J14
225	----------	----------
250	J15, J25, J33	J25

Corresponding Schedule be constructed as:



Dynamic Priority Algorithms:

Dynamic priority algorithms are:

1. Earliest Deadline First (EDF)
2. Least Slack Time (LST)
3. First In First Out (FIFO)
4. Last In First Out (LIFO)

Earliest Deadline First (EDF):

The EDF algorithms assigns priorities to individual's job in the tasks according to their absolute deadlines. EDF algorithms is a task level dynamic priority algorithm.

For example: consider a system with task

T1 (2, 1)

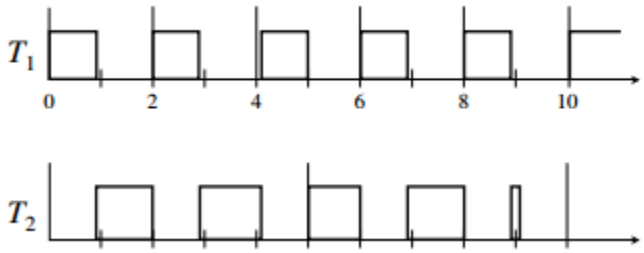
T2 (5, 2.5)

Hyper period (H) = 10

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21	J11
1	J21	J21
2	J12, J21	J12
3	J21	J21
4	J13, J21	J13
4.5	J13	J13
5	J13, J22	J13
6	J14, J22	J14
7	J22	J22
8	J15, J22	J22
9	J15	J15
10	J16, J23	----------

Corresponding Schedule be constructed as:



Least Slack Time (LST):

A well-known dynamic priority algorithms is least slack time. It assigns the priorities to individual jobs in the task according to slack time. If d be the deadline, e be the execution time then at time t slack time be equals to d-e-t.

The scheduler checks the slack of all ready jobs at each time a new job is released and orders the new and existing jobs in accordance with slack time as smaller slack have the higher priority.

For example: consider a system with two jobs:

T1 (2, 1)

T2 (5, 2.5)

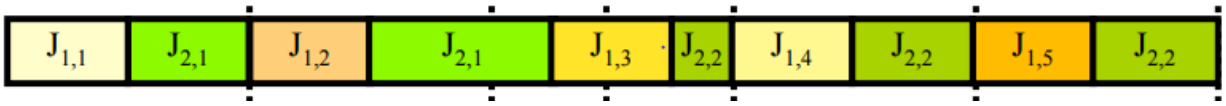
- Two jobs J11 and J21 are released at time 0 then
  Slack for J11 = 2-1-0=1
  Slack for J11 = 5-2.5-0=2.5
  Here T1> T2, the job J11 is completely executes in between time 0 and 1 then J21 inters into the execution.
- At time 2, new job J12 is released then
  Slack for J12 = 4-1-2=1
  Slack for J21 = 5-1.5-2= 1.5
  Here T1>T2, the job J12 is preempted and J12 enters into the execution and released at 3. The job J21 resumed for execution after time 3.
- At time 4, new job J13 is released then
  Slack for J13 = 6-1-4=1
  Slack for J21 = 5-0.5-4=0.5
  Here T2>T1, the job J21 executes up to 4.5 and J13 enters into execution.
- At time 5, new job J22 is released then
  Slack for J22= 10-2.5-5=2.5
  Slack for J13 = 6-0.5-5=0.5
  Here T1> T2, the job executes up to 5.5 and job J22 enters into execution.
- At time 6, the new job J14 released then
  Slack for J14 = 8-1-6=1
  Slack for J22 = 10-2-6=2
  Here T1>T2, so J14 executes up to 7 and J22 resumes its execution.
- At time 8, new job J15 is released then
  Slack for J15 = 10-1-8=1
  Slack for J22 = 10-1-8 = 1
  Here have the equal priority so J15 completes its execution up to 9 and J22 completes its execution up to 10.

The corresponding schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21	J11
1	J21	J21
2	J12, J21	J12
3	J21	J21
4	J13, J21	J13
4.5	J13	J13
5	J13, J22	J13

5.5	J22	J22
6	J14,J22	J14
7	J22	J22
8	J15,J22	J15
9	J22	J22

The required schedule is:



In this approach, the priority order is only changed only when the new job is released or any job finishes its execution such that there is no follow of LST algorithms at any time. This type of priorities is non strict and corresponding algorithms is called non strict LST algorithms otherwise it is called strict algorithms. In case of strict approach, scheduler follow the LST rule during the change of slack at any time and updates the job list of queue according to change of priority.

For example: in above, at time 2.6

Slack time for J12 = 4-0.4-2.6 = 1

Slack time for J21 = 5- 1.5- 2.6 = 0.9

Here J21 gets higher priority than J12. Due to continuous observation and computation of slacks at each period of time, strict LST algorithms suffers from run-time overhead.

First in, first out (FIFO):

Job queue is first-in-first-out by release time

Last in, first out (LIFO):

Job queue is last-in-first-out by release time

Maximum Schedule Utilization-Theorem:

A system is schedulable (and feasible) if and if it is schedulable by some algorithms. A scheduling algorithms can feasibly schedule any set of periodic tasks on a processor if the total utilization of tasks is equal to or less than schedulable utilization of algorithms.

The higher the schedulable utilization of an algorithms, better the algorithms. Since, no algorithms can feasibly schedule a set of tasks with a total utilization greater than 1, an algorithms can feasibly whose schedulable utilization is equal to 1 is an optimal algorithms.

Schedulable Utilization of the EDF Algorithm:

- A system T of independent, preemptable tasks with relative deadlines equal to their respective periods can be feasibly scheduled on one processor if its total utilization is equal to or less than 1.

$u1 = \frac{e1}{p1}$, $u2 = \frac{e2}{p2}$ then $u = u1 + u2 < 1$

- A system T of independent preemptable tasks can be feasibly scheduled on one processor if and only if its density is equal to or less than 1.

Density of Task = $\frac{e_k}{\min(D_k, P_k)}$

Example: T_1= (2, 0.9); T_2= (5, 2.3, 3) then

$\Delta = 0.9/2+2.3/3 = 7.3/6 > 1$, not feasible

Optimality of RM and DM Algorithms:

- The fixed priority algorithms can be optimal in restricted systems. Since RM and DM are fixed priority algorithms, they are optimal in simply periodic system. A system of periodic task is a simply periodic period if the period of each task is an integer multiple of the period of order task that is $P_k = nP_i$ where $P_i < P_k$ and n is positive integer.
- A system of simple periodic independent preemptable tasks with $Di \leq Pi$ is schedulable on one processor using RM algorithms if and only if $U \leq 1$. The RM algorithms is optimal among all the fixed priority algorithms whenever the relative deadlines of their task are proportional to their periods.
- When RM algorithms is optimal then DM is also optimal.

Scheduling Test for Fixed Priority Tasks with Short Response Times-Critical Instants:

A scheduling test is required to schedule the fixed priority task with shortest response time which is smaller than or equal to respective periods and total utilization is less than 1. Generally fixed priority algorithms are not suffers from the scheduling anomalies and requires following task to generate a schedule without anomalies.

- Find the critical instant when the system is most loaded and its worst response time.
- Use time demand analysis to determine if the system is schedulable at that instant.

Critical Instant:

A critical instant for a job is the worst-case release time for that job, taking into account all jobs that have higher priority i.e. a job released at the same instant as all jobs with higher priority are released, and must wait for all those jobs to complete before it executes.

A critical instant of a task Ti is a time instant such that:

- the job of Ti released at this instant has the maximum response time of all jobs in Ti, if the response time of every job of Ti is at most Di, the relative deadline of Ti and
- the response time of the job released at this instant is greater than Di of the response time of some jobs in Ti exceeds Di

The schedulability test involves for checking each task as it is scheduable or not in the critical instant. The critical instant theorem states that, a fixed-priority system where every job completes before the next job of the same task is released, a critical instant of any task Ti occurs when one of its job Ji,c is released at the same time with a job of every higher priority task.

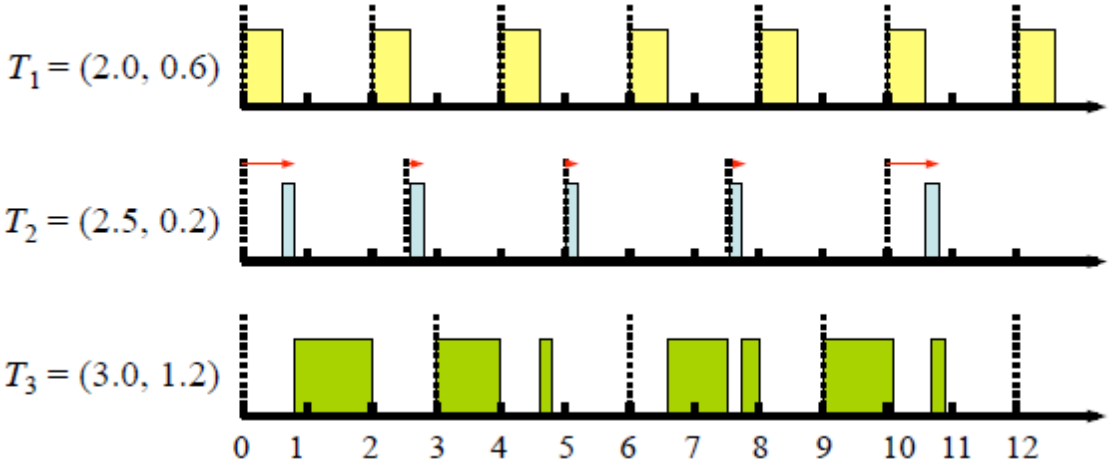
For example: Consider three tasks T1 (2, 0.6), T2 (2.5, 0.2), T3 (3, 1.2)

Here T1>T2>T3

The RM schedule table be

Time	Ready to Run	Scheduled
0	J11, J21, J31	J11
0.6	J21,J31	J21
0.8	J31	J31
2	J12,	J12
2.5	J12, J22	J12
2.6	J22	J22
2.8	----------	----------
3	J32	J32
4	J13,J32	J13
4.6	J32	J32
4.8	----------	----------
5	J23	J23
5.2	----------	----------
6	J14, J33	J14
6.6	J33	J33
7.5	J24, J33	J24
7.7	J33	J33
8	J15	J15
8.6	----------	----------
9	J34	J34
10	J16, J25,J34	J16
10.6	J25, J34	J25
10.8	J34	J34
11	----------	----------

The corresponding schedule be constructed as below



Response time for all jobs of Task T1 = 0.6

Response time for Job J21 of task T2 = r21= 0.8

Response time for Job J22 of task T2 = r22 = 2.8 - 2.5 = 0.3

Response time for Job J23 of task T2 = r23 = 5.2 – 5 = 0.2

Response time for Job J24 of task T2 = r24 = 7.7 -7.5 = 0.2

Response time for Job J25 of task T2 = r25= 0.8

The response time all jobs of task T2 never exceed the response time of first job. Hence the critical instant is equal to t=0 and t=10.

Response time for Job J31 of task T3 = r31 = 2

Response time for Job J32 of task T3 = r32 = 1.8

Response time for Job J33 of task T3 = r33 = 2

Response time for Job J34 of task T3 = r34 = 2

Here the response time for jobs in T3 never exceed the response time of first job in T3. Hence critical instant of T3 are t= 0, 6 and 9.

Time- Demand Analysis:

The process used to determine whether a task can meet its deadline is called time demand analysis. Methods to time-demand analysis are:

1. Compute the total demand for processor time by job released at a critical instant of task and by higher priority task as a function of time from the critical instant.

2. Check weather tis demand can be met before the deadline of job as follows:

   -Consider the scanning is done one task at a time from higher priority and working down to lowest priority.

   -Focused on a job in Ti where the released at time t0 of that job is a critical instant of Ti.

   -At time t_0+t for$t \geq 0$, compute the processor time demand w_i (t) of this job and all higher priorities job released in [t0, t] using

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

$w_i(t)$ = the time-demand function

Execution time of job J_i

Execution time of higher priority jobs started during this interval

3. Compare the time demand w_i (t) with available time t as

   -If $w_i(t) \leq t \; for \; t \leq D_i$, the job meets its deadline $t_0 + D_i$
   -If $w_i(t) > t \; for \; 0 \leq t \leq D_i$, the task may not complete by its deadline and system cannot scheduled by using fixed priority algorithms.

Schedulability test for fixed priority tasks with arbitrary response times-busy intervals:

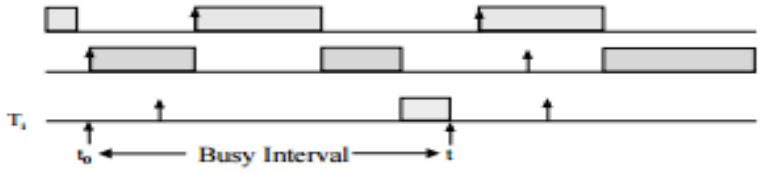
It is the schedulability test for tasks with relative deadlines greater than respective periods. Since the response time of a task may be larger than its period, it may have more than one job ready for execution at any time. Ready jobs in the same task are usually scheduled on the FIFO basis. This policy is used here named as busy interval.

Busy Interval:

A *level-π_i busy interval (t_0, t]* begins at an instant t_0 when

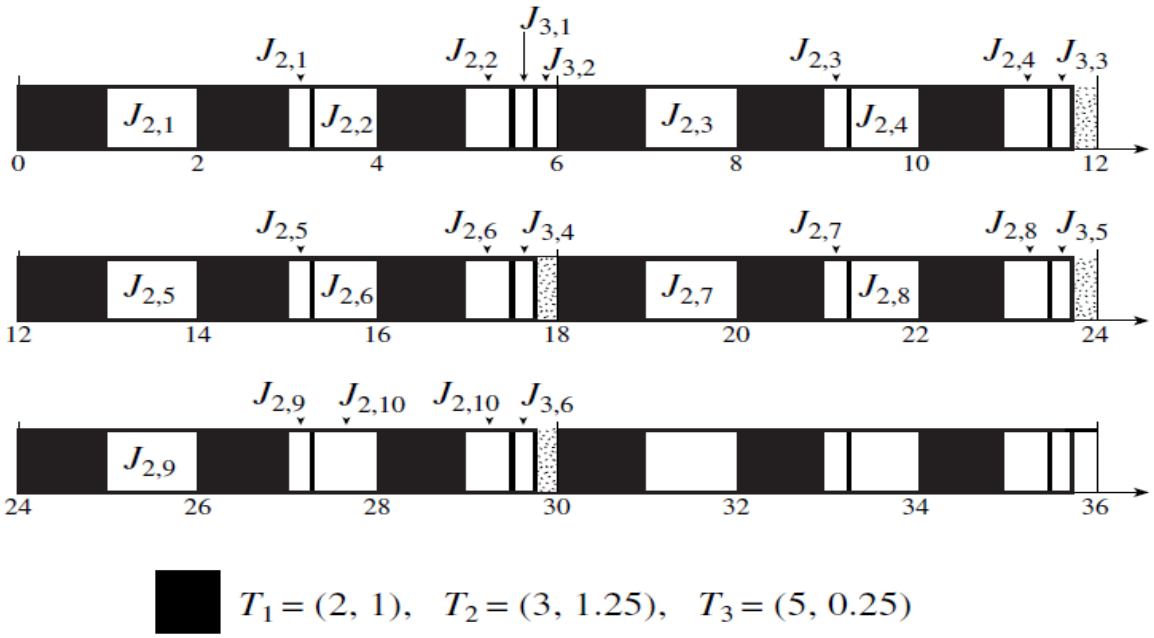
- all jobs in $\mathbf{T}_i$ released before the instant have completed and
- a job in $\mathbf{T}_i$ is released.

The interval ends at the first instant *t* after *t*0 when all the jobs in **T***i* released at *t*0 are complete i.e. the interval *(t*0*, t]*, the processor is busy all the time executing jobs with priorities π_i or higher, all the jobs executed in the busy interval are released in the interval, and at the end of the interval there is no backlog of jobs to be executed afterwards.



A level-π_i busy interval is *in phase* if the first jobs of all tasks that have priorities equal to or higher than priority π_i and are executed in this interval have the same release time.

For example, Consider a system with three task $T1 = (2, 1)$, $T2 = (3, 1.25)$, and $T3 = (5, 0.25)$ scheduled as



In above example, the priorities of the tasks are $\pi1 = 1$, $\pi2 = 2$, and $\pi3 = 3$, with 1 being the highest priority and 3 being the lowest priority. The filled rectangles depict where jobs in $T1$ are scheduled. The first busy intervals of all levels are in phase.

- Every level-1 busy interval always ends 1 unit time after it begins.
- For this system, all the level-2 busy intervals are in phase since they begin at 0 and ends at 6 which is LCM of T1 and T2 and length of interval equals to 5.5.
- Level -3 busy interval begins at 5.5 and ends at 6 and not in phase since next job released at time 10.

Sufficient Schedulability Conditions for RM & DM algorithms:

Consider case when the relative deadline of every task is equal to its period. For such systems, the RM and DM algorithms are identical. Sufficient condition of RM algorithms states that, system of n independent, preempt able periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U is less than or equal to $U_{RM}(n) = n(2^{1/n} - 1)$.

Q.N A system contains five tasks: T1 (1, 0.25), T2 (1.25, 0.1), T3 (1.5, 0.3), T4 (1.75, 0.07), T5 (2, 1). Find that the system is schedule by using RM algorithms.

Practical Factor for Priority Driven Scheduling of Periodic Task:

We have assumed that:

- Jobs are preemptable at any time.
- Jobs never suspend themselves.
- Each job has distinct priority.
- The scheduler is event driven and acts immediately.

These assumptions are often not valid so following practical factors must be considered for priority driven scheduling.

1. Blocking and Priority Inversion
2. Self-Suspension
3. Context Switches
4. Thick Scheduling

Blocking and Priority Inversion:

⇨ A ready job is *blocked* when it is prevented from executing by a lower-priority job; a *priority inversion* is when a lower-priority job executes while a higher-priority job is blocked.

⇨ These occur because some jobs cannot be pre-empted:

- Many reasons why a job may have non-preemptable sections

⇨ Critical section over a resource Some system calls are non-preemptable

⇨ Disk scheduling

- If a job becomes non-preemptable, priority inversions may occur, these may cause a higher priority task to miss its deadline.
- When attempting to determine if a task meets all of its deadlines, must consider not only all the tasks that have higher priorities, but also non-preemptable regions of lower-priority tasks.

⇨ Add the blocking time in when calculating if a task is schedulable.

Self-suspension:

⇨ A job may invoke an external operation (e.g. request an I/O operation), during which time it is suspended.

- ⇨ This means the task is no longer strictly periodic… again need to take into account self-suspension time when calculating a schedule.

Context Switches:

- ⇨ Assume maximum number of context switches *Ki* for a job in *Ti* is known; each takes t_{CS} time units.
- ⇨ Compensate by setting execution time of each job, *e*actual = $e + 2t_{CS}$
  (more if jobs self-suspend, since additional context switches)

Tick Scheduling:

- ⇨ All of our previous discussion of priority-driven scheduling was driven by job release and job completion events.
- ⇨ Alternatively, can perform priority-driven scheduling at periodic events (timer interrupts) generated by a hardware clock i.e. tick (or time-based) scheduling.
- ⇨ Additional factors to account for in schedulability analysis.
- ⇨ The fact that a job is ready to execute will not be noticed and acted upon until the next clock interrupt; this will delay the completion of the job.
- ⇨ A ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue, the *pending job* queue.
- ⇨ When the scheduler executes, it moves jobs in the pending queue to the ready queue according to their priorities; once in ready queue, the jobs execute in priority order.

Chapter- 6

Priority-Driven Scheduling of Periodic Tasks:

Assumptions:

1. The priority driven scheduling of periodic tasks is perform on single processor.
2. A restricted periodic task is used. The restricted model means:
   - A fixed number independent periodic task exist.
   - Jobs comprising these tasks are ready for execution as soon as they are released.
   - Jobs can be preempted at any time.
   - Jobs never suspend themselves.
   - New task only admitted after an acceptance test.
   - The period of task is the minimum inter release time of jobs.
   - There are no aperiodic or sporadic task.
   - Scheduling decisions are made immediately upon the job release and complete.
   - Algorithms are event driven, not clock driven.
   - The content switched overload are negligibly small.

Fixed – Priority Vs. Dynamic – Priority Algorithms:

- A priority – driven scheduler is an on-line scheduler. It does not pre-compute a schedule of tasks, instead if assigns priority to jobs after they are released and places the jobs in a ready job queue in priority order.
- When preemption is allowed at any time, a scheduling decision is made whenever a job is released or completes. At each scheduling decision time, the scheduler update the ready job and then schedules and executes the job a head of head of the queue.
- Priority driven algorithms differ from each other in how priorities are assigned to jobs
  - Fixed priority
  - Dynamic priority
- A fixed priority algorithms assigns the same priorities to all the jobs in each tasks. The priority of each periodic task is fixed relative to other tasks.
- A dynamic priority algorithms assigns different priorities to the individual jobs in each task. Hence the priority of the task with respect to that of other tasks changes as jobs are released and completed.

