GUDLAVALLERU ENGINEERING COLLEGE
(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
SeshadriRao Knowledge Village, Gudlavalleru – 521 356

Department of Information Technology

REAL TIME SYSTEMS

(2019-2020)

Vision

To be a center of innovation by adopting changes in Information Technology and imparting quality education and research to produce visionary computer professionals and entrepreneurs.

Mission

- To provide an academic environment in which students are given the essential resources for solving real-world problems and work in multidisciplinary teams.
- To impart value based education and research among students, particularly belonging to rural areas, for their sustained growth in technological aspects and leadership.
- To collaborate with the industry for making the students adoptable to evolving changes in Information Technology and related areas.

Program Educational Objectives

PEO 1:

To exhibit analytical skills in modeling and solving computing problems by applying mathematical, scientific and engineering knowledge and to pursue their higher studies.

PEO 2:

To communicate effectively with multi-disciplinary teams to develop quality computing systems with an orientation towards research and development for lifelong learning.

PEO 3:

To address industry and societal needs for the growth of global economy using emerging technologies following professional ethics.

HANDOUT ON REAL-TIME SYSTEMS

Class & Sem. :IIIB.Tech – II Semester (OE) Year : 2019-20

Branch : IT Credits :3

Faculty Name: Sri. B. ANAND KUMARDesignation:Assistant Professor Dept.: IT

==

1. Brief History and Scope of the Subject

The term real-time derives from its use in early simulation, in which a real-world process is simulated at a rate that matched that of the real process (now called real-time simulation to avoid ambiguity). Analog computers, most often, were capable of simulating at a much faster pace than real-time, a situation that could be just as dangerous as a slow simulation if it were not also recognized and accounted for.

This subject introduces diverse aspects of real-time systems including architecture, principles, specification and verification, scheduling and real world applications. It is useful for studentsin a wide range of disciplines impacted by embedded computing and software. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring.

2. Pre-Requisites

- Operating Systems
- Networking

3. Course Objectives:

- To familiarize with the concepts of Real – Time systems.

4. Course Outcomes: Students will be able to

CO1:make use of hard and soft real time systems.

CO2: evaluate Clock driven scheduling, weighted round-robin, priority driven
approaches in real time systems.

CO3: compare rate monotonic and deadline monotonic algorithms.

CO4: analyze multi task scheduling algorithms for periodic, aperiodic and sporadictasks.

CO5: demonstrate temporal distance and DCM.

CO6: outline real time communications architecture.

5. Program Outcomes:

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

6.PROGRAM SPECIFIC OUTCOMES

Student will be able to

1. Organize, monitor and protect IT Infrastructural resources.
2. Design & Develop software solutions to the real world problems in the form of web, mobile and smart apps.

2. Mapping of Course Outcomes with Program Outcomes:

	1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2
CO1	2	2	2	2									1	
CO2	2	2	2									3	1	2
CO3	2	2	2									3		2
CO4	3	3	2									3		2
CO5	2	2	1											
CO6	2	2	2									3		2

7. Prescribed Text Books

- Jane Liu, Real-Time Systems, Prentice Hall, 2000.
- Philip.A.Laplante, Real Time System Design and Analysis, 3rd Edition, PHI, 2001.

8. Reference Text Books

- Laplante and Ovasaka, “Real-Time Systems Design and Analysis: Tools for the Practitioner” (4th Edition).
- Cheng, A. M. K.: Real-Time Systems: Scheduling, Analysis, and Verification.
- Krishna, C. M., Shin, K. G.: Real-Time Systems. McGraw-Hill, 1997.
- Levi, S. T., Agrawala, A. K.: Real-Time System Design. McGraw-Hill, 1990.

9. URLs and Other E-Learning Resources

http://nptel.ac.in/courses/106105036/2#

http://www.cse.unsw.edu.au

10. **Digital Learning Materials:**

- http://ppedreiras.av.it.pt/resources/str1112/apresentacoes_pesquisa/Real-time-CS.pdf
- http://www.cse.unsw.edu.au/~cs9242/08/lectures/09-realtimex2.pdf
- https://pdfs.semanticscholar.org/927e/5c953268beeb71b74f7300f744c27ca76efe.pdf
- https://link.springer.com/journal/11241

11. Lecture Schedule / Lesson Plan

Topic	**No. of Periods**
	Theory
UNIT –1:	
Introduction: Real Time Systems	1
Typical Real Time Applications	3

Hard Versus Soft Real-Time Systems	1
Hard realtime systems and soft real-time systems	2
A reference model of real-time systems	2
Tutorial	1
UNIT-II Commonly Used approaches to hard real-time scheduling	
➢ Clock-Driven Approach	1
➢ Weighted Round-Robin Approach	1
➢ Priority driven approach	1
Clock-driven scheduling	1
➢ Scheduling Sporadic Jobs: Acceptance Test, EDF scheduling of accepted jobs	2
➢ Algorithm for Constructing Static Schedules	2
Tutorial	1
UNIT-III Priority-driven scheduling of periodic tasks	1
Fixed Priority and Dynamic priorityalgorithms-Rate monotonic and Deadline monotonic algorithms.	5
Tutorial	1
UNIT-IV Resources and Resource access control	2
➢ Basic Priority-Inheritance Protocol	2
Multiprocessor scheduling and resource access control	1
➢ Identical Versus Heterogeneous processors	2
➢ Inter process communication	1
➢ Multiprocessor priority-ceiling protocol	2
Tutorial	1
UNIT-V Scheduling flexible computations and tasks with temporal distance constraints	2
➢ Flexible Applications	3
➢ Tasks with Temporal distance constraints	3
Tutorial	1
UNIT-VI Real-Time Communications	1
➢ Model of real-time communication	2
➢ Medium Access-control protocols of broadcast networks	2

Operating Systems	2
➢ Threads and tasks, Kernel	1
➢ Memory Management	1
➢ I/O and Networking	2
➢ Open system architecture	2
Tutorial	1
Total Number of Periods:	**60**

12. Seminar Topics

- Real-Time Systems Applications
- Resources and Functional Parameters of RTS
- Clock-driven approach of RTS scheduling
- Operating System functions
- Real Time Operating System

Real-Time Systems
UNIT – I

Objectives:

➢ To gain the knowledge typical applications and structure of real time systems, differentiate hard and soft real-time systems.

Syllabus:

Real time Systems, typical real-time systems, hard versus soft real-time systems, a reference model of real-time systems.

Outcomes:

Students will be able to

➢ explain the purpose and structure of a real time system.
➢ illustrate differences between Digital control systems, High-level control
system.
➢ analyze the performance of soft and hard real time systems.
➢ know reference model of real time system.

Learning Material

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

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

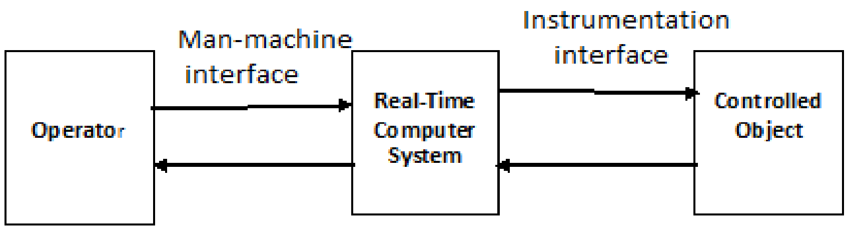
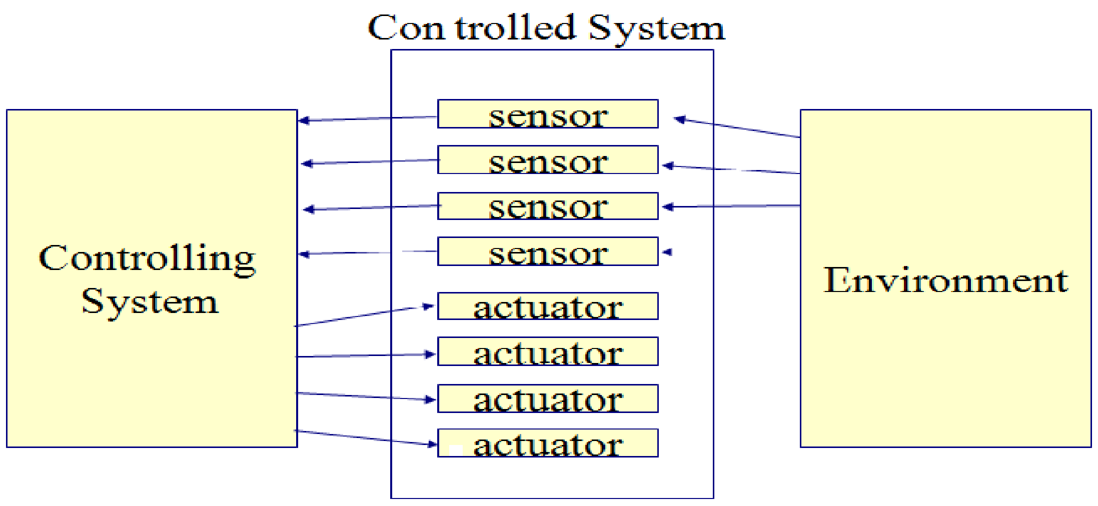


Fig1: Real-Time System

- Real-time systems often are comprised of a *controlling* system, *controlled* system and *environment*.
  Controlling system: acquires information about environment using *sensors* and controls the environment with *actuators*.



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

    Periodic Tasks : Time-driven recurring at regular intervals.

    Aperiodic Tasks : event-driven

Needs of the Real-Time Systems:

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

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

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

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

A Simple Example:

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

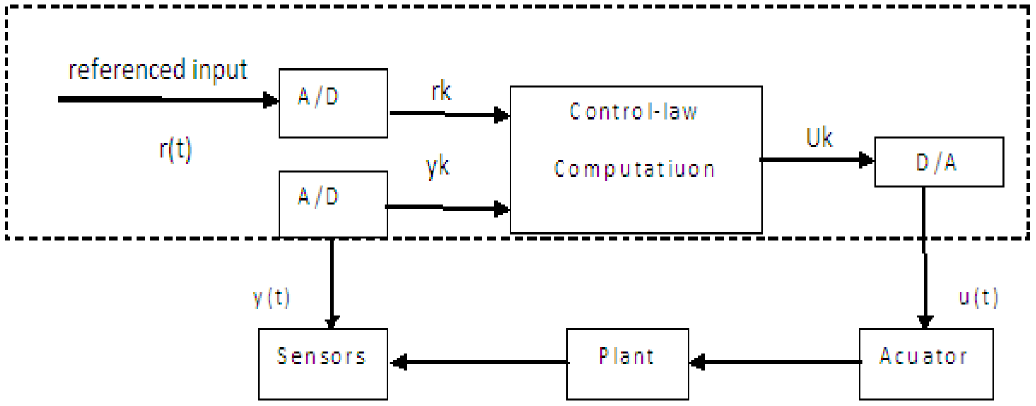


Fig : A Digital Controller

`

- Let $e(t)=r(t)-y(t)$ denote the difference between the desired state *r(t)* and the measured state *y(t)* at time *t* . The output *u(t)* of the controller consists of three terms.
- During any sampling period (say the *k*th), the control output u_k depends on the current and past measured values y_i for $i \le k$. The future measured values y_i 's for $i>k$ in turn depend on u_k . Such a system is called a *(feedback) control loop* or simply a *loop*. We can implement it as an infinite timed loop:

```
set timer to interrupt periodically with period $T$ ;

at each timer interrupt, do

  do analog-to-digital conversion to get $y$;
```

```
        compute control output $u$;

        output$u$ and do digital-to-analog conversion;

    end do;
```

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

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

High-level Control: Controllers in a complex monitor and control system are typically organized hierarchically. One or more digital controllers at the lowest level directly control the physical plant. Each output of a higher-level controller is a reference input of one or more lower-level controllers. With few exceptions, one or more of the higher-level controllers interfaces with the operator.

