Unit 1: Introduction

A. Digital Controls
B. High-level Controls
C. Signal Processing
D. Real time application

1.1 Digital Control:

In order to form digital control, there should be incorporation of different concepts of theories, controller, signaling system and conversion strategies.
Digital Control consists of following sub-control concepts:

- Control Theory
- Digit computer to act as system controller
- Discrete system
- A/D conversion
- D/A conversion

The application of digital control can readily be understood in the use of feedback circuit. Since the creation of first digital computer in 1940s the price of computer has dropped considerably, which has made them key prices to control system for several reasons:

a. Inexpensive
   The price is normally under $5 for micro controller, which has made us to use it.
b. Flexible
   With the introduction of digital control and flexibility to use and configure, it has made to create the situation to drop its prices.
c. Scalable
   There is no burden of memory or extra space even if the programs get changed while using the digital control.
d. Adaptable
   Parameters of the program can be changed at any time which preserves its adaptable features.

Digital Controller Implementation

A digital controller is usually cascaded with the plant in a feedback system. The rest of the system can be digital or analog.
Typically, a digital controller requires the following steps:

a. A/D conversion to convert analog to machine readable (digital) format.
b. D/A conversion to convert digital outputs to foam that can be input to the plant (analog).
c. A system that relates the outputs to he inputs.

The state of a plant is monitored by sensors and can be changed by actuators. The real time (computing) system estimates from the sensor readings the current state of the plant and computes a control output based on the difference between current state and desired state. We call this computation the control law computation. The output terms generated activates the actuators, which brings the plant closer to the desired state.

Sampled Data Systems:

- Long before digital competes became cost effective and widely used, analog controllers were in use.
- The analog version is then transferred to digital version.
- The resultant controller is called sampled data system.
- It samples (reads) and digitals the analog sensor reading periodically and carries out its control law computation.

A simple example:

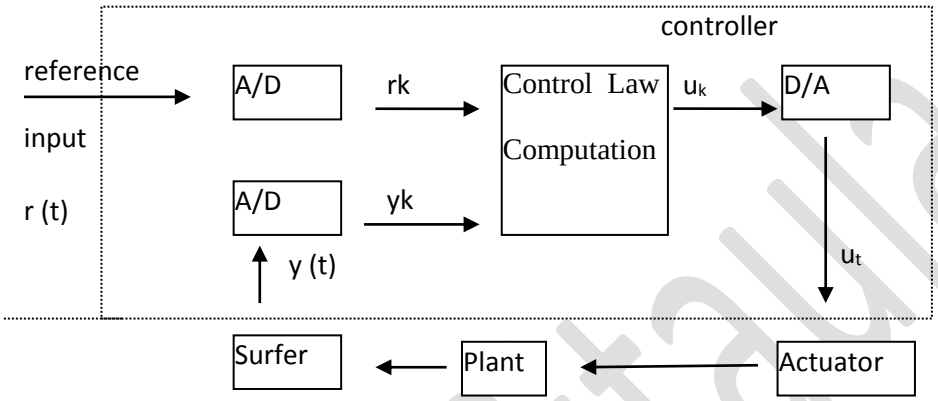


Figure 1.1: A digital controller

We can observe the following things after studying the above diagrams:

→We are given the single input and single output.
→ These are commonly used approaches for digital controller.
→The analog surface reading y (t) gives the measured state of the plant at time it.
→r (t) is the reading the desired state values of the plant. The error value is calculated with the following formulae as given below:

$$e(t) = r(t) - y(t)$$

→The output value u(t) of the controller consists of three terms to be discussed:

i. Firstly, the term is directly proportional to the error value.
ii. Secondly, it is directly proportional to the integration of error value.
iii. And thirdly, it is directly proportional to the differentiation of error values.

→In general, we use trapezoidal rule of numerical integration to transform a continuous integral into discrete integral form. It is same strategy what we used in Numerical Methods.

Algorithms for digital controller

This section is devoted about the steps in order to perform the operations in digital controller.
The basic steps are as follows:

- ✓ Start
- ✓ Set timer to interest periodically with time t
- ✓ At each timer interrupts to do
- ✓ Do analog to digital
- ✓ Compute control output u,
- ✓ Output u and do digital to analog conversion.
- ✓ End do
- ✓ Stop

Selection of Sampling Period

The duration T of time between any two consecutive instants at which y(t) & r(t) are sampled is called sampling period. T is the key design choice.
In making selection, we need to consider two factors.

1. First choice is the perceived responsiveness of the overall system (plant & controller). The operations may issue a command at any time t.
2. Dynamic behavior is the second choice.

Signal Types:

Normally, there are two types of signals- analog and digital signal. If the signal does not show any break point while plotting the coordinate, it is called continuous, whereas if it shows some break points, it is called discrete signals. It can be better explained in the below diagram in a much more refined way.

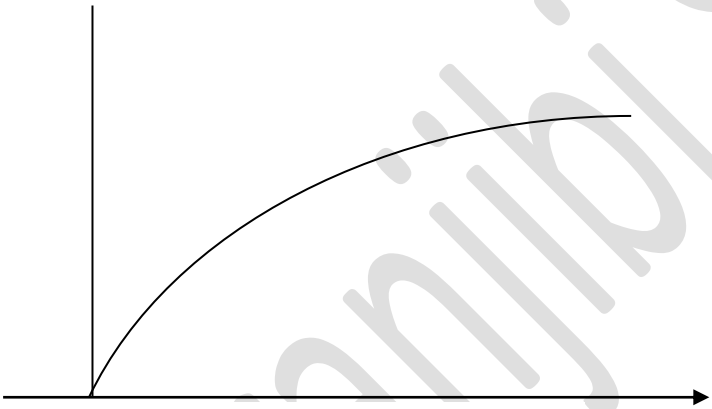


Figure 1.2: Continuous Signal

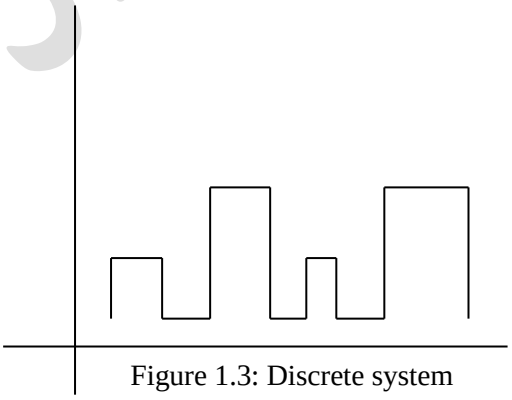


Figure 1.3: Discrete system

1.2 High Level Controls

High level controls means maintaining the hierarchies of different steps in real time system. While designing any real time system, there needs to be arrangements of different upper levels and lower levels for performing different action

- Controller in a complex monitor and control system are typically organized ……
- One or more digital controller at the lowest level controls the physical plant.
- Each output of higher land is input to lower level.
- Exceptionally, one or more higher level controllers interface with the operators.

For better illustrating the high level controls, let's see an example below. It has upper system called patient care system, which remains at the uppermost level. When the user goes to hospital, initially, it will be associated with the patient care system. It is, then, goes to downward where it checks blood pressure, glucose, etc. After checking the health status, the doctors are assigned to the patients depending on the reports.

Patient care system

↓

Microprocessor based controller (blood pressure, respiration, glucose etc)

↓

Doctors interact with operators.

Processing Bandwidth Demands

$$X(k) = \sum_{i=l}^{n} a(k,i)\,y(i)$$

↓ ↓ ↑

Output constant input

Weighted sum of n inputs.

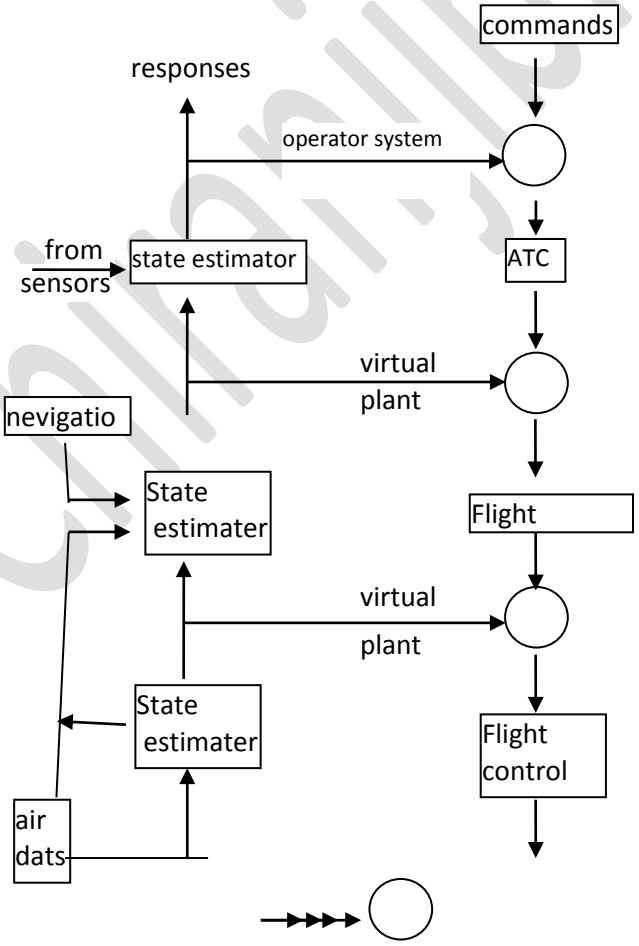


Figure 1.4: Traffic/Flight Control hierarchy

Dead beat Control:

In discrete time control theory, the dead beat control problem consists of finding what input signal must be applied to a system in order to bring the output to the steady state in the smallest number of time steps.

The solution is to apply feedback. Dead beat controllers are after used in process control due to their good dynamic properties. They are a classic feedback controller where the control gains are set using a table based on the plant system order.

The deadbeat response has the following characteristics.

- Very high control signal output.
- Less than 2 % over short/ undershoot.

The ATC system is at the highest level. It regulates the flow of lights to each destination airport. The flight management system chooses a time-refereed 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 input at the lowest level. In general there may be

- ground task
- altitude profile
- cruise speed
- ascent/descent
- cost function
- minimum fuel problems
- time optional algorithms

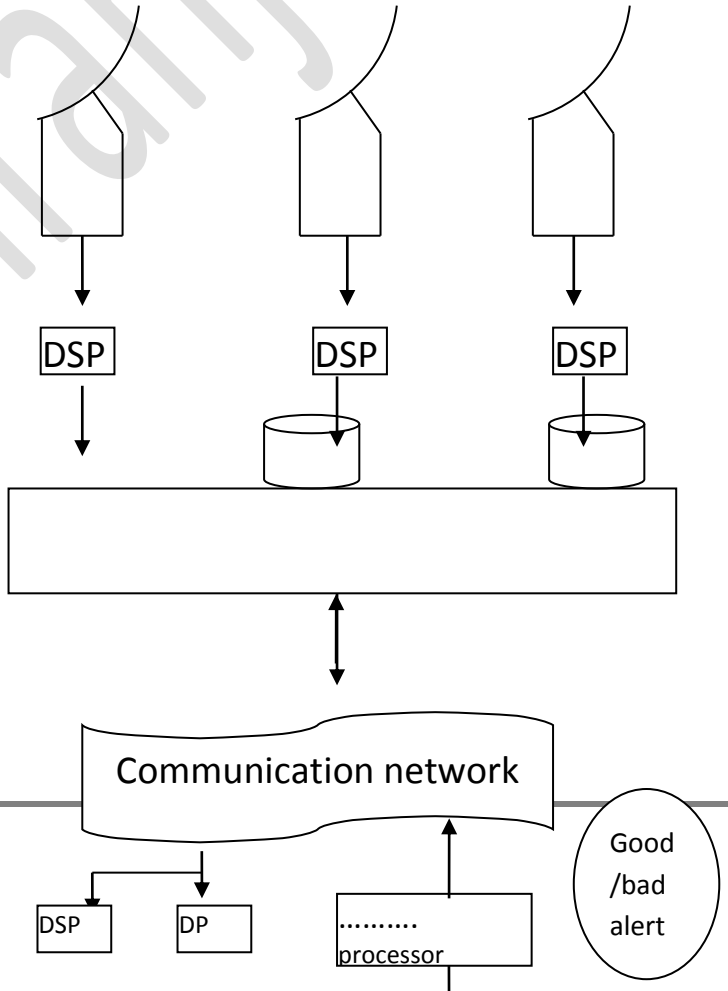


Figure 1.5: Air Traffic Control

High level controls

While the period of a low level control law computation ranges from milliseconds to seconds the period of high level control- law computations may be minutes, even hours.

The (ATC) Air Traffic Control System is at the highest level. It regulates the f low of flights to each destination airport.

1.3 Signal Processing

- Digital filter
- Video/voice compressing decompressing
- Radar signal processing

Digital filter:

In signal processing the function of a filter is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as components lying within certain frequency range.

Figure 1.7 Digital Filters

A digital filter uses a digital process to perform numerical calculates on sampled value of signal. The processor may be a general purpose computer. Such as a PC, or a specialized DSP (Digital signal Processor) chip.

Radar Signal Processing:

If there is an object near the antenna then it defects echo signal reflected by an object at approximately 2x/C seconds
Where, $C = 3 \times 10^8$ m/s
x = distance
Based on the characteristics of Discrete Fourier Transformation, the system can determine position and velocity of an object and place the records in shared memory.

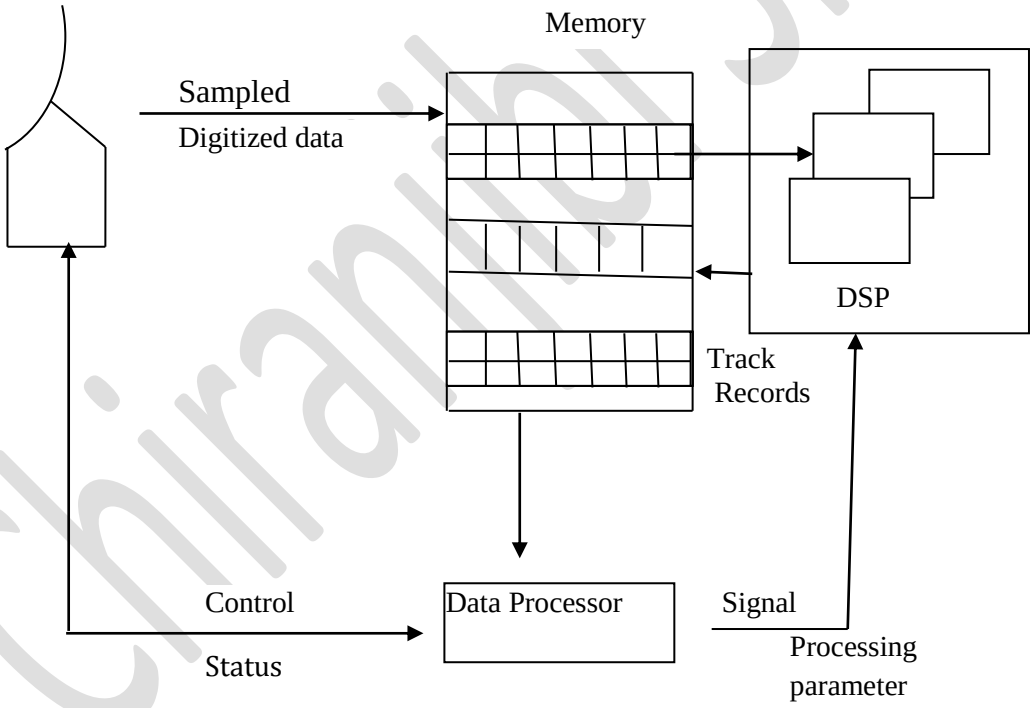


Figure 1.6: Radar Signal Processing & Tracking System

Tracking:

→ Strong noise & man made interferences, including electronic counter measure can lead he signal processing & detection process to wrong conclusion about the presence of object.

→ It wrong output is called false return.
→ Tracker detects false return & soul out.
Tracking is carried out in two steps:

a. Gating
b. Data Acquisition

a. Gating:

→ It is the process of putting each measured value into one of the categories depending on whether it can or cannot be tentatively assigned to one or more established trajectories.
→ The threshold G is called the truck gate.

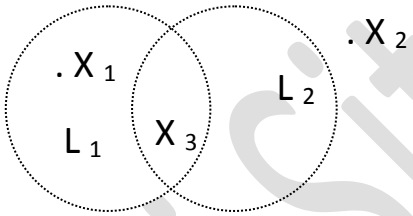


Figure 1.8 Gating

At the start, the tracker computes the predicted position (& velocity) or each object on each established trajectory.
In this example, there are two established trajectories. , L_1& L_2
X_1, X_2, & X_3 are three measured values given by three track records.
X_1 is assigned to L_1 it is within distance G from L_1
X_3 is assigned to both L_1 & L_2
On the other hand X_2 is no assigned to any of the trajectories.

b. Data Acquisition:

The data acquisition step is then carried to complete the assignments & resolve ambiguities.
There are many data association algorithms; one of the most intuitive is the nearest neighbor algorithms.
$X_1 \rightarrow L_1$
$X_3 \rightarrow L_2$
X_2 is removed from the trajectories.

1.4 Real time applications:

a. Real Time Database
b. Multimedia application

a. <u>Real Time database</u>: The term real time database refers to a diverse spectrum of Information systems, ranging from stock price quotation system, to track records database, to real time file systems.

- ❖ Perishable nature
- ❖ Image objects

b. <u>Multimedia application :</u> Frequency encountered real time applications: multimedia

MPEG compression/Decompression:

a. Motion estimations(blocking)

b. Discrete cosine Transform & Encoding
c. Decompression : (inverse transform)

Consistency models:

- Concurrency control mechanism for real time data.

Absolute Temporal Consistency:

A Set of data objects is said to be absolutely (temporally) consistent if the more age of object is no greater than certain threshold.

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 larger than relative consistent threshold.

Questions:

- ✓ Explain the operations of digital controller in detail.
- ✓ What do you mean by digital control? Explain in detail.
- ✓ What do you mean by High Level Controls?
- ✓ How do you define deadbeat control? Simulate it with the help of Air Traffic Systems?
- ✓ Draw the High Level Controls for Traffic or Flight Control Hierarchy.
- ✓ What do you know about Signal Processing? Explain with Radar Signal Processing.
- ✓ What are the application areas of Real Time Systems?
- ✓ What do you know about Tracking? Explain about its strategies.[T.U]

Unit 2: Hard versus Soft Real Time Systems

A. Jobs and processors
B. Release times
C. Deadlines and timing constraints
D. Hard and soft timing constraints
E. Hard real-time systems
F. Soft real-time systems

Jobs & Processors:-

We call each unit of work that is scheduled and executed by the system a job and a set of related job which jointly provide some system a task.

Hence, the computation of a control law is a job. So, the computation of a FFT (Fast Fourier Transform) of sensor data, or the transmission of a data Packet, or the retrieval of a file, and 80 %. We call them a control-law computation.

Every job executes on some resource. For example, the jobs mentioned above execute on a CPU, a network and a disk respectively.

These resources are called servers in queuing theory. Literature and sometimes, active resources in real time system literatures.

To avoid overloading this team, we call all these resources processor except occasionally when we want to be specific about what they are.

Release Time, Deadlines & Timing Constraints

We focus on release time & deadline of a job, two parameters that distinguish job in real time system from those in non real time systems.