Rate – Monotonic (RM) algorithms:

A well-known fixed priority algorithms is the rate monotonic algorithms. The algorithm assigns priorities to task based on their periods as shorter period have higher priority. The rate (of job release) of a task is inverse of periods that means higher rate gets higher priority.

For example: consider a system contains three tasks T1 (4, 1), T2 (5, 2) and T3 (20, 5) then

Rate for task T1 = ¼

Rate for task T2 = 1/5

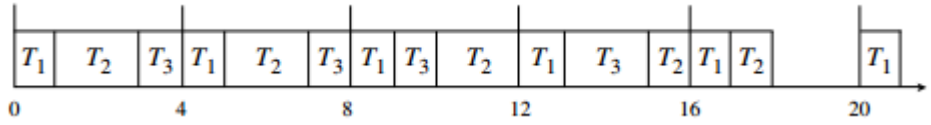
Rate for task T3 = 1/ 20

So the priority be assign as T1>T2>T3

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21, J31	J11
1	J21, J31	J21
3	J31	J31
4	J12, J31	J12
5	J22, J31	J22
7	J31	J31
8	J13, J31	J13
9	J31	J31
10	J23, J31	J23
12	J14, J31	J14
13	J31	J31
15	J24	J24
16	J15, J24	J15
17	J24	J24

The required schedule be constructed as:



Deadline Monotonic Algorithms:

This algorithms assigns priorities to tasks according to their relative deadlines as the shorter deadline gets higher priority. When the relative deadlines are arbitrary, DM algorithms performs better in the sense that it can sometimes produce the feasible schedule if RM algorithms fails that means RM algorithms always fails if DM algorithms fails.

For example: consider three tasks

T1 (50, 50, 25, 100)

T2 (0, 62.5, 10, 20)

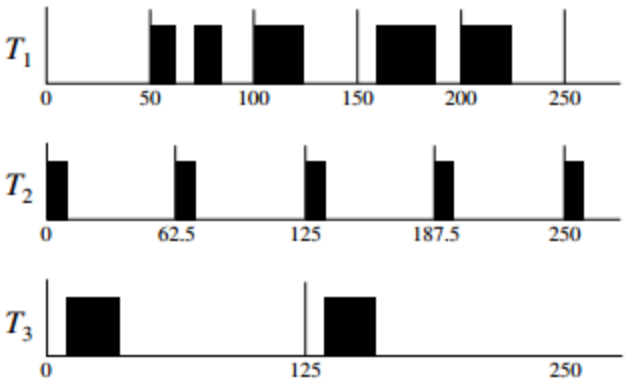
T3 (0, 125, 25, 50)

According to DM priority be assigned as T2> T3 >T1

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J21, J31	J21
10	J31	J31
35	----------	----------
50	J11	J11
62.5	J22, J11	J22
72.5	J11	J11
85	----------	----------
100	J12	J12
125	J23, J32	J23
135	J32	J32
150	J32, J13	J32
160	J13	J13
185	----------	----------
187.5	J24	J24
197.5	----------	----------
200	J14	J14
225	----------	----------
250	J15, J25, J33	J25

Corresponding Schedule be constructed as:



Dynamic Priority Algorithms:

Dynamic priority algorithms are:

5. Earliest Deadline First (EDF)
6. Least Slack Time (LST)
7. First In First Out (FIFO)
8. Last In First Out (LIFO)

Earliest Deadline First (EDF):

The EDF algorithms assigns priorities to individual's job in the tasks according to their absolute deadlines. EDF algorithms is a task level dynamic priority algorithm.

For example: consider a system with task

T1 (2, 1)

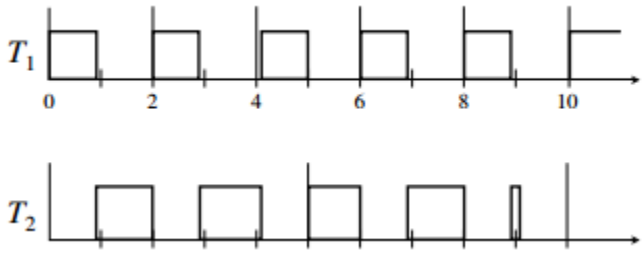
T2 (5, 2.5)

Hyper period (H) = 10

The schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21	J11
1	J21	J21
2	J12, J21	J12
3	J21	J21
4	J13, J21	J13
4.5	J13	J13
5	J13, J22	J13
6	J14, J22	J14
7	J22	J22
8	J15, J22	J22
9	J15	J15
10	J16, J23	----------

Corresponding Schedule be constructed as:



Least Slack Time (LST):

A well-known dynamic priority algorithms is least slack time. It assigns the priorities to individual jobs in the task according to slack time. If d be the deadline, e be the execution time then at time t slack time be equals to d-e-t.

The scheduler checks the slack of all ready jobs at each time a new job is released and orders the new and existing jobs in accordance with slack time as smaller slack have the higher priority.

For example: consider a system with two jobs:

T1 (2, 1)

T2 (5, 2.5)

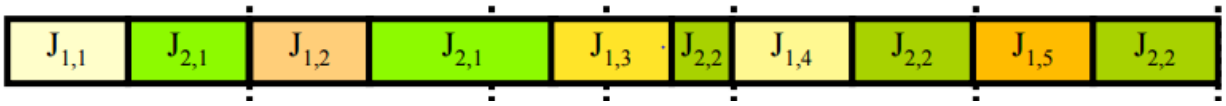
- Two jobs J11 and J21 are released at time 0 then
  Slack for J11 = 2-1-0=1
  Slack for J11 = 5-2.5-0=2.5
  Here T1> T2, the job J11 is completely executes in between time 0 and 1 then J21 inters into the execution.
- At time 2, new job J12 is released then
  Slack for J12 = 4-1-2=1
  Slack for J21 = 5-1.5-2= 1.5
  Here T1>T2, the job J12 is preempted and J12 enters into the execution and released at 3. The job J21 resumed for execution after time 3.
- At time 4, new job J13 is released then
  Slack for J13 = 6-1-4=1
  Slack for J21 = 5-0.5-4=0.5
  Here T2>T1, the job J21 executes up to 4.5 and J13 enters into execution.
- At time 5, new job J22 is released then
  Slack for J22= 10-2.5-5=2.5
  Slack for J13 = 6-0.5-5=0.5
  Here T1> T2, the job executes up to 5.5 and job J22 enters into execution.
- At time 6, the new job J14 released then
  Slack for J14 = 8-1-6=1
  Slack for J22 = 10-2-6=2
  Here T1>T2, so J14 executes up to 7 and J22 resumes its execution.
- At time 8, new job J15 is released then
  Slack for J15 = 10-1-8=1
  Slack for J22 = 10-1-8 = 1
  Here have the equal priority so J15 completes its execution up to 9 and J22 completes its execution up to 10.

The corresponding schedule table be constructed as:

Time	Ready to Run	Scheduled
0	J11, J21	J11
1	J21	J21
2	J12, J21	J12
3	J21	J21
4	J13, J21	J13
4.5	J13	J13
5	J13, J22	J13

5.5	J22	J22
6	J14,J22	J14
7	J22	J22
8	J15,J22	J15
9	J22	J22

The required schedule is:



In this approach, the priority order is only changed only when the new job is released or any job finishes its execution such that there is no follow of LST algorithms at any time. This type of priorities is non strict and corresponding algorithms is called non strict LST algorithms otherwise it is called strict algorithms. In case of strict approach, scheduler follow the LST rule during the change of slack at any time and updates the job list of queue according to change of priority.

For example: in above, at time 2.6

Slack time for J12 = 4-0.4-2.6 = 1

Slack time for J21 = 5- 1.5- 2.6 = 0.9

Here J21 gets higher priority than J12. Due to continuous observation and computation of slacks at each period of time, strict LST algorithms suffers from run-time overhead.

First in, first out (FIFO):

Job queue is first-in-first-out by release time

Last in, first out (LIFO):

Job queue is last-in-first-out by release time

Maximum Schedule Utilization-Theorem:

A system is schedulable (and feasible) if and if it is schedulable by some algorithms. A scheduling algorithms can feasibly schedule any set of periodic tasks on a processor if the total utilization of tasks is equal to or less than schedulable utilization of algorithms.

The higher the schedulable utilization of an algorithms, better the algorithms. Since, no algorithms can feasibly schedule a set of tasks with a total utilization greater than 1, an algorithms can feasibly whose schedulable utilization is equal to 1 is an optimal algorithms.

Schedulable Utilization of the EDF Algorithm:

- A system T of independent, preemptable tasks with relative deadlines equal to their respective periods can be feasibly scheduled on one processor if its total utilization is equal to or less than 1.

$u1 = \frac{e1}{p1}$, $u2 = \frac{e2}{p2}$ then $u = u1 + u2 < 1$

- A system T of independent preemptable tasks can be feasibly scheduled on one processor if and only if its density is equal to or less than 1.

Density of Task = $\frac{e_k}{\min(D_k, P_k)}$

Example: T_1= (2, 0.9); T_2= (5, 2.3, 3) then

$\Delta = 0.9/2+2.3/3 = 7.3/6 > 1$, not feasible

Optimality of RM and DM Algorithms:

- The fixed priority algorithms can be optimal in restricted systems. Since RM and DM are fixed priority algorithms, they are optimal in simply periodic system. A system of periodic task is a simply periodic period if the period of each task is an integer multiple of the period of order task that is $P_k = nP_i$ where $P_i < P_k$ and n is positive integer.
- A system of simple periodic independent preemptable tasks with $Di \leq Pi$ is schedulable on one processor using RM algorithms if and only if$U \leq 1$. The RM algorithms is optimal among all the fixed priority algorithms whenever the relative deadlines of their task are proportional to their periods.
- When RM algorithms is optimal then DM is also optimal.

Scheduling Test for Fixed Priority Tasks with Short Response Times-Critical Instants:

A scheduling test is required to schedule the fixed priority task with shortest response time which is smaller than or equal to respective periods and total utilization is less than 1. Generally fixed priority algorithms are not suffers from the scheduling anomalies and requires following task to generate a schedule without anomalies.

- Find the critical instant when the system is most loaded and its worst response time.
- Use time demand analysis to determine if the system is schedulable at that instant.

Critical Instant:

A critical instant for a job is the worst-case release time for that job, taking into account all jobs that have higher priority i.e. a job released at the same instant as all jobs with higher priority are released, and must wait for all those jobs to complete before it executes.

A critical instant of a task Ti is a time instant such that:

- the job of Ti released at this instant has the maximum response time of all jobs in Ti, if the response time of every job of Ti is at most Di, the relative deadline of Ti and
- the response time of the job released at this instant is greater than Di of the response time of some jobs in Ti exceeds Di

The schedulability test involves for checking each task as it is scheduable or not in the critical instant. The critical instant theorem states that, a fixed-priority system where every job completes before the next job of the same task is released, a critical instant of any task Ti occurs when one of its job Ji,c is released at the same time with a job of every higher priority task.

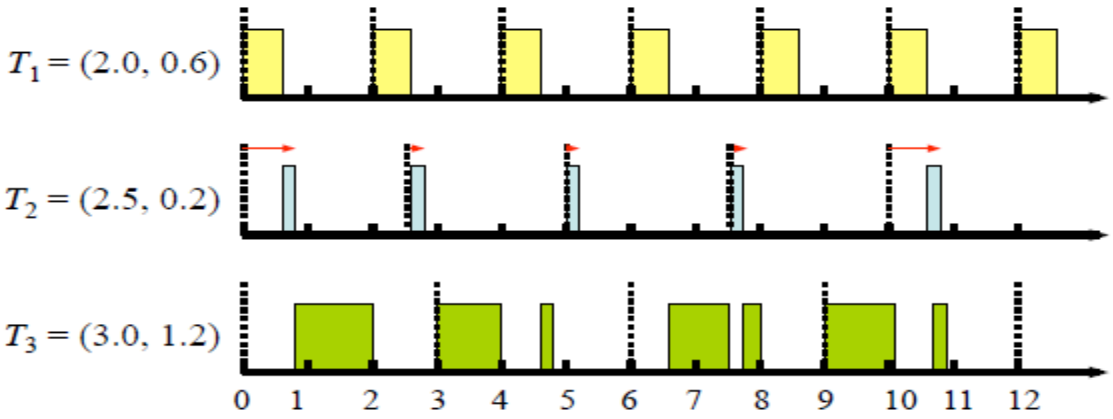
For example: Consider three tasks T1 (2, 0.6), T2 (2.5, 0.2), T3 (3, 1.2)

Here T1>T2>T3

The RM schedule table be

Time	Ready to Run	Scheduled
0	J11, J21, J31	J11
0.6	J21,J31	J21
0.8	J31	J31
2	J12,	J12
2.5	J12, J22	J12
2.6	J22	J22
2.8	----------	----------
3	J32	J32
4	J13,J32	J13
4.6	J32	J32
4.8	----------	----------
5	J23	J23
5.2	----------	----------
6	J14, J33	J14
6.6	J33	J33
7.5	J24, J33	J24
7.7	J33	J33
8	J15	J15
8.6	----------	----------
9	J34	J34
10	J16, J25,J34	J16
10.6	J25, J34	J25
10.8	J34	J34
11	----------	----------

The corresponding schedule be constructed as below



Response time for all jobs of Task T1 = 0.6

Response time for Job J21 of task T2 = r21= 0.8

Response time for Job J22 of task T2 = r22 = 2.8 - 2.5 = 0.3

Response time for Job J23 of task T2 = r23 = 5.2 – 5 = 0.2

Response time for Job J24 of task T2 = r24 = 7.7 -7.5 = 0.2

Response time for Job J25 of task T2 = r25= 0.8

The response time all jobs of task T2 never exceed the response time of first job. Hence the critical instant is equal to t=0 and t=10.

Response time for Job J31 of task T3 = r31 = 2

Response time for Job J32 of task T3 = r32 = 1.8

Response time for Job J33 of task T3 = r33 = 2

Response time for Job J34 of task T3 = r34 = 2

Here the response time for jobs in T3 never exceed the response time of first job in T3. Hence critical instant of T3 are t= 0, 6 and 9.

Time- Demand Analysis:

The process used to determine whether a task can meet its deadline is called time demand analysis. Methods to time-demand analysis are:

1. Compute the total demand for processor time by job released at a critical instant of task and by higher priority task as a function of time from the critical instant.
2. Check weather tis demand can be met before the deadline of job as follows:

-Consider the scanning is done one task at a time from higher priority and working down to lowest priority.

-Focused on a job in Ti where the released at time t0 of that job is a critical instant of Ti.

-At time t_0+t for$t \geq 0$, compute the processor time demand w_i (t) of this job and all higher priorities job released in [t0, t] using

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

$w_i(t)$ = the time-demand function

Execution time of job J_i

Execution time of higher priority jobs started during this interval

3. Compare the time demand w_i (t) with available time t as

-If $w_i(t) \leq t \; for \; t \leq D_i$, the job meets its deadline $t_0 + D_i$
-If $w_i(t) > t \; for \; 0 \leq t \leq D_i$, the task may not complete by its deadline and system cannot scheduled by using fixed priority algorithms.

Schedulability test for fixed priority tasks with arbitrary response times-busy intervals:

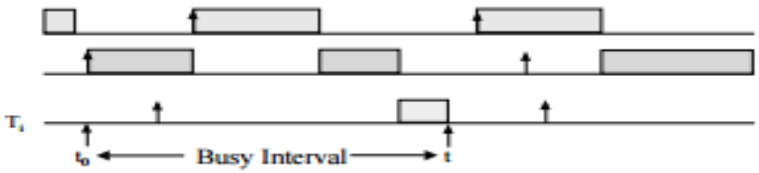
It is the schedulability test for tasks with relative deadlines greater than respective periods. Since the response time of a task may be larger than its period, it may have more than one job ready for execution at any time. Ready jobs in the same task are usually scheduled on the FIFO basis. This policy is used here named as busy interval.

Busy Interval:

A *level-π_i busy interval (t_0, t]* begins at an instant t_0 when

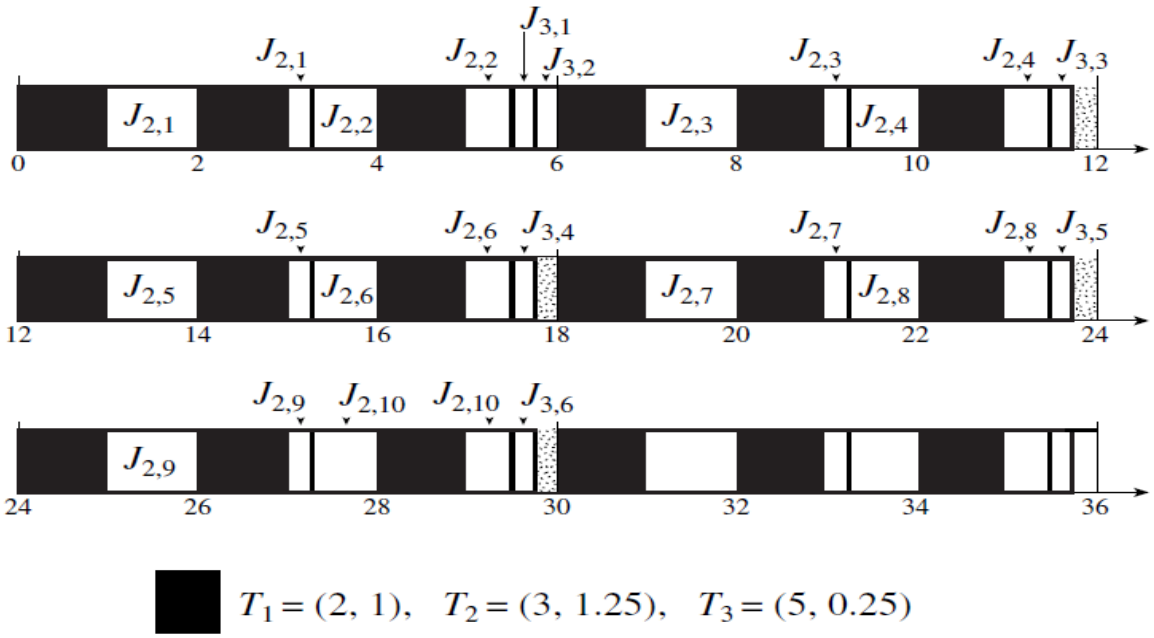
- all jobs in $\mathbf{T}_i$ released before the instant have completed and
- a job in $\mathbf{T}_i$ is released.

The interval ends at the first instant *t* after *t*0 when all the jobs in **T***i* released at *t*0 are complete i.e. the interval *(t*0, *t]*, the processor is busy all the time executing jobs with priorities π_i or higher, all the jobs executed in the busy interval are released in the interval, and at the end of the interval there is no backlog of jobs to be executed afterwards.



A level-π_i busy interval is *in phase* if the first jobs of all tasks that have priorities equal to or higher than priority π_i and are executed in this interval have the same release time.

For example, Consider a system with three task $T1 = (2, 1)$, $T2 = (3, 1.25)$, and $T3 = (5, 0.25)$ scheduled as



In above example, the priorities of the tasks are $\pi1 = 1$, $\pi2 = 2$, and $\pi3 = 3$, with 1 being the highest priority and 3 being the lowest priority. The filled rectangles depict where jobs in $T1$ are scheduled. The first busy intervals of all levels are in phase.

- Every level-1 busy interval always ends 1 unit time after it begins.
- For this system, all the level-2 busy intervals are in phase since they begin at 0 and ends at 6 which is LCM of T1 and T2 and length of interval equals to 5.5.
- Level -3 busy interval begins at 5.5 and ends at 6 and not in phase since next job released at time 10.

Sufficient Schedulability Conditions for RM & DM algorithms:

Consider case when the relative deadline of every task is equal to its period. For such systems, the RM and DM algorithms are identical. Sufficient condition of RM algorithms states that, system of n independent, preempt able periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U is less than or equal to $U_{RM}(n) = n(2^{1/n} - 1)$.

Q.N A system contains five tasks: T1 (1, 0.25), T2 (1.25, 0.1), T3 (1.5, 0.3), T4 (1.75, 0.07), T5 (2, 1). Find that the system is schedule by using RM algorithms.

Practical Factor for Priority Driven Scheduling of Periodic Task:

We have assumed that:

- Jobs are preemptable at any time.
- Jobs never suspend themselves.
- Each job has distinct priority.
- The scheduler is event driven and acts immediately.

These assumptions are often not valid so following practical factors must be considered for priority driven scheduling.

3. Blocking and Priority Inversion
4. Self-Suspension
5. Context Switches
6. Thick Scheduling

Blocking and Priority Inversion:

⇨ A ready job is *blocked* when it is prevented from executing by a lower-priority job; a *priority inversion* is when a lower-priority job executes while a higher-priority job is blocked.

⇨ These occur because some jobs cannot be pre-empted:

- Many reasons why a job may have non-preemptable sections

⇨ Critical section over a resource Some system calls are non-preemptable

⇨ Disk scheduling

- If a job becomes non-preemptable, priority inversions may occur, these may cause a higher priority task to miss its deadline.
- When attempting to determine if a task meets all of its deadlines, must consider not only all the tasks that have higher priorities, but also non-preemptable regions of lower-priority tasks.

⇨ Add the blocking time in when calculating if a task is schedulable.