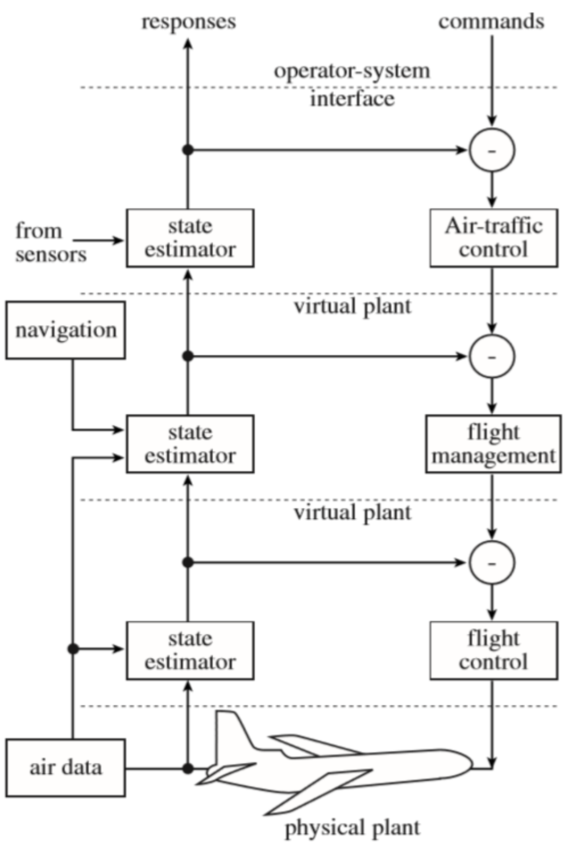
Examples of Control Hierarchy

A patient care system may consist of microprocessor-based controllers that monitor and control the patient's blood pressure, respiration, glucose, and so forth. There may be a higher-level controller (e.g., an expert system) which interacts with the operator (a nurse or doctor) and chooses the desired values of these health indicators.

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

Flight Control System:

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

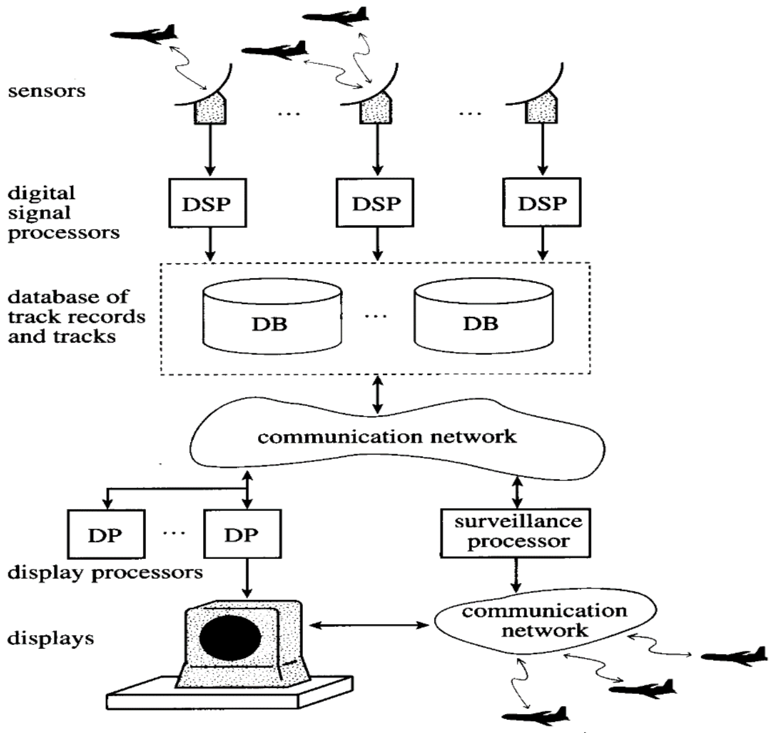


- The Air Traffic Control (ATC) system is at the highest level.
- It regulates the flow of flights to each destination airport.
- It does so by assigning to each aircraft an arrival time at each metering fix (known geographical point, adjacent points are 40-60 miles apart) in the route to destination.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

Real-Time Command and Control

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

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



An architecture of air traffic control system.

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

SIGNAL PROCESSING:

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

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

Processing Bandwidth Demands:

Typically, a real-time signal processing application computes in each sampling period one or more outputs. Each output *x(k)* is a weighted sum of *n* inputs *y(i)*'s

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

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

Radar System

- A signal processing application is typically a part of a larger system.

- The system consists of an Input/Output (I/O) subsystem that samples and digitizes the echo signal from the radar and places the sampled values in a shared memory.
- An array of digital signal processors processes these sampled values.
- The data thus produced are analyzed by one or more data processors, which not only interface with the display system, but also generate commands to control the radar and select parameters to be used by signal processors in the next cycle of data collection and analysis.

Radar Signal Processing:

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

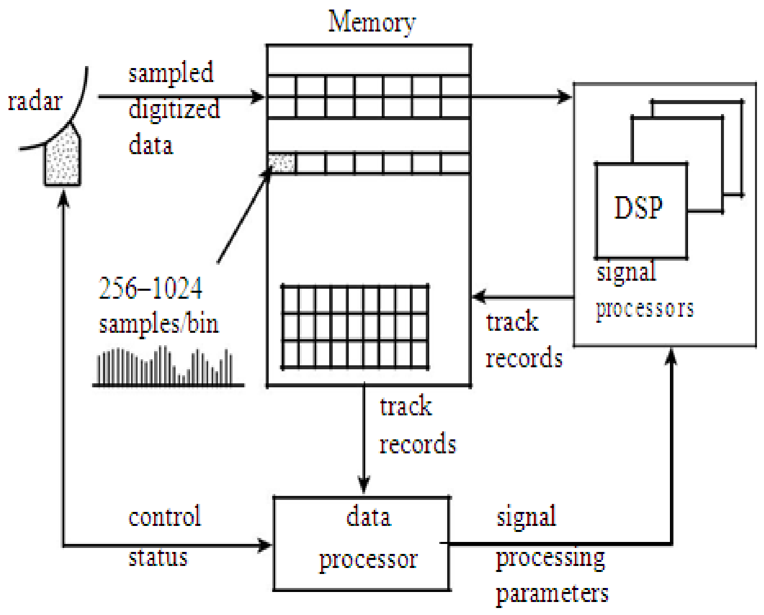


FIGURE 1–6 Radar signal processing and tracking system

Tracking:

- Noise and man-made interferences, including electronic countermeasure (i.e., jamming), can lead the signal processing and detection process to wrong conclusions about the presence of objects.
- A track record on a non existing object is called a false return. An application that examines all the track records in order to sort out false returns from real ones and update the trajectories of detected objects is called a *tracker*.

➢ Gating:

- Typically, tracking is carried out in two steps: gating and data association.
- *Gating* is the process of putting each measured value into one of two categories depending on whether it can or cannot be tentatively assigned to one or more established trajectories.
- The gating process tentatively assigns a measured value to an established trajectory if it is within a threshold distance G away from the predicted current position and velocity of the object moving along the trajectory.
- The threshold G is called the track gate. It is chosen so that the probability of a valid measured value falling in the region bounded by a sphere of radius G centered around a predicted value is a desired constant.

Process:

- At the start, the tracker computes the predicted position (and velocity) of the object on each established trajectory.
- In this example, there are two established trajectories, L_1 and L_2. We also call the predicted positions of the objects on these tracks L_1 and L_2. X_1, X_2, and X_3 are the measured values given by three track records.
- $X1$ is assigned to L_1 because it is within distance G from L_1. X_3 is assigned to both L_1 and $L_2$for the same reason.
- On the other hand, $X_2$is not assigned to any of the trajectories.
- Itrepresents either a false return or a new object. Since it is not possible to distinguish between these two cases, the tracker hypothesizes that X_2 is the position of a new object.
- Subsequent radar data will allow the tracker to either validate or invalidate this hypothesis. In the latter case, the tracker will discard this trajectory from further consideration.

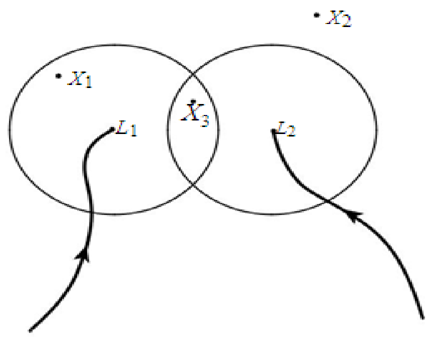


FIGURE 1–7 Gatingprocess.

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

Other Real-Time Applications:

- **Real-time databases:**
  - Transactions must complete by deadlines.
  - **Main dilemma:** Transaction scheduling algorithms and real-time scheduling algorithms often have conflicting goals.
  - Data may be subject to **absolute** and **relative temporal consistency** requirements.
  - Overall goal: reliable responses
- **Multimedia:**
  - Want to process audio and video frames at steady rates.
  - TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
  - Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec

Hard versus Soft Real-Time Systems:

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

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

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

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

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

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

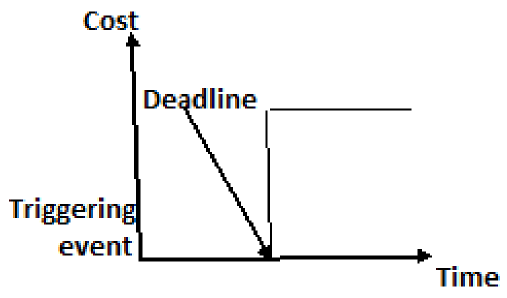


Fig : Hard Real-Time System

Some Reasons for Requiring Timing Guarantees:

- Many embedded systems are hard real-time systems. Deadlines of jobs in an embedded system are typically derived from the required responsiveness of the sensors and actuators monitored and controlled by it.
  - As an example, consider an automatically controlled train. It cannot stop instantaneously. When the signal is red (stop), its braking action must be activated a certain distance away from the signal post at which the train must stop.
  - This braking distance depends on the speed of the train and the safe value of deceleration. From the speed and safe deceleration of the train, the controller can compute the time for the train to travel the braking distance.
  - This time in turn imposes a constraint on the response time of the jobs which sense and process the stop signal and activate the brake. No one would question that this timing constraint should be hard and that its satisfaction must be guaranteed.
- Jobs in some non-embedded systems may also have hard deadlines. An example is a critical information system that must be highly available: The system must never be down for more than a minute.
- In recent years, this observation motivated a variety of approaches to soften hard deadlines. Examples are to allow a few missed deadlines or premature terminations as long as they occur in some acceptable way.

More on Hard Timing Constraints:

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

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

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

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

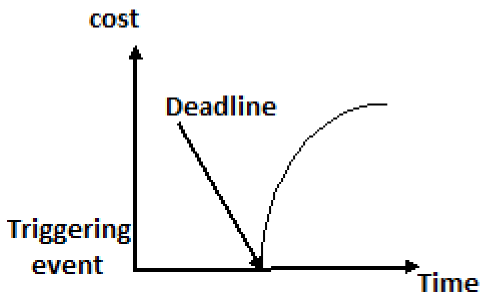


Fig: Soft Real-Time System

A Reference Model of Real-Time Systems:

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

Task ,Job, Processors and Resources:

- ➢ Task (*T_i*): Set of related jobs jointly provide function.
- ➢ Job (*J_{ij}*): Unit of work, scheduled and executed by system. characterized by the following parameters:
  1. Temporal parameters: timing constraints and behavior
  2. Functional parameters: intrinsic properties of the job.
  3. Interconnection parameters: how it depends on other jobs and how other jobs depend on it.
  4. Resource parameters: resource requirements.
- ➢ Processors and Resources: we can divide the system resources into two major types: processors and resources. Processors are often called servers and active resources.
  - Computers, transmission links, disks, and database server are examples of processors. They carry out machine instructions, move data from one place to another, retrieve files, process queries, and so on. Every job must have one or more processors in order to execute and make progress toward completion
- Resources can be divided into passive and active:
  - Active resources = *Processors* (*P_i*): they execute jobs.
    - ▪ Every job must have one or more processors
    - ▪ Same type if functionally identical and used interchangeably.
  - Passive resource = *Resource* (*R_i*):
    - ▪ Job may require Resources in addition to processor.
    - ▪ Reusable resources are not consumed

Task Temporal Parameters:

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

Release time (r_i) : is the time at which the job becomes ready for execution.
Absolute deadline(d_i): is the time at which the job should be completed.
Relative deadline(D_i): is the time length between the arrival time and the absolute deadline.
Start time (sj): is the time at which the job starts its execution.

Finishing time (fj): is the time at which the job finishes its execution.
Execution time (e_i): May know range [e^-, e^+]. Most deterministic models use e^+.