The release time of a job is the instant of time at which the job becomes available for execution. The job can be schedule and executed at any time at or after its release time.

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

The deadline of a job is the instant of time at which its execution is required to be completed,

For example, each control law computation job must complete by the release time of the subsequent job.

We say that jobs have no deadline if its deadline is at infinity.

It is more natural to state the timing requirement of a job in terms of its response time, that is, the length of time from the release time of job to the instant when it completes.]

We call the maximum allowable response time of a job its relative deadline.

Absolute deadline $\rightarrow$ release time + release deadline

Constraints imposed by timing behaviour of a job called timing constraints. In its simplest form, a timing constraint of a job can be specified in terms of its release time & relative or absolute deadlines.

Timing constraints

- Hard Timing Constraints
- Soft Timing Constraints

Before explaining about timing constraints, let's be familiar with three definitions.

a. Functional criticality of jobs
b. Usefulness of late results.
c. Deterministic or probabilistic nature of the constraints.

a. **Functional Critically of jobs**: - It means how critical the job is depending on the nature of critically, the jobs can be categorized.
b. **Usefulness of late results**: What happens when the results are produced lately. Whether that meets user's require must or not. It also helps to categorize the job.
c. **Deterministic or Probabilistic nature of constraints**: - Predefined or it can be easily determined or it happens randomly or probabilistically. Depending on those natures, it can be categorized.

Common Definition of deadlines in teams of soft & hard.

Commonly accepted definition has defined hard & soft deadline in the following manner.
A timing constraint or deadline is hard is the failure to meet it is considered to be a total fault. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences. (Example, a late command to stop a train may cause a collision, and a bomb dropped too late may hit a civilian population instead of the intended military target.

A timing constraint or deadline is soft if the failure to meet is considered to be not so total fault. In contrast, the late completion of a job that, has a soft deadline is undesirable.

The main distinction of hard & soft real time systems are explained in terms of tardiness of a job.
It tardiness is equal to zero if the job completes at or before its deadline; otherwise, if the job is late, its tardiness is equal to the difference between its completion time and its deadline.
The usefulness of result produced by soft real time job decreases gradually as the tardiness of a job increases, but the usefulness of a result produced by hard real time falls off abruptly and may become negative when the tardiness of a job becomes larger than 0.

If a job never miss its deadline, then the deadline is hard, on the other hand, if its deadline can be missed occasionally with some acceptably low probability, then its timing constraint is soft.

Hard Timing Constraints & Temporal Quality of Service Guarantees:

Distinction between hard and soft timing constraint is compatible with the distinction between guaranteed and best effort services. Stated another way, if the user wants the temporal quality (e.g. response time) of the service provided by a task guaranteed and the satisfaction of the timing constraint defining the temporal quality, then the timing constraint are hard.

Hard Real Time Systems:

To justify the requirement of hard real time systems, the several examples of hard real time systems and discussed why hard timing constraints are imposed.

Some reasons for requiring Timing Guarantees:

- Most embedded systems are hard real time systems:
- Deadlines of jobs in an embedded system are typically derived from the required responsiveness of sensors & ..... monitored & controlled by it.
- Let's take an example of automatically controlled train.
- When the signal is red(stop), its braking action must be activated a certain distance away from the signal post at which the train must stoop.
- This braking action depends on the speed of the train & the safe value of deceleration.
- With the help of speed & safe value deceleration, the controller can compute the time for the train to travel the braking distance.
- This time in turn imposes a constraint on the response time of the jobs which sense and process the stop signal & activate the brake.
- Some concept is also applied in air craft system also.

More on Hard Timing Constraints:

- Deterministic Constraints
- Probabilistic (less than certain probability.
- Constraints in terms of some usefulness function.

A. Soft Real Time System:

A system in which jobs have got deadlines is a soft real time systems. Examples of such systems include on –line transaction, electronic games.

The less rigorous validation required of the system and, often more relaxed timing constraints allow develops to consider other performance metrics equally seriously.
An occasional missed deadline or aborted execution is equally considered tolerable.

The timing requirement of soft real-time systems are often specified in probabilistic terms.
e. g. telephone network, multimedia systems.

Telephone Network Example:-

In response to our dialing a telephone number, a sequence of jobs executes in turn, each routes the control signal from one switch to another in order to set up a connections through the network on our behalf.
The users are usually satisfied if after extensive simulation and trial use, the system indeed appears to meet this requirement.

Multimedia System:

For example, a frame of movie must be delivered every thirtieth of a second, and the difference in the times when each video frame is displayed and when the accompanied speech is presented should be no more than 80 .....
However, the ..... are often willing to tolerate a few glitches, as long as the glitches occur rarely and for short length of time.

Questions:

- ✓ What do you mean by Jobs and Processors? Elaborate the terms Release time, Deadlines and Timing constraints.
- ✓ What are the types of deadlines? Explain.
- ✓ How many types of timing constraints are there in Real time systems?
- ✓ Define Hard Real Time system in terms of Functional criticality of jobs, usefulness of late results and deterministic or probabilistic nature of constraints.
- ✓ Define Soft Real Time system in terms of Functional criticality of jobs, usefulness of late results and deterministic or probabilistic nature of constraints.
- ✓ Explain about the examples of Soft Real time systems in detail.
- ✓ Explain about the examples of Hard Real time systems in detail.

Unit 3: Reference Model of Real Time Systems

A. Processor and resources
B. Temporal parameters of real-time workload
C. Periodic Task Model
D. Precedence constraints and data dependency
E. Other dependencies
F. Functional parameters
G. Resource parameters of jobs and parameters of resources
H. Scheduling hierarchy

According to this model, each system is c..... by three elements:

a. a workload model that describes the applications supported by the system.
b. a resource model that describes the system resources available to the applications.
c. algorithm that define how the application system uses the resources at all times.

First two elements of the reference model include description of the application & resources.
Third element is the set of scheduling and resource management algorithms.

Processors & Resources: -

We divide all the system resources into two major types: processors & resources. Again, Processors are often called servers and active resources.
Computer, transmission links, disks and database servers are example of processors.
Two processors are of same type if they are functionally identical & they can be used interchangeably.
Processors that are functionally different or for some other reason cannot be used interchangeably are of different types.
Symmetric multiprocessor is identical similarly, a transmission links connecting an on-board flight management system to the ground controller is a different type of processor from the link connecting two air traffic control centre even when the links have same characteristics, because they can't be used interchangeably.
We will use P to represent processors. The processors are P_1, P_2..... P_m
Resources are passive resources. Example of resources are memory, sequence number, muter & database locks. A job may need some resources in addition with processor to make progress.
For example, a computation job may share data with other computation, and the data may be guarded by semaphores. We model each semaphore as a resource when a fit wants to access the shared data guarded by a semaphore R, it must first lock the semaphore, and then it enters to access the data, semaphore R is resource. We will use the letter R to denote resources. The resources in the example mentioned above are reusable because they are not consumed during use.
A resource is plentiful if no jobs are ever prevented from execution by the lack of this resource.

Temporal Parameters of Real-Time Workload

The workload on processors consists of jobs, each of which is a unit of work to be allocated processor time and other resource.
Many perimeters of hard real-time jobs and tasks are known at all times, otherwise it wouldnot be possible to ensure that the system meet its hard real-time requirements. The number of tasks may change as tasks are added & deleted while the system executes.

Each job Ji is characterized by its temporal parameters, functional parameter resource parameters and interconnection parameters.

Temporal parameter tells us its timing constraints & behaviour.
Functional parameter specify the intrinsic properties of the job.
Resource parameter gives us its resource requirement.
Interconnection parameter describes how it depends on other jobs and how other depends on it.

Fixed, Jittered & Sporadic Release Time:

Release time of a job is represented by r_i of each job j_i
r_i^- is called earliest release time & r_i^+ is called latest release time.
If there is range like
$[r_i^- \ r_i^+]$, it is called jitter release time.

Fixed release time is the predefined fixed time of release. If, for all practical purposes, we can approximate the actual release time of each job by its earliest or latest release time, then we say that the job has a fixed release time.
Almost all every real-time system is required to respond to external events which occur at random instants of time when such an event occurs, the system executes a set of jobs in response. The release time of these jobs are not known until the event triggering them occurs. These jobs are called sporadic jobs or aperiodic jobs.
[sporadic jobs → hated deadlines]
[aperiodic jobs → soft deadlines or no deadline]

Execution Time:

Another temporal parameter of a job, J_i, is its execution time, ei. ei is the amount of time required to complete the execution of J_i when it executes alone and has all resources it requires.
We say that execution time e_i of the job J_i is in the range $[e_i^- \ e_i]$ where e_i^- and e_i^+ are the minimum execution time & maximum execution of j_i respectively. We assume that e_i^- & e_i^+ are known for every hard real time jobs but the actual execution time of the jobs if unknown. We use deterministic model to job which assumes it takes it maximum execution time to complete.

Periodic Task Model

→ It is a well known deterministic workload model.
→ With the help of this model, the model characterizes accurately many traditional hard real time systems applications such as digital control, real time monitoring and constant ..... rate voice/video transmission.
→ In this model, each computation of data transmission is executed regular of semi regular.

Periods, Execution Time & Phases of Periodic Tasks:

Periods: -
The period P_i of the periodic task T_i is the minimum length of all time intervals between release times of consecutive jobs in T_i
Execution Time: -
Its execution time is the maximum execution time of all the jobs in it.
The accuracy of the periodic task model decreases with increasing ..... in release time & variations in execution time. So, a periodic task model is an inaccurate model on the transmission of a variable bit-rate video, because of large variation in the execution time of jobs (transmission time of individual frame).
The release time $r_{i,1}$ of the first job J_i in each task T_i is called phases T_i. For convenience we can use $Q_i = r_{i,1}$

Hyperperiod:

LCM of P_i [i =1, 2, ...n]
H = LCM of P_i

Utilization of Tasks:

$u_i = e_i / p_i$

u_i is equal to the fraction of time truly periodic task with period p_i & execution time e_i keeps processor busy.

Aperiocic & Sporadic Tasks:-

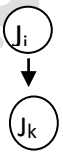
Real time system is invariably required to respond to external events, and to respond, it executes aperodic or sporadic jobs whose release time are not known priori.
We say a task is aperiodic if the jobs have either soft deadline or no deadlines. Example, the task to adjust radar's sensitivity.
Tasks containing jobs that are released at random time instants and have hard deadlines are sporadic jobs. We treat them as hard real time tasks.

Precedence Constraints & Data Dependency:

Data & control dependency among jobs may constrain the order in which they can execute.
In classical scheduling theory, the jobs are said to have precedence constraint if they are constrained to execute in some order. Otherwise, if jobs execute in any order they are said to be independent.
Example:
Signal processing tasks (producer)
↓
tracker tasks (consumer)
Example:
Database Authentication
↓
..... in database

Precedence Graph & Task Graph:

$<$ ← Precedence relation we use $<$ over the set of jobs to specify the precedence constraints among jobs.
A job J_i is predecessor of another job J_k (& J_k is successor of J_i)
Immediate predecessor:-
Immediate successor:-
$J_i < J_j < J_k$
J_i ← predecessor of J_k
J_j ← immediate predecessor of J_K
Similarly,
J_k is immediate successor of J_j & successor of J_i.
Precedence graph is a directed graph. Each vertex in this graph represents a job in J. There is directed edge from the vertex J_i to the vertex J_k.



A task graph which gives us a general way to describe the application system is an extended precedence graph.
As in precedence graph, the vertex in a task graph represents jobs. They can be either circle or square.

(0, 7) (2, 9) (4, 11) (6, 13) (8, 15)
0 0 0 0 0
Phase →0
Period →2, relative deadline → 7 (independent)

(2, 5) (5, 8) (8, 11) (11, 14 (14, 17)
0 → 0 → 0 → 0 → 0 → 0
Phase → 2
Period → 3 (Dependent)
Relative deadline → 5

Data Dependency:

In task graph, data dependencies among jobs are represented explicitly by data dependency edges jobs. There data dependency edges from a vertex J_i to vertex J_k in the task graph if the job J_k consumes data generated by J_i or the job J_i sends message to J_K.

Other Types of Dependencies:

a. Temporal Dependency:

Some jobs may be constrained to complete within a certain amount of time relative to one another. We call the distance between these as temporal distance.

Example person & audio should match when person wants to have lip synchronization.

b. AND/OR Precedence constraints:

In the classical model, a job with more than one immediate predecessor must wait until all immediate predecessors have been completed before its execution can begin.

Whenever, it is necessary to be specific, we call such jobs as AND jobs & they are represented as circles.

In contrast, OR job is one which can begin execution at or after its release time provided one or some of its immediate predecessors has been completed. We represent OR jobs by square vertices.

c. Conditional Branches:

Checking the coordination in predecessor & moving ahead.

d. Pipeline Relationship:

A dependency between a pair of producer-consumer job that are piped can theoretically be represented by precedence graph.

Functional Parameters:

Main function parameters include preemptively, Verticality, optional interval & laxity types.

- Preemptively of Jobs:

  The Scheduler may suspend the execution of a less urgent job & give the processor to a more urgent job. Later when the more urgent job completes, the scheduler returns the processor to the less urgent job so the job can resume execution. This interruption of job execution is called preemption.

  A job is preemptable if its execution can be suspended at any time to allow the execution of other jobs and, later on, can be resumed from the point of suspension. Computation job that execute on CPUs are examples of preemptable jobs.

  A job is non preemptable if it must be executed from start to completion without interruption.

- Critically of jobs:

  In any system, jobs are not equally important. The importance (or criticality) of a job is a positive number that indicates how critical the job is with respect to other job, the more critical the job, the larger its importance. We use importance to measure criticality in order to avoid overloading these terms.

  For example, in a flight control and management system, the job that controls the flight of the aircraft is more critical than the navigation job that determine the critical position. The cabin air flow and temperature control jobs are more critical than the job that runs in-flight movies. In the model of this system, the designer may give these jobs different importance values. In this way, the different degrees of criticality of the jobs are explicitly specified.

- Optional Executions:

  It is possible to structure an application so that some jobs or portions of jobs are optional. If an optional job or an optional portion of a job complete late or is not executed at all, the system performance may degrade, but nevertheless function satisfactorily. For example, collision avoidance system. The collision avoidance system may still function satisfactorily if it skips this computation as long as it generates a warning & displays the course of the object about to collide with it in time.

- Laxity Type & Laxity Function:

  The laxity type of a job indicates whether its timing constraints are soft or hard.

  Laxity type of a job is sometimes supplemented by a usefulness function. This function gives the usefulness of the result produced by the job as a function of its tardiness.

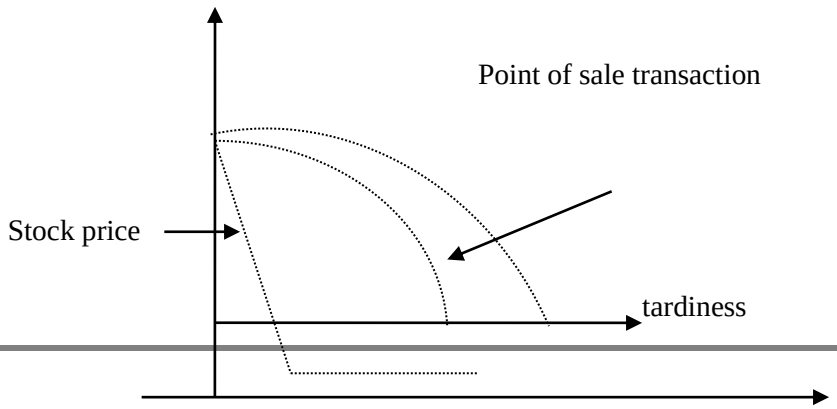


Figure 3.1 Laxity Type and Laxity function

❖ Resource Parameters of Jobs :

a. Preemptivity of resources: A reources parameter is preemptive in nature. A resource is non-preemptable if each unit of resource is constrained to be used serially. In other words, once a unit of a non-preemptable resource is allocated to a job, other job needing the unit must wait until the job completes its use. Otherwise, if jobs can use every unit of a resource in an interleafed fashion, the resource is preemptable. The fact that transmission of a message over a token ring, it interrupted, must be redone from the beginning. Hare, non-preemptivity is a property of the token ring.

b. Resource Graph: We describe the configuration of the resource using a resource graph. In a resource graph, there is a vertex K_i for every processor or resource R_i for every processor or resource R_i in the system. The attributes of the vertex are the parameter 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.
There are two types of edges in resource graphs. An edge from a vertex R_i to another vertex R_k mean that R_k is a component of R_i. This edge is an is –o-past-of edge.
Some edges in resources graphs represent connectivity between components. These edges are called accessibility edges. If there is a connection between two CPUs in the two computers, we can draw accessibility edge.
A parameter of an accessibility edge from a processor P_i to another P_k is the cost for sending unit of data from a job.

Scheduling Hierarchy :

The application system is represented by a task graph. This gives processor time of resource requirement of jobs, the timing constraints of each job & dependency of jobs. The resource graph describes amount of resource available to execute the application system, attributes of resources.
Between them are the scheduling & the resource access-control algorithms used by the operating system.

Processors & Resources Scheduling:

a. Scheduler & Schedules: Scheduler means the module that implements the algorithm to assign & allocate resources. By a schedule, we mean an assignment of all the jobs in the system on the available processor produced by the scheduler.
For having valid schedules, it satisfies following conditions:
   i. Every processor is assigned to at most one job at any time.
   ii. Every job is assigned at most one processor at any time.
   iii. No job is scheduled before its release time.
   iv. All the precedence & resource usage constraints are satisfied etc.

b. Feasibility, Optimality & Performance Measures: A valid schedule is a feasible schedule if every job completes by its deadlines (or in general, meets its timing constraints), we say that a set of jobs is schedulable according to a scheduling algorithms if when using the algorithm the scheduler always produces feasible schedule. If the algorithm produces feasible schedule it is optional.
Other commonly used performance measure includes the maximum and average tardiness, late and response time and the miss, loss and invalid rates.

c. Interaction Among Schedulers:
In the previous example, we treated database locks as resources. In fact, these resources are implemented by a database management system whose execution must be scheduled on one or more processors. The scheduler that schedules the database management system may be different from the application system using the locks. Now, we have two levels of scheduling. In the higher level, the application system is scheduled on the resources. In the lower level, the job that execute in order to implement the resources are scheduled on the processors & resources needed by them.
The scheduler that schedules the database management system may be different from the schedules that schedules application system using locks.

Questions:

- ✓ What do you mean by Reference Model of Real Time Systems?
- ✓ Distinguish between Jobs and Processors.
- ✓ Explain about Temporal Parameters. What do you know about sporadic and aperiodic tasks?
- ✓ What do you mean by periodic task model? Define periods, execution time and phases of such model.
- ✓ Write short note about precedence constraint and data dependency in each paragraph.
- ✓ Explain about precedence graph and task graph.
- ✓ What are the types of dependencies?
- ✓ Explain about functional parameters.
- ✓ Explain in detail about resource parameters.
- ✓ What do you know about scheduling hierarchy? Explain in detail.