Self-suspension:

⇨ A job may invoke an external operation (e.g. request an I/O operation), during which time it is suspended.

- ⇨ This means the task is no longer strictly periodic… again need to take into account self-suspension time when calculating a schedule.

Context Switches:

- ⇨ Assume maximum number of context switches *Ki* for a job in *Ti* is known; each takes t_{CS} time units.
- ⇨ Compensate by setting execution time of each job, *e*actual = $e + 2t_{CS}$
  (more if jobs self-suspend, since additional context switches)

Tick Scheduling:

- ⇨ All of our previous discussion of priority-driven scheduling was driven by job release and job completion events.
- ⇨ Alternatively, can perform priority-driven scheduling at periodic events (timer interrupts) generated by a hardware clock i.e. tick (or time-based) scheduling.
- ⇨ Additional factors to account for in schedulability analysis.
- ⇨ The fact that a job is ready to execute will not be noticed and acted upon until the next clock interrupt; this will delay the completion of the job.
- ⇨ A ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue, the *pending job* queue.
- ⇨ When the scheduler executes, it moves jobs in the pending queue to the ready queue according to their priorities; once in ready queue, the jobs execute in priority order.

Chapter-7

Scheduling aperiodic and sporadic jobs in priority driven systems:

Assumptions and Approaches-Objectives:

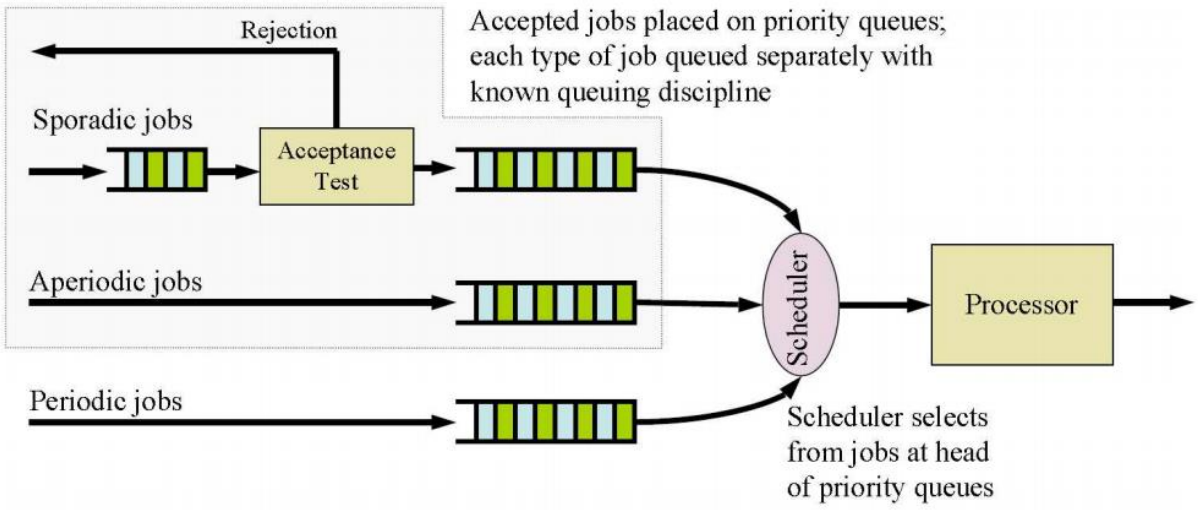
Assumptions:

- The parameters of each sporadic job become known after it is released.
- Sporadic jobs may arrive at any instant, even immediately after each other.
- Moreover, their execution times may vary widely, and
- Their deadlines are arbitrary.

It is impossible for some sporadic jobs to meet their deadlines no matter what algorithm we use to schedule them. The only alternatives are:

- To reject the sporadic jobs that cannot complete in time or
- To accept all sporadic jobs and allow some of them to complete late. Which alternative is appropriate depends on the application.

System Model:



System model consists the separate queue for the aperiodic, periodic, sporadic tasks in a well-known order. The main objective of this system model is to complete each aperiodic job as soon as possible, without causing periodic tasks or accepted sporadic jobs to miss deadlines such that aperiodic jobs are always accepted. Based on the execution time and deadline of each newly arrived sporadic jobs are decided as whether to accept or reject. When the job will complete within its deadline, without causing any periodic task or previously accepted sporadic job to miss its deadline then job be accepted. If sporadic job cannot guarantee it will meet its deadline then it is

rejected from the scheduling. This is done by acceptance test. When the sporadic job is accepted then it enters into the priority queue based on known discipline.

Correctness:

- A correct schedule or correctness of schedule is one where all periodic tasks, and any sporadic tasks that have been accepted, meet their deadlines.
- A scheduling algorithm supporting aperiodic and/or sporadic jobs is a correct algorithm if it only produces correct schedules for the system.

Optimality:

- The system is said to be optimal when the queuing discipline is well-known and corresponding scheduling algorithms are optimal to schedule the periodic, aperiodic and sporadic jobs.
- A periodic job scheduling algorithms is said to be optimal algorithm when it always produces a feasible schedule accepted periodic jobs.
- A sporadic job scheduling algorithm is optimal if it accepts a new sporadic job, and schedules that job to complete by its deadline, if and only if the new job can be correctly scheduled to complete in time.
- An aperiodic job scheduling algorithm is said to be optimal if it minimizes either the response time of the job at the head of the aperiodic job queue or the average response time of all aperiodic jobs for a given queuing discipline.

Scheduling Aperiodic Jobs:

Generally three approaches are used as

- Background approach
- Interrupt driven execution approach
- Polled approach

Background Approach:

In this approach, aperiodic jobs are executed in the background i.e. Aperiodic jobs are scheduled and executed only at times when there are no periodic or sporadic jobs ready for execution.

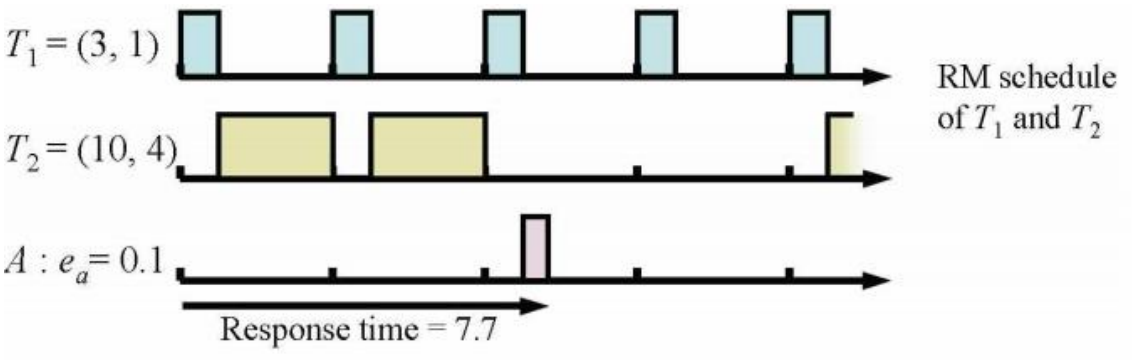
Advantages:

- Clearly produces correct schedules
- Extremely simple to implement

Disadvantage:

- It is not optimal since it is almost guaranteed to delay execution of aperiodic jobs in favor of periodic and sporadic jobs, giving unduly long response times for the aperiodic jobs. For example consider two periodic task T1 (3, 1), T2 (10, 4) and a aperiodic task A released at

0.1 with execution time 0.8. when the periodic tasks are executed by using RM algorithms and aperiodic task be executed in background as :



Here, response time for aperiodic job = 7.8 – 0.1 = 7.7

It is a worst case response time.

Interrupt Driven Execution Approach:

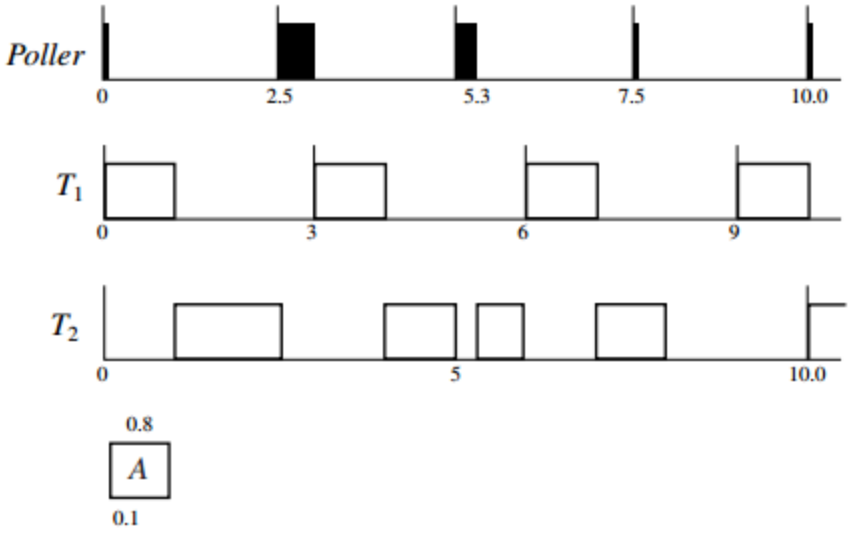
Whenever an aperiodic job arrives, the execution of periodic tasks is interrupted, and the aperiodic job is executed then it is called as interrupt driven execution. This approach reduces response times of aperiodic jobs but will not be correct and will often cause periodic/sporadic tasks in the system to miss some deadlines.

Polled Approach:

It is a common way to schedule aperiodic jobs using a polling server. A periodic job (ps, es) is a periodic task called as poller or polling server where ps is the polling period and es execution time. The poller is ready for execution periodically at integer multiple of Ps and scheduled together with periodic tasks in the system according to priority-driven algorithms. When executed, it examines the aperiodic job queue then

- If an aperiodic job is in the queue, it is executed for up to e_s time units
- If the aperiodic queue is empty, the polling server self-suspends, giving up its execution slot
- The server does not wake-up once it has self-suspended, aperiodic jobs which become active during a period are not considered for execution until the next period begins

For example: Consider a system with two periodic task T1 (3, 1) , T2 (10, 4), a periodic task A with r = 0.1 and e = 0.8 and a poller (2.5, 0.5) then the aperiodic task be scheduled along with scheduling of periodic task using RM as :



Response time of task A = 5.3 – 0.1 = 5.2

Periodic Servers:

A task that behaves much like a periodic task, but is created for the purpose of executing aperiodic jobs, is called as periodic server. A periodic server, $T_{PS} = (P_{PS}, e_{PS})$ never executes for more than e_{PS} units of time in any time interval of length P_{PS} where the parameter e_{PS} is called the execution budget (or simply budget) of the periodic server. When a server is scheduled and executes aperiodic jobs, it consumes its budget at the rate of 1 per unit time and the budget has been exhausted when it reaches 0. A time instant when the budget is replenished is called a replenishment time.

A periodic server is backlogged whenever the aperiodic job queue is nonempty i.e. it is idle if the queue is empty. The periodic server is scheduled as any other periodic task based upon the priority scheme used by the scheduling algorithm. A polling server is a simple kind of periodic server. The budget is replenished to e_s at the beginning of each period and the budget is immediately consumed if there is no work when the server is scheduled

Bandwidth-Preserving Servers:

- A deficiency of the polling server algorithm is that if the server is scheduled when it is not backlogged, it loses its execution budget until it is replenished when it again released.
- An aperiodic job arriving just after the polling server has been scheduled and found the aperiodic job queue empty will have to wait until the next replenishment time.
- Algorithms that improve the polling approach in this manner are called bandwidth-preserving server algorithms
- Bandwidth-preserving servers are periodic servers with additional rules for consumption and replenishment of their budget. The working principle or rules of working of this server are:
    1. A backlogged bandwidth-preserving server is ready for execution when it has budget

2. The scheduler keeps track of the consumption of the budget and suspends the server when it is exhausted, or the server becomes idle.
3. The scheduler moves the server back to the ready queue once it replenishes its budget, if the server is backlogged at that time.
4. If arrival of an aperiodic job causes the server to become backlogged, and it has budget, the server is put back on the ready queue.

The last rule overcomes the deficiency of pooling server. Different types of bandwidth-preserving server are:

- Deferrable servers
- Sporadic servers
- Constant utilization servers
- Total bandwidth servers
- Weighted fair queuing servers

Deferrable Server:

It is a simplest bandwidth-preserving server that improves response time of aperiodic jobs as compared to polling server.

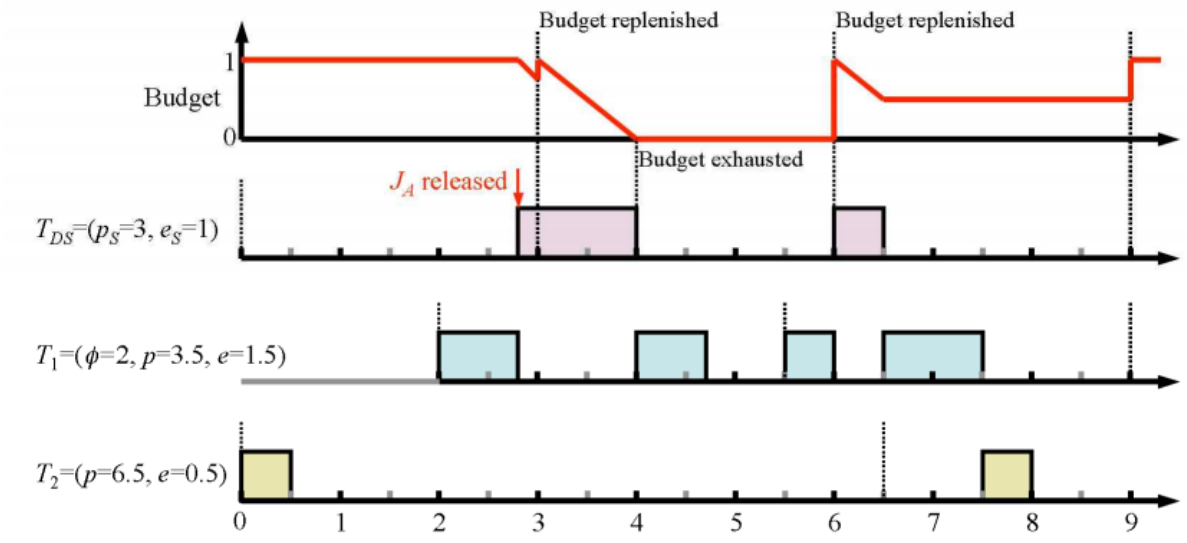
Consumption rule:

- The budget is consumed at the rate of one per unit time whenever the server executes.
- Unused budget is retained throughout the period, to be used whenever there are aperiodic jobs to execute.
- If no aperiodic job is available to execute at start of period then keep it instead of discarding with the hope of new arrival of new aperiodic job.

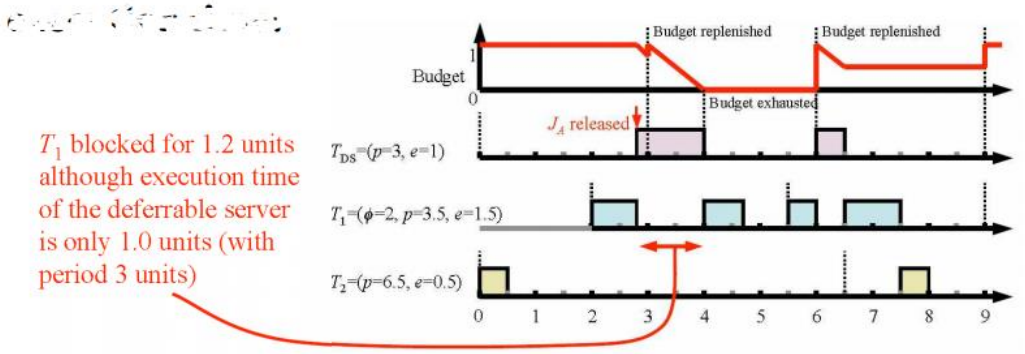
Replenishment rule:

- The budget is set to e_S at multiples of the period i.e. time instants $k \cdot P_S$, for $k = 0, 1, 2, \ldots$

For example: consider a system with two independent periodic tasks T1 (F = 2, P = 3.5, e = 1.5) and T2 (P = 6.5, e = 0.5) with aperiodic jobs A (r=2.75, e = 1.75) be executed in deferrable server T_{DS} = (Ps = 3, e_s =1). The RM schedule of periodic task and execution of aperiodic task on deferrable server is:



Limitations of Deferrable Servers:



They may delay lower-priority tasks for more time than a periodic task with the same period and execution time.

Sporadic Server:

A different type of bandwidth preserving server that is designed to eliminate the limitation of the deferrable server is called sporadic server. It has more complex consumption and replenishment rules ensure that a sporadic server with period p_S and budget e_S never demands more processor time than a periodic task with the same parameters. Consider a system T of N independent preemptable periodic tasks, plus a single sporadic server task with parameters (p_s, e_s) then

- T_H is the subset of periodic tasks with higher priorities than the server.
- t_r defines the last time the server budget replenished.
- t_f defines the first instant after t_r at which the server begins to execute.
- t_e denotes the latest effective replenishment time.

- At any time t define:
  - ✓ BEGIN as the start of the earliest busy interval in the most recent contiguous sequence of busy intervals of T_H starting before t.
  - ✓ END as the end of the latest busy interval in this sequence if this interval ends before t; define END = ∞ if the interval ends after t.

The consumption and replenishment rules be stated as:

Consumption rule

1. At any time t after t_r , server's budget is consumed at the rate of 1 per unit time until the budget is exhausted when either one of following two conditions is true.
   C1: The server is executing
   C2: The server has executed since t_r and END < t
2. When they are not true, the server holds its budget

Replenishment rules

R1: When system begins executing and budget is replenished as:

Budget = e_S and t_r = the current time.

R2: When server begins to execute i.e. at time t_f do

if END = t_f then

t_e = max(t r, BEGIN)

else if END < tf then

$t_e = t_f$

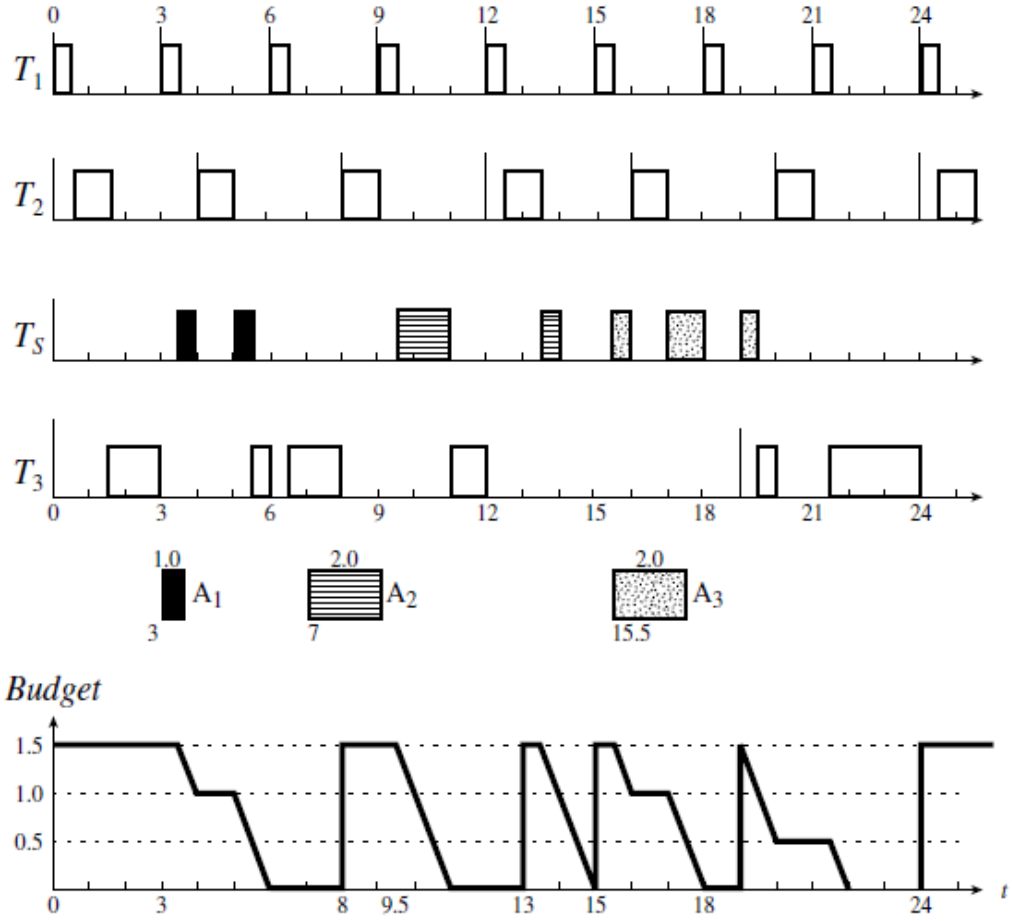
The next replenishment time is set to $t_e + p_S$.