- **Periodic Task Model:** In periodic task, jobs are released at regular intervals. A periodic task is one which repeats itself after a fixed time interval. A periodic task is denoted by five tuples: Ti = < Φi, Pi, ei, Di >
  Where,
  Task T_i is a serious of periodic Jobs J_{ij}.
  φ_i - phase of Task T_i, equal to r_{i1}.
  p_i - period, minimum inter-release interval between jobs in Task T_i. Must be bounded from below.
  e_i - maximum execution time for jobs in task T_i.
  Di – is the relative deadline of the task.
  r_{ij} - release time of the j^{th} Job in Task i (J_{ij} in T_i).
  H –Hyperperiod = Least Common Multiple of p_i for all i:
  H = lcm(p_i), for all i.
  u_i - utilization of Task T_i.
  U - Total utilization = Sum over all u_i.

1. Aperiodic Tasks: In this type of task, jobs are released at arbitrary time intervals i.e. randomly. Aperiodic tasks have soft deadlines or no deadlines.
2. Sporadic Tasks: They are similar to aperiodic tasks i.e. they repeat at random instances. The only difference is that sporadic tasks have hard deadlines. A sporadic task is denoted by three tuples: Ti =(ei, gi, Di)
Where
ei – the execution time of the task.
gi – the minimum separation between the occurrence of two consecutive instances of the task.
Di – the relative deadline of the task.

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

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

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

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

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

1. Preemptivity of resources
2. Resource Graph

Preemptivity of resources:

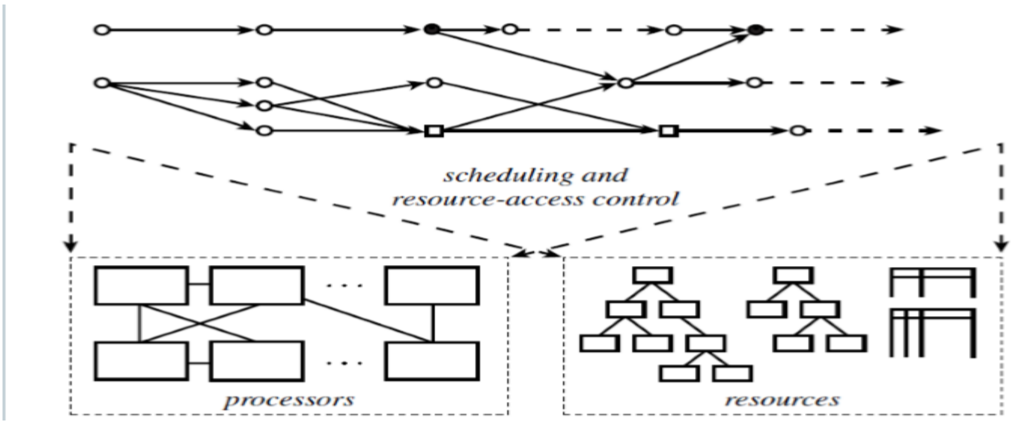
- A resource in non-preemptable if each unit of resource is constrained to be used serially.
- Once the unit of non-preemptable resource is allocated to a job, the other job needing the unit must wait until the job completes its use.
- If a job can use every resource in an interleaved manner, then the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

Resource Graph:

- Resource graph is used to describe the configuration of the resources.
- In a resource graph, there is a vertex for every processor or resource Ri in the system.
- The attributes of the vertex are the parameter of the resources.
- Resource type of a resource tell if the resource is processor or passive resource and number gives the available number of units.

There are two types of edges in resource graphs.

- An edge from vertex Ri to Rk can mean Rk is component of Ri(memory is a part of computer).
- This edge is an is-a-part-of-edge.
- Some edges in resource graph represent the connectivity between the components.
- These edges are called accessibility edges.(Connection between two CPUs)



****

Unit – II

Objectives:

To gain the knowledge on commonly used approaches to hard real-time scheduling, Clock driven scheduling.

Syllabus:

Commonly used approaches to hard real-time scheduling

- Clock-driven scheduling, weighted round-robin approach, priority driven approach.

Clock-Driven scheduling:

- Scheduling sporadic jobs: Acceptance test, EDF scheduling of accepted jobs.
- Algorithm for constructing static schedules: scheduling independent preemptable tasks-Network flow graph

Outcomes:

Students will be able to

- describe different hard real-time scheduling algorithms.
- use clock driven scheduling to schedule sporadic jobs.
- construct network flow graph by using static schedules.
- differentiate Weighted Round Robin over Round Robin scheduling.
- implement priority driven , clock driven approaches in scheduling jobs.

SCHEDULING ALGORITHMS:

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

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

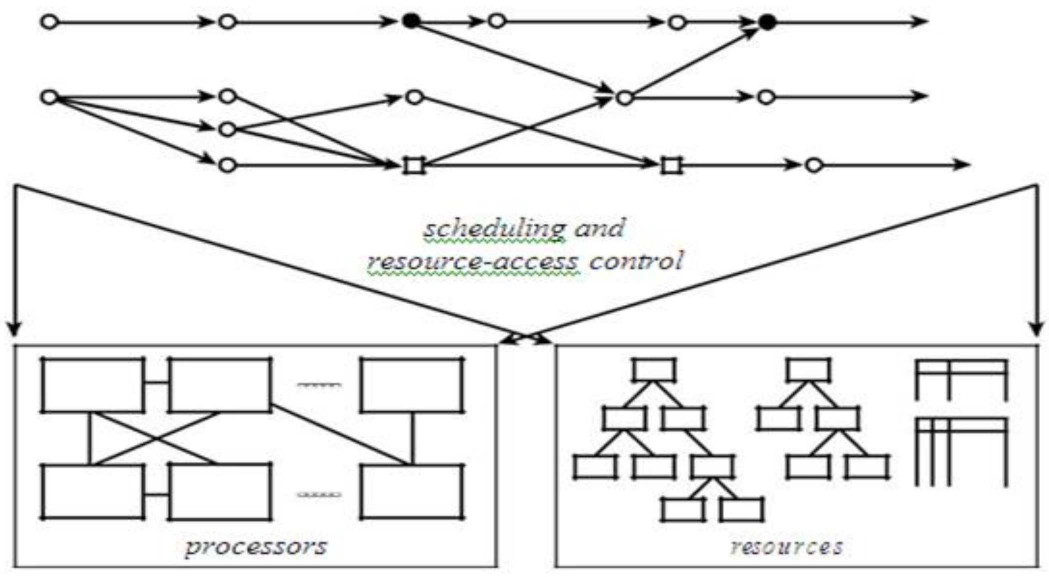


Fig: Model of Real-time systems

Scheduler and Schedules

- Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols. The module which implements these algorithms is called the scheduler.
- Every job is scheduled in a time interval on a processor if the processor is assigned to the job, and hence the job executes on the processor, in the interval. The total amount of processor)time assigned to a job according to a schedule is the total length of all the time intervals during which the job is scheduled on some processor.
- By a schedule, we assignment of all the jobs in the system on the available processors produced by the scheduler. A scheduler is ***valid schedule*** , if it satisfies the following conditions:
    1. Every processor is assigned to at most one job at any time.
    2. Every job is assigned at most one processor at any time.
    3. No job is scheduled before its release time.

Depending on the scheduling algorithm(s) used, the total amount of processor time assigned to every job is equal to its maximum or actual execution time.

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

- A valid schedule is a ***feasible schedule*** if every job completes by its deadline.
- A hard real-time scheduling algorithm is ***optimal*** if the algorithm (the scheduler) always produces a feasible schedule if the given set of jobs has feasible schedules.
- The ***lateness*** of a job is the difference between its completion time and its deadline. The lateness of a job which completes early is negative, while the lateness of a job which completes late is positive.
- For many soft real-time applications, it is acceptable to complete some jobs late or to discard late jobs. For such an application, suitable performance measures include the ***miss rate and loss rate***.
- *Miss rate* is the percentage of jobs that are executed but completed too late.
- *Loss rate* is the percentage of jobs that are discarded, that is, not executed at all.

2.1 Commonly used approaches to hard real-time scheduling:

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

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

2.1.1 Clock-driven scheduling:

As the name implies, when scheduling is clock-driven (also called time-driven), decisions on what jobs execute at what times are made at specific time instants.

- Primarily used for hard real-time systems where all properties of all jobs are known at design time, such that offline scheduling techniques can be used.
- These instants (parameters) are chosen a priori before the system begins execution.
- A schedule of the jobs is computed off-line and is stored for use at runtime; as a result, scheduling overhead at run-time can be minimized.
- Decisions about what jobs execute at what times are made at specific time instants
- Usually regularly spaced, implemented using a periodic timer interrupt
- Scheduler awakes after each interrupt, schedules the job to execute for the next period, then blocks itself until the next interrupt
- One way to implement a scheduler that makes scheduling decisions periodically is to use a hardware timer.
- The timer is set to expire periodically without the intervention of the scheduler.
- When the system is initialized, the scheduler selects and schedules the job(s) that will execute until the next scheduling decision time and then blocks itself waiting for the

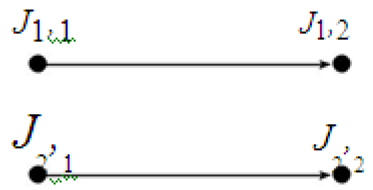
expiration of the timer. When the timer expires, the scheduler awakes and repeats these actions.

2.1.2 Weighted round-robin approach:

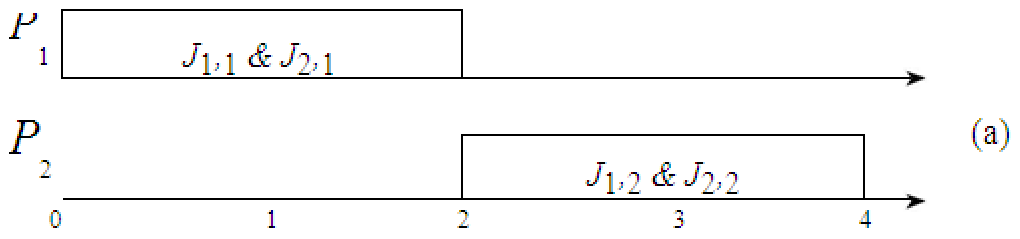
- Regular ***Round-robin*** scheduling is commonly used for scheduling time-shared applications
    - Every job joins a FIFO queue when it is ready for execution
    - When the scheduler runs, it schedules the job at the head of the queue to execute for at most one time slice (A time slice is the basic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

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

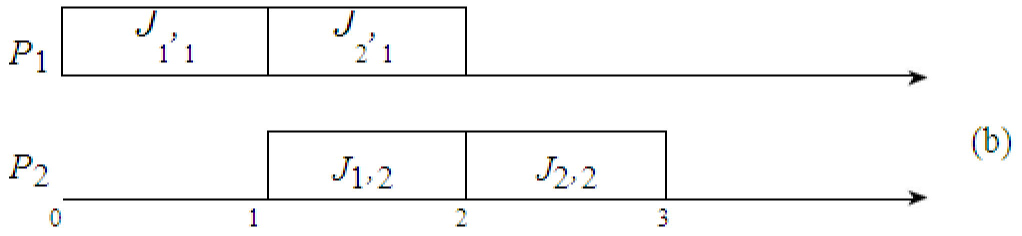
➢ As an example, we consider the two sets of jobs, J1 = {J1,1, J1,2} and J 2 = {J2,1, J2,2}, shown in Figure. The release times of all jobs are 0, and their execution times are 1. J1,1 and J2,1 execute on processor P1, and J1,2 and J2,2 execute on processor P2. Suppose that J1,1 is the predecessor of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



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



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

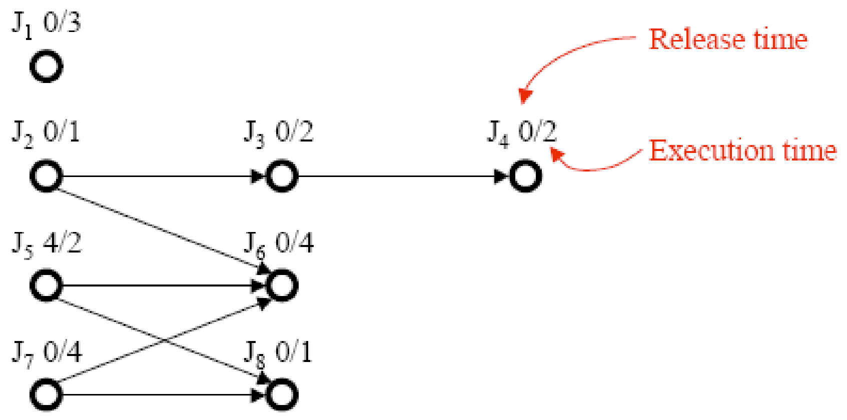


➢ On the other hand, suppose that the result of the first job in each set is piped to the second job in the set. The latter can execute after each one or a few time slices of the former complete. Then it is better to schedule the jobs on the round-robin basis because both sets can complete a few time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