Unit 4: Commonly Used Approaches in Real Time Systems

A. Clock-driven approach
B. Weighted round-robin approach

C. Priority-driven approach
D. Dynamic versus static system
E. Effective release times and deadlines
F. Optimality of EDF and LST algorithms
G. Non optimality of EDF and LST algorithms
H. Challenges in validating timing constraints in priority-driven systems
I. Offline versus online scheduling

In this chapter we discuss about three commonly used approaches to scheduling. They are clock driven, weighted round robin & priority driven approaches.

- **Clock driven Approach**:
  As the name implies, when scheduling is clock-driven (also called time time driven), decisions on what jobs execute at what time are made at specific time instants. These instants are chosen a priori before the system begins execution.
  A frequently adopted choice is to make scheduling decisions at regularly spaced time instants.
- **Weighted Round-Robin Approach:**
  The round-robin approach is commonly used for scheduling time-shared applications. When jobs are scheduled on a round-robin basis, every jobs joins a First-in-first out (F_1F_0) queue when it becomes ready for execution. A time slice is the basic granule of time that is allocated to jobs. The weighted round-robin algorithm has been used for scheduling real-time traffic in high-speed switched networks. It builds on the basic round-robin scheme. Rather than giving all the ready jobs equal shares of the processor, different jobs may be given different weights. Here, weight of job refers to the fraction of processor time allocated to the job.

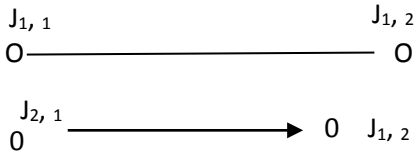
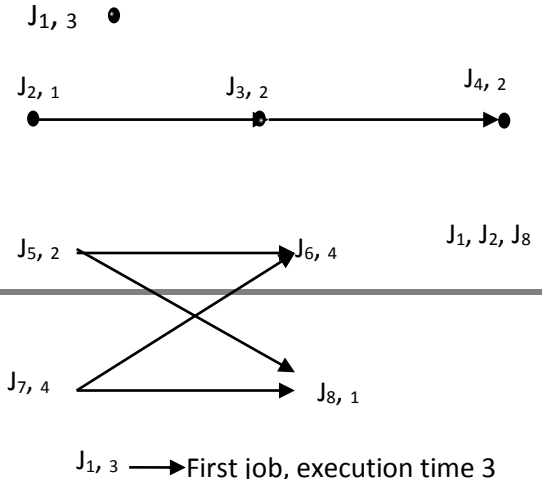


Figure 4.1 Different jobs scheduled using algorithms

❖ **Priority-Driven Approach:**

The term priority driven algorithms refers to a large class of scheduling algorithm that never leave any resource idle intentionally. Stated in another way, a resource idles only when no job requiring the resources is ready for execution. Other commonly used names for this approach are greedy scheduling, list scheduling & work-conserving scheduling. Most scheduling algorithms used in non real time systems are priority driven. Examples include the F_1 F_0, L_1F_0, SETF (Shortest Execution Time First), LETF (Longest Execution Time First) which assign priorities on the basis of job execution time.



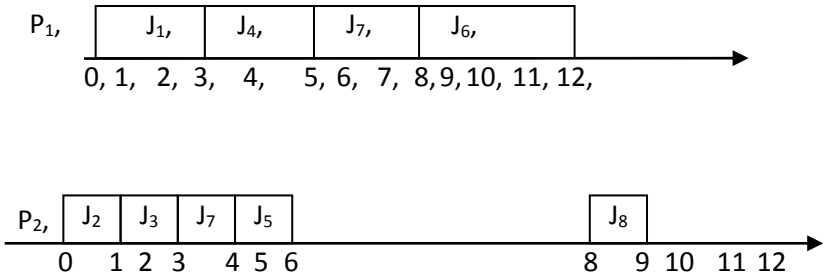


Figure 4.2 Jobs with their execution times

❖ **Dynamic Versus Static System**:

Jobs that are ready for execution are placed in a priority queue common to all processors. When a processor is available, the job at the head of the queue executes on a processor. We will refer to such a multiprocessor system as a dynamic system, because jobs are dynamically dispatched the processors.

Another approach to scheduling in multiprocessor and distributed systems is to partition the jobs in the system into subsystems and assign and bind the subsystem statically to the processors. Such a system is called a static system, because the system is statically configured.

❖ **Effective Release Time & Deadlines**:

a. Effective Release Time: The effective release time of a job without predecessor is equal to its given release time. The effective release time of a job with predecessor is equal to the maximum value among its given release time and the effective release time of its entire predecessor.
b. Effective Deadline: The effective deadline of a job without a successor is equal to the minimum value among its given deadline and the effective deadline of all its successors.

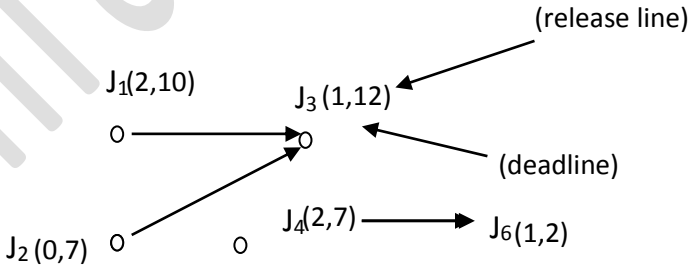


Figure 4.3 Jobs with their release time and deadline

J_1 & J_2 have no predecessor, their effective release time are the given release time, that is, 2 and 0 respectively. The given release time of J_3 is 1, but the latest effective release time of its predecessor is 2, that of J_1. Hence, the effective release time of J_3 is 2.

J_6 has no successor. So the effective deadline of J_6 is same as deadline similarly, J_4 has successor. The effective deadline is 2.

❖ Optimality of EDF & LST Algorithm:

EDF(Earliest-Deadline First)
LST (Least-Slack Time First)
Theory: When preemption is allowed and jobs do not contend for resources, the EDF algorithm produces a feasible schedule of a set J of jobs with arbitrary release time & deadline on a processor ..... J ahs feasible schedule.

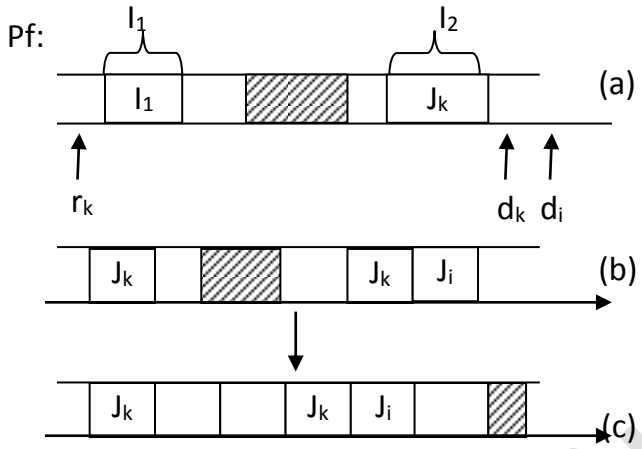
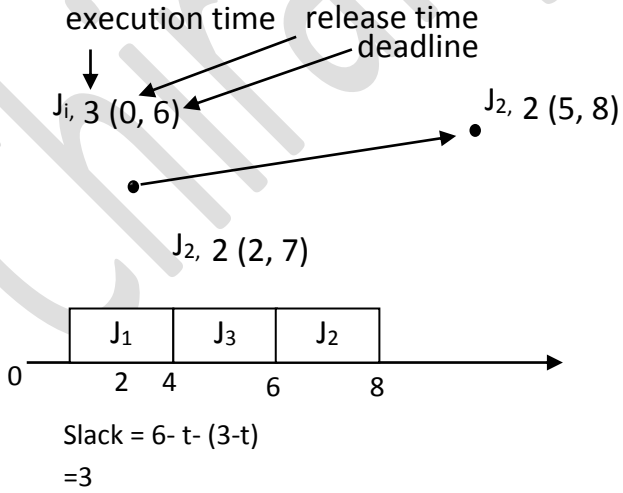


Figure 4.4 Transformation of a non- EDF Schedule into an EDF Schedule.

→Any feasible schedule of J can be systematically transformed into an EDF schedule. Suppose that in a schedule, parts of J_i & J_k are scheduled in intervals I_1 & I_2, respectively. Furthermore, the deadline d_i of J_i is greater than deadline d_k of J_k but I_1 is earlier than I_2.
Without loss of generality, we need to consider release time r_n of J_k is before the end of I_1 but I_1 is earlier than I_2.
To transform the given schedule, we swap J_i and J_k. If the interval I_1 is shorter than I_2, we more the portion of J_k that fits in I_1 forward to I_1 and more the entire portion of J_I scheduled in I_1 backwards to I_2 & place it after J_k. Similar swap if the interval I_1 is longer than I_2.
In this way, we obtain feasible EDF schedule & vice versa.
Theory: 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 feasible schedules of J exist.



Slack = 6- t- (3-t)
=3

Now suppose that it is preempted at time 2 by J_3, which executes from time 2 to 4. During this interval, the slack of J_1 decrease from 3 to 1.

i. e. 6-4-1 =1

slack = (d-t) – time required to complete remaining portion of a job.

t = At any time.

The LST algorithms assigns priorities of jobs based on their slacks, the smaller the slack, the higher the priority.

Non Optimality of EDF & LST Algorithm:

EDF & LDT algorithm gives non optimal solution if non –preemption is applied. When preemption is not applied in the EDF algorithms the remaining portion of time is appended & it produces waste time.

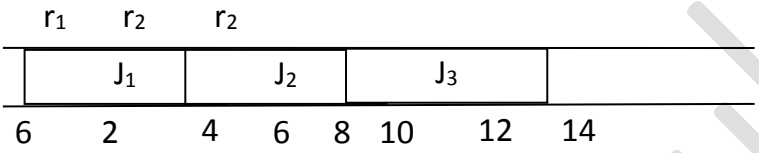


Figure 4.5 Optimal EDF Schedule

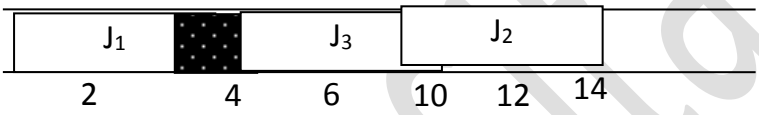


Figure 4.6 Non optional EDF Schedule

Challenges in Validating Timing Constraints in Priority-Driven Systems:

Priority-driven approach has not been widely used in hard real-time systems, especially safety-critical systems, until recently. The major season is that the timing behaviour of a priority-driven system is non deterministic when job parameters vary. Consequently, it is difficult to validate the deadlines of all jobs scheduled priority-driven manner.

Validation means given a set of jobs, the set of resources available to the jobs, and the scheduling algorithms to allocate processors & resource to jobs, determine whether it meat or not.

Anomalies Behaviour of Priority Driven Systems:

J_i	0	10	5
J_2	0	10	2, 6
J_3	4	15	8
J_4	0	20	10

Table 1: Anomalies Behavior of priority driven systems

Unexpected timing behaviour of priority driven system makes anomaly. Scheduling anomalies make the problem of validating a priority driven system difficult whenever job parameter may vary.

Predictability of Executions:

When the timing constraints are specified in teams of deadlines of jobs, the validation problem is the same as that of finding the worst-case (the largest) completion time of every job.

Maximum schedule: Max ^{m} ″

Minimum Schedule: Min ^{m} ″

Actual Schedule: actual execution time.

Validation Algorithm & their Performances:

Validation algorithm allows us to determine whether all jobs in a system in deed meet their timing constraints. While there are nature validation algorithm and tools for static systems, good validation algorithm for dynamic priority-driven systems are not yet available.

We say that a validation algorithm is correct if it never declares that all timing constraints are meet when some constraints may not be. The correct validation algorithms are measured in terms of their complicity, robustness, and accuracy. A validation algorithm is robust if it remains correct even if some assumptions of its underlying workload model are not valid. It improves accuracy.

Off –Line Vs. Online Scheduling:

Pre-computed schedules are called off line schedules. Offline scheduling has several advantages, the deterministic timing behaviours of the resultant system.

We say that scheduling is done on-line or that we use an on-line scheduling algorithms if the scheduler makes each scheduling decision without knowledge about the jobs that will be released in the future the parameters of each job become known to the on-line scheduler only after the job is released.

The system is said to be overload when the job offered to the scheduler cannot be feasibly scheduled.

For performance measure, we say that the value of a job is equal to its execution time if the job complete by its deadline according to a given schedule and is equal to zero if the job fails to complete in time according to the schedule.

The value of a schedule of a sequence of jobs is equal to the sum of the values of all the jobs in the sequence according to the schedule.

Questions:

- ✓ What are the commonly used approaches for Real Time Scheduling?
- ✓ Distinguish between Clock Driven Approach and Weighted Round Robin Approach.
- ✓ What do you know about Priority Driven Approach? Explain in detail.
- ✓ Explain about dynamic vs. static systems, effective release time and deadline.
- ✓ State and Prove optimality of EDF Algorithms
- ✓ State and Prove optimality of LST Algorithms.
- ✓ Prove the non optimality of EDF and LST Algorithms.
- ✓ What are the anomalies behaviors of priority driven systems.
- ✓ Write short notes about offline and online schedules.

Unit 5: Clock Driven Scheduling

A. Notations and assumptions
B. Timer-driven scheduler
C. General structure of cyclic schedules
D. Cyclic executives
E. Improving the average response time of aperiodic jobs
F. Scheduling sporadic jobs

G. Practical considerations and generalization
H. Algorithms for constructing static schedules
I. Pros and Cons of clock-driven scheduling.

Notations & Assumptions:

Conversions:

- There are a periodic tasks in the system. As long as the system stays in an operation made, n is fixed.
- The parameters of all periodic tasks are known ……
  (for all practical purpose, each job in T_i is released P_i units of time after the …… job in T_i)
- Each job $J_{i,k}$ is ready at its release time $r_{i,k}$.

  We refer a periodic task T_i with phase Q_i, period P_i, execution time e_i, relative deadline D_i by …….

  (Q_i, P_i, e_i, D_i)

  e.g. (1, 10, 3, 6) is a periodic tasks whose phase is 1, period is 10, execution time is 3, & relative deadline is 6. Phase is 0 & relative deadline is equal to its period.
- Static, Timer-Driver Scheduler whenever the parameters of jobs with hard deadlines are known before the system begins to execute, a straight forward way to ensure that they meet their deadline is to construct a static schedule of the jobs off-line.

This schedule …….. exactly when each job executes. According to the schedule, the amount of processor time allocted to every job is equal to its maximum execution time, and every job completes by its deadline.

A straight forward way to implements the schedule is to store the precomputed schedule as a table.
Each entry { t_k, $T(t_k)$} in this table gives a decision time t_k, which is an instant when a scheduling decision is made, & $T(t_k)$, which is either the name of the task whose job should start at t_k or I.
We call a periodic static schedule a cyclic schedule. Again, this approach to scheduling hard real-time jobs is called the dock-driver or time driver approach because each scheduling decision is made at a specific time, independent of events, such as job release and completions, in the system.

General Structure of Cyclic Schedule:

- Frame & Major Cycles: The scheduling decision time partition the time line into intervals called frames every frame has length f, f is the frame size.
  Because scheduling decisions are made only at the beginning of every frame, there is no preemption within each frame.
- Frame size constraints: Ideally, we want the frames to be sufficiently long so that every job can start and complete its execution within a frame.
  We can meet this objective if we make the frame size f larger than execution time e_i of every task T_i.
  - $f \geq \max(e_i)$
  - $[p_i/f] - p_i/f = 0$
  - $2f - \text{get}(p_i, f) \leq D_i$
- Job slice: Sometimes, the given parameters of some tasks systems cannot meet all these frame size constraints simultaneously.
  For T= { (4,1), (5, 27)}, we can divide each job in (20, 5) into a chain of three slices with execution time l, 3 & 1. In other words, the task (20, 5) now consists of three subtasks (20, 1), (20, 3) and (20, 1). The resultant system has five tasks for which we can choose the frame size.
  The three original tasks are called T_1, T_2 & T_3 respectively, and the three subtasks of T_3 are called $T_{3,1}$, $T_{3,2}$ & $T_{3,3}$.
  [Note: (Q_i, p_i, e_i, D_i) (0,10, 3, 10)
  As examples (0, 10, 3, 6), (10, 3, 6) & (10, 3) have zero phase. Their deadline is 6 & 10 respectively.]
- Cyclic Executive: The clock-driver scheduler described above must be modified to accommodate the restriction that scheduling decisions are made only at frame boundaries.
  The cyclic execution is a way. In real time system literatures, the term "cyclic executives" refers to a scheduler that deterministically interleaves and sequentializes the execution of periodic tasks on a CPU according to a given cycle schedule.
  In essence, the cyclic executive takes over the processor and executes at each of the clock interrupts, which occur at the beginning of frames.

Improving the average response time of aperiodic jobs.

- Slack Stealing: A natural way to improve the response times of aperiodic jobs is by executing the aperiodic jobs ahead of the periodic job whenever possible. This approach is called slack stealing, was originally proposed for priority driver systems. For the slack stealing scheme described below to work, every periodic job must be scheduled in a frame that ends later than its deadline.
  Let the total amount of time allocated to all the slices scheduled in the frame k be x_k.
  The slack (time) available in the frame is equal to $f-x_k$ at the beginning of the frame.
- Average Response Time: While we are not required to ensure the completion of aperiodic jobs by some specific times, we are often require to guarantee that their average response time is no grater than some value. To give this guarantee, we need to be able estimate the average response time these jobs.
  Suppose that the average rate of arrival of aperiodic jobs in the i^{th} aperiodic task is λ_i jobs per unit time. The sum λ & λ_i over all i= 1, 2, ……. a is the total number of aperiodic job arrival per unit of time.

The mean & the mean square values of the execution times of jobs in the i^{th} aperiodic task are $\epsilon[\beta_i]$ & $\epsilon[\beta_i^2]$ respectively.

We can estimate the average response time of the system for this average arrival rate.

Scheduling Sporadic Jobs:

- Acceptance Test: During the acceptance test, the scheduler checks whether the newly released sporadic job can be feasibly scheduled with all the jobs in the system at the time.

  Here, by a job in the system, we mean either a periodic job, for which time has already been allocated in the precomputed cyclic schedule or a sporadic job which has been scheduled but not yet completed. If according to the existing schedule, there is a sufficient amount of time in the frames before its deadline to complete the newly released sporadic job without causing any job in the system to complete too late.

  To illustrate that this approach is a reasonable one, we consider a quality control system. A sporadic job that activities a robotic arm is released when a defective past is defected. When the job cannot be scheduled to complete in time, it is better for the system to have this information as soon as possible.
- EDF Scheduling of the Accepted jobs: By virtue of its optimality, the EDF algorithm is a good way to schedule accepted sporadic jobs. For this purpose, the scheduler maintains a queue of accepted sporadic jobs in non decreasing order of their deadlines. Whenever all the slices of periodic tasks scheduled in each frame are completed, the cyclic executive lets the jobs in the sporadic job queue execute in the order they appear in the queue.
- Implementation of the Acceptance Test: We now describe how to implement the acceptance test when accepted sporadic jobs are scheduled on the EDF basis.

  Specially, we focus on an acceptance test at the beginning of frame …… to decide whether a sporadic job s(d, e) should be accepted or rejected. The acceptance test consists of the following two steps.

  a. The scheduler first determines whether the current total amount of slack in the frames before the deadline of job s is at least equal to the execution time e of s. If the answer is no, it rejects s. If the answer is yes, the second step is carried out.
  b. In the second step, the scheduler determines whether any sporadic job in the system will complete late if it accepts s. if the acceptance of s will not cause any sporadic job in the system to complete too late, it accepts s, otherwise, it rejects s.