R3: The next replenishment occurs at the next replenishment time (= $t_e + p_S$), except under the following conditions:

a. If t e + p_S is earlier than t_f the budget is replenished as soon as it is exhausted
b. If T becomes idle before $t_e + p_S$, and becomes busy again at t_b, the budget is replenished at min(t_b, $t_e + p_S$)

For example:

Consider, the budget of the server (5, 1.5) is 1.5 initially. It is scheduled rate-monotonically with three periodic tasks: T1 = (3, 0.5), T2 = (4, 1.0), and T3 = (19, 4.5). They are schedulable even when the aperiodic job queue is busy all the time. Here the priority order be maintained as: T1>T2>Ts>T3 where TH= (T1, T2)



1. From time 0 to 3, the aperiodic job queue is empty and the server is suspended. Since it has not executed, its budget stays at 1.5. At time 3, the aperiodic job A1 with execution time 1.0 arrives; the server becomes ready. Since the higher-priority task (3, 0.5) has a job ready for execution, the server and the aperiodic job wait.
2. The server does not begin to execute until time 3.5. At the time, tr is 0, BEGIN is equal to 3, and END is equal to 3.5. According to rule R2, the effective replenishment time te is equal to max(0, 3.0) = 3, and the next replenishment time is set at 8.
3. The server executes until time 4; while it executes, its budget decreases with time.
4. At time 4, the server is preempted by T2. While it is preempted, it holds on to its budget.
5. After the server resumes execution at 5, its budget is consumed until exhaustion because it executes (C1) and then, when it is suspended again, T1 and T2 are idle (or equivalently, END, which is 5.0, is less than the current time) (C2).
6. When the aperiodic job A2 arrives at time 7, the budget of the server is exhausted; the job waits in the queue.
7. At time 8, its budget replenished (R3), the server is ready for execution again.
8. At time 9.5, the server begins to execute for the time since 8. te is equal to the latest replenishment time 8. Hence the next replenishment time is 13. The server executes until

its budget is exhausted at 11; it is suspended and waits for the next replenishment time. In the meantime, A2 waits in the queue.

9. Its budget replenished at time 13, the server is again scheduled and begins to execute at time 13.5. This time, the next replenishment time is set at 18. However at 13.5, the periodic task system T becomes idle. Rather than 18, the budget is replenished at 15, when a new busy interval of T begins, according to rule R3b.
10. The behavior of the later segment also obeys the above-stated rules. In particular, rule R3b allows the server budget to be replenished at 19.

Constant Utilization Server:

- The constant utilization server is based on the size of the server.
- It reserves a known fraction ($\tilde{u}_s$) of the processor time for execution of the server.
- When the budget is non-zero then the server is scheduled with other tasks on an EDF basis.
- The budget and deadline of the server are chosen such that the utilization of the server is constant when it executes, and that it is always given enough budget to complete the job at the head of its queue each time its budget is replenished.
- The server never has any budget if it has no work to do.

Here

$\tilde{u}_s$ = size of server

e_s = budget of server

d = deadlines

t = current time

e= execution time of job at the head of aperiodic job que

Then the consumption and replenishment rule of constant utilization server is:

Consumption Rule:

- A constant utilization server only consumes budget when it executes.

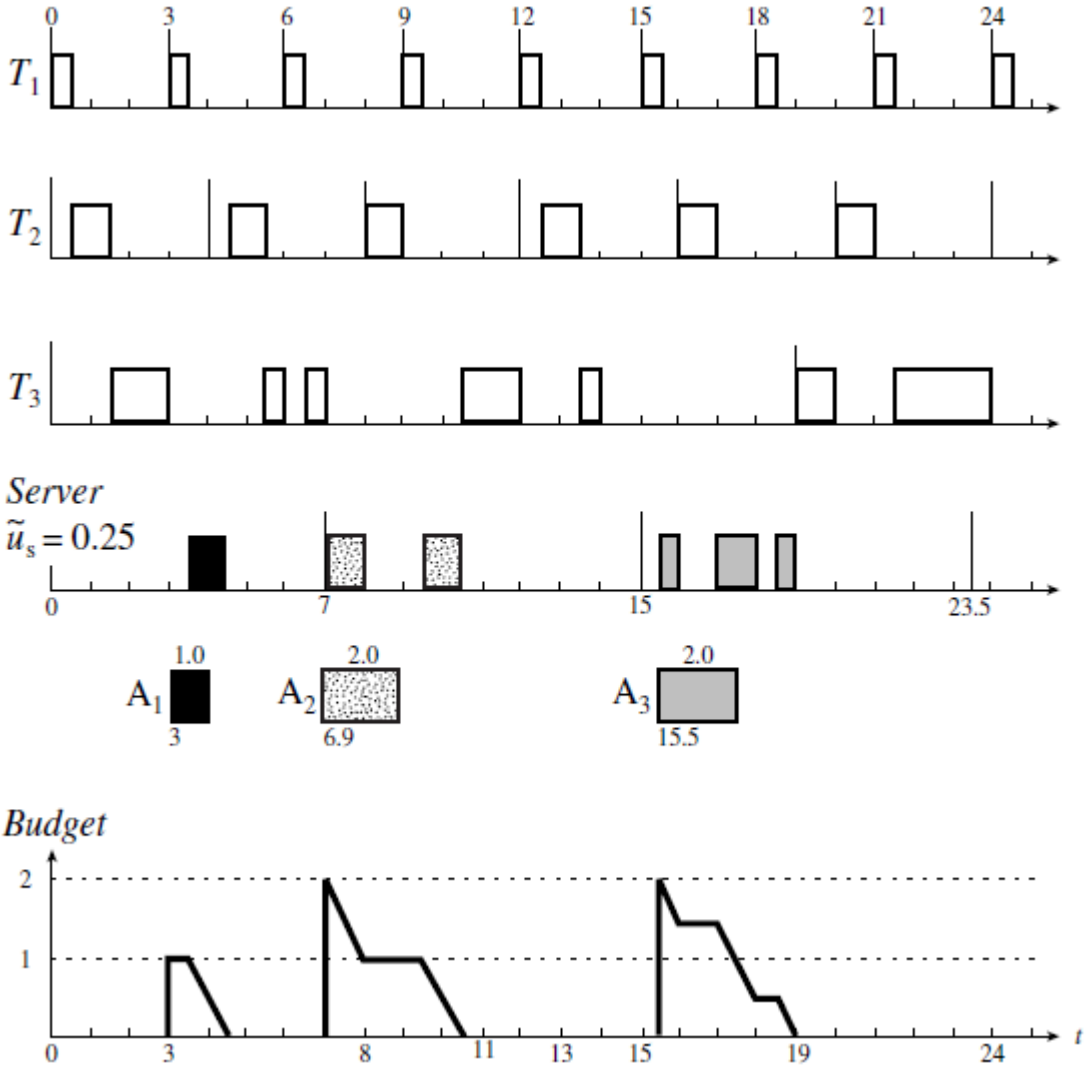
Replenishment Rules:

- Initially, set budget $e_s = 0$ and deadline $d = 0$
- When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue then
    - If $t < d$, do nothing ($\Rightarrow$ server is busy; wait for it to become idle)
    - If $t \geq d$ then set $d = t + e/\tilde{u}_s$ and $e_s = e$
- At the deadline d of the server
    - If the server is backlogged, set $d = d + e/\tilde{u}_s$ and $e_s = e$ $\Rightarrow$ was busy when job arrived

- If the server is idle, do nothing

I.e. the server is always given enough budget to complete the job at the head of its queue, with known utilization, when the budget is replenished.

For example: ũ s = 0.25 T1 = (3, 0.5), T2 = (4, 1), T3 = (19, 4.5)



Total Bandwidth Server:

- A constant utilization server gives a known fraction of processor capacity to a task but cannot claim unused capacity to complete the task earlier.
- A total bandwidth server improves responsiveness by allowing a server to claim background time not used by the periodic tasks.

- This is done by having the scheduler replenish the server budget as soon as the budget is exhausted if the server is backlogged at the time or as soon as the server becomes backlogged.

Consumption Rule:

- A total bandwidth server only consumes budget when it executed.

Replenishment Rule:

- Initially, e s = 0 and d = 0
- When an aperiodic job with execution time e arrives at time t to an empty aperiodic job queue
    - Set d = max(d, t) + e/ũ s and e $_s$ = e
- When the server completes the current aperiodic job, the job is removed from the queue then
    - If the server is backlogged, set d = d + e/ũ s and e s = e
    - If the server is idle, do nothing

Total bandwidth server is always ready for execution when backlogged and assigns at least fraction ũ s of the processor to a task foe execution.

For example: ũ $_s$ = 0.25 T1 = (3, 0.5), T2 = (4, 1), T3 = (19, 4.5)

Do your self

Weighted Fair Queuing Server:

The constant utilization server and total bandwidth server are used to assign some fraction of processor capacity to an aperiodic task. During assignment of task, there is an issue of fairness and starvation i.e. the fairness cannot maintained in case of total bandwidth and constant utilization server approach. A scheduling algorithms is fair with in any particular time interval if fraction of processor time in the interval attained by each backlogged server is proportional to size of server.

In such system not only all the task meet their deadlines, but they all make a continuous process according to their share of processor and there is no starvation. The waited fair queuing algorithms are used to share processor time between sever and designed to ensure fairness is allocation among multiple servers.

Consumption Rule:

It consumes budget when it executed.

Replenishment Rule:

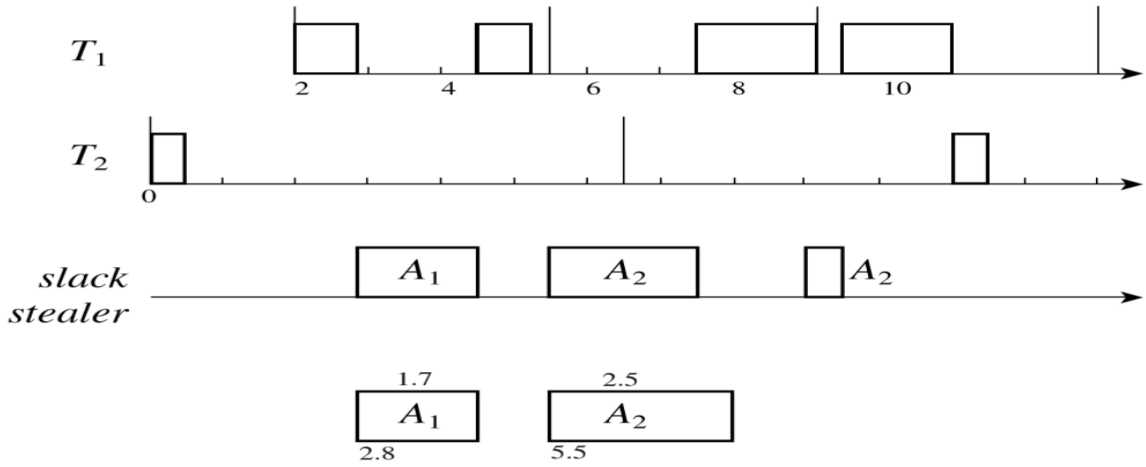
- Its budget is replenishment when it first becomes backlogged after being idle.
- As long as it is backlogged, its budget is replenished each time it completes a job.

- At each replenished time, the server budget is set to the execution time of the job at the head of queue.

Slack Stealing in Deadline Driven System:

- The slack stealer is a periodic server to execute aperiodic job using slack time.
- The slack stealer is ready for execution whenever the aperiodic job queue is nonempty and is suspended when the queue is empty.
- The scheduler monitors the periodic tasks in order to keep track of amount of available slack.
- Whenever the slack stealer executes, it executes the aperiodic job at the head of aperiodic job queue.
- This kind of slack stealing algorithms is said to be greedy and non-optimal.
- The available slack is always used if there is an aperiodic job ready to be executed.
- Slack computation algorithm is correct if it never says that the system has slack when the system does not, since doing so may cause a periodic job to complete too late

Consider, a system of two periodic tasks, T1 = (2.0, 3.5, 1.5) and T2 = (6.5, 0.5). In addition to the aperiodic job that has execution time 1.7 and is released at 2.8, suppose another aperiodic job with execution time 2.5 is released at time 5.5. These jobs are A1 and A2, respectively. The figure below shows the operation of a slack stealer.



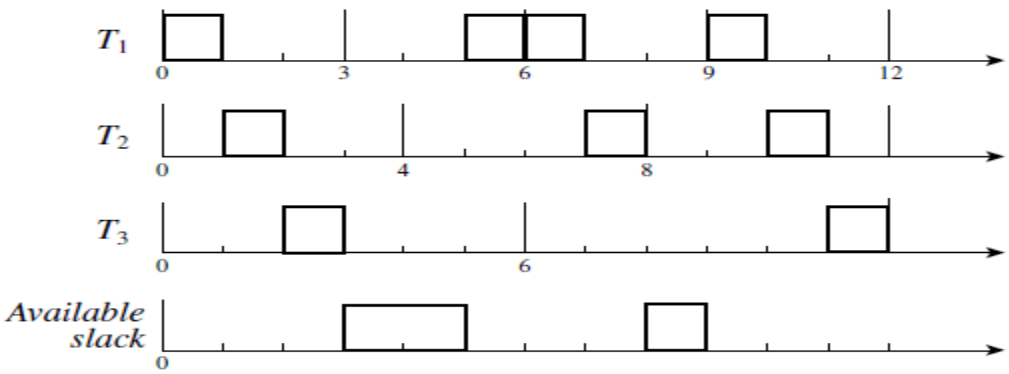
- Initially, the slack stealer is suspended because the aperiodic job queue is empty. When A1 arrives at 2.8, the slack stealer resumes. Because the execution of the last 0.7 units of J1, 1 can be postponed until time 4.8 (i.e., 5.5−0.7) and T2 has no ready job at the time, the system has 2 units of slack. The slack stealer is given the highest priority. It preempts J1, 1 and starts to execute A1. As it executes, the slack of the system is consumed at the rate of 1 per unit time.

- At time 4.5, A1 the slack stealer is suspended. The job J1, 1 resumes and executes to completion on time.
- At time 5.5, A2 arrives, and the slack stealer becomes ready again. At this time, the execution of the second job J1, 2 of T1 can be postponed until time 7.5, and the second job J2, 2 of T2 can be postponed until 12.5.Hence, the system as a whole has 2.0 units of slack. The slack stealer has the highest priority starting from this time. It executes A2.
- At time 7.5, all the slack consumed, the slack stealer is given the lowest priority. J1, 2 preempts the slack stealer and starts to execute.
- At time 9, J1, 2 completes, and the system again has slack. The slack stealer now has the highest priority. It continues to execute A2.
- When A2 completes, the slack stealer is suspended again. For as long as there is no job in the aperiodic job queue, the periodic tasks execute on the EDF basis.

Slack Stealing in Fixed priority System (Optimality Criterion and Design Consideration):

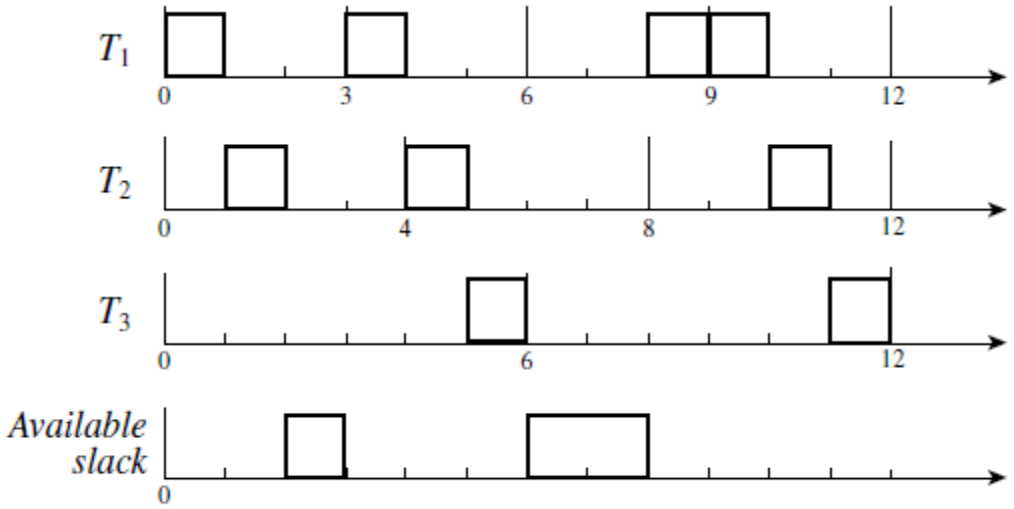
In principle, slack stealing in a fixed-priority system works in the same way as slack stealing in a deadline-driven system. However, the computation and the usage of the slack are both more complicated in fixed-priority systems. Consider, the system contains three periodic tasks T1 = (3, 1), T2 = (4, 1), and T3 = (6, 1) such that,

- Job J11, J21 and J31 released at time 0 with priority order J11>J21>J31 where J11 can be postponed up to 2, J21 up to 3 and J31 up to 5. Similarly job J12 and job J22 are released at 3 and 4 and they are postponed up to 5 and up to 7 such that slack of 1 be available in between 0 and 3. Similarly a slack of 1 available between 3 and 6. When slack of 1 before 3 is not used then slack stealer can use the available slack of 2 in between 3 to 5.
- Similarly job J13 released at 6 and can be postponed up to 8. Job J23 released at 8 and can be postponed up to 11. Job J32 released at 6 and can be postponed up to 11. Hence a slack of 1 available between 6 to 12 and slack stealer can use it and uses in between 8 to 9.
- The corresponding schedule without using slack of 1 before 3 be constructed as:



(a)

- When the 1 unit of slack before 3 is used in between 2 to 3 then job J31 postponed to 5 and completes its execution up to 6. The job J12 and J22 released at 3 and 4 can be postponed up to 5 and 7. The slack of 2 available in between 3 to 8. When the slack stealer use the slack of 2 in between 6 to 8 the schedule be generated as:



(b)

- When an aperiodic job of 1 execution time released in between 0 to 3 be executed following schedule (b) provides the better response time. If the aperiodic job of execution time not more than 2 provides the better response time when it is executed by fallowing schedule (a).

The example points out the following:

- No slack-stealing algorithm can minimize the response time of every aperiodic job in a fixed-priority system even when prior knowledge on the arrival times and execution times of aperiodic jobs is available i.e. low chance to meet optimality
- The amount of slack a fixed-priority system has in a time interval may depend on when the slack is used. To minimize the response time of an aperiodic job, the decision on when to schedule the job must take into account the execution time of the job.

Model for Scheduling Sporadic Jobs:

When sporadic jobs arrive, they are both accepted and scheduled in EDF order

- In a dynamic-priority system, this is the natural order of execution
- In a fixed-priority system, the sporadic jobs are executed by a bandwidth preserving server, which performs an acceptance test and runs the sporadic jobs in EDF order
- In both cases, no new scheduling algorithm is required

Definitions: –

- Sporadic jobs are denoted by Si(ri, di, ei) where ri is the release time, di is the (absolute) deadline, and ei is the maximum execution time
- The density of a sporadic job $\Delta i = ei/(di - ri)$ • The total density of a system of n jobs is $\Delta = \Delta 1 + \Delta 2 + \ldots + \Delta n$
- The job is active during its feasible interval (ri,di]

Sporadic Jobs in Dynamic Priority System:

Theorem:

A system of independent preemptable sporadic jobs is schedulable according to the EDF algorithm if the total density of all active jobs in the system ≤ 1 at all times.

- This is the standard schedulability test for EDF systems, but including both periodic and sporadic jobs.
- This test uses the density since deadlines may not equal periods; hence it is a sufficient test, but not a necessary test.

This means:

- If we can bound the frequency with which sporadic jobs appear to the running system, we can guarantee that none are missed.
- Alternatively, when a sporadic job arrives, if we deduce that the total density would exceed 1 in its feasible interval, we reject the sporadic job.

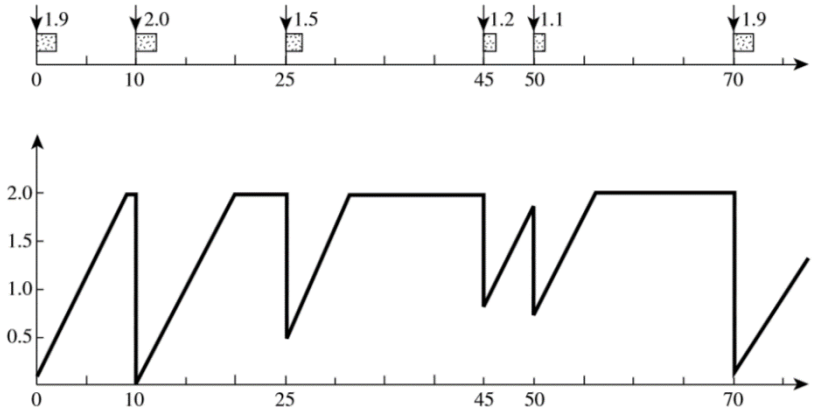
Real Time Performance of Jobs with Soft Timing Constraints- Traffic Models:

- Specifically, when requesting admission into the system, each sporadic task presents to the scheduler its traffic parameters and the collection of these parameters are called the flow specification of the task in communications literature.
- They define the constraints on the inter-arrival times and execution times of jobs in the task.
- The performance guarantee provided by the system to each task is conditional which means that the system delivers the guaranteed performance conditioned on the fact that the task meets the constraints define by its traffic parameters.
- The flip side is that if the task misbehaves (that is, its actual parameters deviate from its traffic parameters), the performance experienced by it may deteriorate
- . Different traffic models use different traffic parameters to specify the behavior of a sporadic tasks.