2.1.3 Priority-Driven approach:

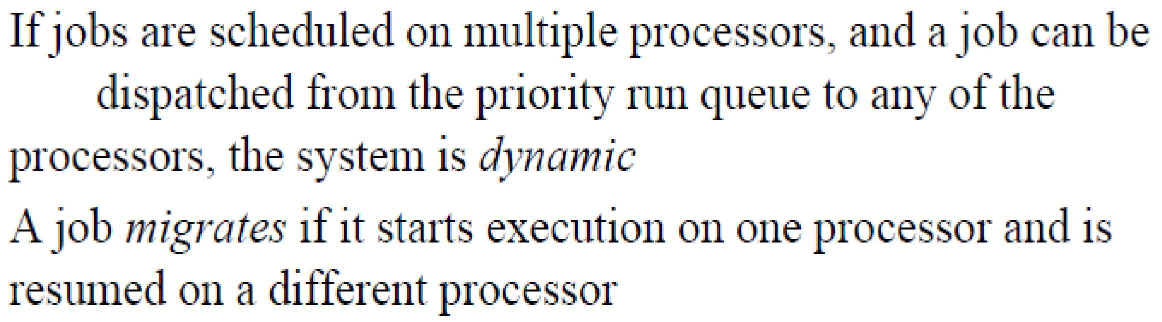
- The term priority-driven algorithms refer to a large class of scheduling algorithms that never leave any resource idle intentionally. Scheduling decisions are made when events such as releases and completions of jobs occur. Hence, priority-driven algorithms are event-driven.

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



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

This scheduling can also be achieved by static system.
Let us assign J1,J2,J3 and J4 to P1 and remaining to P2.
Jobs on P1 are completed by time 8 and the jobs on P2 by time 11.



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

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

2.2 Clock-Driven Scheduling:

2.2.2 Scheduling sporadic tasks:

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

<u>Acceptance Test:</u>

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

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

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

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

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

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

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

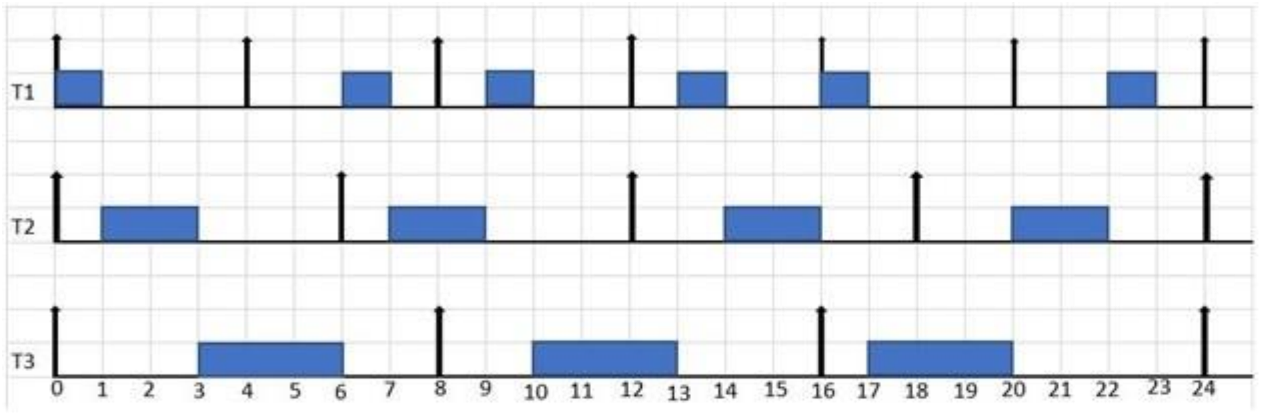


Figure 2. Earliest deadline first scheduling of task set in Table

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

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

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

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

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

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

2.2.2 Algorithm for constructing static schedules:

- First consider the special case where the periodic tasks contain no non preemptable section. After presenting a polynomial time solution for this case, we then discuss how to take into account practical factors such as non preemptivity.
- **Scheduling Independent Preemptable Tasks**
    - A system of independent, preemptable periodic tasks whose relative deadlines are equal to or greater than their respective periods is schedulable if and only if the total utilization of the tasks is no greater than 1.
    - Because some tasks may have relative deadlines shorter than their periods.
    - The iterative algorithm described below enables us to find a feasible cyclic schedule if one exists. The algorithm is called the **iterative network-flow algorithm, or the INF** algorithm.
    - Its key assumptions are that tasks can be preempted at any time and are independent.
    - Before applying the INF algorithm on the given system of periodic tasks, we find all the possible frame sizes of the system
    - The INF algorithm iteratively tries to find a feasible cyclic schedule of the system for a possible frame size at a time, starting from the largest possible frame size in order of decreasing frame size.
    - A feasible schedule thus found tells us how to decompose some tasks into subtasks if
      their decomposition is necessary. If the algorithm fails to find a feasible schedule after all the possible frame sizes have been tried, the given tasks do not have a feasible cyclic schedule that satisfies the frame size constraints even when tasks can be decomposed into subtasks.

<u>Network-Flow Graph</u>:

- The algorithm used during each iteration is based on the well known network-flow formulation of the preemptive scheduling problem.

- In the description of this formulation, it is more convenient to ignore the tasks to which
  The jobs belong and name the jobs to be scheduled in a major cycle of F frames J1, J2. JN
- The constraints on when the jobs can be scheduled are represented by the network-flow
  graph of the system.
- This graph contains the following vertices and edges; the capacity of an edge is a Non-negative number associated with the edge.

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

- A flow of an edge is a nonnegative number that satisfies the following constraints:
  (1) It is no greater than the capacity of the edge.
  (2) With the exception of the source and sink, the sum of the flows of all the edges into
  every vertex is equal to the sum of the flows of all the edges out of the vertex.

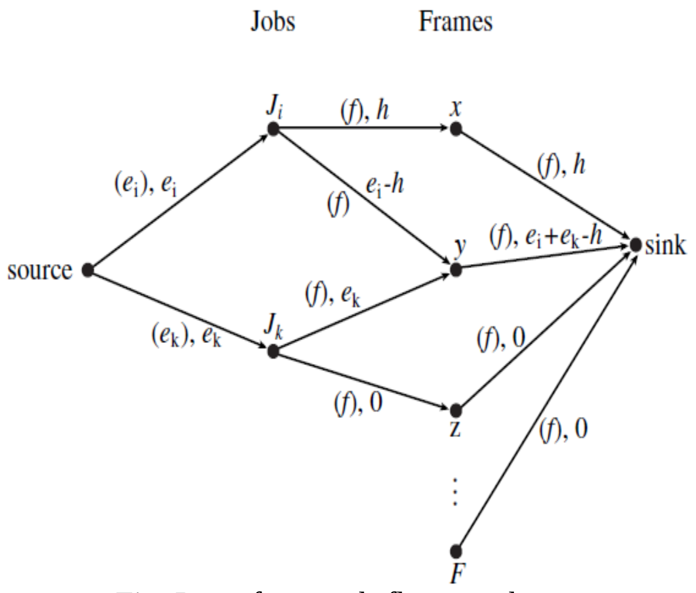


Fig: Part of network-flow graph

- For simplicity, only job vertices Ji and Jk are shown. The label "(capacity), flow" of the each edge gives its capacity and flow. This graph indicates that job Ji can be scheduled in frames x and y and the job Jk can be scheduled in frames y and z.
- A **flow of a network-flow graph, or simply a flow**, is the sum of the flows of all the edges from the source; it should equal to the sum of the flows of all the edges into the sink. There are many algorithms for finding the maximum flows of network-flow sgraphs.

UNIT – III

Priority-driven scheduling of periodic tasks

Objectives:

To gain the knowledge on commonly used priority driven scheduling algoriths to periodic tasks.

Syllabus:

- **Priority-driven scheduling of periodic tasks:** Fixed priority and Dynamic priority algorithms- Rate monotonic and deadline monotonic algorithms.
- **Scheduling aperiodic and sporadic jobs in priority- driven system:**
    - Deferrable servers: Operations on Deferrable servers

Outcomes:

Students will be able to

- describe Rate Monotonic and Deadline Monotonic fixed priority algorithms and dynamic priority algorithms.
- Scheduling deferrable server using RM algorithm
- Scheduling deferrable server using EDF algorithm

PRIORITY-DRIVEN SCHEDULING OF PERIODIC TASKS :

- A priority-driven scheduler is an on-line scheduler. It does not pre compute a schedule of the tasks.
- Rather, it assigns priorities to jobs after they are released and places the jobs in a ready job queue in priority order.
- When preemption is allowed at any time, a scheduling decision is made whenever a job is released or completed.
- At each scheduling decision time, the scheduler updates the ready job queue and then schedules and executes the job at the head of the queue.
- We classify algorithms for scheduling periodic tasks into two types: **fixed priority and dynamic priority.**
- **A fixed-priority** algorithm assigns the same priority to all the jobs in each task. In other words, the priority of each periodic task is fixed relative to other tasks.
- **A dynamic-priority** algorithm assigns different priorities to the individual jobs in each task. The priority of the task with respect to that of the other tasks changes as jobs are released and completed. This is why this type of algorithm is said to be "dynamic."
- Once assigned, the priority of the job relative to other jobs in the ready job queue does not change. In other words, at the level of individual jobs, the priorities are fixed, even though the priorities at the task level are variable.
- We have three categories of algorithms: fixed-priority algorithms, task-level dynamic-priority (and job level fixed-priority) algorithms, and job-level (and task-level) dynamic algorithms.

Rate-Monotonic and Deadline-Monotonic Algorithms:

Rate-Monotonic Algorithm:

- A well-known fixed-priority algorithm is the rate-monotonic algorithm.
- This algorithm assigns priorities to tasks based on their periods: the shorter the period, the higher the priority.
- The rate (of job releases) 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.

- Rate monotonic scheduling Algorithm works on the principle of preemption. Preemption occurs on a given processor when higher priority task blocked lower priority task from execution. This blocking occurs due to priority level of different tasks in a given task set. rate monotonic is a preemptive algorithm which means if a task with shorter period comes during execution it will gain a higher priority and can block or preemptive currently running tasks. In RM priorities are assigned according to time period. Priority of a task is inversely proportional to its timer period. Task with lowest time period has highest priority and the task with highest period will have lowest priority. A given task is scheduled under rate monotonic scheduling Algorithm, if its satisfies the following equation:
- $$\sum_{k=1}^{n} \frac{C_k}{T_k} \leq U_{RM} = n(2^{1/n} - 1)$$

Where C_k is worst case execution time/computing time.
Where T_k Deadline (D=T)
Where U_{RM} is CPU utilization of task set.
Where n is the no of tasks.

Example of RATE MONOTONIC (RM) SCHEDULING ALGORITHM

For example, we have a task set that consists of three tasks as follows

Tasks	Release time(ri)	Execution time(Ci)	Deadline (Di)	Time period(Ti)
T1	0	0.5	3	3
T2	0	1	4	4
T3	0	2	6	6

Table 1. Task set

U= 0.5/3 +1/4 +2/6 = 0.167+ 0.25 + 0.333 = 0.75

As processor utilization is less than 1 or 100% so task set is schedulable and it also satisfies the above equation of rate monotonic scheduling algorithm.