Practical Considerations & Generalizations:

In this topic, we discuss about how to handle frame overrun, how to do made changes, and how to schedule tasks in multiprocessor systems.

- Handling Frame Overrun: A frame overrun can occur for many reasons. For example, when the execution time of a job is input data dependent, it can become unexpectedly large for some rare combination of input values which is not taken into account in the precomputed schedule. A transient hardware fault in the system may cause some job to execute longer than expected.

  A way to handle overrun is to simply about the overrun job at the beginning of the next frame and log the premature termination of the jobs. Such a fault can be handled by some recovery mechanism later when necessary. This way seems attractive for applications where late results are no longer useful.
- Mode changes: During a mode change, the system is reconfigured. Some periodic tasks are deleted from the system because they will not execute in the new mode. Periodic tasks that execute in the new mode but not in the old mode are created and added to the system.

  The periodic tasks that execute in both modes continue to execute in a timely fashion. When the mode completes, the new set of periodic tasks are scheduled & executed.

  a. Aperiodic Mode Change: (soft)

     A periodic task that will not execute in the new mode can be deleted and its memory space and processor time freed as soon as the current job in the task completes. This scheme can be implemented by letting the scheduler or the mode-change job mark each periodic task that is to be deleted.

     During mode change, the scheduler continues to use the old schedules table.
  b. Sporadic Mode Change: (hard)

     If the mode change job is not schedulable & is therefore rejected, the application can either postpone the mode change or take some alternate action.

     General workloads & Multiprocessor Scheduling:

Global clock is used to use multi-process to handle them for scheduling.

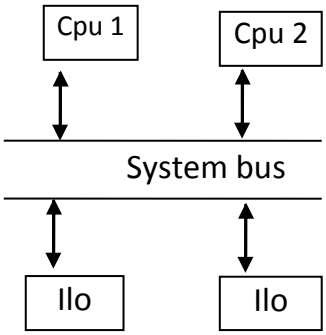


Figure 5.1 Multi Processor

Algorithm for Constructing Static Schedules:

- Scheduling Independent Preemptive Tasks:

  For this interactive Network Flow algorithm is used.

A. Network-Flow Graph:

This graph contains the following vertices & edges; the capacity of an edge is a non negative number.

a. There is a job vertex J_i representing each J_i, for I = 1, 2,…… N
b. There is a frame vertex named j representing each frame j in the major cycle, for j = 1, 2, …….. N.
c. There is a directed edge (j_i, j) form a job vertex J_i to from vertex j if the job J_i can be scheduled in the frame j, and the capacity of the edge is the frame size f.
d. There is a directed edge from the source vertex to every job vertex J_i and the capacity of this edge is the execution time e_i of the job.
e. There is a directed edge from every frame vertex to the sink and the capacity of the edge is f.

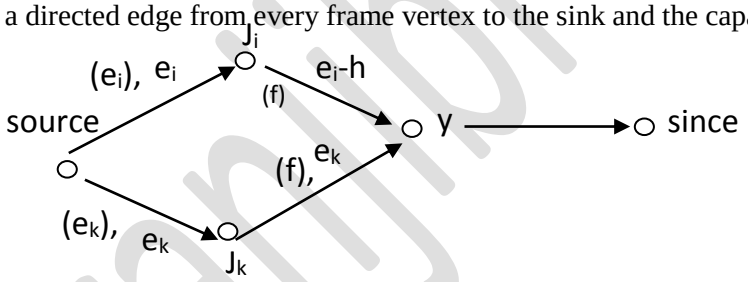


Figure 5.2 Network-Flow

B. Maximum Flow & Feasibility Preemptive Schedule: Clearly, the maximum flow of a network flow graph defined above is at most equal to the sum of the execution time of all the jobs to be scheduled in a major cycle. The set of flows of edges from job vertices to frame vertices that gives this maximum flow represents a feasible preemptive schedule of the jobs in the frames. Specially, the flow of an edge (J_i, J) from a job vertex J_i to a frame vertex j gives the amount of time in frame j allocated to job J_i.

The total amount of time allocated to a job J_i is represented by the total flow out of all edges from the job vertex J_i.

C. Generalization to Arbitrary Release Times and Deadlines: By the way, the network-flow formulation of the preemptive scheduling problem can easily be generalized to find schedules of independent jobs that have arbitrary known release time & deadlines.

In this case, the release time and deadline of all N jobs to be scheduled partition the time from the earliest release time to the deadline into at most 2N-1 intervals.

- Post Processing: The feasible schedule found by the INF algorithm may not meet some of the constraints that are imposed on the cyclic schedule.

  Examples are precedence order of jobs and restrictions on preemptions.

  We can always transform the schedule into one that meets

  Precedence constraints of the jobs by swapping the time allocating of jobs that are scheduled in wrong order.

  Because the jobs may be scheduled in some wrong order according to a feasible schedule found by INF algorithm.

Pros & Cons of Clock-Driven Scheduling:

The clock-driven approach to scheduling has many **advantages.**

a. Conceptual Simplicity:
b. Time triggered systems based on clock-driven scheduling approach are relatively easy to validate, test & certify.

Disadvantages:

a. The release time of all jobs must be fixed.
b. In clock driven system, all combination of periodic tasks that might execute at the same time must be known a prior so a schedule for the combination can be precomputed which may not be acceptable for on-line schedules.
c. The pure clock-driven approach is not suitable for many systems that contain both hard and soft real time application.

Questions:

- ✓ What do you mean by clock driven approach? Explain about static timer driven scheduler.
- ✓ What are the general structure constituents of cyclic schedules?
- ✓ How do you improve the average response time of aperiodic jobs?
- ✓ How do you schedule the sporadic jobs?
- ✓ What are the practical considerations and generalizations for clock driven approach?
- ✓ What are the strategies for constructing static schedules?
- ✓ Explain about the merits and demerits of clock driven approach.

Unit 6: Priority driven scheduling of Periodic Tasks

A. Static assumption
B. Fixed-priority versus dynamic-priority algorithms
C. Maximum schedule utilization
D. Optimality of RM and DM algorithms
E. Schedulability test of fixed-priority tasks with short response times
F. Schedulability test for fixed-priority tasks with arbitrary response time
G. Sufficient Schedulability conditions for RM and DM algorithms
H. Practical factor

For consideration, let us suppose two things throughout this chapter as

- the tasks are independent and
- there are no aperiodic and sporadic tasks.

We hereafter use the term period to mean the minimum interrelease time of jobs in a task.

Static Assumption

Multiprocessor priority-driven system is either dynamic of static. In processor, and tasks on each processor are scheduled by themselves. In contrast in a dynamic system, jobs ready for execution are placed in one common priority queue and dispatched to processors for execution as the processors become available.

The dynamic approach should allow the processors to be more fully utilized on average as the workload fluctuates. Indeed, it may perform well most of the time. However, in the worst case, the performance of priority-driven algorithms can be unacceptably poor.

In most cases, the performance of dynamic systems is superior to static system.

In static, all the tasks are partitioned into subsystems. Each subsystem is assigned to a processor, and tasks on each processor are scheduled by themselves. In contrast, in a dynamic system, jobs ready for execution are placed in one common priority queue and dispatched processors for execution as the processors become available.

Fixed-Priority versus Dynamic-Priority Algorithms:

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

A fixed-priority algorithm assigned the some priority to all the jobs in each task. In other words, the priority of each periodic task is fixed relative to other tasks. In contrast, a dynamic priority algorithm assigned different priorities to the individual jobs in each task. Hence, the priority of the tasks with respect to that of the other task changes as jobs are released and completed. This is why this type of algorithm is said to be 'dynamic'.

Rate-Monotonic & Deadline Monotonic Algorithms:

Rate-Monotonic algorithm is well known fixed priority algorithm. This algorithm assigns priorities to tasks based on their periods, the shorter the period, the higher the priority. The rate of a task is the inverse of its period. Hence, the higher its rate, the higher its priority. We will refer to this algorithm as the RM algorithm for short and a schedule produced by the algorithm as an RM schedule.

For Example, the system contains three task:

T_1 = (4, 1), T_2= (5, 2) & T_3 = (20, 5). The priority of T_1 is highest because its rate is the highest (or equivalently its period is the shortest).

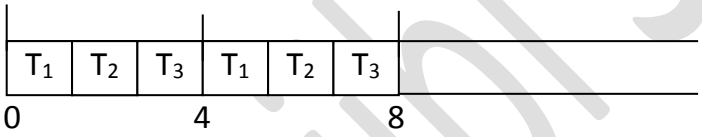


Figure 6.1 RM Schedule

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

For example, the system consists of three tasks. They are T_1 = (50, 50, 25, 100), T_2 = (0, 62.5, 10, 20) & T_3 = (0, 125, 52, 50).

According to the DM algorithm, T_2 has the highest priority because it relative deadline 20 is the shortest among the tasks. T_1 with a relative deadline 100, has the lowest priority.

Clearly, when the relative deadline of every task is proportional to its period, the RM & DM algorithm are identical.

Well –Known Dynamic Algorithm

The EDF algorithm assigns priorities to individual jobs in the tasks according to their absolute deadline, it is dynamic-priority algorithms.

Least-Slack-Time first is also another dynamic –priority algorithms.

LIFO & FIFO are also well known dynamic-priority algorithms.

Relative Merit:

Algorithms that do not take into account the urgencies of jobs in priority assignment usually perform poorly. Dynamic priority algorithm such as FIFO & LIFO are examples.

Hereafter, we confine our attention to algorithms that assign priorities to jobs based on one or more temporal parameters of jobs & have either optional or reasonably good performance. The RM, DM, EDF & LST algorithm are such algorithm.

Optional dynamic algorithm outperform fixed priority algorithm, an advantage of fixed priority algorithm is predictably. In contrast, when the tasks are scheduled according to a dynamic algorithm, it is difficult to predict which tasks will miss their deadlines during overloads.

Maximum Schedulable Utilization:

We say that a system is schedulable by the algorithm if the algorithm always produces a feasible schedule of the system. We now ask low large the total utilization of a system can be in order for the system to be surely schedule.

Schedulable Utilization of an EDF Algorithms:

Theorem:

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

PF: For proving this concept let's consider two cases:

a. The current period of every task begins at or after $r_{i,\,c}$.
b. The current periods of some tasks begin $r_{i,\,c}$.

The total utilization is larger than 1 if it is not feasible.

Let suppose contrast

case I:

The fact that $J_{i,\,C}$ misses its deadline at t tells us that any current job whose deadline is after t is not given any processor time to execute before t and that total processor time required to complete $J_{i,\,C}$ and all the jobs with deadlines at or before t exceeds the total available time.

$$t < \frac{(t-Q_i)\,e_i}{p_i} + \sum_{k \neq i} \left[\frac{t-Q_k}{p_k}\right] e_k \quad \ldots\ldots\ldots\ldots (i)$$

Similarly,

Since,

$$\frac{(t-Q_i)\,e_i}{p_i} + \sum_{k \neq i} \left[\frac{t-Q_k}{p_k}\right] e_k$$

$$\leq + \frac{e_i}{p_i} + t \sum_{k \neq i} \frac{e_k}{p_k} +$$

$$\geq + \sum_{k=i}^{n} y_{k=tU} \ldots\ldots\ldots\ldots\ldots\ldots (II)$$

Com…….. this inequality with eq$^{n}{}_{(i)}$, we have U>1

Case II

Let T^1 devote the subset of T containing all the tasks whose current jobs were released before $r_{i\ C}$ & have deadlines after t.

$$t - t_{-1} < \frac{\left(t-t_{-1}-Q_i^{\,1}\right) e_i}{p_i} + \sum_{T_K T - T^1} \left[\frac{t-t_{-1}-Q_k^{\,1}}{p_k}\right] e_k$$

This inequality is some as (i) except that t is replaced by $t - t_{-1}$ & Q_k is replaced by Q_{k1}. We can use same concept to prove it.

Hence, contradiction proved.

Schedulability Test for the EDF Algorithms:

Hereafter, we call a test for the purpose of validating that the given application system can indeed meet its hard deadlines when scheduled according to the chosen scheduling algorithm a schedulability test. If a schedulability test is efficient, it can be used as an on-line acceptance test.

We are given,

a. the period P_i, execution time e_i, & relative deadline D_i of every task T_i in a system.

$T = \{ T_1, T_2 \ldots\ldots\ldots\ldots T_n$ of independent periodic tasks.

b. a priority driven algorithm used to schedule the task in T preemptively on one processor.

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

We check the (i) in eq for checking its scheduling.

Optimality of the RM and DM algorithms

Th:

A system of simply periodic, independent, preemptable tasks whose relative deadlines are equal to or larger than their periods is schedule on one processor according to RM algorithm if it total utilization is equal to or less than 1.

Proof:

Let's prove it by contradiction. We now suppose for that the tasks are in phase (i.e, the tasks have identical phase) and processor never idles before the Task T_1 misses a deadline for the first time at t.

Now, the total time required to complete all the jobs with deadlines before and at t is equal to .....

$\sum_{k=1}^{i} e_k \left(\frac{t}{p_k}\right)$, Which is equal to t times the total utilization $U_i = \sum_{k=1}^{i} u_k$ of the i.

highest priority tasks. That T_i misses a deadline at t means this demand exceeds to in other words, $U_i > 1$ proved.

A Schedulability Test for fixed priority tasks with short response time:

- Critical Instants: The schedulability test uses as input the given sets $\{p_i\}$ and $\{e_i\}$ of periods and execution times of the tasks in T & checks one task T_i at a time to determine whether the response times of all its jobs are equal to or less than its relative deadline D_i.

  A critical instant of a task T_i is a time instant which is such that

  a. the job in T_i released at the instant has the maximum response time of all jobs in T_i if the response time of every job in T_i is equal to or less than the relative deadline D_i of T_i and
  b. the response time of the job released at the instant is greater than D_i if the response time of some jobs in T_i exceeds D_i.

  We call the response time of a job in T_i released at a critical instant the maximum response time of the task & denoted by W_i.

- Time-Demand Analysis: To determine whether a task can meet all its deadline we first compute the total demand for processor time by a job released at which instant of the task we than check whether this demand can be met before the deadline of the job. For this, we name this test a time-demand analysis.

  To carry out the time-demand analysis on T, we consider one task at a time, starting from the task T, with highest priority in order of decreasing priority.

  The total time demand W_i (t) are given as

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

  to …….. release time

  $t \geq 0$

- ***Alternative to Time Demand Analysis:***

In other words, a way to test the schedulability of such a system is to construct a schedule of it according to the given scheduling algorithms. For the alternative to time-demand analysis, we perform simulation method.

For new, it appears that the more conceptually complex time-demand analysis method has no advantage over the simulation method.

For large number of worst case condition, the complexity of the simulation method and make it impractical for large systems. For non-preemptive system, time-demand analysis can be extended but simulation cannot be extended.

Schedulability Test for fixed-priority job with arbitrary response time:

- Busy Intervals:

We will use the term level π_i busy interval (t_0, t) begins at an instant t_0 when.

a. all jobs in T_i released before the instant have completed.

b. a job in T_i is released the interval ends at the first instant t after to when all the jobs in T_i released since t_0 are complete.

- General Schedulability Test:

For general Schedulability Test, we use general time-demand analysis method.

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

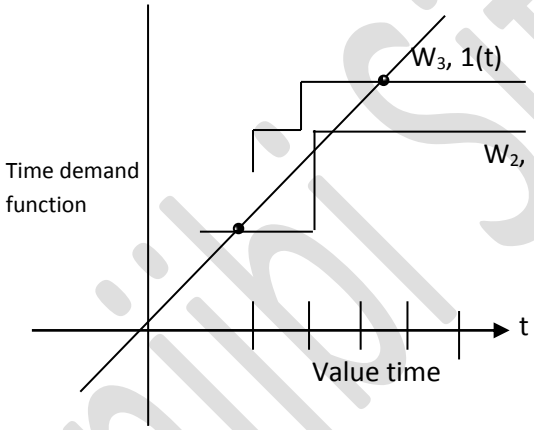


Figure 6.2 Schedulability test

- Correctness of the general Schedulability Test:

The general schedulability test described above makes a key assumption. The maximum response time of some job J_i, j in an in-phase level π_i busy interval is equal to the maximum possible response time of all jobs in T_i. Therefore, to determine whether T_i is schedulable, we only need to check whether the maximum response time of all jobs in this busy interval are no greater than the relative deadline of T_i.

Practical Factors:

We will need to discuss about two new terms as blocking & priority inversion.

A ready job J_i is blocked when it is prevented from executing by a lower-priority job. The lower-priority job executes while J_i waits. We say that a priority inversion occurs whenever a lower-priority job executes while some ready higher priority job waits.

- Non Preemptability:

a. Blocking time due to non-preemptivity. A higher-priority job that becomes ready when a non-preemptable lower-priority job is execution is blocked until the non preemptable portion of lower priority job completes. This delay due to blocking may cause the higher-priority job to miss its deadline.

b. Effects of Blocking on schedulability:

The time demand function including blocking time is given by

$$w_i\,(t) = e_i + b_i \quad \sum_{k=1}^{i-1} \left[\frac{t}{p_k}\right] e_k, \text{ for } o < t \leq \min\,(D_i, P_i)$$

- Self-suspension: (self-blocking): Self-blocking or self suspension occurs when the job is suspended and waits until such an operation completes before its execution can continue.
  A job may self-suspend after its execution has begun, and the amount of time for which job in a task self suspend may differ.
  In a special case, every job in a task T_i self-suspend for x units of time immediately after it is released. The job is ready for execution x time units after it is released.
  Hence, the time from the instant when the job is ready to its deadline is only $D_i - x$ not D_i.
  To determine whether the task T_i is schedulable, we use the shortened deadline D_i-x.
- Context Switches: We let CS denote the context-switch time of the system, that is, the maximum amount of time the system spends per context switch. CS includes the time required to maintain the context of the jobs involved in the context switch, as well as the time spent by the scheduler to service the event interrupt that triggers the context switch & to carry out the scheduling action at the context switch.
- Tick Scheduling: A way to implement the scheduler is to make it time-driven. By this we mean that the execution of the scheduler is triggered by a timer which is set to expire periodically. This method is called tick scheduling or time-based scheduling.
  Tick scheduling introduces two additional factors that must be accounted for in schedulability analysis.
  First, the fact that a job is ready may not be noticed and acted upon by the scheduler until the next clock interrupt.
  Second, a ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue.
- Varying Priority in Fixed Priority Systems:
  a. **Subtasks, canonical form & Interference Block:**
     A task is in canonical form if every later subtask has a higher priority than its immediate predecessor subtask.
     An interference block of a subtask T_i is a chain of one or more contagious subtasks having following properties.
     - All of those subtasks have equal or higher priorities than the priority $\pi_{i,k}$ of $T_{i,k}$.
     - either T_i has no predecessor or the priorities of its immediate predecessor are lower than $\pi_{i,k}$.
     - either $T_{l,y}$ has no successor or the priority of its immediate successor is lower than $\pi_{i,K}$.

  b. Extended General Time-Demand Analysis:
     - Transforming the target task.
     - Identifying the Interference Blocks.
     - Computing the length of level $\pi_{i,l}$.
- Schedulability Test for Hierarchically Schedule Periodic Tasks:
  Two common hierarchical scheduling schemes are the priority-driven/round –robin system, the scheduler schedules the subsystems in a priority-driven manner. The tasks in each subsystem are scheduled in a round robin manner in the interval assigned to the subsystem.