Leaky Bucket Model:

We first introduce the notion of a (Λ, E) leaky bucket filter, then define the leaky bucket model. This kind of fillter is specified by its input rate Λ and size E: The filter can hold at most E tokens

at any time and it is filled with tokens at a constant input rate of Λ tokens per unit time. A token is lost if it arrives at the filter if the filter already contains E tokens. It can be assumed of sporadic task that meets the (Λ, E) leaky bucket constraint as if its jobs were generated by the filter in the following manner. The filter may release a job with execution time e when it has at least e tokens. After the release of this job, the e tokens are removed from the filter. A job cannot be released when the filter has no token. Therefore, not any job in a sporadic task that satisfies the (Λ, E) leaky bucket constraint, has execution time larger than E. The total execution time of all the jobs which are released within any time interval of length E/Λ is less than 2E. A periodic task with period equal to or larger than E/ Λ and execution time equal to or less than E satisfies the (Λ, E) leaky bucket constraint. A sporadic task S = (1, p, p', I) that fits the FeVe model satisfiÂÂÂÂÂ • es the constraint if p' = 1/ Λ and E = (1− p/p')I/p'.



Leaky Bucket traffic model

Chapter- 8

Resources and Resource Access Control

Resources may represents

- Hardware devices such as sensor and actuators
- Disk or memory capacity and buffer space
- Software resources such as locks or queues

Resource Access Control Protocols work to reduce the undesirable effect of resource contention. Resource contention affects the execution behavior and schedulability of jobs. It is focused on the Priority Driven Systems. Clock Driven Systems can avoid resource contention among jobs by scheduling them according to cyclic schedule that keeps jobs‟ resource access serialized.

Assumptions on Resources and their Usage:

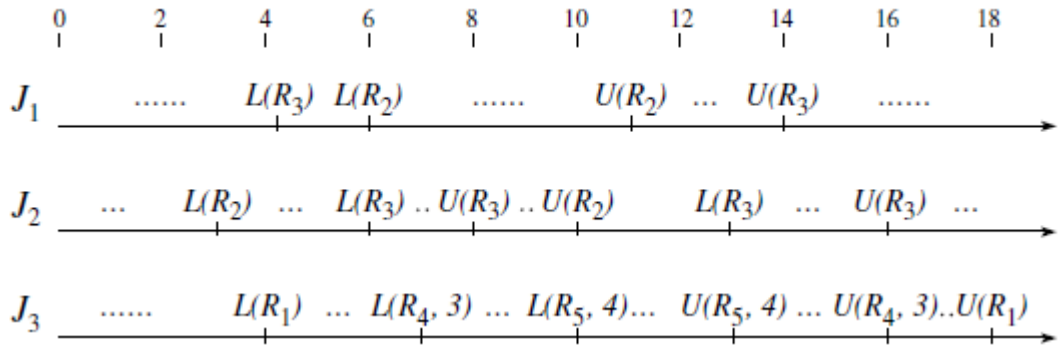
- The system consists of only one processor
- The system also contains p types of serially reusable resources named R1, R2 ---- Rp
- There are v i indistinguishable units of resources of type Ri for $1 \leq i \leq p$
- Serially reusable resources are granted to jobs on a non-preemptive basis and used in a mutually exclusive manner
- When a unit of a resources Ri is granted to a job, this unit is no longer available to other jobs until the job frees the unit. For example mutexes, reader/writer locks, connection sockets, printers.
- A binary semaphore is a resource that has only one unit while a counting semaphore has many units. For example a system containing five printers has five units of the printer resource
- There is only one unit of an exclusive write lock.
- Some resources can be used by more than one job at the same time
- This type of resource has many units each used in a mutually exclusive manner e.g. a file that can be read by at most „v‟ users at the same time

Mutual Exclusion and Critical Sections:

- A lock based concurrency control mechanism is used to enforce mutually exclusive accesses of jobs to resources.
- When a job wants to use n_i units of resource R i, it executes a lock to request them This lock request is denoted by $L(R_i, n_i)$.
- If a lock request fails, the requesting job is blocked and loses the processor.
- When the requested resource becomes available, it is unblocked.
- The job holding a lock cannot be preempted by a higher priority job needing that lock.
- The job continues to execute when it is granted the requested resource.
- When the job no longer needs the resource, it releases the resource by executing an unlock denoted by $U(R_i, n_i)$
- A job that begins at a lock and ends at a matching unlock is called a critical section Resources are released in the last-in-first-out-order.

- The expression [R, n, e] is used to represent the critical section regarding n units of resources with the critical section requiring e units of execution time.
- The critical section may nest if a job needs the multiple simultaneous resources. For example lock R1 then Lock R2 then lock R3 and unlock R3, R2 and R1 is represented as
  [R1, n_1, e_1, [R_2, n_2, e_2, [R3, n_3, e_3]]]
- The critical section that is not included in other critical section is called outer most critical section.

For example:



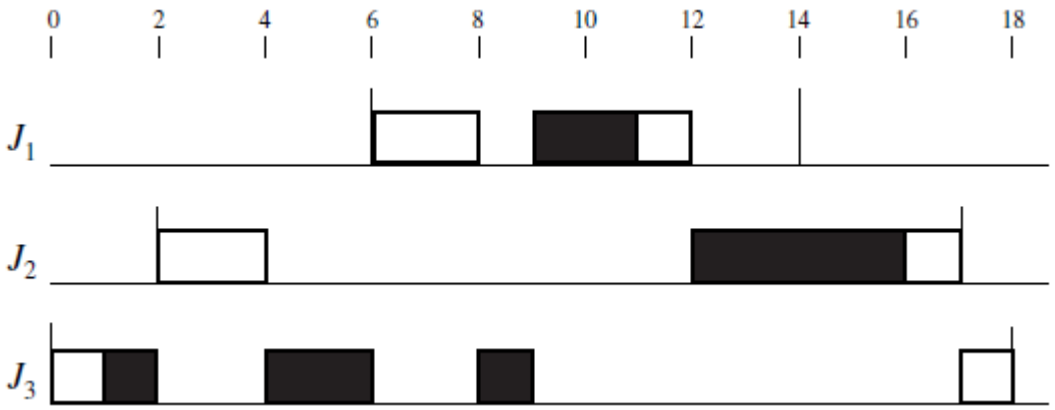
Resource Conflicts and Blocking:

- Two jobs conflict with one another or have a resource conflict if some of the resources they require are of the same type.
- Jobs contend for a resource when one job requests a resource that other jobs already have. The scheduler always denies a request if there are not enough free units of the resource to satisfy the request or to avoid some undesirable execution behavior.
- When the scheduler does not grant n i units of resource R i to the job requesting them, the lock request L(R i, η i) of the job fails.
- When the lock request fails, the job is blocked and loses the processor then blocked job is removed from the ready job queue. It stays blocked until the scheduler grants it n i units of R i for which the job is waiting.
- When the job becomes unblocked it is moved back to the ready queue and executes when it is scheduled.

For example: Consider a system with

- Three jobs J 1, J2, J3, with feasible interval (6, 14], (2, 17], (0 ,18].
- The jobs are scheduled on the processor on the EDF basis.
- J1 has the highest and J3 has the lowest priority.
- All three jobs require the resource R which has only one unit.

Then the critical sections in these jobs are [R; 2], [R; 4], [R; 4] such that



- At time 0 only J3 is ready so it executes
- At time 1, J3 is granted the resource R when it executes L(R)
- J2 is released at time 2, preempts J3 and begins to execute
- At time 4, J2 tries to lock R , because R is in use by J3, this lock request fails . J2 becomes blocked and J3 regains the processor and begins to execute
- At time 6, J1 becomes ready preempts J3 and begins to execute
- J1 executes until time 8 when it executes a L(R) to request R. J3 still has the resource so J1 becomes blocked. Only J3 is ready for execution and it again has the processor and executes.
- The critical section of J3 completes at time 9. The resource R becomes free when J3 executes U(R) both J1 and J3 are waiting for it. The priority of the former is higher. Therefore, the resource and the processor are allocated to J1, allowing it to resume execution.
- J1 releases the resource R at time 11. J2 is unblocked. Since J1 has the highest priority it continues to execute.
- J1 completes at time 12. Since J2 is no longer blocked and has a higher priority than J3, it has the processor, holds the resource and begins to execute. When it completes at time 17, J3 resumes and executes until completion at 18.

This example illustrates how resource contention can delay the completion of higher-priority jobs. It is easy to see that if *J*1 and *J*2 do not require the resource, they can complete by times 11 and 14, respectively.

Effects of Resource Contention and Resource Access Control:

A resource access control protocol is a set of rules that govern

- When and under what conditions each request for resource is granted
- How jobs requiring resources are scheduled.

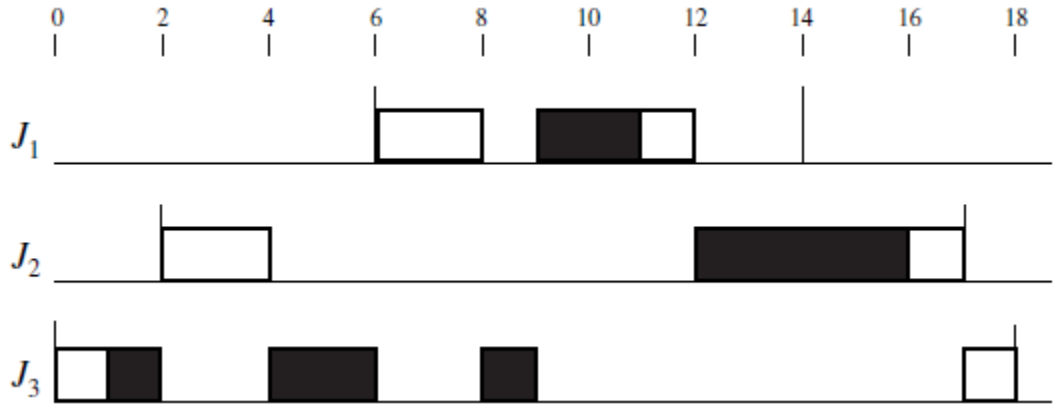
One of the major objective of resources access control is to minimize the undesirable effects of resource allocation. The undesirable effects of resource contention are

1. Priority Inversion

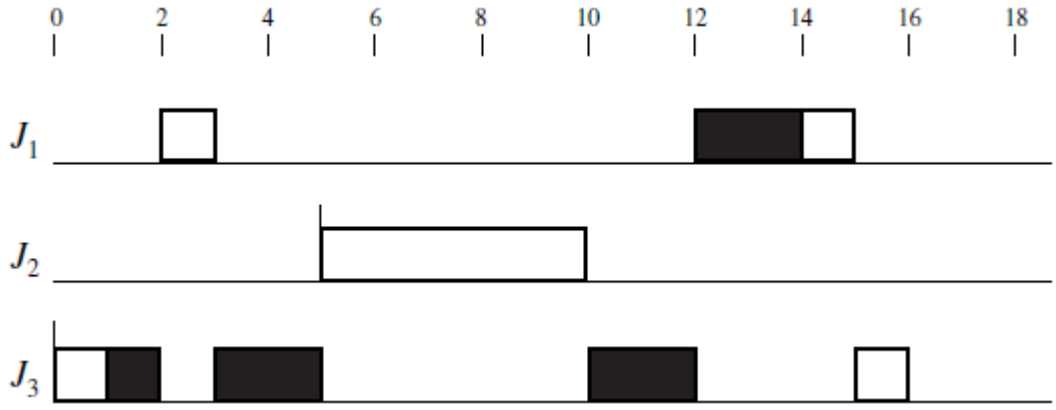
2. Timing anomalies
3. Deadlock

Priority Inversion:

Priority inversion can occur when execution of some jobs or portions of jobs is non-preemptable. As resources are allocated to jobs on a non-preemptive basis, a higher priority job can be blocked by a lower priority job if the jobs conflict, even when the execution of both jobs is preemptable.



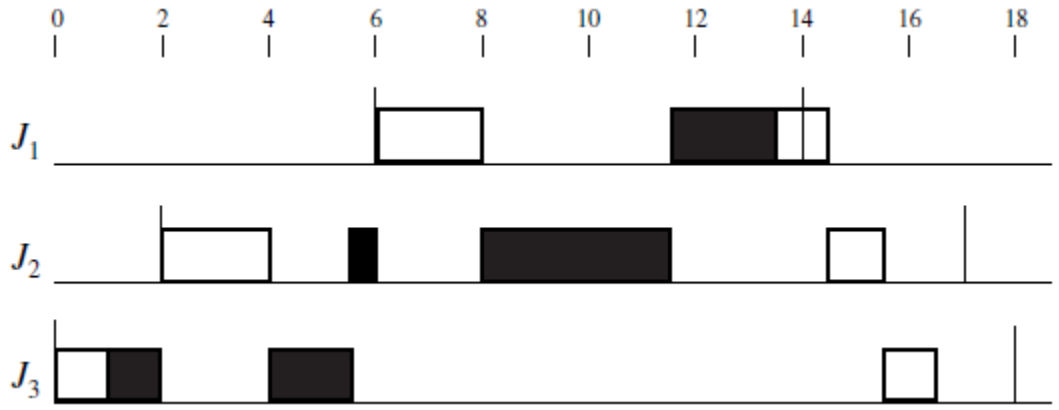
In this example, there are three jobs, J1, J2, and J3, whose feasible intervals are (6, 14], (2, 17] and (0, 18], respectively. Here the lowest priority job J3 first blocks J2 and then blocks J1 while it holds the resource R such that Priority Inversion occurs in intervals (4,6] and (8,9] .
Let us take another example



Here, jobs J1 and J3 have the highest priority and lowest priority, respectively. At time 0, J3 becomes ready and executes. It acquires the resource R shortly afterwards and continues to execute. After R is allocated to J3, J1 becomes ready. It preempts J3 and executes until it requests resource R at time 3. Because the resource is in use, J1 becomes blocked, and a priority inversion begins. While J3 is holding the resource and executes, a job J2 with a priority higher than J3 but lower than J1 is released. Moreover, J2 does not require the resource R. This job preempts J3 and executes to completion. Thus, J2 lengthens the duration of this priority inversion. In this situation, the priority inversion is said to be uncontrolled.

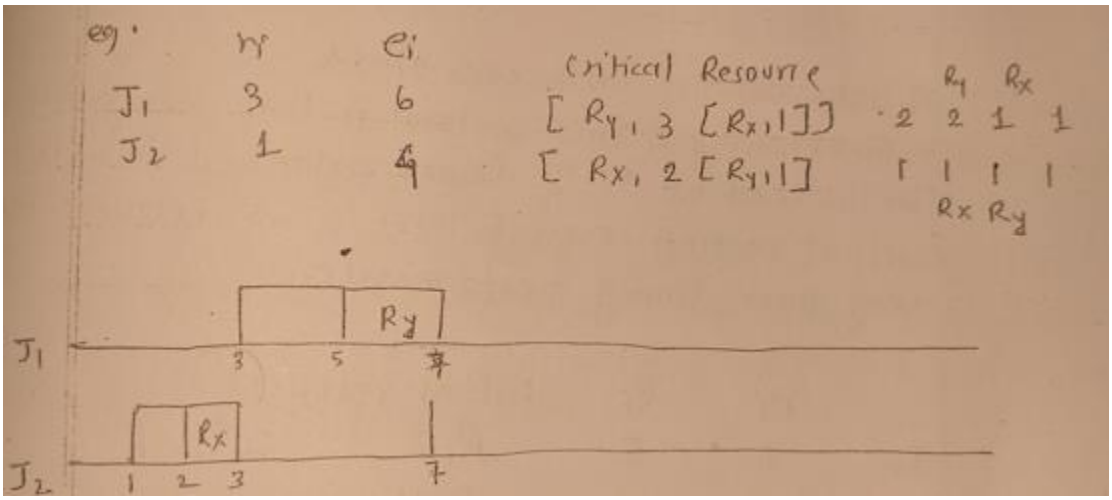
Timing anomalies:

When priority inversion occurs, timing anomalies invariably follow i.e. there may be the chance of miss the deadline. This condition is called timing anomalies. Suppose the critical section J3 is [R; 2.5] in the above example then J1 misses its deadline as it does not complete until 14.5.



Deadlock:

If two jobs need resource Rx and Ry and if one acquires the lock in the ordre Rx then Ry and the other job acquires them in opposite order Ry then Rx then deadlock occurs.



Here the deadlocks occurs when J1 needs Rx locked by J2 and J2 needs Ry locked by J1.

Non-Preemptive Critical Section:

- The simplest way to control access of resources is to schedule all critical sections on the processor non- preemptively.
- When a job holds any resources, it executes at priority higher than the priority of all jobs. This protocol is call non-preemptive critical section protocol (NPCS). Because no job is ever preempted when it holds any resource, deadlock can never occur.

- The most important advantages of NPCS protocol is its simplicity especially when the number of resource units are arbitrary.
- The protocol does not need any prior knowledge about resource requirement of jobs.
- It is simple to implement and can be used in both fixed priority and dynamic priority systems.
- It is clearly a good protocol when all the critical sections are short and when most of jobs conflict with each other.
- Short coming of this protocol is that every job can be blocked by every lower priority jobs with a critical section even if there is no resource and shows very poor performance.

For example:

Job	r_i	e_i	Critical Section
J1	6	5	R,2
J2	2	7	R,4
J3	0	6	R,4

Basic-Priority Inheritance Protocol:

It Works with any preemptive, priority driven scheduling algorithm and does not require prior knowledge on resource requirements of jobs. The priority inheritance protocol does not prevent deadlock .When there is no deadlock the protocol ensures that no job is ever blocked for an indefinitely long time because uncontrolled priority inversion cannot occur.

Definition:

At any time t, each ready job J_l is scheduled and executes at its current priority $\pi_l(t)$ which may differ from its assigned priority and may vary with time. The current priority $\pi_l(t)$ of a job Jl may be raised to the higher priority $\pi h(t)$ of another job J_h. When this happens, we say that the lower priority job J_l inherits the priority of the higher priority job J_h and that J_l executes at its inherited priority $\pi_h(t)$.

Rules of Priority Inheritance Protocol:

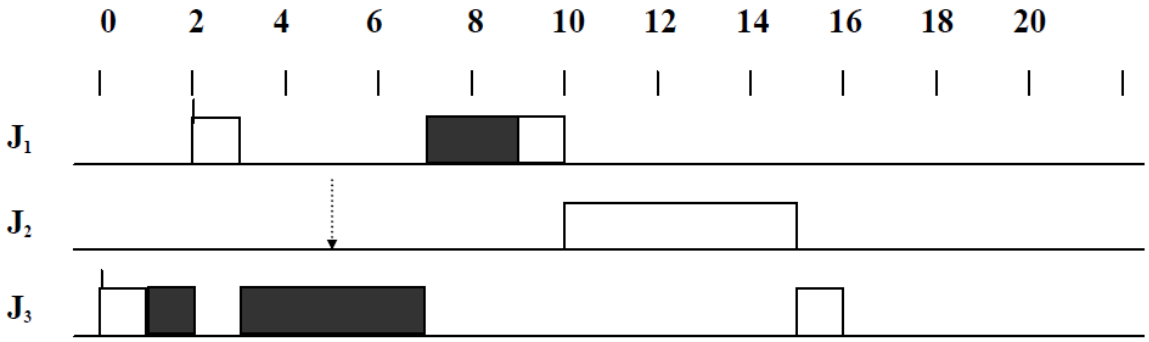
1. Scheduling Rule: Ready jobs are scheduled on the processor preemptively in a priority driven manner according to their current priorities. At its release time t, the current priority π (t) of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

2. Allocation Rule: When a job J requests a resource R, at time t
   - If R is free, R is allocated to J until J releases the resource.
   - If R is not free, the request is denied and J is blocked.

3. Priority Inheritance Rule: When the requesting job J becomes blocked the job J_l which blocks J inherits the current priority $\pi(t)$ of J . The job J_l executes at its inherited priority π

(t) until it releases R at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it acquires the resource R .

According to this protocol, a job J is denied a resource only when the resource requested by it is held by another job.