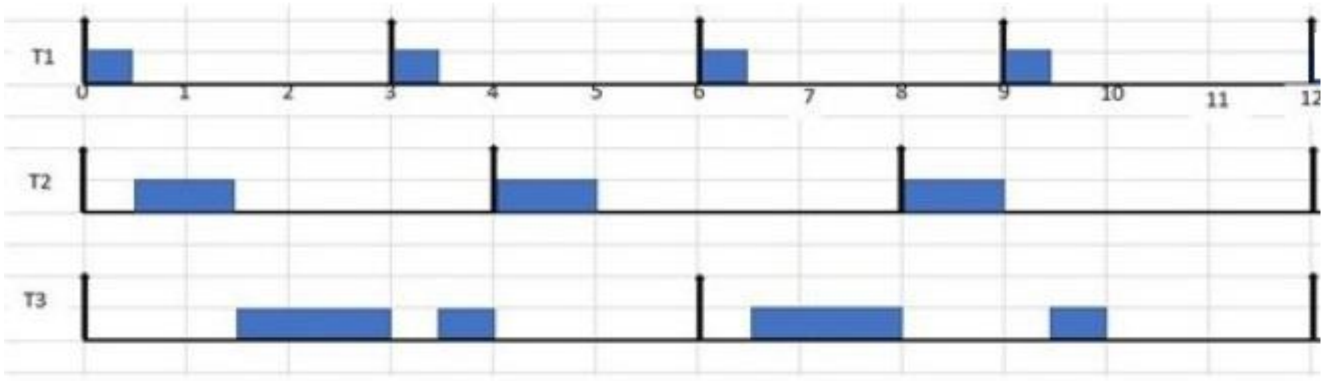


Figure 1. RM scheduling of Task set in table 1.

A task set given in table 1 it RM scheduling is given in figure 1. The explanation of above is as follows

1.According to RM scheduling algorithm task with shorter period has higher priority so T1 has high priority, T2 has intermediate priority and T3 has lowest priority. At t=0 all the tasks are released. Now T1 has highest priority so it executes first till t=0.5.
2.At t=0.5 task T2 has higher priority than T3 so it executes first for one-time units till t=1.5. After its completion only one task is remained in the system that is T3, so it starts its execution and executes till t=3.
3.At t=3 T1 releases, as it has higher priority than T3 so it preempts or blocks T3 and starts it execution till t=3.5. After that the remaining part of T3 executes.
4.At t=4 T2 releases and completes it execution as there is no task running in the system at this time.
5.At t=6 both T1 and T3 are released at the same time but T1 has higher priority due to shorter period so it preempts T3 and executes till t=6.5, after that T3 starts running and executes till t=8.
6.At t=8 T2 with higher priority than T3 releases so it preempts T3 and starts its execution.
7.At t=9 T1 is released again and it preempts T3 and executes first and at t=9.5 T3 executes its remaining part. Similarly, the execution goes on.

Dead-Line Monotonic Algorithm:

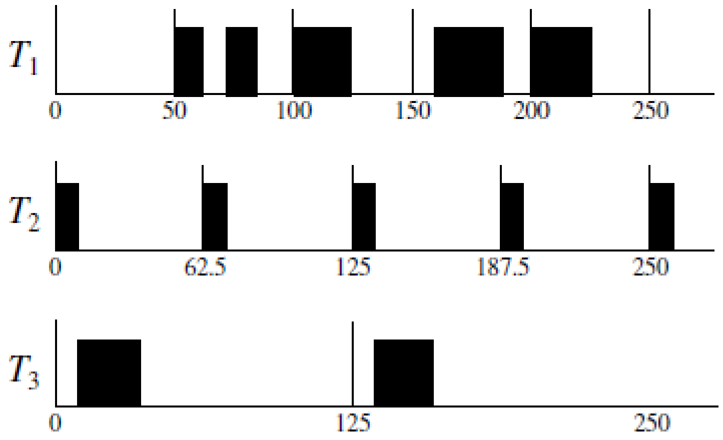
- Another well-known fixed-priority algorithm is the deadline-monotonic algorithm, called the DM algorithm.

- This algorithm assigns priorities to tasks according their relative deadlines: the shorter the relative deadline, the higher the priority.

Example:

Tasks	Release time(ri)	Execution time(Ci)	Deadline (Di)	Relative Deadline(Ti)
T1	50	25	50	100
T2	0	10	62.5	20
T3	0	25	125	50

- Their utilizations are 0.5, 0.16, and 0.2, respectively. The total utilization is 0.86. According to the DM algorithm, T2 has the highest priority because its relative deadline 20 is the shortest among the tasks. T1, with a relative deadline of 100, has the lowest priority. The resultant DM schedule I shown as



○

- Clearly, when the relative deadline of every task is proportional to its period, the RM and DM algorithms are identical.
- When the relative deadlines are arbitrary, the DM algorithm performs better in the sense that it can sometimes produce a feasible schedule when the RM algorithm fails.

DEFERRABLE SERVERS:

- A deferrable server is the simplest of bandwidth-preserving servers.
- The execution budget of a deferrable server with period ps and execution budget es is replenished periodically with period ps .
- When a deferrable server finds no aperiodic job ready for execution, it preserves its budget.

Operations of Deferrable Servers:

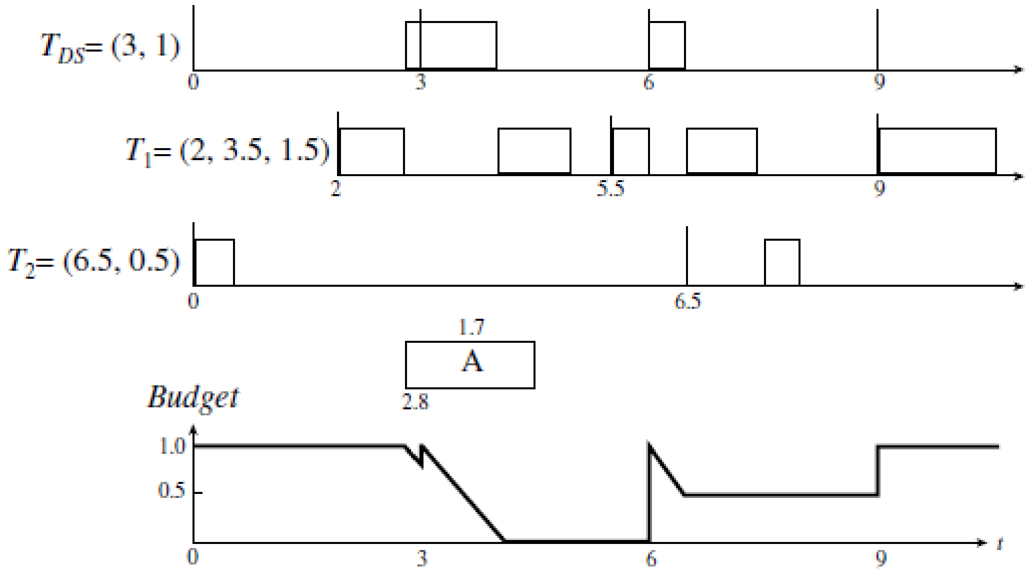
- Specifically, the consumption and replenishment rules that define a deferrable server (ps , es) are as follows.
- Consumption Rule: The execution budget of the server is consumed at the rate of one per unit time whenever the server executes.
- Replenishment Rule: The execution budget of the server is set to es at time instants kpk, for k = 0, 1, 2, . . We note that the server is not allowed to cumulate its budget from period to period.

Example:

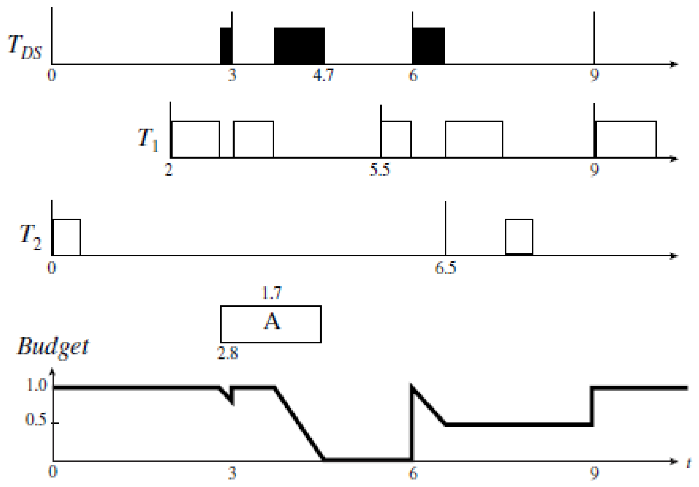
Tasks	Release Time	Time Period	Execution Time	Deadline
T1	2	3.5	1.5	3.5
T2	0	6.5	0.5	6.5
DS	0	3.0	1	3.0

Consider a deferrable server DS has the highest priority. The periodic tasks T1 and T2 and the server are scheduled rate-monotonically. Suppose that an aperiodic job A with execution time 1.7 arrives at time 2.8.

- At time 0, the server is given 1 unit of budget. The budget stays at 1 until time 2.8.When *A* arrives, the deferrable server executes the job. Its budget decreases as it executes.
- Immediately before the replenishment time 3.0, its budget is equal to 0.8. This 0.8 unit is lost at time 3.0, but the server acquires a new unit of budget. Hence, the server continues to execute.
- At time 4.0, its budget is exhausted. The server is suspended, and the aperiodic job *A* waits.
- At time 6.0, its budget replenished, the server resumes to execute *A*.
- At time 6.5, job *A* completes. The server still has 0.5 unit of budget. Since no aperiodic job waits in the queue, the server suspends itself holding this budget.

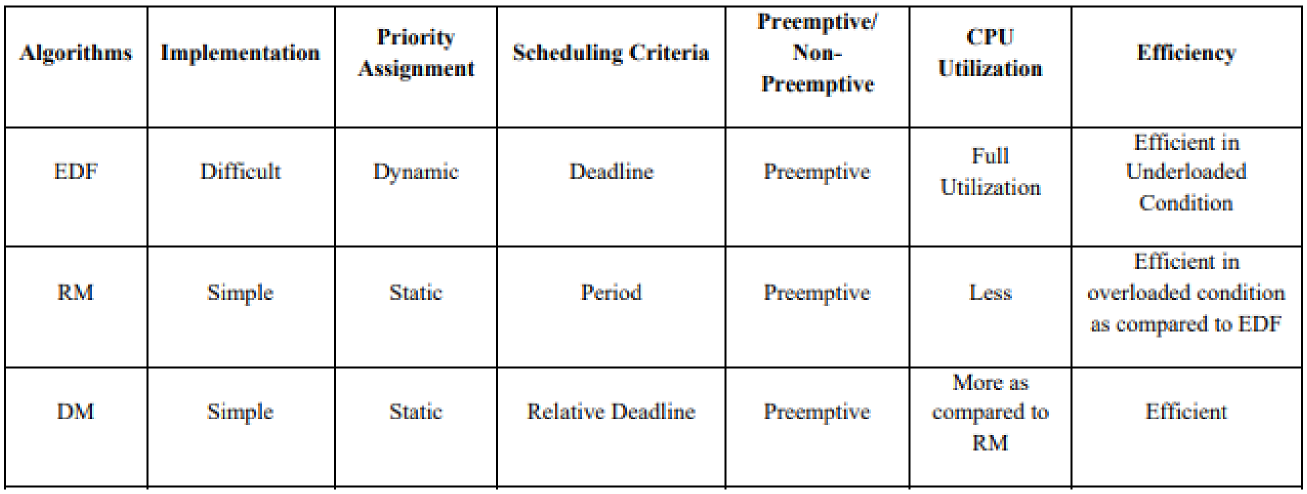


1. Consider another system that shows the same periodic tasks and the deferrable server scheduled according to the EDF algorithm. At any time, the deadline of the server is equal to the next replenishment time.
   - ➢ At time 2.8, the deadline of the deferrable server is 3.0. Consequently, the deferrable server executes at the highest-priority beginning at this time.
   - ➢ At time 3.0, when the budget of the deferrable server is replenished, its deadline for consuming this new unit of budget is 6. Since the deadline of J1,1 is sooner, this job has a higher priority. The deferrable server is preempted.
   - ➢ At time 3.7, J1,1 completes. The deferrable server executes until time 4.7 when its budget
     is exhausted.
   - At time 6 when the server's budget is replenished, its deadline is 9, which is the same as the deadline of the job J1,2. Hence, J1,2 would have the same priority as the server. The figure shows that the tie is broken in favor of the server.



COMPARISON OF REAL TIME TASK SCHEDULING ALGORITHMS:

Table 2. Comparison of real time scheduling algorithm



Algorithms	Implementation	Priority Assignment	Scheduling Criteria	Preemptive/ Non-Preemptive	CPU Utilization	Efficiency
EDF	Difficult	Dynamic	Deadline	Preemptive	Full Utilization	Efficient in Underloaded Condition
RM	Simple	Static	Period	Preemptive	Less	Efficient in overloaded condition as compared to EDF
DM	Simple	Static	Relative Deadline	Preemptive	More as compared to RM	Efficient

Unit – IV

Objectives:
To gain the knowledge on commonly used priority driven scheduling algoriths to schedule aperiodic and sporadic tasks.