Questions:

- ✓ Explain about fixed priority versus dynamic priority.
- ✓ What do you know about Rate Monotonic and Deadline Monotonic?
- ✓ Explain about the well known dynamic algorithms.
- ✓ State and Prove Maximum Scheduler Utilization.
- ✓ State and Prove Schedulability test for the EDF algorithms.
- ✓ Explain about Schedulability Test for fixed priority tasks with short response time.
- ✓ Explain about Schedulability Test for fixed priority tasks with arbitrary response time.
- ✓ What are the practical factors for priority scheduling? Explain in detail.
- ✓ Explain about the operations when varying priority in fixed priority systems.
- ✓ How to test the Schedulability for hierarchical systems?

Unit 7: Scheduling Aperiodic and Sporadic Jobs in Priority Driven Systems

A. Assumptions and approaches
B. Deferrable servers
C. Sporadic servers
D. Constant Utilization
E. Total bandwidth
F. Weighted fair queuing servers
G. Slack stealing in deadline-driven systems
H. Slack stealing in fixed-priority systems
I. Scheduling of sporadic jobs
J. Real time performance for jobs with soft timing constraints
K. Low-level scheme for integrated scheduling

❖ ***Assumption & Approaches:***

We donot make any assumption on the inter-release-times and execution time of aperiodic jobs. Sporadic job may arrive at any instant, even immediately after each other. It is impossible for some sporadic jobs to meet their deadlines no matter what algorithm we use to schedule them. The only alternative are

a. to reject the sporadic jobs that cannot complete in time or
b. to accept all sporadic jobs & allow some of them to complete late.

Objectives, Correctness and Optimality:

The algorithm described in this topic determine when aporiodic or sporadic jobs are executed. We call them aperiodic job and sporadic job scheduling algorithms, they are solutions to the following problems.

a. Based on the execution time & deadline of each newly arrived sporadic job, the scheduler decides whether to accept or release the job.
b. The scheduler tries to computer each aperiodic job as soon as possible. The problem is now to do so without causing periodic tasks & accepted sporadic jobs to miss their deadlines.

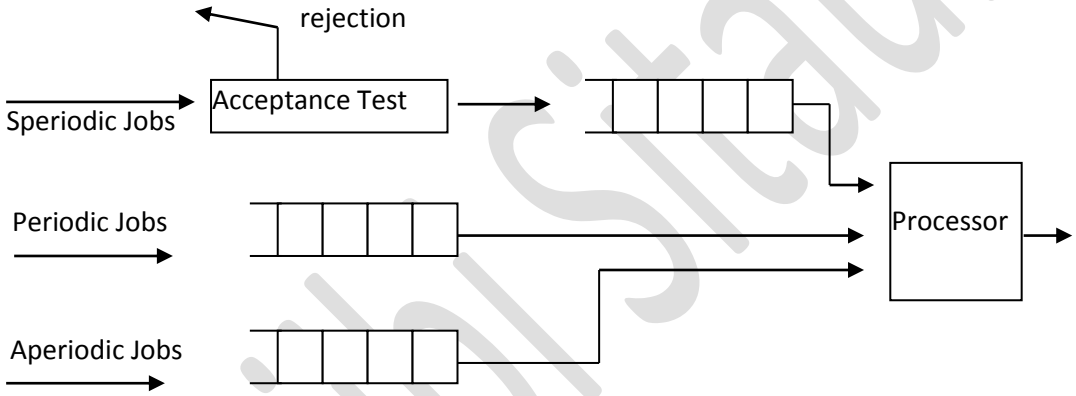


Figure 7.1 priority queues maintained by operating system

❖ ***Alternative Approaches:***

All the algorithms described in this chapter attempts to provide improved performance over the three commonly used approaches. These are background,..... and interrupt driven executions. For the sake of simplicity & clarity, we ignore sporadic jobs for now.

a. Background and Interrupt-Driven execution versus Slack Stealing:

According to the background approach, aperiodic jobs are scheduled and executed only at times when there is no periodic or sporadic job ready for execution.

However, the execution of aperiodic jobs may be delayed and their response time prolonged unnecessarily.

An obvious way to make the response times of aperiodic jobs as short as possible to make their execution interrupt-driven.

Whenever an aperiodic job arrives, the executions of periodic tasks are interrupted, and the aperiodic job is executed. Algorithms that make use of the available slack times of periodic and sporadic jobs to complete aperiodic jobs are called slack-stealing algorithms.

In system, where slack-stealing can be done with an acceptable amount of overhead, it allows us to realize the advantage of interrupt driven execution without its disadvantage.

b. Polled Executions Vursus Bandwidth-Preserving Service:

- Polling is another commonly used way to execute aporiodic jobs. In our terminology, a poller or polling server (P_s, e_s) is a periodic task: P_s is its polling period, & e_s is ist execution time when it –executes, it examines the aperiodic job queue. If the queue is non-empty, the poller executes the job at the head of the queue. The poller suspends its execution or is suspended by the scheduler when it has executed e_s units of time in the period or when the aporidic job queue becomes empty.

A periodic server (p_s, e_s) is defined partially by its period p_s and execution time e_s.

Algorithms that improve the polling approach in this manner are called bandwidth preserving server algorithms. Bandwidth preserving servers' also periodic server. Each type of server is defined by a set

of consumption & replenishment rules. The former give the conditions under which its execution budget is preserved & consumed. The latter specify when and how much the budget is replenished.
The three sections describe three types of bandwidth-preserving servers. They are:

a. Deferred server
b. Sporadic server
c. Constant utilization (total bandwidth servers and weighted fair-queuing servers).

❖ ***Deferred Servers;***

A deferred server is the simplest of bandwidth-preserving servers. Like a poller, the execution budget of a deferred server with period p_s and execution budget e_s is replenished periodically with period p_s. Unlike a ..... however, when a deferred server finds no aperiodic job ready for execution, it preserves its budget.

❖ ***Operations of Deferrable Servers:***

Specifically, the consumption and replenishment rules that define a deferrable server (p_s, e_s) are as follows.

a. Consumption Rule: The execution budget of the server is consumed at the rate of one per unit time whenever the server executes.
b. Replenishment Rule: The execution budget of the server is set to e_s at time instants kp_k, for k= 0, 1, 2

❖ Schedulability of Fixed-Priority .....

Sporadic Servers:

A deferrable server may delay lower-priority task for more time than a periodic tasks with the same period and execution time. This section describes a class of bandwidth-preserving server, called sporadic servers that are designed to improve over a deferrable server in this respect. The consumption & replenishment rules of sporadic server algorithm ensure that each sporadic server with period p_s and budget e_s never demands more processor time than the periodic task (p_s, e_s) in any time interval. With the use of sporadic servers, we can either increase server execution time or decrease period which is held in deferred servers.

❖ ***Sporadic sever in Fixed Priority Systems:***

Since our focus here is on the consumption and replenishment rules of the server, we assume that we have chosen the parameters p_s and e_s and have validated that the periodic task (p_s, e_s) and the system T are schedulable according to the fixed-priority algorithm used by the system, while doing the schedulability test we assume that the relative deadline for consuming the server budget is finite but arbitrary. In particular, we allow this deadline to be larger than p_s. during the interval, when the aperiodic job queue is never empty, the server behaves like the periodic task (P_s , e_s) in which some jobs may take longer than are period to complete.

$t_r \rightarrow$ latest (actual) replenishment time.
$t_f \rightarrow$ first instant after t_r,
$t_e \rightarrow$ latest effective replenishment time.
BEGIN $\rightarrow$beginning busy interval at t.
END $\rightarrow$ latest busy interval.

Simple Sporadic Servers:

Consumption:

At any time t after t_r, that server's execution budget is consumed at the rate of 1 per unit time the budget is exhausted when either one of two conditions is true.

$\rightarrow$ The server is executing.
$\rightarrow$ The server has executed since t_r and END < t.
(otherwise the server holds its budget)

Replenishment:

R_1 initially when the system begins execution and each time when the budget is replenished.
The t_r = the current time.
R_2 At time t_f, if END = t_f

t_e = max (t_r, BEGIN)

the replenishment time = $t_e t p_s$.

R_3 a. If the next replenishment time t_e t p_s is earlier then t_f, the budget is replenishment as soon as it is exhausted.

b. If the system T because idle before the next replenishment time $t_e t p_s$ and becomes busy again at t_b, the budget is replenished at min (t_e p_s, t_b).

❖ **Enhancement of Fixed-Priority Sporadic Server:**

a. Sporadic/Background Server:

It is a combination of a sporadic server and a background server & is defined by the following rules.

→ Consumption rules of simple sporadic/background server are the same as the rules of simple sporadic servers expect when the period task system is idle. As long as the periodic task system is idle, the execution budget of the server stays at e_s .

→ Replenishment rules of simple sporadic/background servers are same as those of the simple sporadic server. The budget of a sporadic/background server is replenished at the beginning of each idle interval of the periodic task system t_r is set at the end of the idle interval.

b. Cumulative Replenishment:

In other words, rather than setting the budget to e_s, we increment its budget by e_s units at each replenishment time. Hence, the budget of a server may exceed e_s. As a result of this change in replenishment, the server emulates a periodic task in which some job do not complete before the subsequent jobs in the task are released.

❖ **Simple Sporadic Servers in Deadline Driven Systems:**

The consumption & replenishment rules of simple sporadic servers in a system where all task are scheduled according to the EDF algorithm are stated as follows.

→ We again us t_r denote the latest replenishment time. However, the notation, the effective latest replenishment time, has a different meaning from that of a fixed-priority sporadic server. The server is ready for execution only when it is back logged (i.e, the aperiodic job queue is non empty) and its deadline d (and hence its priority) is set. The server is suspended whenever its deadline is undefined or when the server it idle (i.e., the aperiodic job queue become empty).

Constant Utilization, Total Bandwidth And Weighted Fair-Queuing Servers:

These are three bandwidth preserving server algorithms. These algorithms sometimes called Generalized Processor Sharing (GPS) similar to round robin algorithm.

❖ **Constant Utilization Server Algorithms:**

→ Consumption and Replenishment Rule: The Consumption rule of constant utilization server, as well as that of a total bandwidth or weighted fair-queuing server, is quite simple. A server consumes its budget only when it executes.

→ Replenishment Rules of a Constant Utilization Server of size u_s.

R_1: Initially, $e_s = 0$ and $d = 0$

R_2: When an aperiodic job with execution time e varies at time to to an empty aperiodic job queue.

a. if $t < d$ do nothing

b. if $t \geq d$, $d = t + e/ \sim_{us}$ & $e_s = e$

R_3: At the deadline d of the server

a. if the server is backlogged, set the server deadline to d_t e/ ..... &

b. if the server is idle, do nothing.

❖ ***Total Bandwidth Server Algorithms:***

Specifically, the total bandwidth server algorithm. Improves the responsiveness of a constant utilization server by allowing the server to claim the background time not used by periodic tasks.

Replenishment Rules of a Total Bandwidth Servers of Size $\tilde{u}_s$

R_1 : Initially $e_s = 0$ & $d = 0$

R_2 : When an aperiodic job with the execution time e arrives at time t to an empty aperiodic job queue, set d to max (d,t) $t_e/\tilde{u}_s$ and $e_s = e$

R_3: When the server completes the current aperiodic job, the job is removed from its queue.

a. If the server is backlogged, two server deadline is set to $d + \frac{e}{\tilde{u}_s}$, and $e_s = e$

b. If the server is idle, do nothing.

❖ ***Preemptive Weighted Fair:***

Queuing Algorithm:

We now ready to state the rules that define the WFQ algorithm. The scheduling and budget consumption rules of a WFQ server are essentially the same as those of a total bandwidth server.

a. **Scheduling Rule:**
A WFQ Schedule is ready for execution when it has budget and a finish time. The scheduler assigns priorities to ready WFQ server based their finish numbers. The smaller the finish number, the higher the priority.

b. **Consumption Rule:** A WFQ server consumes its budget only when it executes.

Slack-Stealing in Deadline Driven System:

We will see shortly that slack stealing algorithms for deadline-driven systems are conceptually simpler than slack-stealing algorithms for fixed priority systems.

The scheduler monitors the periodic tasks in order to keep track of the amount of available slack. It gives the slack stealer the highest priority whenever there is slack and the lowest priority whenever there is no slack. When the slack stealer executes, it executes the appropriate job queue. This kind of slack stealing algorithm is said to be greedy.

❖ ***Static-slack Computation:***

The slack of the system $\sigma(t_c)$ at time t_c is the minimum of the slacks of all the jobs with deadlines after t_c.

e.g. : $\sigma(o) = d_i$ - .....

let, w (j, k) denote the minimum of all σ_i (o) for i = J, j+ 1, ..., k -1, k

❖ ***Dynamic Slack Computation:***

In the presence of release time jitters, the slack time may have to be computed dynamically during runtime. We focus here on an algorithm that computes a lower bound on the slack of the system at time t_c, when the scheduler needs the bound, without relying on any pre-computed slack information. So, the adjective "current" and "next" are relative with respect to the slack computation time t_c.

The steps for dynamic- slack computation are

- Information maintained by the scheduler.
- Estimated busy interval length.
- Slack computation.

As an example,

We consider a system of three periodic tasks:

$T_1 = (4, 1)$

$T_2 = (5, 2)$

& $T_3 = (11, 2.1)$

Suppose that the scheduler computes the slack of the system at time.

t = 2.0, At time, ... is 1.0, ... is 1.0 & ..... is 0.

The execution time of remaining..... of job are 0, 1.0 & 2.1.

The earliest release time of the next job in three periodic task are 4, 5 & 11.

According to the above expression of the processor-time demand w(x),

We have,

$$W(x) = 0 + 1.0 + 2.1 + \left\lceil \frac{x-2}{4} \right\rceil + \left\lceil \frac{x-3}{5} \right\rceil$$

The solution $\propto$ of w (x) = x is 7 %.

Hence, EN = 2+ 7.1 = 9.1

BEGIN is equal to

Min [4+ 4 $\left(\frac{9.1-4}{4}\right)$, 5+ 5 $\left(\frac{9.1-5}{5}\right)$, 11

= Min (12, 10, 11) =0

Formulae,

w(x) = $\sum_{i=i}^{n}$ $(e_i - E_i)$+ $\sum_{i=i}^{n}$ ……..

Begin = ……

END = t+ w(n)

Slack = BEGINE – END = 10 – 9.1 = 0.9

$\Sigma_G{}^1$ → execution time of the completed portion of current job J_{ci}.

t_c → slack computation time

u_i…… → unit step function is 0 for + <0, is 1 for t >0

Q_i →release time.

Scheduling of Sporadic Jobs:

In our subsequent discussion, we refer to each individual sporadic job as s_i. When we want to call attention to the fact that the job is released at time t and has maximum execution time e and (absolute) deadline d, we call the job s_i(t, d, e).

❖ ***A simple Acceptance test in Deadline Driven System:***

a. Acceptance test procedure:
The acceptance test on the first sporadic job s(t, d, e) is simple indeed.
The scheduler accepts s if its density $\frac{e}{d-t}$ is no greater than 1-Δ.
Let I_1 be the time interval containing the deadline d of the new sporadic job s (t, d, e). The scheduler accepts the job s if
$\frac{e}{d-t}$ +Δ, k ≤ 1- Δ ∀ k= 1, 2.....n
Δ ← Density

Real Time Performance for Jobs With Soft Timing Constraints:

❖ Traffic Models: Performance based on maximum tardiness and miss rate of jobs in it.
a. F_e V_e and (λ, β) Models:
According to the F_e V_e model, each sporadic task is ..... b 4 type (e, p, T, I) i, e is the maximum execution time of all jobs. In the task, p is the minimum interval time of the jobs, T is their average interval time, and this average interval time, and this average is taken over a time interval of length I.
The (λ, β), model, characterize each sporadic task by a rate parameter λ and a burst parameter β. The total execution time of all jobs that are released in any time interval of length x is no more than β + λx.
b. Leaxy Buket Model:

To define the leaky bucket model, we first introduce the notion of a (Δ, ∈) leaky bucket filter. Such a filter is specified by its (input) rate Δ and size∈. the filter can hold at most ∈ tokens at any time and it is being filled with tokens at a constant rate of Δ tokens per unit time. A token is lost if it arrives at the filter when the filter already contains ∈ tokens.

A two-level scheme for integrated scheduling:

By design, the two-level scheme allows different applications to use different scheduling algorithms (e.g. some may be clock-driven manner while others in priority driven manner.)

- Overview & Terminology:

  OS Scheduler ← lower level scheduler in replenishes the budget and sets the deadline of each server in the manner described below.

  Server scheduler← Higher Level Scheduler, this scheduler schedules the job in the application to the algorithm chosen for the application.

  a. Required Capacity:

  $\frac{1}{5} > 1$

  S← processor speed

  The minimum fraction of speed at which the application is schedulable is called its required capacity s_i. The required capacity of every application must be less than 1.

  b. Predictable versus Non-predictable Applications:

  We call an application that is scheduled according to a preemptivem priority driven algorithm and contains aperiodic and sporadic tasks and /or periodic tasks with release time jitters an unpredictable application.

  The season for this name is that the OS scheduler needs an estimate of its next event time at each replenishment time of its server but its server scheduler cannot compute an accurate estimate.

  All other types of applications are predictable. An application that contains only periodic tasks with fixed release times and known resource request times is predictable because its server scheduler can compute accurately the occurrence times of its future events.

- ***Scheduling Predictable Applications:***

  a. Non- preemptively scheduled Applications: Specifically, according to the two-level scheme, each non preemptively scheduled application T_i is executed by a constant utilization server whose size $\tilde{u}_i$ is equal to required capacity s_i of the application.

  b. Applications Scheduled according to a cyclic Schedule: This can also be scheduled according to a cyclic schedule is also executed by a constant utilization server of size equal to s_i.

  c. Preemptivity Scheduled Predictable Applications: As with other predictable applications, the size $\tilde{u}_i$ of the server for a predictable application T_i, that is scheduled in a preemptive, priority-driven manner is equal to its required capacity. However, the OS scheduler cannot maintain the server according to the constant utilization/total bandwidth server algorithms.

- ***Scheduling Non-predictable Applications:***

  a. Budget Replenishment:

  In the absence of any knowledge of the release-times of jobs in an application, the OS scheduler replenishes the budget of its server every ∈ units of time. Hence, the smaller the quantum size, the more frequent the server replenishments, and the higher the server maintenance overhead.

  b. Required Server Size:

  $$\tilde{u}_i = s_i \frac{D_i\, min}{D_i\, \min - \epsilon}$$

  D_i min → minimum relative deadline

  ∈ → positive constants.