Controlling Priority Inversion by using Priority Inheritance

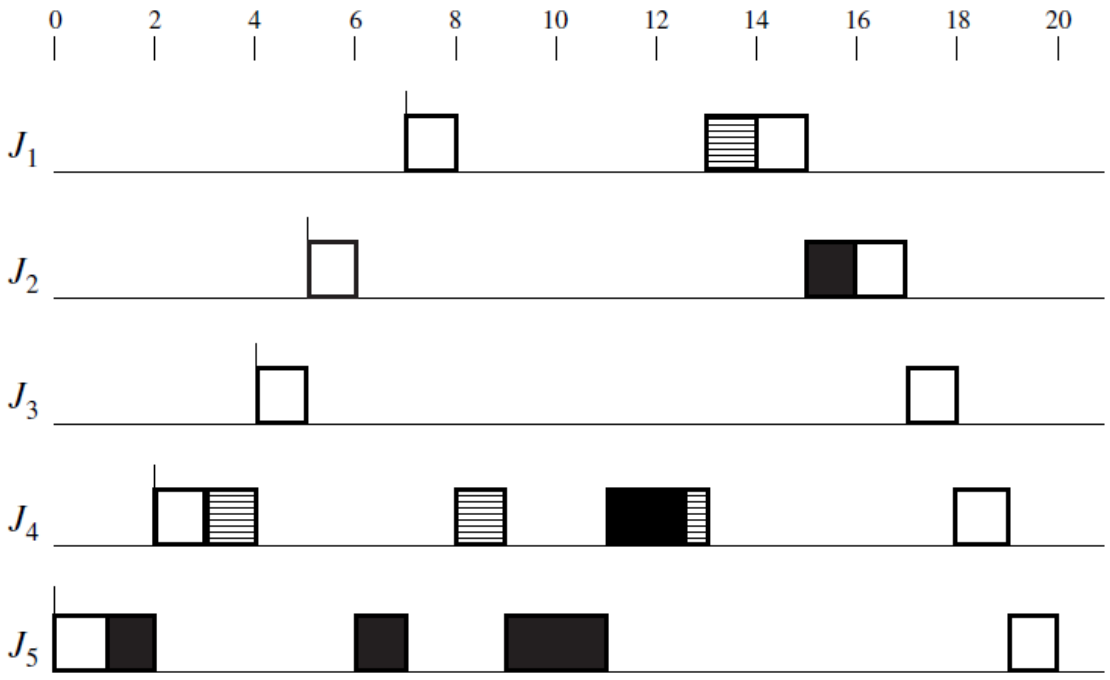
When Jl requests R and becomes blocked by J3 at time 3 , job J3 inherits the priority π1 of Jl When job J2 becomes ready at 5, it cannot preempt J3 as its priority π2 is lower than the inherited priority π1 of J3 So J3 completes its critical section as soon as possible The protocol ensures that the duration of priority inversion is never longer than the duration of an outermost critical section each time a job is blocked.

For another example:

Consider a system with following parameters:

Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]

Then the schedule time diagram using priority inheritance rule can be constructed as:



- At time 0, job J5 becomes ready and executes at its assigned priority 5. At time 1 it is granted the resource black
- At time 2, J4 is released it preempts J5 and starts to execute
- At time 3 J4 requests shaded. Shaded being free is granted to the job. The job continues to executes
- At time 4, J3 is released and preempts J4. at time 5, J2 is released and preempts J3
- At time 6, J2 executes L (black) to request black; L (black) fails as black is in use by J5. J J2 is a now directly blocked by J5. According to rule 3, J5 inherits the priority 2 of J2. as J5‟s priority is now the highest among already jobs J5 starts to execute
- J1 is released at time 7. Having the highest priority 1 it preempts J5 and starts to execute
- At time 8, J1 executes L (shaded) which fails, and becomes blocked. Since J4 has shaded at the time, it directly blocks J1 and consequently inherits J1‟s priority 1. J4 now has the highest priority among the ready jobs J3, J4, J5 therefore it starts to execute
- At time 9, J4 requests the resource black and becomes directly blocked by J5. At this time the current priority of J4 is 1. The priority it has inherited from J1 since time 8. therefore , J5 inherits priority 1 and begins to execute
- At time 11, J5 releases the resource black. Its priority returns to 5, which was its priority when it acquired black. The job with the highest priority among all unblocked jobs is J4. J4 enters its inner critical section and proceeds to complete this and the outer critical section

- At time 13, J4 releases shaded. The job no longer holds any resource. Its priority returns to 4, its assigned priority, J1 becomes unblocked acquires shaded and begins to execute
- At time 15, J1 completes. J2 is granted the resource black and is now the job with the highest priority. Hence it begins to execute.
- At time 17, J2 completes. Later on, jobs J3, J4 and J5 execute in turn to completion

Properties of Priority Inheritance Protocol:

- Simple to implement, does not require the prior knowledge of resource requirement.
- Job exhibits different types of blocking
  - ⇨ Direct blocking: due to resource lock.
  - ⇨ Priority inheritance blocking
  - ⇨ Transitive blocking
- Deadlock is not prevented.
- It can be reduce blocking time compared to non-preemptive critical section but does not guarantee to minimize blocking.

Priority Ceiling Protocol:

The priority ceiling protocol extends the priority inheritance protocol to prevent deadlocks and to further reduce the blocking time. This protocols makes following assumptions.

- The assigned priorities of all jobs are fixed.
- The resources required by all jobs are known a priori before the execution of any job begins.

The protocol makes use of a parameter called priority ceiling of every resource. The priority ceiling of any resource Ri is the highest priority of all the jobs that require Ri and is denoted by Π (Ri). At any time t, the current priority ceiling Π (t) of the system is equal to the highest priority ceiling of the resources that are in use at that time, if some resources are in use. If all the resources are free at the time, the current ceiling Π (t) is Ω, a non-existing priority level that is lower than the lowest priority of all jobs.

Rules of Priority Ceiling Protocol:

1. Scheduling rule:

a. At its release time *t*, the current priority π (t) of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

b. Every ready job J is scheduled preemptively and in a priority driven manner at its current priority π (t).

2. Allocation rule:

Whenever a job J requests a resource R at time *t*, one of the following two conditions occurs:

a. R is held by another job J"s request fails and J becomes blocked.

b. R is free then

i. If J's priority *π (t)* is higher than current priority ceiling, R is allocated to J.

ii. If J's priority π (t) is not higher than the current priority ceiling of the system, R is allocated to J only if J is the job holding the resource whose priority ceiling is equal to current priority ceiling otherwise J's request is denied and J becomes blocked.

3. Priority Inheritance Rule:
   When J becomes blocked the job J_l which blocked J inherits the current priority π (t) of J. J_l executes at its inherited priority until the time when it releases every resource whose priority ceiling is equal to or higher than π (*t*). At that time, priority of J_l returns to its priority π_l (*t'*) at time *t'* when it was granted the resource.

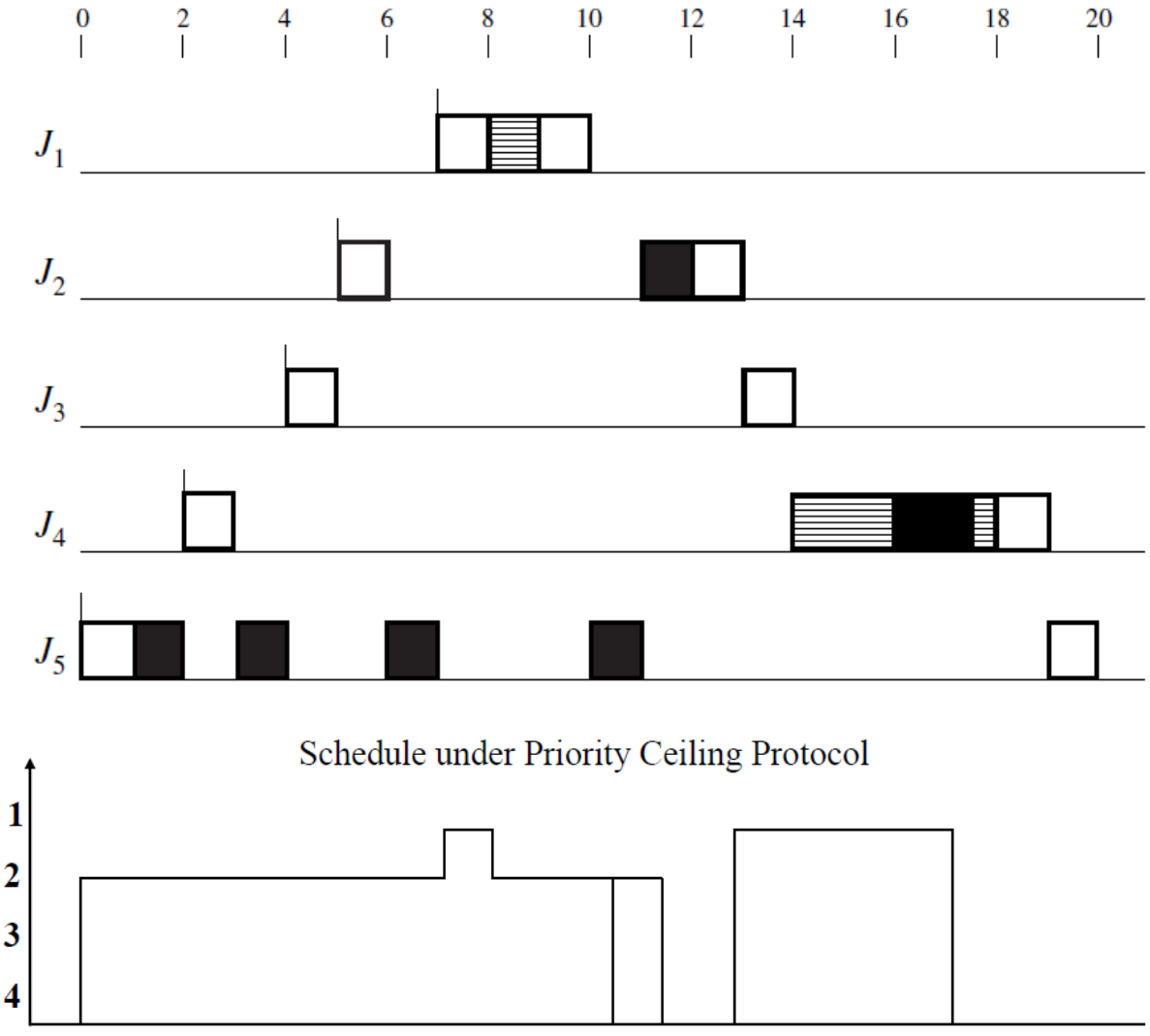
For example:

Consider a system with following parameters:

Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]

- The priority ceiling of the black resource is two since J2 is the highest priority job among the jobs requiring it.
- The priority ceiling of the shaded resource is one.
- In the priority inheritance protocol figure [0,1) current ceiling of the system is Ω lower than 5.
- In (1, 3] interval black is held by J5 so the current ceiling of the system is 2.
- In (3, 13] interval when shaded is also in use the current ceiling of the system is 1.

The access to resources are controlled by the priority ceiling protocol. Here the priority ceiling of the resources Black and Shaded are two and one respectively.



- In the interval (0,3], this schedule is the same as the schedule which is produced under the basic priority inheritance protocol, the ceiling of the system at time 1 is Ω, when J5 requests black it is allocated the resource according to (i) in part (b) of rule 2 after black is allocated the ceiling of the system is raised to two the priority ceiling of black
- At time 3, J4 requests shaded, shaded is free. However, because the ceiling pi(3) is equal to two of the system is higher than the priority of J4 ,J4's request is denied according to ii in part of (b) of rule two J4 is blocked and J5 inherits J4‟s priority and executes at priority 4
- At time 4, J3 preempts J5 and at time 5 J2 preempts J3 at time 6, J2 requests black and becomes directly blocked by J5. So J5 inherits the priority 2. It executes until J1 becomes ready and preempts it. During all this time , the ceiling of the system remains at two
- When J1 requests shaded at time 8, its priority is higher than the ceiling of the system. So its request if granted according to i in part (b) of rule 2 allowing it to enter its critical

section and complete by the time 10 at time 10, J3 and J5 are ready. The latter has a higher priority 2 ; It resumes

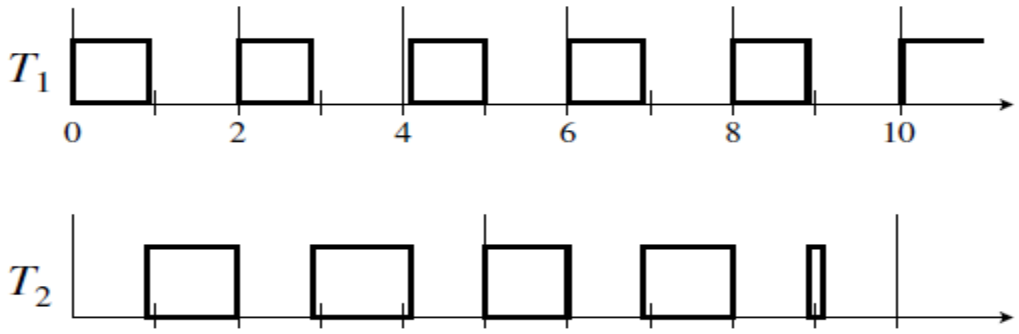
- At 11, when J5 releases black, its priority returns to 5 and the ceiling of the system drops to Ω. J2 becomes unblocked is allocated black and starts to execute
- At time 14 ,after J2 and J3 complete J4 has the processor and is granted the resource shaded because its priority is higher than Ω, the ceiling of the system at the time. It starts to execute. The ceiling of the system is raised to one. The priority ceiling of shaded.
- At time 16, J4 requests black, which is free, the priority of J4 is lower than current priority ceiling Π(16), but J4 is the job holding the shaded resource whose priority ceiling is equal to Π(16) hence J4 is granted black . It continues to execute

Difference between Priority Inheritance and Priority Ceiling Protocols:

- Priority Inheritance protocols are greedy while Priority Ceiling protocols are not The allocation rule of priority inheritance protocol lets the requesting job have a resource whenever the resource is free but in case of priority ceiling protocol, a job may be denied its requested resource even when the resource is free at the time (J4 at time 3 above)
- The priority inheritance rules of these two protocols are the same
- In principle, both rules say that whenever a lower priority job Jl blocks the job J whose request is just denied, the priority of Jl is raised to J"s priority π(t)
- The difference arises because of the non-greedy nature of the priority ceiling protocol
- If is possible for J to be blocked by a lower priority job which does not hold the requested resource according to the priority ceiling protocol while this is impossible according to the priority inheritance protocol.
- The set of priority ceiling of resources impose a linear order on all the resources, so deadlock can never occur under priority ceiling protocol.
- The priority celling protocol gives the good performance but defining rules are complex.
- Priority celling protocol falls under high context swith overhead due to frequent switching as compared with inheritance protocol.

Use of Priority Ceiling Protocol in Dynamic Priority Systems:

In a dynamic priority system the priorities of the periodic tasks change with time while the resources required by each task remain constant. Hence, the priority ceilings of the resources may change with time. Consider a system with two tasks Tasks T1 (2, 0.9), T2 (5, 2.3) executed in deadline driven system as below.



In its first two periods from time 0 to 4. Tl has priority 1 while T2 has priority 2, but from time 4 to 5 T2 has priority 1 and Tl has priority 2. Suppose that the task Tl requires a resource X while T2 does not. The priority ceiling of X is 1 from time 0 to 4 and becomes 2 from time 4 to 5 and so on .For dynamic systems, we can use the priority ceiling protocol to control resource accesses provided we update the priority ceiling of each resource and the ceiling of the system each time task priorities change.

Stack based Priority Ceiling Protocol:

- So far we have assumed that each job has its own run time stack
- In systems where the number of jobs is large, it may be necessary for the jobs to share ta common run time stack, in order to reduce overall memory demand.
- Space in the shared stack is allocated to jobs contiguously in the last in first out manner.
- When a job J executes, its stack space is on the top of the stack
- The space is freed when the job completes.
- When J is preempted the preempting job has the stack space above J‟s.
- J can resume execution only after all the jobs holding stack space above its space complete, free their stack spaces and leave J‟s stack space on the top of the stack again.
- According to this scheduling rule, when a job begins to execute all the resource it will ever need during its execution are free.
- No job is ever blocked once its execution begins.
- When a job J is preempted, all the resources the preempting job will require are free, ensuring that the preempting job can always complete so J can resume.
- Hence, deadlock can never occur**.**

Controlling Access to Multiple unit Resources:

- The priority ceiling protocols are extended so they can deal with multiple unit resources.
- The priority ceiling $\Pi(R,k)$ of a resource R that has $v \leq 1$ units when k ($v \geq k \geq 0$) are available is the highest priority of all jobs that require more than k units of R The system ceiling at any time is equal to the highest of priority ceilings of all resources in the system.
- Except for this modification the multiple unit priority ceiling protocol is the same as the basic version

Controlling Concurrent Access to Data Objects:

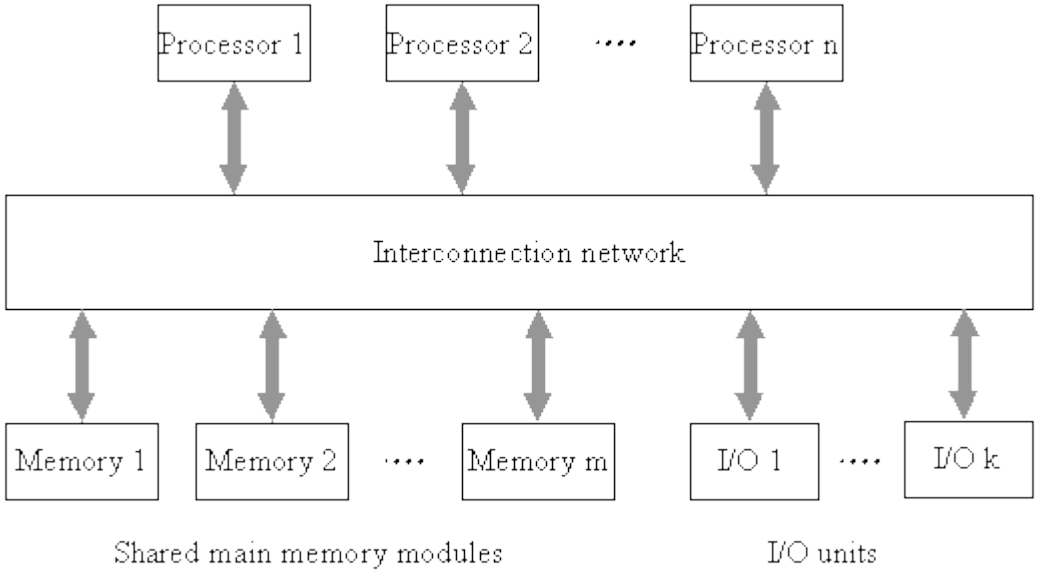
- Data objects are a special type of shared resource.
- When jobs are scheduled preemptively, their accesses to data object may be interleaved.
- To ensure data integrity it is common to require that reads and writes be serializable.
- A sequence of reads and writes by a set of jobs is serializable if the effect produced by the sequence on all the data objects shared by the jobs is the same as the effect produced by a serial sequence.
- Two phase locking protocol is used to access the data objects. It is :
  - ⇨ A way to ensure serializability
  - ⇨ A job never requests any lock once it releases some lock

Chapter-9

Multiprocessor Scheduling Resource Access Control and Synchronization

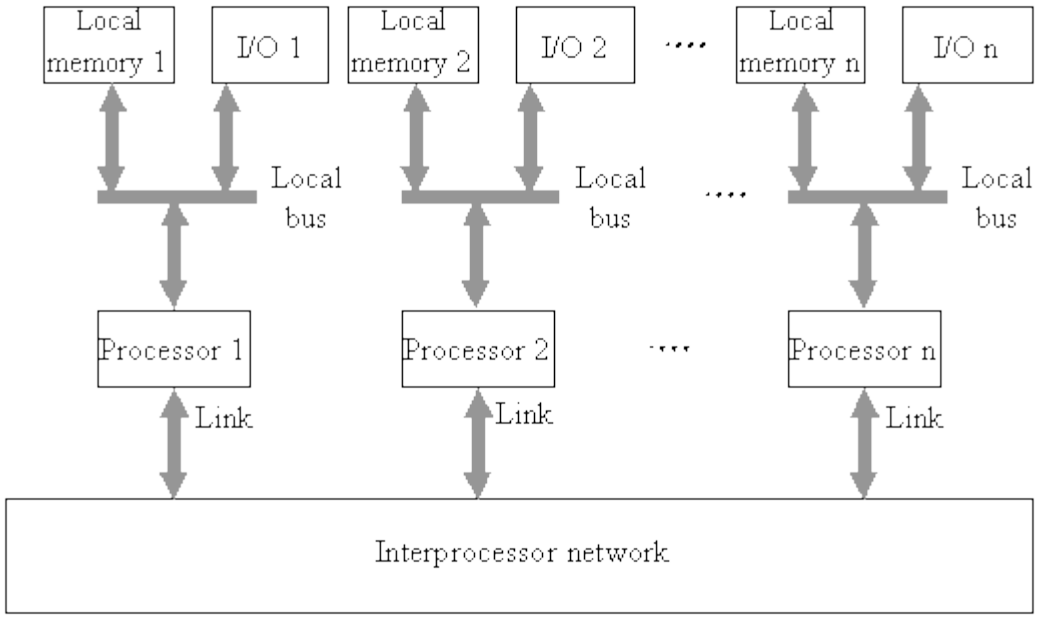
Model of Multiprocessor and Distributed System

Multiprocessor system is tightly coupled system where global status and workflow information on all processor can be kept at a low cost. These systems may use centralized scheduler or each processor may have its own scheduler. Fig below shows a typical multiprocessor system.