Syllabus:

- **Resources and resource access control**:
  - Priority-Inheritance Protocol- rules, properties.
- **Multiprocessor scheduling and resource access control and synchronization:**
  - Model of multiprocessors and distribution systems: Identical versus Heterogeneous Processors, Inter process communication.
  - Multiprocessor priority ceiling protocol-blocking time due to resource contention, upper bounds to factors of blocking time.

Outcomes:
Students will be able to

- describe Priority-Inheritance Protocol- rules, properties
- use Priority-Inheritance Protocol
- describe properties of multiprocessor and distributed systems

- use Multiprocessor priority ceiling protocol(MPCP) in blocking time due to resource contention and upper bounds to factors of blocking time.

4.1 RESOURCES AND RESOURCE ACCESS CONTROL

4.1.1 PRIORITY-INHERITANCE PROTOCOL:

- The priority-inheritance protocol proposed by Sha.
- It works with any preemptive, priority-driven scheduling algorithm.
- It does not require prior knowledge on resource requirements of jobs.
- The priority-inheritance protocol does not prevent deadlock.
- When there is no deadlock, the protocol ensures that no job is ever blocked for an indefinitely long time.
- We call the priority that is assigned to a job according to the scheduling algorithm its assigned priority.
- At any time t, each ready job J_l is scheduled and executes at its current priority $\pi_l(t)$, which may differ from its assigned priority and may vary with time.
- The current priority $\pi_l(t)$ of a job J_l may be raised to the higher priority π_h (t) of another job J_h . When this happens, we say that the lower-priority job J_l inherits the priority of the higher priority job J_h and that J_l executes at its inherited priority π_h (t).
- The priority-inheritance protocol is defined by the following rules
- It assumes that every resource has only 1 unit.
- Rules of the 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 π(t) of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

2. Allocation Rule: When a job J requests a resource R at time t, (a) if R is free, R is allocated to J until J releases 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 Jl which blocks J inherits the current priority π(t) of J . The job Jl executes at its inherited priority π(t) until it releases R; at that time, the priority of Jl returns to its priority πl(t') at the time t' when it acquires the resource R.

Example:

- There are five jobs and two resources *Black* and *Shaded*.
- The parameters of the jobs and their critical sections are listed in part (a).
- As usual, jobs are indexed in decreasing order of their priorities: The priority *πi* of *Ji* is *i* , and the smaller the integer, the higher the priority.
- In the schedule in part (b) of this figure, black boxes show the critical sections when the jobs are holding *Black*. Shaded boxes show the critical sections when the jobs are holding *Shaded*.

1. At time 0, job *J*5 becomes ready and executes at its assigned priority 5. At time 1, it is granted the resource *Black*.
2. At time 2, *J*4 is released. It preempts *J*5 and starts to execute.
3. At time 3, *J*4 requests *Shaded*. *Shaded*, being free, is granted to the job. The job continues to execute.
4. At time 4, *J*3 is released and preempts *J*4. At time 5, *J*2 is released and preempts *J*3.
5. At time 6, *J*2 executes *L(Black)* to request *Black*; *L(Black)* fails because *Black* is in use by *J*5. *J*2 is now directly blocked by *J*5. According to rule 3, *J*5 inherits the priority 2 of *J*2. Because *J*5's priority is now the highest among all ready jobs, *J*5 starts to execute.
6. *J*1 is released at time 7. Having the highest priority 1, it preempts *J*5 and starts to execute.
7. At time 8, *J*1 executes *L(Shaded)*, which fails, and becomes blocked. Since *J*4 has *Shaded* at the time, it directly blocks *J*1 and, consequently, inherits *J*1's priority 1. *J*4 now has the highest priority among the ready jobs *J*3, *J*4, and *J*5. Therefore, it starts to execute.
8. At time 9, *J*4 requests the resource *Black* and becomes directly blocked by *J*5. At this time the current priority of *J*4 is 1, the priority it has inherited from *J*1 since time 8. Therefore, *J*5 inherits priority 1 and begins to execute.
9. At time 11, *J*5 releases the resource *Black*. Its priority returns to 5, which was its priority when it acquired *Black*. The job with the highest priority among all unblocked jobs is *J*4.

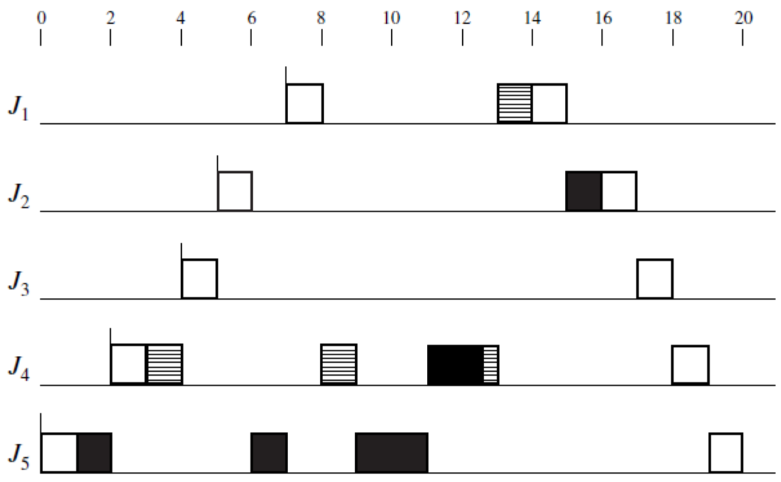
Consequently, *J*4 enters its inner critical section and proceeds to complete this and the outer critical section.

10. At time 13, *J*4 releases *Shaded*. The job no longer holds any resource; its priority returns to 4, its assigned priority. *J*1 becomes unblocked, acquires *Shaded*, and begins to execute.

11. At time 15, *J*1 completes. *J*2 is granted the resource *Black* and is now the job with the highest priority. Consequently, it begins to execute.

12. At time 17, *J*2 completes. Afterwards, jobs *J*3, *J*4, and *J*5 execute in turn to completion

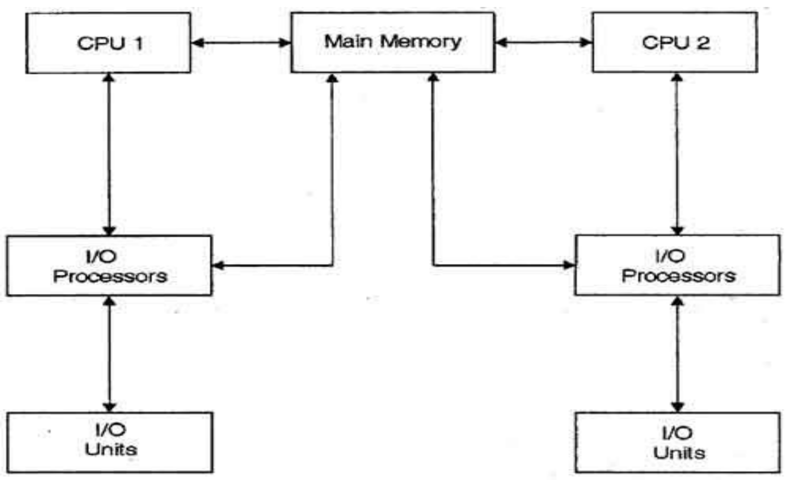
Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]



4.2 Multiprocessor scheduling and resource access control and synchronization:

4.2.1 Model of multiprocessor and distributed system:

- **Multiprocessor s**ystem refers to the use of two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral device



- We can classify multiprocessor systems as follows:
  - **Tightly coupled multiprocessing:** Consists of a set of processors that share a common main memory and are under the integrated control of an operating 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.
    - When each processor has its own scheduler, the actions and the decisions of the scheduler of all processors are coherent.
  - **Loosely coupled or distributed multiprocessor, or cluster:**
    - A distributed system is loosely coupled in such a system that it is costly to keep global status and workload information current.
    - The schedulers on the different processors may make scheduling and resource access control decisions independently. As a result, their decisions may be incoherent as a whole.
- It is assumed that each processor has its own scheduler in this chapter. Each scheduling, resource access control or synchronization algorithm will be evaluated to see how much the algorithm relies on the current global information, how much coordination among schedulers is required and therefore how suitable the algorithm is for loosely coupled systems.

4.2.2 Identical versus Heterogeneous Processors : There are two types of processors

- **Identical Processors:** The processors are of the same type or identical, if the processors can be used interchangeably.
  - For example, in a parallel machine, each of the CPUs can execute every computation job in the system, so the CPUs are identical.
  - If any message from a source to a destination can be sent on any of the data links connecting them, then the links are identical.

- **Heterogeneous Processors**: Different types of processors may have different functions. As an example, CPUs, file disk, and transmission links are functionally different. So, they cannot be used interchangeably. Processors can be of different types for many reasons.
    - For example, if the designer decides to use some CPUs for only some components of the system but not others, then the CPUs are divided into different types according to the components that can execute them. In a static system, the application system is partitioned into μ components and jobs in each component execute on a fixed CPU. CPUs are viewed as μ different processors.
- The model of heterogeneous processors used here is known as the unrelated processors model in scheduling theory literature. According to this model, each job can execute on some types of processors but, in general, not all types.
- Different types of processors may have different speeds. The execution times of each job on different types of processors are unrelated hence the name of the model.
    - For example, the execution time of a computation intensive job is 1 second in CPU1 but is 5 seconds on a less powerful CPU2. Because CPU2 has better interrupt handling and I/O capabilities, the execution time of an I/O intensive job is 10 seconds on CPU1 but is only 3 seconds in CPU2. Both jobs cannot execute on a transmission link and signal processor, so their execution time on these kinds of processors is infinite. The unrelated models allow us to characterize all system.

4.2.3. Interprocessor Communication:

- A special case of practical interest is where inter processor communication is via shared memory.
- Sometimes, we treat shared memory as a plentiful resource and do not include this resource in our model.
- The implication is that execution of a job is never adversely affected by memory contention; therefore, the cost of inter processor communication is negligible.
- We can model shared memory explicitly either as a resource or as a processor.

Example:

- The following figure(a) shows a possible configuration of a real-time monitor system that has many field processors.
- The producer jobs that collect and process sensor data execute on these processors.
- Consumer jobs that correlate and display the data execute on the control processor.
- The jobs communicate via a shared dual-port memory.
- Each filed processor is connected via a dedicated link to a port of the memory that is shared by all field processors
- For the effect of memory contention, we can model the shared memory as a processor.
- The system contains three types of processors
    - Field processor
    - Shared-memory processor
    - Control processor

- The workload consists of end-end jobs and each containing a memory-access job.
- The delay in the completion of each job caused by memory content is taken into account by the response time of the memory access component of the job.
- We can model the shared memory as a global resource whose synchronization processor is the shared memory processor and both producer and consumer tasks require this resource.
- Figure (b) shows another configuration where all the processors are connected via a network.
- Specifically the network is a Controller Area Network (CAN).
- The results produced by each job on the field processor are sent via the network to the control processor.
- Such a network can be modeled as a processor on which message transmission jobs are scheduled non preemptively on a fixed-priority basis.
- We do not need to include the interprocessor communication costs in our model because they are taken into account by the response times of the message-transmission jobs on the network.

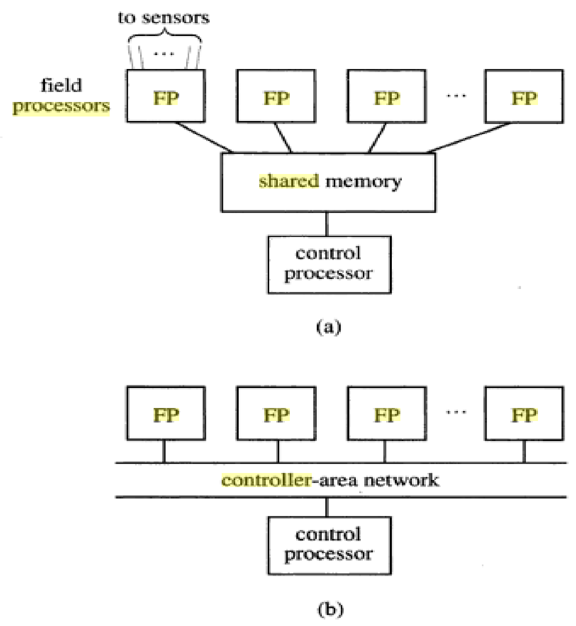


Fig: Inter processor communication architecture

- In general, communication networks can be modeled naturally as a processors.
- The scheduling algorithms used on some networks such as the IEEE 802.5 token ring and

CAN, more or less resemble priority-driven algorithms used for CPU scheduling.