Questions:

- ✓ Explain about different jobs executed in priority manner by the operating systems.
- ✓ What are the alternative approaches of priority systems?
- ✓ Explain about bandwidth preserving servers.
- ✓ Distinguish between Polled Executions and Bandwidth preserving servers.
- ✓ Explain the consumption and replenishment rule for simple sporadic servers.
- ✓ What are the enhancements of fixed priority sporadic servers?
- ✓ Explain the consumption and replenishments rule for constant utilization servers.
- ✓ Define static and dynamic slack computation in detail.
- ✓ How do you perform acceptance test in deadline driven systems?
- ✓ Explain about Real time performance Models.
- ✓ What are the two level schemes for integrated scheduling?
- ✓ How do you schedule predictable applications? Explain.
- ✓ How do you schedule non predictable applications? Explain.

Unit 8: Resource and Resource Access Control

A. Assumptions on resources and their usages
B. Effects of resources contention and resource access control
C. Non preemptive critical sections
D. Basic priority –inheritance protocol
E. Basic priority-ceiling protocol
F. Stack-based priority ceiling protocol
G. Use of priority-ceiling protocol in dynamic-priority system
H. Controlling accesses to multiple-unit resources
I. Controlling concurrent accesses to data objects

❖ ***Assumption on Resources and their Usage:***

We consider the system contains only one processor. In addition the system also contains …….. types of serially reusable resources named R_1, R_2…….. R_3.

When a unit of a resource R_i is granted to a job, this unit is no longer available to other jobs until the job froes the unit. Again, examples of ……. Resources are mutates, reader/writers locks printers & remote sewers.

❖ ***Enforcement of Mutual Exclusion and critical sections:***

We assume that 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 look to request them. We denote this lock request by L (R_i, n_i). The job continues its execution when it is granted the requested resource.

For unlock U (R_i, n_i), when a resource R_i has only 1 unit, we use the simpler notation L(R_i) and U(R_i) for lock and unlock respectively.

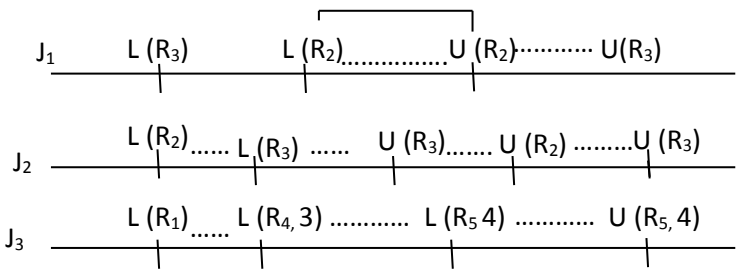


Figure 8.1 Locking and Unlocking in Critical Sections

❖ ***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.
The jobs contend for a resource when one job requests a resource that the other job already has.
When the scheduler does not grant n_i units of resource R_i to the job requesting them, the lock request $L(R_i, n_i)$ of the job fails (or is denied). When its lock request fails, the job is blocked and loses the processor. It stays blocked until the scheduler grants it n_i units of R_i for which job is waiting. At that time, the job becomes unblocked, is moved backed to the ready job queue, and executes when it is scheduled.
Critical section (R, n, i, e)
n ← member of unit n
e ← execution time of critical section(……)

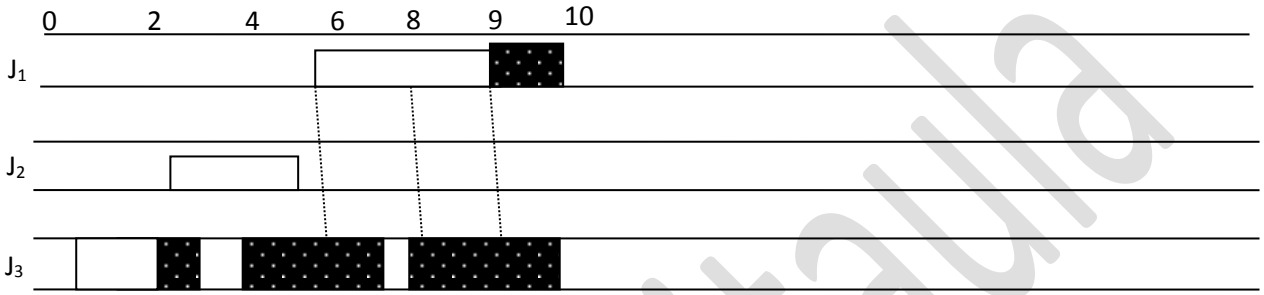


Figure 8.2 Resource Conflicts

1. At time 0, only J_3 is ready, it executes.
2. At time 1, J_3 is granted the resource R when it executes L(R).
3. J_2 is released at time 2, preempts J_3 and begins to execute.
4. At time 4, J_2 tries to lock R. because R is in use by J_3, this lock request fails. J_2 becomes blocked, and J_3 regains the processor and begins to execute.
5. At time 6, J_1 becomes ready, preempts J_3 and begins to execute.
6. J_i executes until time 8 when it executes a L(R) to request R. J_3 still has the resource. Consequently, J_1 becomes blocked. Only J_3 is ready for execution and it again has the processor and executes.
7. The critical section of J_3 completes at some points & it can be used.

Effects of Resource contention & Resource Access Control:

A resource access-control protocol, or simply an access-control protocol, is a set of rules that govern
1. When and under what conditions each request for granted &
2. How jobs requiring resources are scheduled.

❖ ***Priority Inversion, Timing Anomalies and Deadlock:***

We saw that priority inversion can occur when the execution of some jobs or portions of jobs is non preemptable. Resource contentions among jobs can also cause priority inversion. Because resources are allocated to jobs on a non preemptive basis, a higher-priority job can be blocked by a lower-priority job jf the job conflict, even when the execution of both job is preemptable- priority inversion occurs in intervals (4,6) & (8,9).
When priority inversion occurs, timing anomalies invariably follow.

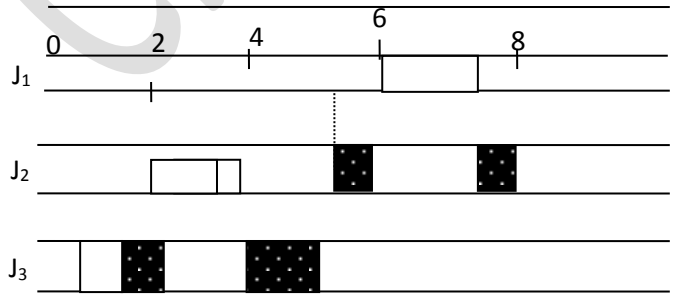


Figure 8.3 Priority Inversion

The three jobs J_1, J_2 & J_3 are shown in the above diagram. The critical section in J_3 is (R, 2.5).

Our intuition might tell us that as a consequence of this reduction in J_3's execution time, all jobs should complete sooner. Indeed, this reduction does allow jobs J_2 and J_3 to complete sooner.

More seriously, without good resource access control, the duration of a priority inversion can be unbounded.

❖ ***Additional Terms, Notations and Assumptions:***

We use J_n and J_l to denote a higher-priority job and a lower-priority job respectively. The priorities of these jobs are denote by π_n and π_1 respectively.

A higher-priority job J_n is said to be directly blocked by a lower-priority job J_l when J_h holds some resource which J_n requests and is not allocated. In figure (1) J_3 blocks J_2 at time 4.

We describe the dynamic-blocking relationship among jobs using a wait for graph. In the wait-for graph of a system, every job that requires some resource is represented by vertex labeled by the name of the job.

A cyclic path in a wait for graph indicates a deadlock.

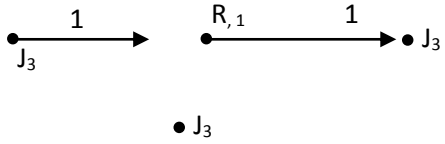


Figure 8.4 wait-for graph

R, $_1$ ← name and number of unit or resource

edge ← if x, x units of resource are allocated. It represents the state of the system in above figure. The resource R is allocated to J_3 and J_2 is waiting for the resource. The path from J_2 to J_3 indicates that J_2 is directly blocked by J_3, later, J_1 will also be directly blocked by J_3 when it requests and is denied the resource, and the wait for graph of the system will have an additional edge from J_1 to R. since, there is only one resource, deadlock can never occur.

Non Preemptive Critical Sections:

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

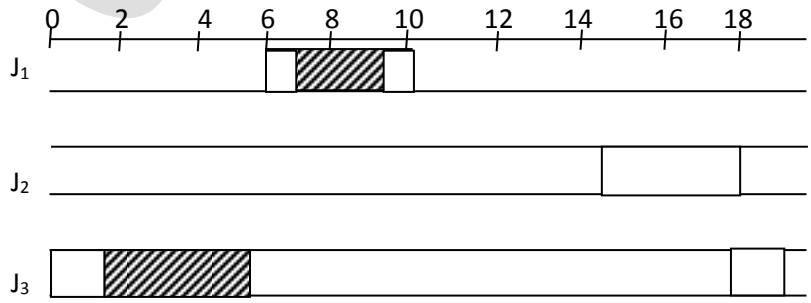


Figure 8.5 Non Preemptive critical sections

Figure (3) shows the schedule of these jobs when their critical sections are scheduled non preemptively on the processor. However, as soon as J_3 release the resource, J_1 becomes unblocked and executes to completion. Because J_1 is not delayed by J_2, it completes at time 10 rather than 14 in figure (1)

Basic Priority-Inheritance Protocol:

- Definition of Basic Priority Inheritance Protocol:-
  In the definition of the protocol, as well as other protocols described later, we call the priority that is assigned to a job according to the scheduling algorithm its assigned priority.

***Rules for Basic 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 $\pi(t)$ of every job J is equal to its assigned priority.
2. **Allocation Rule:** When a job J requests a resource R at time t
   a. if R I free, R is allocated to J until J release the resource, and
   b. 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_1 which blocks J inherits the current priority π (t) of J. The job J_1 executes at its inherited priority $\pi(t)$ until it release R, at that time, the priority of J_2 returns to its priority π_1 (t^1) at t^1 when it acquires the resource R.

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

Table 8.1 Table with Critical Section representation

- **Properties of the priority inheritance Protocol:**
  This example illustrate that when resource accesses are controlled by the priority inheritance protocol there are two types of blocking.

→priority –inheritance blocking(or simply inheritance blocking): The priority-inheritance protocol does not reduce the blocking time suffered by jobs as small as possible.

Basic Priority-Ceiling Protocol:

The priority-ceiling protocol extends the priority-inheritance protocol to prevent deadlocks and to further reduce the blocking time. This protocol makes two key assumptions.

→ The assigned priorities are fixed

→ The resources required by all jobs are known a priori before the execution of nay job begins.

To define the protocol, we need two additional terms. The protocol makes use of a parameter, called priority ceiling, of every resource. The priority ceiling of any resource R_i is the highest priority of all the jobs that require R_i and is denoted by $\pi(R_i)$.

For example,

π (Block) = 2

π(Black) =2

π (Black) = 4 } minimum means highest priority.

π (Black) =5

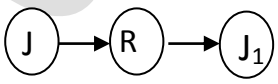
❖ **Definition of the Basic Priority-ceiling Protocol:**

Rules of Basic Priority-ceiling Protocol:

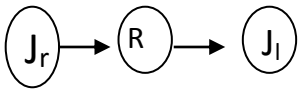
1. **Scheduling Rule:**
   a. At its release time t, the current priority π (t) of every job J is equal to its assigned priority. The job remains at his priority except under the condition stated in rule 3.
   b. Every ready job J is scheduled preemptively and in a priority driven manner at its current 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.
      i. If J's priority π (t) is higher than the current priority ceiling $\hat{\pi}$ (t) R is allocated to J.
      ii. If J's priority π (t) is not higher than the ceiling $\hat{\pi}$ (t) of the system R is allocated to J only if J is the job holding the resource(s) whose priority ceiling is equal to $\hat{\pi}$ (t). otherwise, J's request is denied, and J is blocked
3. **Priority-Inheritance Rule:** When J becomes blocked, the job J_1 which blocks J inherits the current priority π(t) of J. J_1 executes of 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, the priority of J_1 returns to its priority π_1 (t^1) at the time t^1 when it was granted the resource.

❖ ***Difference Between the Priority-Inheritance and Priority-Ceiling Protocols:***

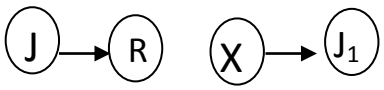
A fundamental difference between the priority-inheritance and priority-ceiling protocols is that the former is greedy while the latter is not, you may recall that the allocation rule of the priority-inheritance protocol lets the requesting job have a resource whenever the resource is free. In contrast, according to the allocation rule of the priority-ceiling protocol, a job may be denied its requested resource even when the resource is free at the time.



a. Direct blocking



b. Priority-Inheritance blocking



Π (x) is equal to or higher than (x)

c. Avoidance blocking

❖ **Duration of Blocking**:

Computation of Blocking Time:

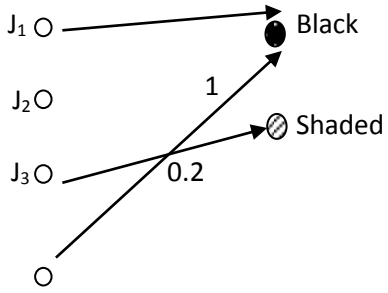


Figure 8.6 Example of blocking:

When resource accesses of preemptive priority-driven jobs on one processor are controlled by the priority-ceiling protocol, a job can be blocked for at most the duration of one critical section.

To illustrate how to do this computation, let us consider the system of jobs whose resource requirement are given above. As always, the jobs are indexed in order of decreasing priorities. We see that J_1 can be directed blocked by j_4 for 1 unit of time. The blocking time $b_1(r_c)$ of J_1 is clearly 1. Although J_2 and J_3 denote require the resource Block, they can be priority inheritance blocked by J_4 since J_4 can inherit priority J_1. Hence, the blocking times $b_2(r_c)$ and $b_3(r_c)$ are also one.

Stack-Based, Priority-Ceiling Protocol:

In this section, we two different definitions of a protocol that is simple than the priority-ceiling protocol but has the same worst-case performance as the priority-ceiling protocol.

❖ **Motivation and Definition of Stack-Sharing Priority-Ceiling Protocol:**

Rules Defining Basic Stack-Based Priority –Ceiling Protocol:

0. Update of the current ceiling: whenever all the resources are free, the ceiling of the system is …….. The ceiling $\hat{\pi}(t)$ is updated each time a resource is allocated or freed.
1. Scheduling Rule: After a job is released, it is blocked from starting execution until its assigned priority is higher than the current ceiling $\hat{\pi}(t)$ of the system. At all times, jobs that are not blocked are scheduled on the processor in a priority driven, preemptive manner according to their assigned priorities.
2. Allocation Rule: Whenever a job requests a resource, it is allocated the resource.

❖ **Definition of Ceiling-Priority Protocol:**

Rules Defining the Ceiling-Priority Protocol:

1. Scheduling Rule:
   a. Every job executes at its assigned priority when it does not hold any resource. Jobs of the same priority are scheduled on the FIFO basis.
   b. The priority of each job holding any resource is equal to the highest of the priority ceiling of all resources held by jobs.
2. Allocation Rule: Whenever a job requests a resource, it is allocated the resource.

❖ Preemption Levels of Job and Periodic Tasks:

Validity condition: If π is higher than π_k and $r_i > r_k$, then ψ_i is higher than ψ_k.

Preemption = ψ

❖ **Definition of Protocols and Duration of Blocking:** A preemption –ceiling protocol makes decisions on whether to grant a free resource to any job based on the preemption level of the job in a manner similer to the priority-ceiling protocol. This protocol also assumes that the resource requirements of all the jobs are known a priori.

Controlling Accesses to multiple Unit Resource:

❖ **Priority(Preemption) Ceilings of Multiple-Unit Resources:**

If one or more jobs in the system require more than k units of R_i, $\pi(R_i, k)$ is the highest priority of all these jobs. The preemption ceilings of resources that have multiple units can be defined in a similar manner. The preemption ceiling $\psi(R_i, k)$ of the resource R_i when k units of R_i are free is the highest preemption level of all the jobs that require more than k units of R_i.

Resource	x(2)	y(3)	z(1)
• J_1	1	0	0
• J_2	0	2	0
• J_3	0	2	0
• J_4	0	0	0

Table 8.2 jobs resources with their priority

❖ **Priority-Inheritance Rule:**

When the requesting job J becomes blocked at t, the job with the highest priority among all the jobs holding the resource R that has the highest priority among all the jobs holding the resource R that has the highest priority among all the resource inherits J'_8 priority until it releases its unit or R.

Controlling Concurrent Access to Data Objects:

Data objects are a special type of shared resources. When jobs are scheduled preemptively, their access to data objects may be interfered.

❖ **Convex-Ceiling Protcol:**

For example, both-the NPCS and PC (Priority-and Preemption-Ceiling) protocols allow a higher priority job J_n to……. And write a data object x between two disjoint critical section.

❖ ***Motivation & Assumptions:***

A well known way to ensure…… Is Two –phase Locking (2 pL), according to the 2 pL protocol, a job never requests any lock once it release some lock.
We can easily get concurrency control protocols that not only ensure serializability but also prevent deadlock and transitive blocking by augmenting the protocols.
The augmented protocols have an obvious short comings prolonged blocking.
Following the 2 pL rule, a job may hold a data object even when it no longer require the object. As a consequence, it may block other jobs for a longer duration.
The convex(priority) ceiling protocol described below s another extension of priority-ceiling protocol. It is an improvement over the pcp-2pL protocol because it reduces the duration of blocking.

❖ ***Other Real-Time Concurrency Control Scheme:***

a. Well-known conflict Resolution Policies: To explain, conflict resolution policies that have been proposed for database transactions, we note that each transaction typically keeps a copy of each object it reads and may write in its own memory space. When it completes all the reads, writes and computation, it writes all the data objects it has modified back to the global space. The last step is called commit. So, until a transaction commits, no shared data object in the database is modified, and take back the data objects allocated to it.

b. Optimistic Concurrency Control Scheme: optimistic concurrency control is an alternative approach to two-phase locking. Under the control of an occ scheme, whether a transaction conflict with other executing transaction is checked immediately before the transaction commits. This step is called validation. If the

c. one of them is allowed to proceed and commit, while the conflicting transactions are restarted.

Questions:

- ✓ What do you know about mutual exclusion and its necessity for preserving critical sections?
- ✓ Explain about resource conflict and blocking.
- ✓ What are the effects of resource contention and resource access control?
- ✓ What do you know about Non Preemptive Critical Sections?
- ✓ Explain about the rules of Basic Priority Inheritance Protocol.
- ✓ Explain about the rules of Basic Priority Ceiling Protocol.
- ✓ Distinguish between Basic Priority Inheritance Protocol and Basic Priority Ceiling Protocol.
- ✓ What do you know about duration of blocking? Explain with better illustrations.
- ✓ What do you know about stack based priority ceiling protocol?
- ✓ Define ceiling priority protocol.
- ✓ Define preemption ceiling protocol.
- ✓ How do you control accesses to multiple unit resources?
- ✓ How do you control concurrent accesses to data objects?
- ✓ What are the concurrency control schemes?

Unit 9: Multiprocessor Scheduling and Resource Access Control and Synchronization

A. Model of multiprocessor and distributed systems
B. Task assignment
C. Multiprocessor priority-celing protocol
D. Elements of scheduling algorithms for end-end periodic tasks
E. End-to-end tasks in heterogeneous systems
F. Predictability and validation of dynamic multiprocessor systems

Thus for, we have ignored served realistic facts. Almost every system contains more than one processor, control and data dependencies impose precedence constraints among jobs, and timing constraints of jobs are usually not independent.