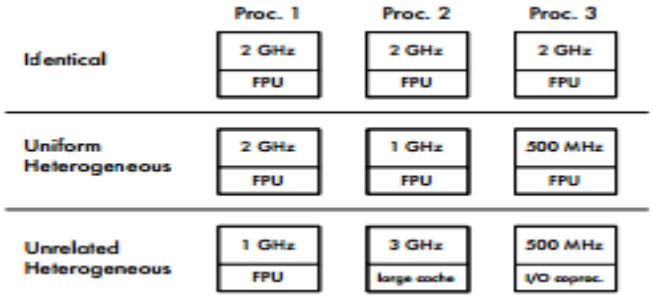
A distributed system is loosely coupled system where global status and workflow information update process is costly due to communication cost. Here each processor of the system has its own scheduler.

Fig below shows the typical distributed system.



Identical vs. Heterogeneous Processors:

Identical processors can be used interchangeably such as each core of multi-core CPU. Heterogeneous processors cannot be used interchangeably such as disk drive, transmission line etc. fig below shows a typical processor organization of the identical as well as heterogeneous processors.



- If all processors of system have equal speed and capabilitie then they are called as identical processor.
- If all processors have equal capabilities but different speeds then they are called as uniform heterogeneous (or homogenous) multiprocessor.

If no regular relation assumed and tasks may not be able to execute on all processors then such system is called unrelated heterogenous multiprocessors.

Problems Associated with Resource Assignment in Multiprocessor Scheduling:

The major problrm in resource assignment in multiprocessor system is the task assignment problems. The task assignment problem is related with following two problems.

- The hard real time system are concern with the static system in which the task are prtitoned and statistically bound to the processor. The problem is how to partition the system of task and passive resources into modules and how to assign the modules to individuals processors.
- Another problem is related with the interprocessor communication and synchronization . A synchronization protocol is required to maintan precedence constraints of jobs on different processors.

Job shops and Flow shops:

According to job shop model, each task T_i in the system is a chain of n(i) jobs, denoted by $J_{i,k}$ for k = 1,2…..n(i). Adjacent jobs $J_{i,k}$ and $J_{i,k+1}$ on the chain execute on different processors. $J_{i,k+1}$ becomes ready for execution when $J_{i,k}$ completes its execution. Processor are specified for each job to run by the visit sequence $V_i = (v_{i,1}, v_{i,2}, \ldots\ldots v_{i,n(i)})$. Visit sequence $V_1 = (p_1, p_2)$ denote the job sequence has two jobs and first job execute on first processor and second job on second processor.

A flow shop is a special job shop which has identical visit sequence.

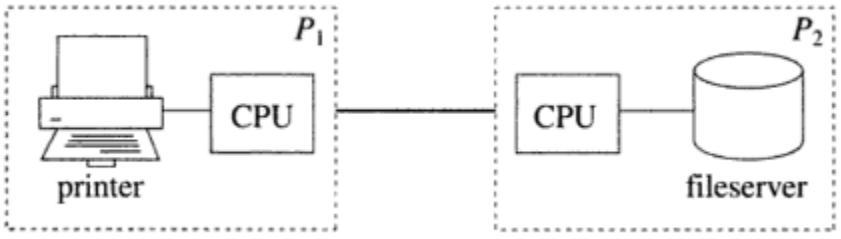
End to End Task:

Let the release time r_i of task T_i be the release time of first job $J_{i,1}$ in the task and the deadline d_i of the task is the deadline of its last job $J_{i,n(i)}$. As long as the last job completes by the task deadline, it is not important when other jobs of the task complete then such model is known as end to end task model.

Periodic End to End tasks:

An end to end task T_i is periodic with period p_i if the chain of n(i) jobs is released every p_i (or more) and the jobs in the chain execute in turn on processors according to visit sequence $V_i = (v_{i,1}, v_{i,2}, \ldots\ldots v_{i,n(i)})$.

Local versus Remote Resource(MPCP Resource Model):



Here **P_1** is synchronization processor of resource printer while **P_2** is the synchronization of resource file server. Printer is local resource of J1 and J2 while fileserver is remote resource of J1. File server is global resource as it is needed by multiple jobs residing on different processor. When

a job access global resource its global critical section executes on synchronization processor of that resource.

End to End resource model:

No jobs can make nested request for the resources residing on different processors.

J1 on P1 and P_1 → R1, P_2 → R_2

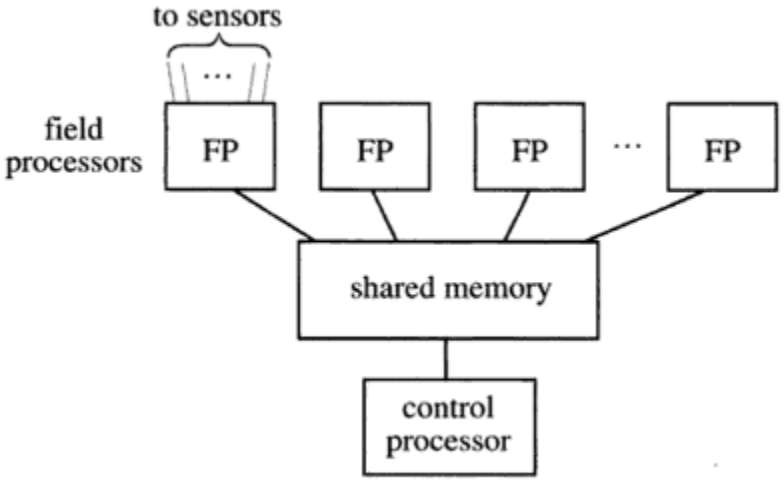
L(R_1) L(R_2) U(R_2) U(R_1) → not allowed

L(R_1) U(R_1) L(R_2) U(R_2) → allowed

Inter-processor Communication:

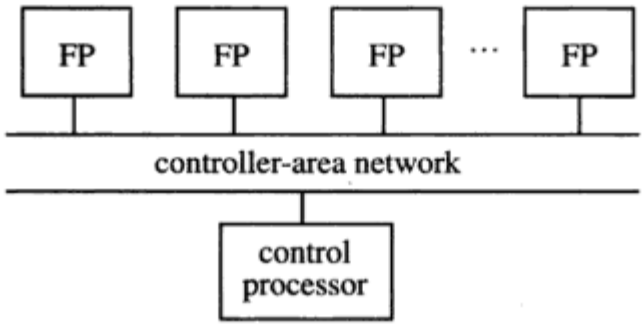
Inter processor communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle many user requests at the same time. Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other. The communication between two or more processor in a multi-processor organization to share the information/data is called inter-processor communication.

By using different algorithms, we can partitionton an application system into the compnents called modules and assigns the modules to processor. These algorithms make the use of information provided by interconnection parameters of jobs like volume of shared data stored on memory exchanged beteen each pair of jobs. The time reqire to synchronize jobs and transmit the data among them comes into account in several ways as inter-processor communication.



(a)

Fig(a). realtime monitor system that consists many field processor. Those jobs that collects and process the sensor data are called producer jobs since they generates the data. Those jobs that correlate and displays the data by executing on central processor are called consumer jobs. The jobs are communicated via. shared memory. Here each field processors are connected with shared memory via. a dedicated link that is shared by all field processors. The system contains three types of processors as field processor, shared-memory, control processor. The workload consists of end to end jobs, each job containing memory accessing jobs. In this way , delay in completion of each job caused by memory contention is taken into account. Hence shared memory cal be modeled as global resource whose synchronization processor is shared memory processor and both consumer and produceser requires this job.



(b)

Fig(b). shows a model in which all the producer are connected via. a network as controller area network(CAN). The result produced by each job on field processor are sent via network to control processor. Here network can be modeled as processor on which message transmission jobs are scheduled on a fixed priority basis. There is no need to account the interprocessor communication cost because they are taken into account by message transmission jobs in network.

Task Assignment:

In hard Real Time System, application system is partitioned into modules and each module is assigned to a processor, and most module assignment in done offline. Assignment is based on execution time requirement of job as the communication cost is minimal as well as minimization of communication cost . Communication cost of individual task is independent of where the task execute. In multiprocessor system task of same module may execute on different processors, the communication cost of these tasks is negligible due to shared memory architecture. During the early designed stage, we want to ignore it but it cost be depens on the system where task is executed.

Simple Bin Packing Formulation/ Algorithms:

- The task assignment problem can be generalized using a simple bin packing problem.
- There can be more than on job.
- The server utilization should not be greater be than 1 but if there are more than one job, the utilization can be greater than one. Therefore, all job cannot execute on a single processor.
- The bin packing formulation is used to find optimal number of processor to execute all jobs to execute all jobs with total utilization less than 1.
- The binpacking algorithms can use two approaches
    a. Uniform size bin packing
    b. Variable size bin packing
- The variable size bin packing is the efficient method because it use the first fit method to select and execute the job on single processor.
- The job if single can be partitioned into different modules and by using bin packing formulation, an appropriate processor can be selected to execute each modules such that total utilization becomes less than 1.

Suppose a system with a 7 modules with utilization as 0.2 0.5 0.4 0.6 0.1 0.3 0.8 and bin size 0.9 then

Total utilization = 0.2 + 0.5 + 0.4 + 0.6 + 0.1 + 0.3 + 0.8 = 2.9

Different approaches like next fit, first fit and best fit are used to formulate the bin packing.

FirstFit

Algorithms:

- Keep a single open bin
- When an item does not fit, close this bin and open a new bin where closed bins are never used again.
- In this method the bin packing be done as below.

Bin1	Bin2	Bin3	Bin4	Bin5
0.2	0.4	0.6	0.3	0.8
0.5		0.1		

Next Fit

Algorithms:

- Keep all bins open.
- An item is assigned to the first bin in which it fits.
- Check all previous bins to see if the next item will fit.
- If item does not fit in any open bin, open a new bin.

Bin1	Bin2	Bin3	Bin4
0.2	0.4	0.6	0.8
0.5	0.3		
0.1			

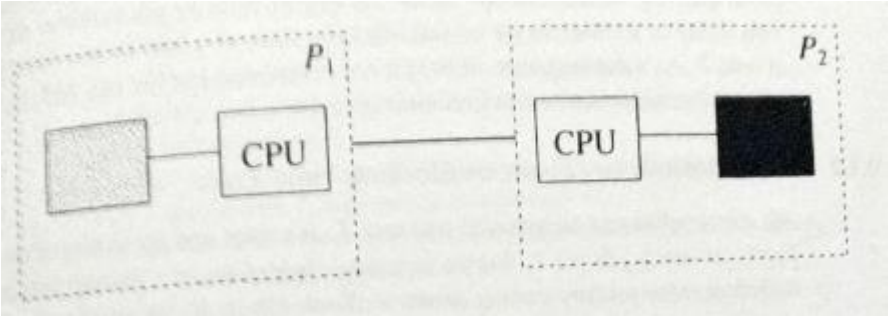
Multiprocessor Priority Ceiling Protocol:

MPCP assumes that task and resources have been assigned and statically bound to process and scheduler of any synchronization processor know the priorities and resource requirement of all task requiring the global resources managed by the processor.

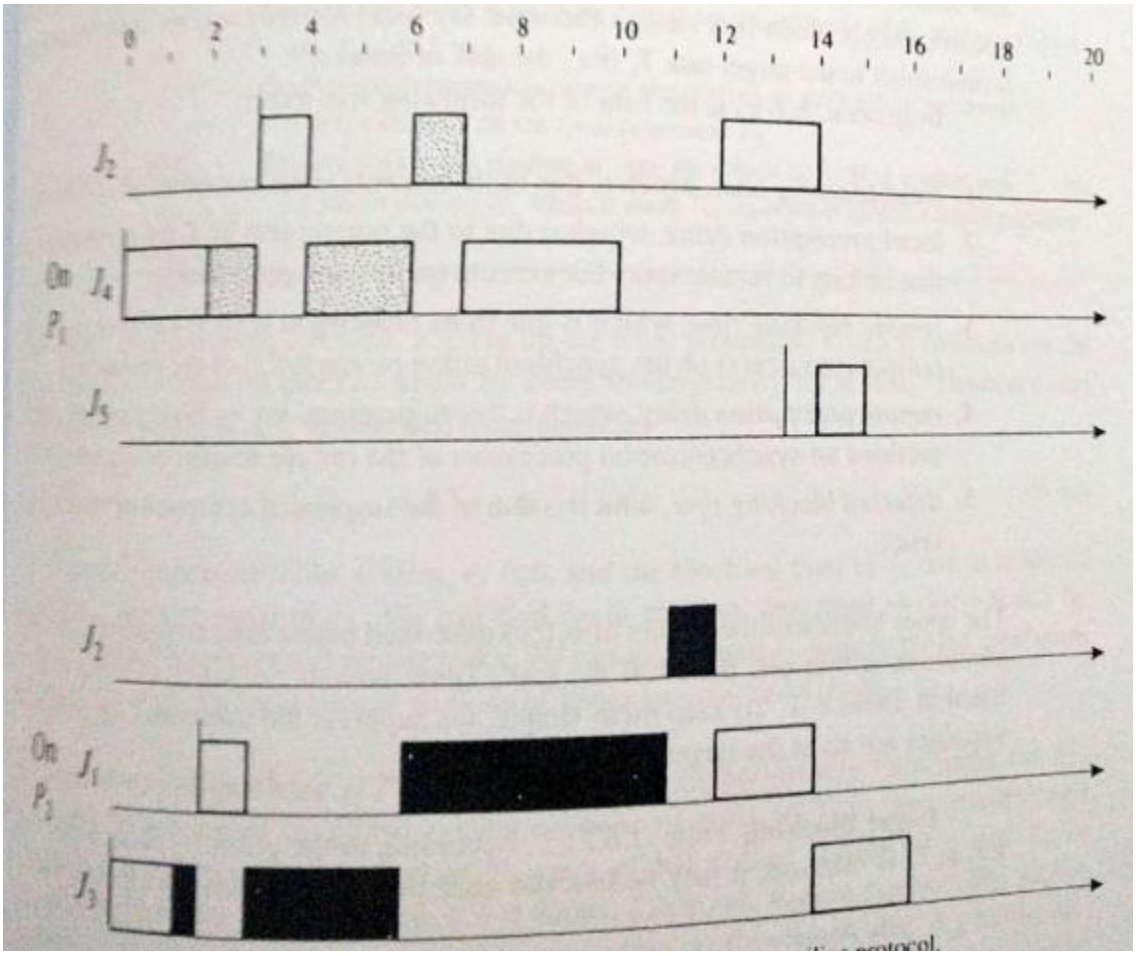
According to this protocol, the scheduler of each processor schedules all local task as well as global critical section on a fixed priority basis. It also controls the resources excess according to priority ceiling protocol. When a task uses a global resource, its global critical section executes on its synchronization processor of resource. If global critical section task has lower priority than local task on synchronization processor then local task may delay the completion of global critical section and increases the blocking time of remote task. To avoid this, MPCP schedules all global critical section with higher priorities than local task. If the lower priority is π_{lower} then scheduler schedules the global critical section of a task with priorities π_i at priority ($\pi_i - \pi_{lower}$).

For example in a system of task with priorities 1 – 5 , global critical section of a task with priority 5 is scheduled at priority 0 which is higher than 1.

Consider a system with two processor P1 and P2 as shown in fig below.



The job J2, J4, J5 are local to processor P1 which is synchronization processor for resources dotted(R2). Dotted is the local resource because it is required by local jobs J2 and J4 only. Job J1 and J3 are local to processor P2 which is synchronization processor for the resource black (R1). Balck is the global resource required by J1, J2 and J3 as shown in timing diagram.



In the example, J2 is directly blocked by J4 when J2 requests dotted at time 4 and J1 is directly blocked by J3 at time 3 when it needs black. The execution of global critical section may be delayed by global critical section of other jobs on synchronization processor. For example, the global critical section of job J2 on P2 is delayed by global critical section of higher priority job J1 in time interval 6 to 11. This is called blocking time of J2.

At time 11, J1 exits from global critical section and its priority becomes lower than priority of global section of J2 i.e. priority of J1 is 1 and priority of J2 = 2- 5 = -3 which is higher than J1. As the result J2 preempts J1 on processor in the interval (11, 12]. The total delay suffered by a job due to preemption by global critical section of lower priority job is a considered as blocking time.

A job may be delayed by a local higher priority job whose execution is suspended on the local processor when higher priority jobs executes on a remote processor. This time is also considered as a blocking time in multiprocessor system. For example J2 is suspended at time 7 and it is still not completed in the time interval (13, 14) and J5 released at 13 cannot start until time 14.

Elements of scheduling algorithm for end to end tasks:

Every periodic task that requires resource on more than one processor can be treated as an end to end periodic task. Each job of task contains a chain of component jobs which execute in sequence on different processors. Suppose, task T_i has n(i) subtasks $T_{i,k}$ for k = 1, 2, …., n(i). These subtasks execute in turn in different processors to its visit sequence $V_i = (v_{i,1}, v_{i,2}, \ldots\ldots v_{i,n(i)})$ where $V_{i,k} = P_j$ which means the k^{th} subtask executes on processor P_j. The two essential components to the end to end scheduling schemes are 1) protocols for synchronizing the execution of sibling subtasks on different processor. So that precedence constraint among subtasks is maintained. 2) Algorithms for scheduling subtasks on each processor.

Inter-process Synchronization Protocol:

Greedy Synchronization Protocol

It is most commonly used in non-real time systems like in wide communication. Transmission of each newly compressed frame is made ready as soon as compression of frame is completed.

Greedy Synchronization protocol states that when j^{th} job of $T_{i,k}$ completes on $V_{i,k}$, the scheduler of $V_{i,k}$ sends a synchronization signal to the scheduler of $V_{i,k+1}$ on which the successor subtask $T_{i,k+1}$ executes. Upon receiving the synchronization signal, the scheduler of $V_{i,k+1}$ releases the corresponding job of task $T_{i,k+1}$.

Example: T_1 (6, 3) T_3 (6, 9, 4) $T_{2,1}$ (9, 3) and $T_{2,2}$ (9, 3)

These tasks run on two processors P_1 and P_2 where T_1 and $T_{2,1}$ run on P_1, $T_{2,2}$ and T_3 on P_2. T_1 has highest priority on P_1 and $T_{2,2}$ has highest priority in P_2.

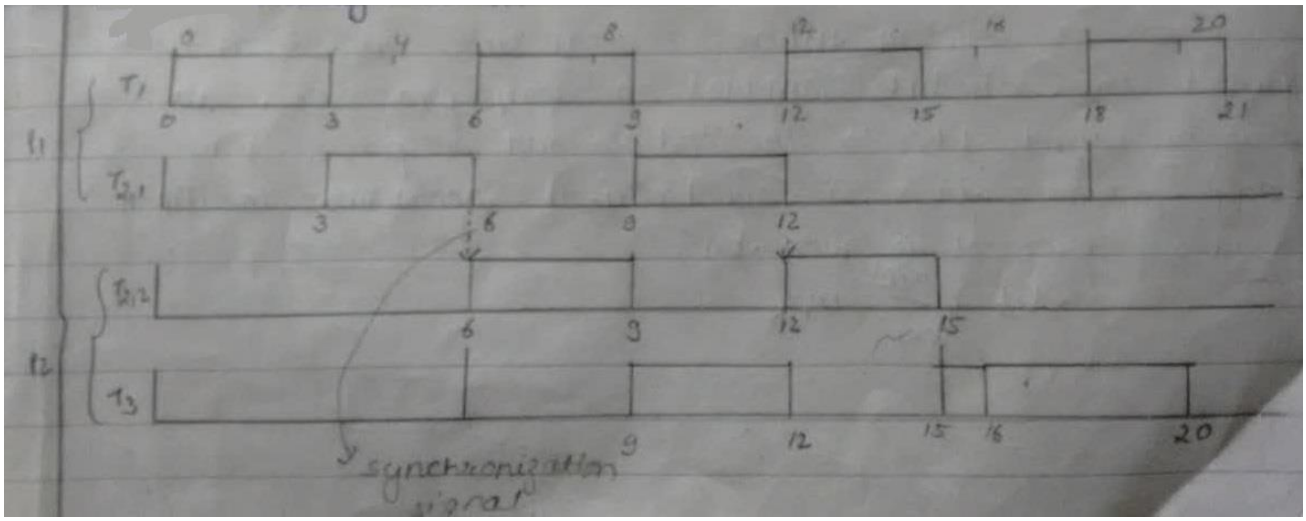


Figure: Illustration of Greedy Synchronization Protocol

Greedy synchronization protocol decreases the response time of end to end periodic task in comparison to non-greedy method. In the example above, the inter-release time of task T_2 is 6 while its period is 9 due to which T_3 misses its deadline.

End to End Task in Heterogeneous System

A simple embedded system contains different types of processors (CPU, disks, and networks). The scheduling algorithm used on each processor keeps the response time to a bound while satisfying the design constraints of the algorithm run by them. The bound is generated by applying non-greedy inter-processor synchronization. The modified phase modification protocol (MPM) and release guard protocol (RG) are such synchronization protocol. These protocols reshapes the release time of subtasks in different processors truly periodic.

Chapter-10

Real Time Communication

Introduction:

Real-time communication is a means of sharing information and interacting with people through network connections just as if they were face-to-face. Digging into the technicalities a little deeper, it's any live (real-time) telecommunications that doesn't have transmission delays – it's usually a peer-to-peer connection with minimal latency, and data from a real-time communications application is sent in a direct path between the source and destination.