- Other types of networks such as FDDI network may use very different scheduling algorithms.

- In all cases, we can take into account inter processor communication costs by including work
processor and message transmission job in our model. There is no need to consider this factor separately in some ad hoc manner.

- Finally, CPUs may be connected via dedicated links. When messages transmissions between a
pair of CPUs are under program control, each simplex link can be modeled as single-unit resource on the sending CPU.

- When a job sends a message, it is in a critical section using this resource.
- When message transmissions are done via a DMA interface, we can take into account the time required for sending each message by lengthening the execution time of job that execute concurrently with DMA controlled message transmission.
- In both cases, we can take into account the effect of inter processor communication by adjusting the execution time and blocking time parameters of all the jobs in the system accordingly.

4.2.4 Multiprocessor priority ceiling protocol

- The multiprocessor priority ceiling assumes that the tasks and resource have been assigned and
statically bound to processors.

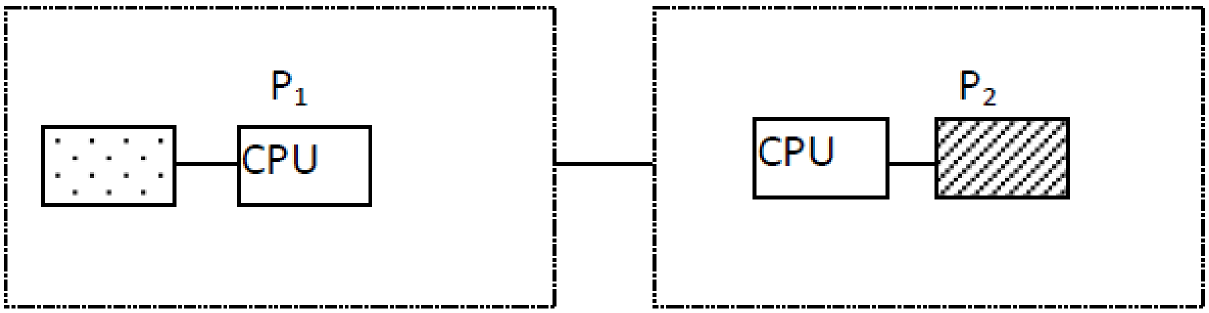
- The scheduler of every synchronization processor knows the priorities and resource requirements of all the tasks requiring the global resources managed by the processor.
- It is also assumed that the resources used by every job during nested critical sections lies on the same processor.
- According to this protocol, the scheduler of each processor schedules all the local tasks and global critical sections on the processor on a fixed priority basis and controls their resource accesses according to the basic priority ceiling protocol.
- According to the MPCP model, when a task uses a global resource, its global critical section executes on the synchronization processor of the resource.
- If the global section of a remote task were to have a lower priority than some local task on the synchronization processor, these local tasks could delay the completion of the global critical section and prolong the blocking time of the remote task.
- For preventing this, the multiprocessor priority ceiling protocol schedules all the global critical sections at higher priorities than all the local tasks on every synchronization processor.
- This can be implemented in a system where the lowest priority πlowest -all the tasks is known.

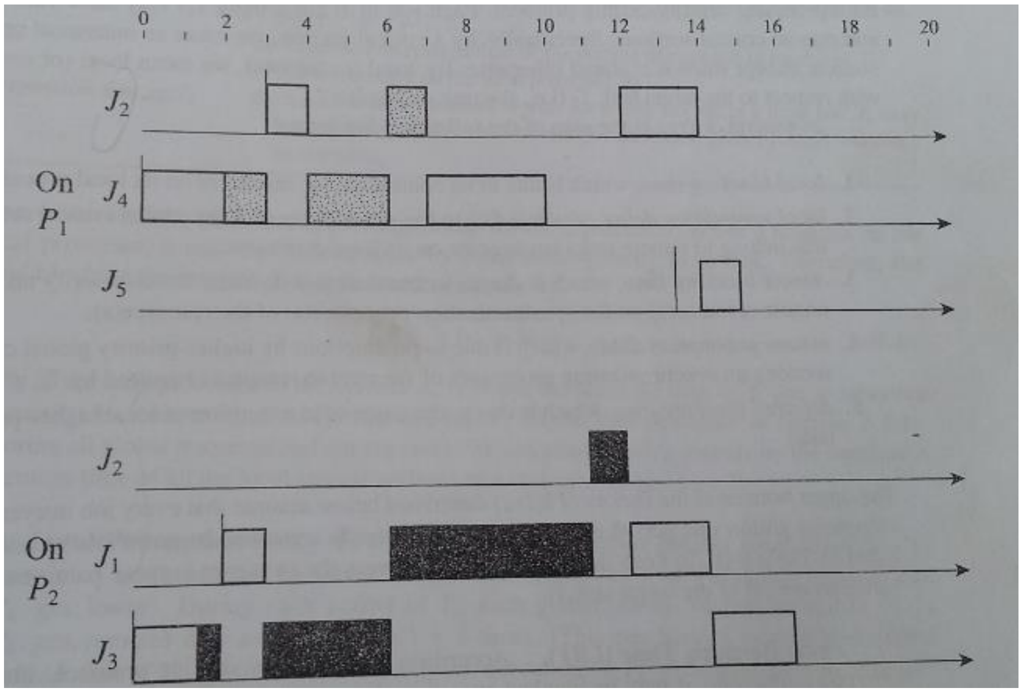
- The scheduler of each synchronization processor schedules the global critical sections of a task with priority πi at priority πi - πlowest.
- As an example:
  - Consider a system where tasks have priorities 1 through 5, πlowest is 5.
  - A global critical section of a task with priority 5 is scheduled at priority 0, which is higher that priority 1.

  - Also, the priority of a global critical section of a task with priority 1 is – 4 which is the highest priority in the system

4.2.4.1 Blocking time due to resource contention

- The figure below shows the illustration that the types of blocking a job may suffer under the multiprocessor priority ceiling protocol.
- The system has two processors P1 and P2. Jobs J2, J4 and J5 are local to processor P1, which is the synchronization processor of the resource dotted.
- Dotted is a local resource since it is only required by local jobs J2 and J4. Jobs J1 and J3 are local to processor P2, which is the synchronization processor of the resource black.
- Black is a global resource and is required by Jobs J1, J2 and J3.
- The jobs are arranged in decreasing order. The short vertical bar on the timeline of each jobs marks the release time of the job.
- Here, J2 is directly blocked by J4 when J2 requests dotted at time 4.
- J1 is directly blocked by J3 when J1 requests black at time 3.
- The global critical section of J2 on P2 is delayed by the global critical section of the higher priority J1 in the time interval (7, 11].
- The preemption delay thus suffered by J2 must be taken into account when the schedulability of J2 is to be determined. This delay is treated as J2's blocking time.
- At time 11, J1 exits from its critical section. Its priority is lower than the priority of the global critical section of J2. As a result, J2 preempts J1 on P2 in the interval (11, 12].
- The total delay experienced by a job as a result of preemption by global critical sections of lower priority jobs is also a factor in the total blocking time of the job.
- Finally, a job can be delayed by a local higher priority job whose execution is suspended on the local processor when the priority job executes on a remote processor.
- This delay is another factor of the blocking time of the job. Here in this example, J2 is suspended at time 7.
- As a result, it is still not completed in (13, 14] and J5 released at time 13.2 cannot start until time 14.





4.2.4.2 Upper bounds to factors of blocking time bi(rc): It is the sum of following five terms

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 Ti 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 Ti ; and
e. deferred blocking time, which is due to the suspended execution of local higher-priority tasks.

1. Local Blocking time(LBT):

$$LBT = (\kappa_r + 1)\hat{c}_L$$

2. Local Preemption Delay(LPD):

$$LPD = \sum_{T_k \in \mathbf{T}(P_L, \mathrm{gcs}, \mathrm{remote})} (\lceil p_i / p_k \rceil + 1)\hat{c}_{k,\mathrm{total}}(P_L) + \sum_{T_k \in \mathbf{T}(P_L, gcs, lower)} (k_r + 1)\hat{c}_{k,\max}(P_L)$$

3. Remote Blocking Time(RBT):

$$RBT = \sum_{j=1}^{\kappa_r} \max_{T_k \in \mathbf{T}(P_j, \mathrm{gcs}, \mathrm{lower})} (\hat{c}_{k,\max}(P_j))$$

4. Remote Preemption Delay(RPD):

$$RPD = \sum_{T_k \in \mathbf{T}(\mathrm{gcs}, \mathrm{higher})} (\lceil p_i / p_k \rceil + 1)\hat{c}_{k,\mathrm{total}}$$

5. Deferred Blocking time(DBT):

$$DBT = \sum_{T_k \in \mathbf{T}(\mathrm{local}, \mathrm{higher})} \min(e_{k,L}, suspension\_times\_\mathrm{of}\_T_k)$$

(Or)