Model of ***multiprocessor and distributed system:***

A multiprocessor system is tightly coupled so that global status and workload information on all processor can be kept current at a low cost. The system may use a centralized dispatcher/scheduler.
In contrast, a distributed system is …… coupled, in such, it is costly to keep global status and workload information current.

❖ ***Identical Versus Heterogeneous Processor:***

We say that processors are of the some type or they are identical if the processors can be used interchangeably. For example, each of the CPUs in a paralled machine can execute every computation job in the system; therefore the CPUs are identical. If any massage from a source to a destination can be sent on any of the data links connecting them, then the links are identical.
In contrast, processor of different types cannot be used interchangeably; Different types of processors may be functionally different. For example, CPUs file disks, and …….. links are functionally different. Obviously, they can't be used interchangeably.

❖ ***End-to-End Jobs and Tasks:***

A task in a real-time monitor system may consists of three jobs sampling, encoding and processing the reading of a sensor on a field processor; sending the sensor data by communication processor to the control processor ; and correlation and displaying the data with other sensor data on the control processor.

a. Job Shops and Flow Shops: The classical job shop and flow shops capture the type of concurrency, according to the job, shop model, each task T_i in the system is a chain of r(i) jobs, denoted by J_i, k for k= 1, 2, ……n(i). For all $l \leq k \leq n(i)$, adjacent jobs J_i, k +1 on the chain execute on different processor. A flow shop is a special job shop in which all tasks have the same visit sequence.
   The visit sequences of all the monitor tasks are (field processor, communicable processor, control processor) if the system has only one processor of each of three types. This system can be molded as a flow shop.
b. End to End Timing Constraints: The timing constraints that can be derived directly from the high level requirements of the application are typically end to end in nature. They give the release time and deadline of

each task as a whole. For example, suppose that the surveillance system is required to issue a command for an evasive action whenever a collision is detected. The high level requirement that a current evasive action impose an overall relative deadline on the collision avoidance task associated with each target monitored by the system.

c. Periodic End-to End Tasks: The second difference between the classical job shop model and ours is mostly a matter of notation. Just like periodic tasks in a uniprocessor environment, end-to-end tasks in a multiprocessor system may be periodic. An end-to-end task T_i is periodic with period P_i if a chain of n(i) jobs is released every P_i units of time and jobs in the chain execute in their on processor according to the visit sequence ($V_{i,1}$, $V_{i,2}$,..... $V_{i,n(i)}$,

d. Parallelism: In addition to pipelined execution of jobs on different types of processors, paralle executions are possible whenever there is more than one processor of the same type.

❖ ***Local Versus Global Resource:***

We now describe two resource models the MPCP model and the end to end model; they give us two somewhat different views of a multiprocessors system.

a. MPCP Resource Model: This model calls the processor on which each resource resides its synchronization processor. From the prospective of a job, a resource that also resides on the local processor of the job is local resource and a resource that resides on another processor is a remote resource. A global resource is required by jobs that have different local processors.

In summary, according to the MPCP model, a job may require both local and remote resources. To access to each resource is controlled by the scheduler of the synchronization processor of the resource.

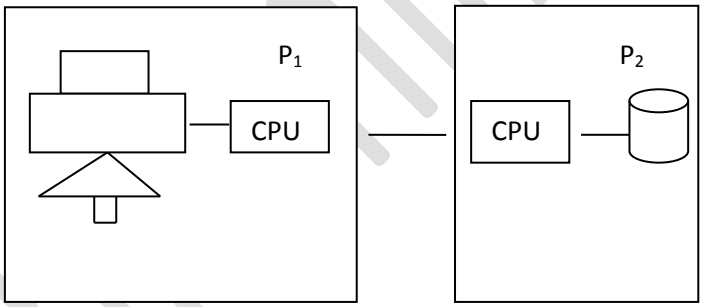


Figure 9.1 Multiple processor with resources

b. End –to –End Resource Model: A reasonable restriction is that no job makes nested requests for resources that reside on different processors. In a system, that satisfies this assumption, we can view each job that requires resource on more than one processor as an end-to-end job. Each job in an end-to-end task requires only resources on its local processor.

❖ ***Interprocessor Communication:*** A special case of practical interest is where interprocessor communication is via shared memory. Sometimes, we treat shared memory as a plentiful resource and do not include this resource in our model.

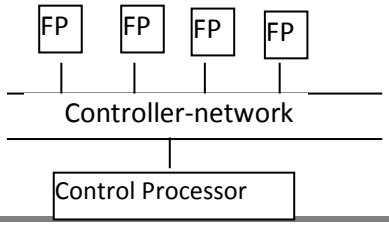


Figure 9.2 Inter processor Communication

Task Assignemnt:

If the system is static, as most current hard-real time systems are, the application system is partitioned into modules, and modules are assigned and bound to processors. We call this step task assignment. Oftentimes, task assignment is done off-line. At some stage in the design and development process of a real –time system, the execution time, resource requirements, data and control dependencies, and timing constraints of all the tasks become known.

- ***Task Assignment Based on Execution Time Requirements:***

Sometimes, it is appropriate to consider only the processing time requirements of the jobs and tasks and ignore communicable costs. For some application (e.g. signal processing), it is possible to provide a sufficient number of memory modules and carefully lay out the address space of tasks to minimize memory contention for these applications, the cost of communication can be made negligibly small.

a. Simple Prin-Packing Formulation: In its simplest form, the task assignments problem can be stated as follows: we are given the utilizations of n periodic tasks, we are asked to partition the system into modules in such a way that tasks in each module are schedulable by themselves on a processor according to a uniprocessors scheduling algorithm of a given class.
   A task assignment is defined by the subject of tasks in every module, that is, the tasks assigned to each processor.
   The quality of a task assignment is measured solely by the number of processor required by the assigned. The smaller the number of processors required by an assigned, the better the assignment.

b. Veriable-Size Bin-Packing Formulation: Example: RMFF Algorithm: the rate –Monotonic First fit algorithm is such a haristic algorithm. According to this algorithm, tasks are first sorted in non decreasing order according to their periods.

- ***Task Assignment to minimize total communication cost:*** When CPUs are connected via some kind of network, the cost of communication between a pair of task is usually significantly; lower when they are on different CPU. In this situation, we want to take into account communication cost when we partition the tasks into modules and assign them to CPUs. The goal of task assignment is, therefore, to partitions all assign each module to a processor in such a way that the number of required processor to feasible schedule all the tasks is minimum and some function of communication cost among tasks on different modules is minimum.

Integration of Task and Resource Assignments:

a. End to End Model: Suppose we are given a task graph description of the system in which what resources each periodic task uses and when the task uses them are specified by the resource parameters of the task. To determine the placements of resource simultaneously while the assignments of periodic tasks, we expand the task graph in the way described below. To capture the resource usage constraints and costs and end-to-end of the tasks.
   - Decomposing Tasks into chains of subtasks.
   - Specifying Resource Usage Constraints & Interpreting the solution.

Multiprocessors Priority Ceiling Protocol:

This section describes the Multiprocessors Priority-Ceiling Protocol (MPCP). This protocol assumes that tasks and resources have been assigned and statically bound to processor and that the scheduler of every synchronization processor knows the priorities and resource requirements of all the tasks requiring the global resources managed by the processor.

In the MPCP, if the global critical section of a remote task were to have a lower priority than some local tasks on the synchronization processors, these local tasks could delay the completion of the global critical section and prolong the blocking time of the remote task. To prevent this, the Multiprocessor Priority Ceiling Protocol Schedules all global critical section at higher priorities than all local tasks on every synchronization processor. This can easily e implemented in a system where the lowest priority π lowest of all tasks is known.

❖ ***Blocking Time Due to Resource Contention:***

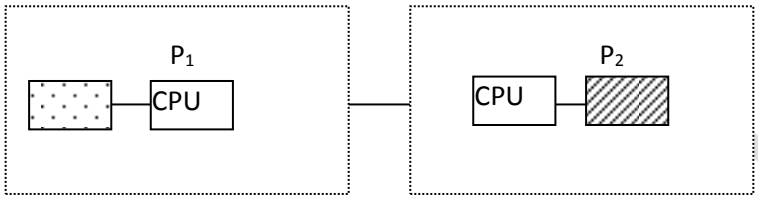


Figure 9.2 Effects of Blocking time due to resource contention

❖ ***Upper Bounds to Factors of Blocking Time :*** $b_i(r_c)$

a. local blocking time, which is due to its contention for resources on its local processors.
b. local preemption delay, which is due to preemption of T_i by global critical sections that belong to remote tasks but execute on its processors.
c. Remote blocking time, which is due to its contention with some lower priority tasks for remote resources on the synchronization processor(s) of the resource(s).
d. remote preemption delay, which is due to preemptions by higher priority global critical sections of synchronization processors of the remote resources required by T_i ; and
e. deferred blocking time, which is due to the suspended execution of local higher-priority tasks.

Element of scheduling algorithm for end-to-end periodic tasks:

The two essential components of any end-to-end scheduling schemes are

a. Protocol (s) for synchronizing the execution of sibling subtasks on different processors so that precedence constraints among subtasks are maintained.
b. Algorithms for scheduling subtasks on each processor. This section describes choices for these elements and their pros and cons.

❖ ***Inter Processor Synchronization Protocols:***

A synchronization protocol is correct if it

1. never releases jobs in any first subtasks, before the end-to-end release times of the jobs.
2. never allows the violation of any precedence constraint among sibling subtasks.

There are two types of synchronization protocols: greedy (or work an serving) and nongreedy(or nonwork conserving)

When sibling subtasks are synchronized according to the greedy synchronization, the j^{th} job $J_{j, K+1\ j}$ of a subtasks $T_{i\ K+1}$ is release on $V_{i,\ k+1}$ as soon as its immediate predecessor $J_{i,\ k\ I}$ j complets on $V_{i,\ k}$ ∀ k = 1,, 2…… n(i)-1.

On the other hand, if the subtasks are synchronized according to a nongreedy synchronization protocol, the completion time of $J_{i,\ k}$; j is the earliest possible release time of $J_{i\ k+1}$; j. the scheduler may delay the release of $J_{i\ k+1}$; j even though its immediate predecessor has already completed.

❖ ***Scheduling of Subtasks on Each Processor:***

Again, we focus here on multiprocessor system in which subtasks on every processor are scheduled according to some priority-driven algorithm. We can use mixture of fixed-priority and dynamic-priority algorithms. Similarly, we can use different processors.

a. Heuristic for Priority Assignments:
b. Deadline-Assignment Algorithms:

Scheduling of Fixed-Priority End-to-End Periodic Tasks:

In this section, we suppose that very subtask $T_{i\,k}$ in the system is assigned a fixed priority …… and ask whether every task can meet its end-to-end deadlines. To answer this question, we need to consider two cases: when tasks are synchronized nongreedily & greedily.

❖ ***Schedulibility of Nongreedy Synchronized Tasks:***

a. Upper Bounds to End-to-End Response Time: When sibling subtasks are synchronized according to any of the non greedy protocols described above, we can treat each subtask $T_{i,\,k}$ like a periodic task. When trying to find its maximum possible response time. The period of this subtask is equal to the period P_i of its parent task T_i and its execution time is $e_{i,k.}$

b. Schedulability of Tasks under Greedy Synchronization: This subsection describes an algorithm that can be used to compute upper bounds to end-to-end response times of tasks synchronized according to the greedy synchronization protocol. This algorithm, called, SA/Ds (Schedulability Analysis for Direct Synchronization Protocol
Rather, to find and upper bound w_i to the end-to-end response time of each task T_i, algorithm Sa/Ds first tries to find on upper bond $I_{w,\,k}$ to the intermediate end-to-end response time of each subtask $T_{i,\,k}$ of the task. The intermediate endd-to-end response time $T_{i,\,k}$ is the maximum length of time between the release of a job in $T_{i,\,1}$ in the first subtask to the completion of the corresponding job in $T_{i,k}$.
By definition, the end-to-end response time of T_i is the intermediate end-to-end response time of its last subtask $T_{i,\,n(i)}$. Hence, $w_i = IW_{i,\,n(i)}$.
Specially, algorithm SA/Ds is an interactive algorithm. It takes an input a set $\{ IW_{i,k}^{(o)} \}$ of initial estimates of intermediate end-to-end response times of all subtasks in the system. During each interaction, say Th (n+1)st, the algorithm uses the set $IW_{i,k}^{n}$ of estimates produced in the n^{th} interaction as input and produces as output a new set of estimates$\{ IW_{i,k}^{n+1} \}$.
If the output estimate for every subtask is equal to the input estimate for the subtask, then the bound $IW_{i,k}^{n+1}$ produced during the iteration is a correct upper bound $IW_{i,k}$ ot he intermediate end-to-end response time of $T_{i,k}$ and an upper bound w_i to the end-to-end response time of the task T_i is equal to $IW_{I,\,e\,n(i)}$ if not equal, another iteration is carried out.

End –to–End Tasks in Heterogeneous System:

Thus for, our discussion on end-to-end scheduling has assumed that a priority-driven scheme is used to schedule every processor. We now remove this restrictive assumption. It is not valid for most real life systems. All but the simplest embedded systems contains different types processors (e.g. CPUs, Disks and networks). Typically, a verity of approaches are taken to scheduling tasks on these processor rather than digressing here to describe some of them, we postpone their descriptions until later chapters and focus here on end-to-end scheduling issues.
In general, during an acceptance ……. to determine whether to admit a new end-to-end periodic task, one or more schedulers choose the processors used to execute its subtasks, that is, the visit sequence of the task. Then the schedulers of the processors its in the visit sequence determine the local relative deadline of its subtasks either sequentially, as in switched network or parallel. The task is acceptable if the sum of the individual local relative deadlines is no greater than the end-to-end relative deadline requested by the new task.

Predictability & Validation of Dynamic Multiprocessor Systems:

Predictable execution of a set J of jobs is defined in terms of three possible scheduler of J according to given scheduling algorithms: the maximal, minimal and actual schedules. The actual start time $s(j_i)$ of a job J_i according to the actual schedule of J can be later (or earlier) than its start time $s^t(J_i)$ [or $s^-(J_i)$] according to the maximal schedule (or minimal schedule)

The start time of J_i is said to be unpredictable when this condition occurs similarly, the actual completion time of (J_i) of J_i according to the actual schedule of J can be later (or earlier) than its completion time $f^t(J_i)$.
[or $f^-(J_i)$] according to the maximal schedule or minimal schedule; the completion time of J_i is unpredictable of this is true. On the other hand if $s^-(J_i) \leq s(J_i) \leq s^t(J_i)$, then J_i is start-time predictable, and if $f^-(J_i) \leq f(J_i) \leq f^t(J_i)$, then J_i is completion-time predictable.

- ***Validation of Preemptable/Migratable Systems:*** A preemptable/migratable system is a dynamic system in which jobs are scheduled in a priority-driven manner on in processor. Every job can be dispatched to execute on any processor, can be preempted can be resumed on any processor. In other words, the job can migrate among processors. A condition for predictability of a preemptable/migratable system is that jobs have no precedence constraints and do not contend for resources.

Questions:

- ✓ What do you mean by multiprocessor and distributed systems? Explain about identical and heterogeneous processor.
- ✓ Define end-to-end jobs and tasks. What are its constituents? Explain.
- ✓ What are the modes used while discussing about Local and Global resources?
- ✓ What do you mean by Task Assignments? What such operations are performed?
- ✓ How to integrate Task and Resource Assignments? Explain.
- ✓ What do you know about Multiprocessor priority ceiling protocol? Explain about blocking time and its upper bounds.
- ✓ Explain about the elements of scheduling algorithms for end-to-end periodic tasks.
- ✓ What are the strategies in order to schedule fixed priority end-to-end task?
- ✓ How to predict and validate dynamic multiprocessor systems?

Unit 10: Real Time Communication

A. Model of real-time communication
B. Priority-based service disciplines for switched networks
C. Weighted round-robin service disciplines
D. Medium access-control protocols of broadcast networks
E. Internet and resource reservation protocols
F. Real time protocol
G. Communication in multi computer systems
H. Signal Processing
I. Real time application

This chapter discuss communication and operating system support for real-time applications.
This chapter is devoted to network services for multiprocessor and distributed real-time applications.
Quality of service is a term commonly used to mean a collection of figures of merits, such as performance, reliability and reconfigurability. We are concerned mainly with real-time performance and we use the team's performance and quality of service interchangeably.

Model of real-time communication:

- Architectural Overview: We focus on message exchanged among applications on different hosts. In others, the source and destinations(s) of every message are application tasks residing on different hosts.

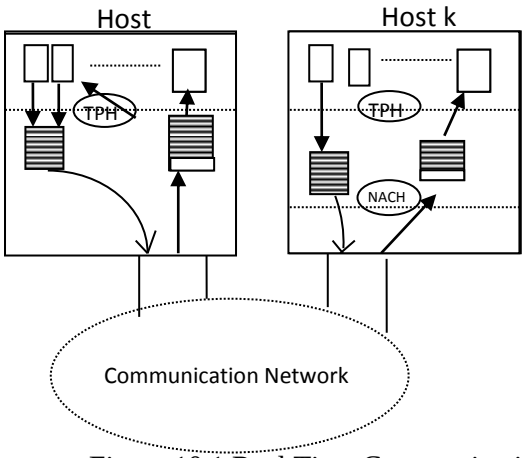


Figure 10.1 Real Time Communication

For the safe of concrete, we assume that these queues are jointly maintained by two (local) servers: the Transport Protocol (TP) handler and Network Access control (NAC) handler. The former interfaces with local applications and provide then with message transport services. The latter interfaces with the network below and provides network-access & message-transmission services to the TP handler.

❖ ***Packets, Network Bandwidth and Physical Size:***

Massage are typically fragmented into segments for their transmission through the communication network. Each segment is handled by the network as a basic transmission unit and the transmission of the unit is nonpreemptable. Such a unit is called a frame, a packed, a cell and the like, in different types of networks. We use the name pocket to refer to all these basic transmission units. We say that the link bandwidth is one (one packet per unit time). When we say that a message stream is allocated a bandwidth of ῦ, we always mean that message is given the fraction ῦ of the link bandwidth.

For example, the time required to transmit a 53-byte packet over a 100-MHg link is 4-24 microsecond and hence a time unit is 4-24 microseconds (plus processing overhand).

For example, a message of length 100 has 100 × 53 = 5.3 k bytes if each packet has 53 bytes, and the length of time to transmit the message is 100 time units.

❖ ***Real-Time Traffic Models:***

In real-time communications literature, the term real-time traffic typically means isochronous (or synchronous) traffic, consisting of message streams that are generated by their sources on a continuing basis and a directed to their respective destinations on a continuing basis, such traffic includes periodic and sporadic messages, which requires some degree of guarantee for on-time delivery. Aperiodic message have soft timing constraints and expect the system to deliver them on a best-effort basis.

a. Period & Aperiodic Message: Period message streams, or simply periodic message, are generated and consumed by periodic tasks, and their characteristics are similar to the characteristic of their respective source tasks. There are also aperiodic message steams; the transmission of an aperiodic message stream is an aperiodic task. Like aperiodic tasks, an aperiodic message stream does not have a relative deadline.
b. Sporadic Messages: A periodic message is an inaccurate model of any sporadic stream whose instances have widely varying length and/or interarrival time. examples include a MPEG-compressed video stream because the length of frames in it vary widely, and command and control messages generated in response to unexpected events because the inter arrival times of messages instance vary widely.