Examples of Real-Time Communications

There's a difference between emailing and chatting with someone. Email is more of a time shifting form of communication – we send emails and expect to hear back from people later, and data is stored between the source and destination. Communicating through methods like email place more emphasis on delivering information reliably, not how quickly the information gets there. When chatting with someone, however, we expect responses just as if we were communicating face-to-face: in real-time. Other examples beyond instant messaging of real-time communications include:

- Video conferencing
- Presence (usually found in UC applications)
- Gaming
- File sharing
- Screen sharing
- Collaboration tools
- Machine to machine technology
- Location tracking
- Online education
- Social networking

Real-time communications applications and solutions can be used in virtually every industry: contact centers, financial services, legal firms, healthcare, education and retail can all benefit and improve processes with real-time communications applications. There are a few trends in play that are helping drive the growth of real-time communications applications.

Model of Real Time Communication:

In the model of the real time communication, end users of the message application systems as source and destination residing in different host. The network interface of each host contains input queue and output queue. Two buffer area called as input / output buffer are allocated to input and output queue to store queuing information. The queue are jointly maintained by two local servers as Transport Protocol Handler (TPH) and Network Access Control Handler (NACH). The former interfaces with local application and provides them with message transport service. The next interface with the network below and provides network access and message transmission services

to the TPH. The client- server architecture produce more delay such that it is not suitable communication network architecture for real time communication.

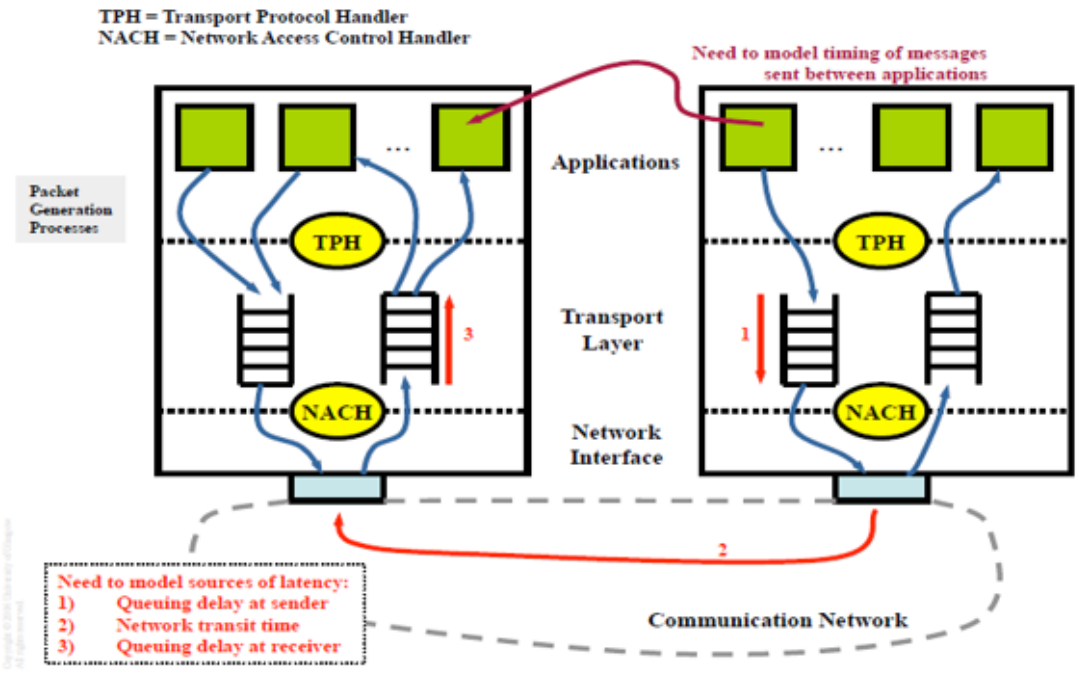


Fig: a real time communication model

The fig. above shows the data path follow to transfer the message in between two hosts. Two marked circle TPH and NACH are transport protocol handler and network access control handler. When requested to send message by a local application task, the source TPH places the massage in output queue. Each outgoing message is delivered to network under the control of source NACH. After the placement of message on network, the NACH of destination host place it on input queue of destination and notify the destination IPH. The destination IPH then moves the message to address space of the destination application task and notify the destination task of arrival of message.

Here the end to end model by chain of jobs is used to represent the message sending activity. The application task available on source and destination are modeled as the predecessor as well as successor of the chain. At the beginning and end of the chain are source and destination chain are the transport protocol processing job. In between them, each job that access the network or transmits the message becomes ready for execution after its predecessor completes. Ideally the network delivers messages to receiver with no delay. In reality there is some of the delay are associated with queuing delay at sender, network transmit time and queuing delay at receiver as well as network.

- Network is not always ready to accept a packet when it becomes available and data may be queued if produced faster than the network can deliver it such that Queuing delay arise at sender.

- Application task are not always ready to accept packets arriving from network and Network may deliver data in bursts form such that Queuing delay occurs at receiver.
- Due to cross-traffic or bottleneck links such that Queuing delay occurs in the network
- Network transit time also generates the delay.

Hence a synchronization protocol is needed to model the real time communication model.

Real Time Traffic Model:

The real time traffic means isochronous or synchronous traffic, consisting stream of message that are generated by their sources and delivered to their respective destination on continuous basis. The traffic includes the periodic, aperiodic and sporadic messages. The periodic and sporadic message are synchronous in nature and there requirement of guarantee of on time delivery whereas aperiodic message are asynchronous in nature and shows the soft timing constraint.

In real time traffic model, each message (M_i) be characterized by tuples of inter-packet spacing (Pi), message length (e_i), reception deadline (Di) as below.

$$M_i = (p_i , e_i , D_i)$$

This traffic model is called peak rate model in real time communication.

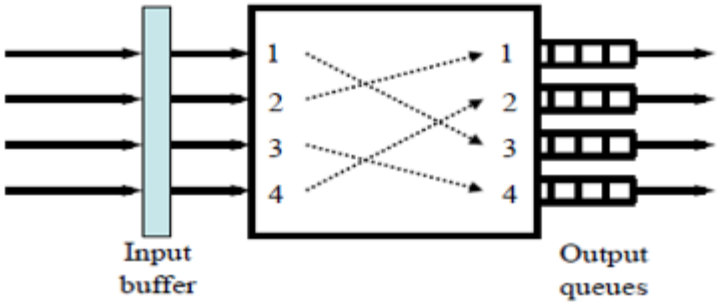
Throughput, Delay and Jitter:

- The throughput is the measure of the numbers of packets or message stream that the network can deliver per unit time.
- The delay (latency) is time taken to deliver the packet or message stream. It is fixed due to minimum propagation delay due to speed of light and varying due to queuing on path.
- The term jitter indicates variance on delay.
- Many real time communication protocol and algorithms are designed to keep not only the worst case delay and jitter as small.

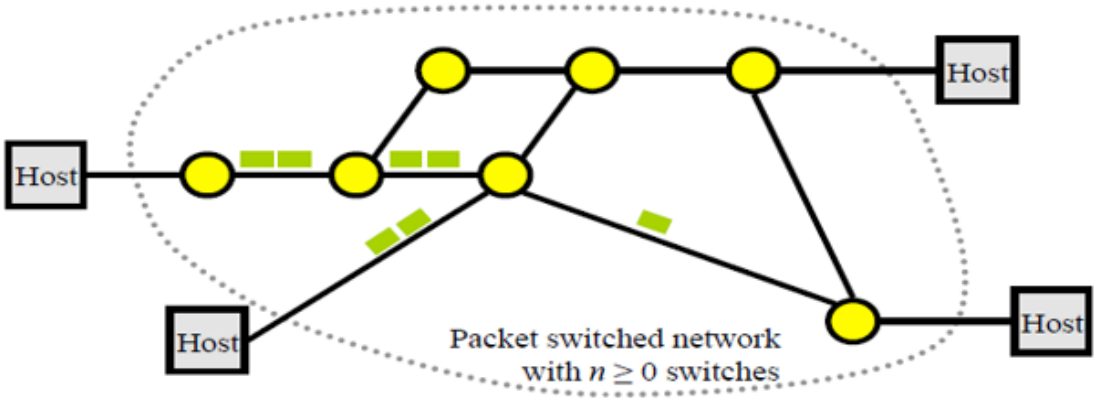
Real Time Connections and Service Disciplines:

The connection oriented approach is used for real time traffic. According to this approach, a simplex logical connection is set up for transmission of each message stream between source and destination. The fixed routing is used to route the packets and chosen route is fixed until torn down of system or invocation of adaptation mechanism. The new connection is established if the existing network meets quality of service parameter like end to end delay, jitter etc. Generally packet switching network is preferred for transmission of message streams.

- Fig below shows the packet switching network along with multi hop switch for multi hop network and circle of the network represents a switch.



- Switches are buffered I.e. there is a buffer pool for each output links, holding the packets that are queued for transmission on the link. Once the switch route the packets to the queue , the packets waits in the queue until schedulaer schedules it for transmission and then it is transmitted to next hop at the other end of the output link.
- The amount of time the switch takes to route the packets is small but the time a packets takes passing trough a swith is equal to its output time at output queue plus packet transmission time and this is called as hop delay o the packet.



- The end to end delay of each packet trough a switched network is equal to the sum of the per hope delays it suffers passing through all switches in the route plus total time it takes to propagate along the all links between the switches.
- The combination of acceptance test and admission control protocol , a synchronization protocol and a scheduling algorithms used for the purpose of rate control (jitter) and scheduling of packets transmission is called a service discipline
- Service disciplines are divided into two categories as rate allocating and rate controlled.
- Rate allocating disciplines allows the packets on each connection to be transmitted at higher rates than guaranteed rate.
- A service disciplines is said to rate controlled if it each connection guaranteed rate but never allows packet to send above guaranteed rate.

Priority – Based Service Disciplines For Switched Network:

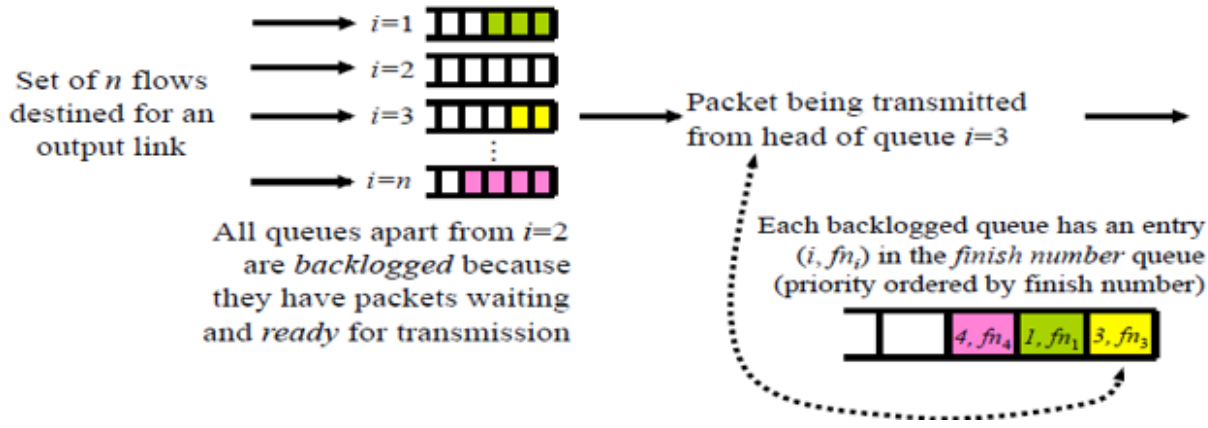
According to a priority based service disciplines, the transmission of ready packets are scheduled in priority driven manner. Waited fair queuing (WFQ) and Waited round robin scheduling are common approach for scheduling the packets in real time communication network.

Waited Fair Queuing (WFQ) Discipline:

It is a rate allocating service discipline and provides each flow with at least its proportional fair share link capacity and isolates the timing between flows such that it is also called as packet by packet generalized processor sharing algorithms. It is define as a packet switch has several inputs, feeding to an output link shared by n established flows where each flows i is allocated to u_i of the link such that total bandwidth allocated to all connections is:

$$U = \sum_{i=1}^{n} ui \quad \text{Where U<=1.}$$

In WFQ, the output buffer comprise of two queues as FIFO and shortest finish number (SFN) for scheduling of packets. FIFO is used to scheduled n flows whereas priority ordered SFN is used to schedule the job obtained from the head FIFO queue on the basis of finish number. The finish number specifies the number of ready packets for transmission. The structure of queue is shown below.



- A packet becomes ready on FIFO queue, a finish number is calculated and SFN queue updated. The newly arrived packet is placed on the head of SFN queue without preempting the current transmission. When packet completes its transmission then it is removed from head of SFN and FIFO queue.

Waited Round Robin Scheduling:

In this approach jobs are placed in FIFO queue. The job at the head of queue executes for one time slice. If it doesnot complete with in the time slice, it is preempted and put back into the queue. There are n jobs in the queue, each job gets one slice every n time slots in a round. The weighted round robin schedule extends this to give each job with weight Wi such that ith job gets Wi lenth time slice for execution. In this method each packets obtained from set of n flows are arranged in

FIFO order and gets the connection for Wi time after connection and preempted and backlogged to the end of the queue if it is not completely transmitted otherwise removed from the queue. This service disciplines is shown below.

When in each round, if more than Wi packets are backlogged on queue i then Wi packets are transmitted such that

- Each flow is guranted Wi slots in each round
- Rate allocating may send more , if nothing to transmit

Then such WRR scheduling scheme is called Greedy WRR scheduling only when there must be a design parameter (RL) satisfies the following conditions.

- At all times $\sum_{i=1}^{n} wt_i \leq RL$
- Each flow is guaranteed a share wt_i/RL of the link capacity
- Provided that:
  - $RL < p_{min}$ (where p_{min} is minimum p_i over all i)
  - $wt_i \geq e_i/(p_i/RL)$ (with appropriate rounding)

Medium Access Control Protocols of Broadcast Networks:

- The transmission medium of the broadcast network is the processor.
- A MAC protocol is a discipline for scheduling this type of processor.
- Scheduling of the transmission medium is done distributedly by network interfaces of hosts in the network.

Medium Access Protocol in CAN (Controller Area Network)

- Controller area network are very small network. CANs are used to connect components of embedded controllers. An example is an automotive control system whose components control the engine, brake, and other parts of an automobile.
- The end to end length of CAN must not exceed 100 meters. This means that within the fraction of a bit time after a station starts to transmit, all stations on the network can hear the transmission.
- The output of all stations are wire-ANDed together by the bus i.e. the bit on the network during bit time is a logical 0 if the output of any station is 0 and logical 1 when the output of all stations is 1.

- CAN MAC protocol is similar to the CSMA/CD (carrier sense Multiple Access/ Collision Detection)
- A station with a packet to send waits until it hears that the network is idle and then commences to transmit the ID number of the packet. At the same time, the station listens.
- Whenever it hears a 0 on the network while it is transmitting 1, it interrupts its own transmission.
- Network connection is resolved in favor of the packet with the smallest ID among all contending packets.

MAC in IEEE 802.5 Token Ring:

In a token ring network, packets are transmitted in one direction along a circular transmission medium. A station transmits a packet by placing its packet on the output link to the network. As the packet circulates around the network, the stations identified by the destination address in the header copies the packet. When the packet returns to the source station, the station removes the packet.

Prioritized Access in IEEE 802.5 Token Ring:

Polling

Network contention is resolved by a polling mechanism called token passing. For the purpose of polling, each packet has in its header an 8-bit Access Control (AC) field. One of the bits in an AC field is called the token bit. By examining this bit in the current packet on the network, a station can determine whether the network is busy. If the network is free the packet is polling packet. As a polling packet circulates around the ring, the stations are polled in a round robin manner in order of physical locations on the ring. When a free token reaches a station that has outgoing packets waiting, it can seize packets if it has the highest priority at that time.

Priority Scheduling:

Prioritized access is made possible by using the two groups of 3 bits each in the AC field: Their values represent the token priority πT and the reservation priority πR. Token priority bits give the priority of the token. A station can seize the free token only when its outgoing packet has an equal or higher priority than the token priority πT.

Reservation bits in the outgoing packets is used to make reservation for future use of the network. When the station seizes the token, it leaves the token priority unchanged but sets the reservation priority to the lowest priority of the network. It then marks the token busy and puts the token in the header of the packet and transmits the packet.

When a source station removes its own packet from the network, it saves the reservation priority carried in the packet. Suppose that when the source station transmit a free token, it sets the token priority of the token to this reservation priority or the highest of its outgoing packets, whichever is higher. In this case the priority arbitration mechanism allows the stations to jointly carry out any fixed scheduling algorithm.

Schedulability Analysis:

The amount of time (execution time) each packet occupies the network is equal to its transmission time plus the round trip delay it takes to return to the source station. The delay is usually on the order of 10-2 or less of the packet transmission time. In addition to this following three factors should be taken account for:

- Context switching: A context switch time is equal to the amount of time required to transmit a free token, plus the round trip delay of the network, which is an upper bound of the time the token takes whose outgoing packets has the highest priority among all outgoing packets during the transmission of the latest data packet.
- Blocking: Since packets are transmitted non-preemptively, we also need to take account the blocking time due to nonpreemptivity. Moreover, a higher priority packet that arrives at the station just after the header of the current data packet passed the station need to wait for a lower priority packet.
- Limited Priority Levels: since the network provides only eight priority levels resulting in schedulability loss.

Internet and Resource Reservation Protocols (see on books)

Issues in Resource Reservation

- Multipoint to Multipoint Communication
- Heterogeneity of Destinations
- Dynamic multicast group membership
- Relation to routing and admission control

Requirements for Multimedia Traffic:

- In order to ensure playback timing and jitter removal timestamps are required.
- In order to ensure presence and order of data a sequence number is required.
- Most of real-time multimedia applications are video conferencing where several clients receive data therefore multicast mode is preferred.
- In order to deal with congestion mechanism for sender notification and change of encoding parameter must be provided.
- In order to display streams generated by different standards the choice of encoding must be provided.
- In order to display audio and videos within a single A/V session mixer are required.
- In order to use high bit rate streams over a low bandwidth network, translator are required.

Why real time data cannot use TCP?

- TCP force the receiver application to wait for transmission in case of packet loss which causes large delay.
- TCP cannot support Multicast.

- TCP congestion mechanism decreases the congestion window when packet loss are detected (slow started). Audio and videos on the other hand have natural rates that cannot be suddenly decreases.
- TCP headers are larger than UDP header (40 bytes for TCP compared to 8 bytes for UDP).
- TCP doesn't contain necessary timestamp and encoding information needed by receiving application.
- TCP doesn't allow packet loss. In A/V however loss of 1-20% is tolerable.

Protocols:

There are several related protocols which support real time traffic over the internet. Some of the important protocols are

RTP (Real Time Protocol): used for real time data transport developed by extending UDP and sits between UDP and application.

RTCP (Real Time Control Protocol): used to exchange the control information between sender and receiver and works conjunction with RTP.

SIP (Session Initiation Protocol): provides the mechanism for establishing calls over IP.

RTSP (Real Time Streaming Protocol): allows user to control display as rewind, pause, forward etc.

RTP (Real Time Protocol):

There was a dilemma weather to implement RTP as a sub layer of transport layer or as a part of application layer. At this point it is common that RTP is implemented as application library, which executes in user space rather in kernel space like all protocol layers bellow RTP. RTP doesn't ensure the real time delivery itself but it provides the means for

- Jitter elimination / reduction
- Synchronization of several audio or video streams that belong to same multimedia session.
- Multiplexing of audio/video streams that belong to different session.
- Translation of audio/video streams from one encoding type to another.
- With the help of RTCP, RTP also provides hooks for adding reliability and flow/congestion control which is implemented within multimedia application. This property sometimes called as application level framing.
- RTP is a protocol that provides the basic transport layer for real time application but doesn't provide any mechanism for error and flow control , congeston control, quality feedback and synchronization. For that purpose RTCP is added as a companion to RTP to provide end to end monitoring, data delivery and QOS.

RTCP (Real Time Control Protocol):

It is responsible for three functions.

- Feedback on performance of application and network.

- Correlation and synchronization of different media streams generated by same sender for example combined audio/video.
- The way to convey the identify sender for display on a user interface.

The volume of RTCP traffic may exceeds the RTP traffic during a conference session involving larger number of participates. Normally only on participant talk at a time while other participants are listing. In mean while RTCO message are not periodically regardless if the participant is taking or not. Therefore the RTCP packet transmission are done dynamically on the basis of participants.

Standard dictates that 20% of the session bandwidth is allocated to RTCP. In other words RTCP RR SDES packets are sent every 5 RTP packets transmitted.

In addition, 5% of the RTCP bandwidth is allocated to particular participants (CNAME). The RTP transmission in the interval of participants is the function of total number of participants in the RTP session that ensures 5% of bandwidth allocation. For that purpose each participants has to continuously estimate the session size. The transmission interval is randomized to avoid synchronization effects.

- RTCP message are stackable.
- To amortize header overhead multiple RTCP message can be combined and send in a compound RTCP message.

A packet is loss if:

- Packet never arrived
- Packet arrived but corrupted.
- Packet arrived after its play out time.