❖ ***Performance Objectives & Constraints;***

We want to measure the performance of scheduling, synchronization, and flow control algorithms used for real-time communication and performance of the resultant communication system along two dimensions from the point of view of the user and the system.

a. Miss, Loss and Invalid Rates: The miss rate of a message stream or a set of messages stream is the fraction of all message instances or packets that are delivered to their destinations too late. When a queue is full or when the queue length reaches a certain (drop) threshold, some packet (s) designated for the queue are dropped. As consequence, some packets may be lost.
b. Delay Jitter, Buffer Requirement, and Throughput: The delay jitter of a message-stream is the verification (e.g. the maximum or average absolute difference) in the delays suffered by different message instants of packets in the stream.

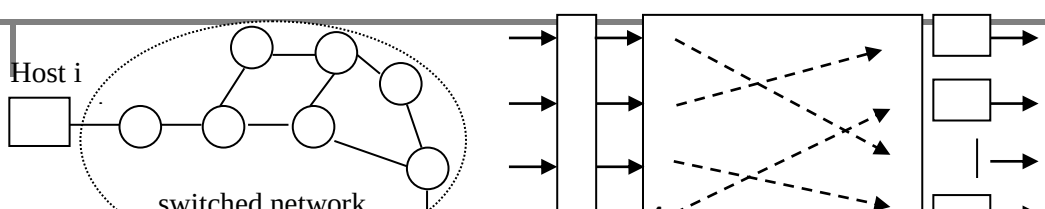
   Hence, a larger delay jitter of a message stream means that more buffers must be provided by the stream. A packet that arrives too early to be processed by destination must be buffered.

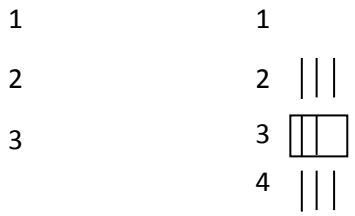
   Finally, the rat of each message stream measures the throughput of the stream. Most of the scheduling and flow control algorithms described in subsequent stations are rate-based meaning that they are designed to provide each message stream with a guaranteed minimum throughput independent of the demands of the other message streams.

❖ ***Real-Time Connections and Service Disciplines:***

It is common to adopt the connection-oriented approach for real-time traffic. According to the connection-oriented approach, a logical simplex connection from the source to the destination is set up for the transmission of each message stream.

a. Admission Control and Connection Establishment: The admission control of each event handler and switch along the chosen …….. uses the parameters like end-to-end delay, jitter, (which are called quality-of-service parameter) as the basis of an acceptance test to determine whether to admit the connection.
b. Packet-Switched Networks:





We can represent every switching pattern by a preemptation of on –tuple (1, 2, …..n). A number I at position k means that the switch is configured to route a packet coming on the input link k at the sometime when it is routing packets. As an examples, for 4×4 switch, the 4 tuple (2, 4, 1, 3) means that a packet of input like 2 goes to a queue of output link 1, a packet on input link 4 goes to the queue of output link 2, and so on.
The waiting of the output queue phis packet transform time is equal to amount of time the switch takes to route packets ⊙ which is called for ho delay of the packet.

c. Taxonomy of Service Disciplined: The combination of an acceptance test and admission control protocol, an execution protocol and a scheduling algorithm used for the purpose of rate control, jitter control and scheduling of packet transmission is called service discipline.

Priority-Based Service Discipline for Switched Networks:

According to a priority-based service discipline, the transmissions of ready packets are scheduled in a priority driven manner. Among this class of disciplines, the most well known are weighted fair queing, Delay Earliest Due-Date and Rate controlled static priority.

❖ ***Weighted Fair – Queuing Discipline:***

The original version of weighted fair queuing algorithm, which is nonpreemptive, was proposed by Domers for scheduling packet transmission in switched network. It is also known as the packet by packet generalized processor-sharing algorithm.

Let n denotes the number of established connections in the link. Each connection i is allocated a fraction $\tilde{\upsilon}_i$ of the link bandwidth. Let $U = \sum_{i=k}^{n} \tilde{u}_i$ denote the total link bandwidth allocated to all n connections. Without loss of generality, $U \leq 1$.

a. Scheduling packets: Scheduling according to the WRQ algorithm is done as follows.
   - When the first packet in a busy interval of the output link arrives, the scheduler compute its finish number and enters this number and connection ID in SFN queue (smallest finished number first). It commences the transmission of the packet immediately.
   - During a link busy interval, the scheduler computes the finish number of each packet that arrives or an idle connection and inserts the corresponding entry into the SFN queue.
   - For as long as link is busy, whenever the transmission of a packet on the link completes, the packet is removed from the connection's FIFO queue and the entry containing the finish number of this packet is

removed from the head of the SFN queue. The scheduler chooses the next packet to transmit in the following manner.

- If connection I is still backlogged, the scheduler computes the finish number of its new ready packet and inserts this number & connection ID in the SFN queue.
- If then connection the transmission of the ready packet on the connection identified by the first entry in the SFN queue.

❖ ***Rote-Proportional Server(RPS) Model and Algorithm:***

While the WFQ algorithm offers advantages in delay and fairness, its implementation is complex: the link finish number may be updated upto o(n) times per finish-number, computation. In contrast, a virtual clock computation according to the virtual clock algorithm takes o(1) time, they call it rate-proportional sever (RPS) model. We now define RPS and describe an algorithm, called the frame-based WFQ, which approximates WFQ but has scheduling complexity o(1).

The Rate-Proportional server (RPS) is an idealized algorithm that cannot be implemented in practice. It also assumes that every connection I is allocated a fraction $\tilde{u}_i$ of the link bandwidth and $\sum_{i=1}^{n} \tilde{u}_i \leq 1$. The scheduler maintains potentials of individual connections and the link as described above. Moreover, when a connection I is backlogged throughout an interval (t_1, t_2) for $t_1 < t_2$.

$\pi_i(t_2) - \pi_i(t_1) = \frac{\tilde{\omega}i(t1,t2)}{\tilde{v}i}$, where the attained time $w_i(t_1, t_2)$ of connection i is the amount of time within (t_1, t_2) used to transmit connection-I packets.

❖ ***Frame-Based Weighted Fair Queueing:***

The FWFQ algorithm provides the some latency as the WFQ, but is not as fair. The key parameter that determines fairness is the frame period F. Each recalibration (say the K^{th}) increase the link potential is smaller than this value at the recalibration time. the larger the frame period, the less frequent the recalibration and the less fair the FWFQ but the lower the recalibration overhead. (The mechanism used by FWFQ to prevent starvation in recalibration).

a. The potential of every back logged connection satisfies the following inequality:- $\pi : (t_k) \geq kF$
b. The potentials of all connections are such that :- $\pi(t_k) < (kH) F$

   At recalibration time t_k, the scheduler sets the link potential according to

   $\pi(t_k) = \max(\pi(t_k), kF)$

   Between recalibrations, the link potential increases at the rate of one per unit time. In other words, $\pi(t) = (\pi(t_k) + (t-t_k)$ for $t_k < t \leq t_{k+1}$

Delay-EDD and JItter-EDD Discipline:

Delay-EDD and Jitter service discipline are also well known priority based service discipline. These service disciplines rely on the EDF algorithm to schedule the transmission of packets on each output link.

Both the delay EDD and jitter-EDD service discipline characterize the message stream on each connection by the periodic message model. In other words, the message stream to be transmitted on connection I is $(p_i, 1, D_i)$; the declared peak rate of the connection is one packet per p_i units of time and the end-to-end relative deadline for their delivery is D_i.

a. Connection Establishment: The source client requests a connection i by declaring parameters p_i and D_i of the packets in the request message is routed from the source to the destination, each switch in route determines whether to accept the connection and if it accepts the connection, what maximum delay it can guarantee, and, hence, what local relative deadline to offer to clients.
b. Rate Control and Scheduling: At switch k, each packet is transmitted with other packet on the same output link on the EDF basis according to its deadline. Unlike the WFQ algo, the EDF algorithm by itself cannot provide timing protection to any connection against overloads on other connections. The deadline $d_{i,j}$ for the transmission of each …….. arrived packet on any connection I is not calculated based on its actual arrival time $a_{i,j}$. Rather, it is based on its effective arrival time. The effective arrival time $a^e_{i,1}$ of the first packet on each connection I is equal to its actual arrival time $a_{i,1}$. The effective arrival time of the j^{th} packet is given by $a^e_{i,j} = \max(a^e_{i,j} + p_i, a_{i,j})$. The local deadline for the transmission of the j^{th} packet at switch k is $d_{i,j} = a^e_{i,j}\ d_{i,k}$.
c. Jitter Control: The jitter-EDF algorithm discipline is an enhancement of the delay-EDD discipline designed to keep the end-to-end delay jitter at switch k under $D_{i,k}$. Hence, the amount of buffer space required for connection I at the switch constant is independent of how many hops the switch is from the source.

Weighted Round-Robin Service Disciplines:

This section describes four variants of the WRR scheme that have been proposed for scheduling message transmission in packet-switched networks. The section focuses on constant bit-rate messages. These messages fit the periodic message model, so the message stream on each connection I is $M_i = (p_i, e_i, D_i)$. Here the parameters

in the types are parameters of instances of M_i, not those of individual packets. So, p_i is the minimum interarrival time of message instance on connection i, e_i is the maximum number of packets in each message & D_i is the end-to-end relative deadline of delivery of every message.

❖ ***Greedy WRR Discipline:***

In its simplest form, the WRR Scheduling algorithm works as follows: During connection establishment, the scheduler at each switch assigns to the new connection i a weight wt_i. This means that the connection i is allocated wt_i slots in each round during which packets on all existing connection sharing the some output link are transmitted in turn.

- Throughput & Delay Guarantees: specifically, during each round, if mare than wt_i packets on connection i are waiting, wt_i packets are transmitted. If wt_i or fewer packets are waiting, all packets on the connection are transmitted. After the scheduler completes the transmission of packets on connection i, it proceeds to transmit packets on connection (i+l) in the same manner, except, of course, the maximum number of packets on connection(i+l) transmitted per round is wt_{i+l}. in this way, each connection i is guaranteed wt_i slots in each round.
- Connection Establishment: Because weights of all connection or an output link depend on the round length used for the link, a change of the latter may require changes of the weights. It would be too costly for the scheduler to adjust the round length dynamically as a p art of acceptance test and admission of new connections.
- Delay jitter and Buffer Requirements: The WRR discipline is greedy and hence, rate allocating. The actual throughput of a connection can exceed its guaranteed rate. Like all other greedy disciplines, the greedy WRR discipline doesnot control delay jitter.

❖ ***Time-Driven WRR Discipline:***

This is non greedy WRR discipline which use different mechanisms to control delay jitters. They are stop & Go algorithm, Hierarchical Round-Robin algorithm, and Budgeted Weighted Round-Robin algorithm. The former two are purely time- driven, while BNRR algorithm is not.

- S & G Algorithm: The S & G algorithm fixes the length of round by dividing the time into interval, called frames, that have fixed length RL. If a connection I is allocated wt_i slots per frame, the scheduler transmits up to wt_i packets in each frame. Frames begin at fixed-time instants; the numbers of packets transmitted per RL units of time is never more than wt_i even when some slots are idle and more packets on connection i are waiting. In this way, the S & G algorithm controls the transmission rate of every connection.
- Hierarchical Round-Robin Algorithm: The HRR algorithm uses different round lengths for different levels of guaranteed service; the higher the level, the shafer, its round length. It is convenient to think of a switch that provides x levels of guaranteed services as loving x servers for each output link.

❖ ***Budgeted Weighted Round-Robin Algorithms:***

Like the S & G and HRR algorithms, the Budgeted Weighted Round Robin (BWRR) algorithm is nongreedy and rate-controlling. It is designed to achieve the performance of synchronized S & G and HRR algorithms. Unlike these algorithms, however, the BWRR algorithm does not divided time into fixed-length frames & hence does not require a synchronized clock.

Medium Access-Control Protocols of Broadcast Networks:

In over terminology, the transmission medium of a broadcast network is a "processor". A Medium Access-Control (MAC) protocol is a discipline for scheduling this type of processor. Scheduling the transmission medium is done distributedly by network interfaces of hosts in the network. This is why MAC protocols typically either look different or are in feet different from the scheduling algorithms. Indeed, some MAC protocols are designed for distributed scheduling by loosely coupled schedulers, and they are fundamentally different from centralized scheduling algorithms.

We call transmission medium of a network. A station on the network is the network interface of a most on the network. We continue to call each basic unit of data transmitted nonpreemptively over the network a packet. The statement that a station listens means that it monitors the data on the network, and that it hears or sees means that it. It senses the presence of data or some special bit patterns on the network.

Real time aspects of MAC protocols:

a. Medium Access Protocol in CAN & IEEE 802.5 Token Ring:

→ Controller area Networks are examples of very small network. CAN's are used to connect components of embedded controllers. An example is an automotive control system, whose components control the engine, the brakes, the environment, and so on, or a car.

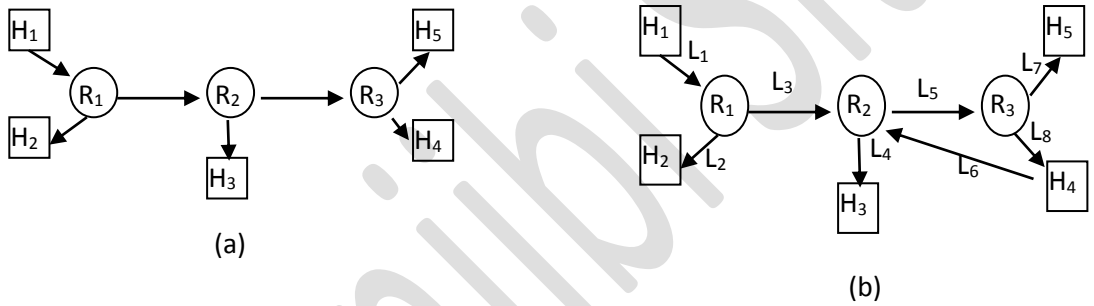
b. Priuitzel Access in IEEE 802.5 Token Rings:

→ In an IEEE 802.5 token ring network, packets are transmitted in an one direction along a circular transmission medium. A station transmits a packet by breaking the network and placing its packet on the output link to the network. As the packet circulates around the network, the station(s) identified by the destination address in the packet header copies the packet. When the packet returns to the source station, the station removes the packet from the network.

Internet & Resource Reservation Protocols:

How hosts and routers operate and interact for the purpose of resource reservation is governed by a resource reservation protocol. This section first discusses issues in resource reservation and then describes the well-known resource-reservation protocol.

❖ ***Issues in Resource Reservation:***



→Multipoint-to-multipoint communication

→Heterogeneity of Destinations

Figure 10.3 Multicast trees of a multicasts group

Real –Time Protocol

Since their advent, the transport protocols TCP and UDP and internet protocol IP have served nonreal-time applications well. Yet these protocols are in suitable for real-time applications for many reasons. IP is connectionless. Previously, we have argued that the connection-oriented approach is more advantageous for real-time applications because of the ability to reserve resources, manage quality of services, and provide isolation in an end-to-end manner for each flow. Therefore, one way to enhances the real time capabilities of the internet is to have a connection-oriented alternative to IP.

TCP is a connection-oriented transport protocol. However, its error-control, flow-control, and sequencing mechanisms are designed to provide users with reliable, sequenced data delivery; they can introduce large delay and jitter and severely limit end-to-end throughput.

❖ ***Data Transport:***

Real Time Protocol is designed to support multicast communication in interactive multimedia applications such as audio and video teleconferencing and distributed simulation. Monitoring, Control, and identification functions are provided by RTCP.

a. RTP Packets:
b. Mixers and Translators:

In addition to hosts, there are also mixers and translators; both are RTP delays. Such a relay connects two or more networks at the transport level. To a relay, each network is defined by its network and transport layer protocols (e.g, IP/VDP) , a multicast address, and a transport-layer destination post.

❖ ***RTCP Control Protol:***

There is a control part associated with each data part. Control packets carrying control and monitoring info. On data transmission to the data part are sent under the control of RTCP to the control part using the same multicast route as the associated data packets.

Specifically, reception quality is feedback to every participant in a multicast group, as well as to a network service provider, or monitors when there Distribution this feedback information is the primary function of RTCP. This info is needed to support adaptive encoding and congestion control. It also helps in fault diagnosis.

Functions:

- Transmission of Reception Quality Reports.
- Computation of Reception Report Retransmission Interval.
- Collision Resolution and Intermediated Synchronization.

Communication in multicomputer System:

We return to the subject of scheduling & flow control in switched networks, this time focusing on those that are used in multicomputer systems.

❖ ***Wormhole Networks:***

In a wormhole network, messages are segmented into very small flow control units, called flits. In the simplest wormhole networks, each switch provides only enough buffer space to hold one flit per input link. The buffer is there in order to decouple the input link from the output links.

- Routing & Transmission.
- Path Selection & Scheduling

Questions:

- ✓ Explain about the model of real time communication in detail.
- ✓ Explain packets, network bandwidth and physical size.
- ✓ What are the types of messages in real time traffic models?
- ✓ What are the performance measure for scheduling, synchronization and flow?
- ✓ Explain about real time communication and service discipline.
- ✓ What do you know about priority based service discipline for switched networks?
- ✓ Explain about Rate Proportional Server (RPS) Model and Algorithms.
- ✓ Explain about Frame-Based Weighted Fair Queuing Approach.
- ✓ Distinguish between Delay-EDD and Jitter-EDD.
- ✓ What do you know about Weighted Round Robin Service Disciplines? How does it differ from Time Driven RR Approach?
- ✓ How do you define Medium Access Control (MAC) Protocol in Broadcast networks?
- ✓ What are the real time protocols?

- ✓ Write short note about wormhole networks. Why it is used in multiprocessor systems?

Bibliography

Real Time systems, Jane W.S Liu, Pearson Education Asia, 2003

Real Time Systems and Programming Languages, Addison Wesley Longman, Third Edition, 2001

Randomized Caches Considered Harmful in Hard Real-Time Systems, J. Reineke , *Leibniz Transactions on Embedded Systems*, 1 (1), June 2014.

Building Timing Predictable Embedded Systems, P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. von Hanxleden, R. Wilhelm, W. Yi *ACM Transactions on Embedded Computing Systems*, 13 (4), February 2014.

Sensitivity of Cache Replacement Policies J. Reineke and D. Grund
ACM Transactions on Embedded Computing Systems, 12 (1), March 2013.

Branch Target Buffers: WCET Analysis Framework and Timing Predictability D. Grund, J. Reineke, and G. Gebhard *Journal of Systems Architecture*, 57 (6), 2011.

Predictability Considerations in the Design of Multi-Core Embedded Systems
C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, S. Wegener, and R. Wilhelm, *Ingénieurs de l'Automobile*, 807, 2010.

Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems
R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, *IEEE Transactions on CAD of Integrated Circuits and Systems*, 28 (7), 2009

Timing Predictability of Cache Replacement Policies J. Reineke, D. Grund, C. Berg, and R. Wilhelm
Real-Time Systems, 37 (2), 2007.