$$DBT = \sum_{T_k \in \mathbf{T}(\mathrm{local}, \mathrm{higher})} e_{k,L}$$

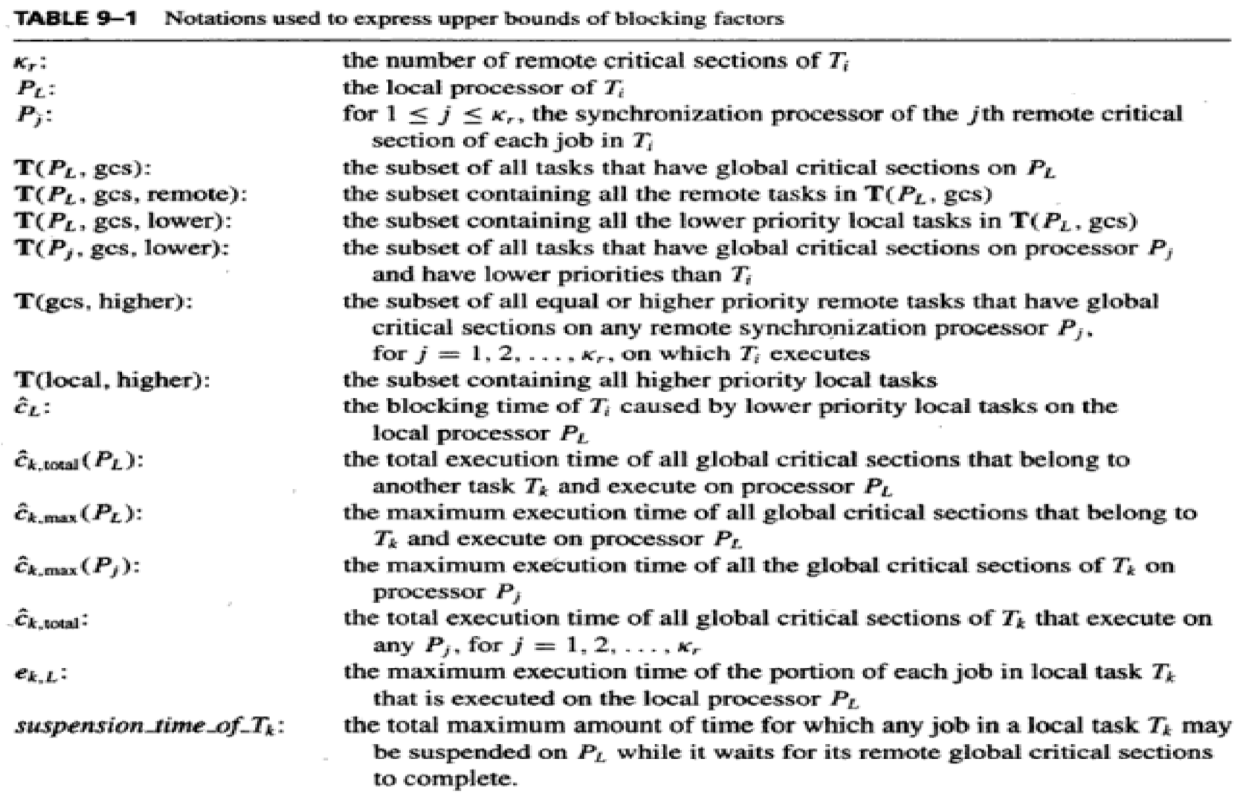


TABLE 9–1 Notations used to express upper bounds of blocking factors

κ_r:	the number of remote critical sections of T_i
P_L:	the local processor of T_i
P_j:	for $1 \le j \le \kappa_r$, the synchronization processor of the jth remote critical section of each job in T_i
$\mathbf{T}(P_L, \text{gcs})$:	the subset of all tasks that have global critical sections on P_L
$\mathbf{T}(P_L, \text{gcs, remote})$:	the subset containing all the remote tasks in $\mathbf{T}(P_L, \text{gcs})$
$\mathbf{T}(P_L, \text{gcs, lower})$:	the subset containing all the lower priority local tasks in $\mathbf{T}(P_L, \text{gcs})$
$\mathbf{T}(P_j, \text{gcs, lower})$:	the subset of all tasks that have global critical sections on processor P_j and have lower priorities than T_i
$\mathbf{T}(\text{gcs, higher})$:	the subset of all equal or higher priority remote tasks that have global critical sections on any remote synchronization processor P_j, for $j = 1, 2, \ldots, \kappa_r$, on which T_i executes
$\mathbf{T}(\text{local, higher})$:	the subset containing all higher priority local tasks
$\hat{c}_L$:	the blocking time of T_i caused by lower priority local tasks on the local processor P_L
$\hat{c}_{k,\text{total}}(P_L)$:	the total execution time of all global critical sections that belong to another task T_k and execute on processor P_L
$\hat{c}_{k,\max}(P_L)$:	the maximum execution time of all global critical sections that belong to T_k and execute on processor P_L
$\hat{c}_{k,\max}(P_j)$:	the maximum execution time of all the global critical sections of T_k on processor P_j
$\hat{c}_{k,\text{total}}$:	the total execution time of all global critical sections of T_k that execute on any P_j, for $j = 1, 2, \ldots, \kappa_r$
$e_{k,L}$:	the maximum execution time of the portion of each job in local task T_k that is executed on the local processor P_L
suspension_time_of_T_k:	the total maximum amount of time for which any job in a local task T_k may be suspended on P_L while it waits for its remote global critical sections to complete.

Unit – V

Objectives:

To gain the knowledge on Scheduling flexible computations and tasks with temporal distance constraints.

Syllabus:

Scheduling flexible computations and tasks with temporal distance constraints

- Flexible applications, algorithms for scheduling flexible application-constrained optimization.
- Tasks with temporal distance constraints: temporal distance model, DCM algorithm.

Outcomes:

Students will be able to

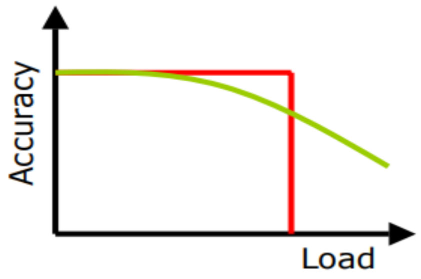
- characterize flexible applications.
- use constrained optimization algorithm for scheduling flexible application.dule t
- schedule tasks with temporal distance constraints.
- use DCM algorithm with Harmonic and Arbitrary distance constraints

5.1 <u>Flexible computations and Flexible Applications:</u>

- ➢ Some real-time applications must tolerate fluctuation in available resources or workload.
- ➢ A real-time network server may receive more traffic than expected.
- ➢ A failure may divert load onto a backup system.
- ➢ In particular, a flexible application can reduce its time and resource demands at expense of the quality of its result.
- ➢ For as long as the user finds its result quality acceptable, a flexible application can degrade gracefully when resources are scarce and the demands of competing workloads are high.
- ➢ In recent years, the flexible computation approach has been proposed as a means for handling overload and increasing availability of applications in domains as diverse as Artificial Intelligence (AI), signal processing and tracking, real-time communication, and databases.
- ➢ The timing constraints of many applications can be characterized more conveniently and naturally by temporal distances rather than deadlines.
- ➢ Real-time performance may degrade due to load from non-real-time tasks sharing the processor A real-time system has two degrees of flexibility when it becomes impossible to meet all deadlines :
    - Graceful degradation in timeliness.
    - Graceful degradation in quality.

Flexible computations : The ability to trade-off, at run time, quality of results for the amount of time and resources used to produce those results .

- As a system moves into overload, it gracefully degrades rather than suddenly failing.
- Assumption: a timely result of poor quality is better than a high quality, but late, result .



Examples:

- Multimedia: a fuzzy picture is better than no picture.
- Air traffic control: prefer system to keep working, with error bars, than to fail completely on overload.

A timely warning of collision, with estimated location better than an exact location, delivered too late to avoid collision.

5.1.1<u>**Characterization of Flexible Applications:**</u> Jobs have an optional component and a mandatory part.

- If sufficient resources, both mandatory and optional parts complete; a precise result.
- If limited resources, the optional component is discarded, giving an imprecise result.

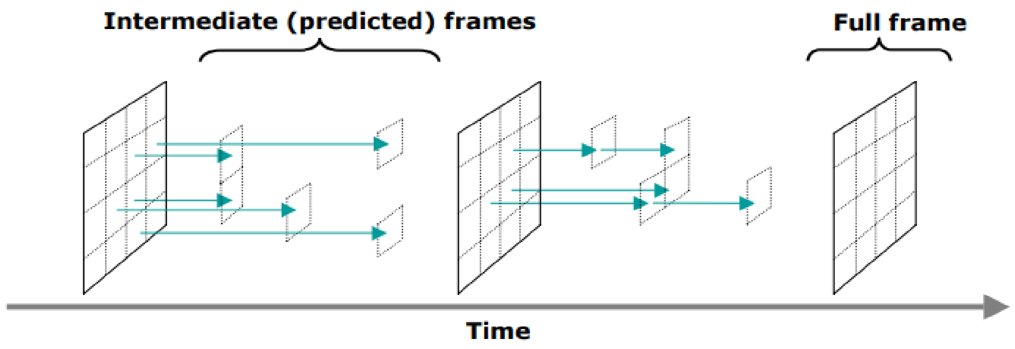
How to implement?

- Sieve method
- Milestone method
- Multiple version method
- Workload Model
- Criteria of Optimality

5.1.2 <u>**Implementation methods :**</u>

1). <u>**Sieve Method:**</u> A flexible task comprises a mixture of mandatory and optional jobs In times of overload, some optional jobs are discarded in their entirety .

- Example:
    - Another example is the job that estimates the current level of background noise at the receiver in a radar signal processing system.
    - Encoding MPEG video



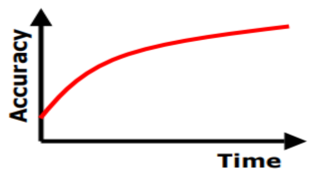
Can be flexible by either:

– stop encoding predicted frames ⇒ reduced frame rate

– delay encoding full frame ⇒ reduced bandwidth, error tolerant.

2). <u>Milestone Method</u> : •The system regularly checkpoints the result of the optional job as a set of milestones. When the deadline is reached, the job terminates and the latest milestone is retrieved.

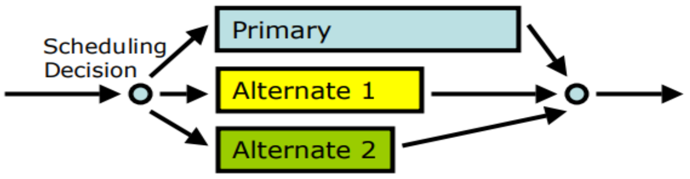
- **Monotone** is a job where the optional component can be stopped at any time, and the quality of the result always increases with longer execution.
  - – Iterative numerical computation
  - – Iterative statistical computation
  - – Layered video encoding



- A monotonic job makes the scheduling decision easier, since longer execution, after the mandatory part, always improves quality – Otherwise needs watch result quality, to know when to stop.

3). <u>Multiple Versions</u> : The flexible job is implemented in multiple versions.

- Primary is high quality, but has a larger execution time and resource usage.
- Alternates are lower quality, but execute quicker or use less resources.



- The scheduler must make an a-priori decision on which version to execute, based on load at the start of the job.
  - Requires more intelligence in the scheduler than sieve or milestone methods.
- Little gain from having more than one alternate

4). **<u>Criteria of Optimality</u>:** Algorithms for scheduling flexible applications have two objectives.

Correctness: Finding a feasible schedule that ensures all mandatory jobs complete.

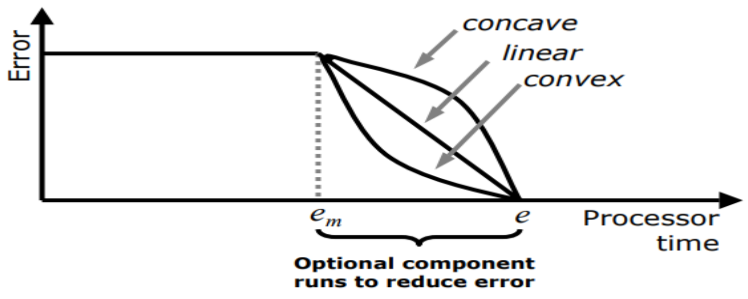
Quality of result: Try to fit in as many optional jobs as possible, to reduce the error in the result.

– Measure the error according to some domain specific metric

• Can be difficult to characterize.

– Clearly desirable if the error function is convex

• May influence choice of algorithm, for milestone based jobs.



5.1. 3 <u>Algorithms for scheduling flexible Applications</u>

- This section gives an overview of algorithms for scheduling flexible applications on a processor.
- Some of these algorithms are off-line and others are on-line.
- off- line algorithms aim at finding an optimal static schedule.
- This term refers to a schedule to a schedule of the given system that

    (1) is feasible whenever the mandatory components of the system are schedule and

    (2) Minimizes one of the static quality metrics (i.e., average error, total error, and maximum error) of the system.

- Most on-line algorithms a static quality metric, while others are designed to prevent dynamic failures.

Constrained optimization formulation:

- When all the parameters of all jobs in the system are known, it is possible to compute an optimal static schedule off- line at design time or configuration time.
- The problem of finding such a schedule is typically formulated as a constrained optimization problem.

<u>Illustrative example:</u>

- *To* illustrate, we consider the problem of scheduling a system of flexible periodic tasks T_i = (p_i , e_i), for i=1,2,3,…n, to minimize average error of the system.
- The tasks are truly periodic.
- The average error is given by Eq. 10.3, and the window Li over which the average error of T_i is computed is equal to H/ p_i , when H is the length of a hyper period of the system.
- When the error functions are linear or convex and feasible schedules of the system exist, there is always an optimal static schedule which allocates the same amount of processor time to all optional jobs of each task by their respective deadlines.
- In other words, we can confine our search for an optimal schedule among those feasible schedules according to which the length x_i ,k of the completed portion of every optional

job j_i ,k in task Ti is equal to x_i for every i=1,2,….n. Because x_i , k= x_i , the average error of T_i is simply equal to $\varepsilon_i(x_i)$.

Thus the problem of finding optimal static schedule of the system is reduced to the problem of finding the set { $x_1, x_2, \ldots x_n$ } of processor allocations to jobs in the optional tasks that satisfies the following constraints :

$$0 \le x_i \le e_0;$$

$$\sum_{i=1}^{n} \frac{e_{m,i} + x_i}{p_i} \le 1 \qquad \text{for i=1,2,3,…n}$$

And minimizes the objective function

$$E_{ave} = \sum_{i=1}^{n} wt_i \varepsilon_i(x_i) \text{ (or any of the static quality metrics).}$$

- The first constraint follows from the definition of valid schedule.
- The second constraint is the necessary and sufficient condition for the existence of a feasible schedule that allocates $e_{m,i} + x_i$ units of processor time by the deadline of the job to every job in every task Ti.
- A system of periodic tasks (p_i , $e_{m,i} + x_i$), for i= 1,2,…n, is schedulable by the EDF algorithm if the above constraints are met .
- By definition, the solution of the problem yields the minimum average error among all feasible schedules of the given systems.
- When the error functions of all tasks are linear, these three expressions define a linear program.
- For a system containing tens or even hundreds of tasks, the running time of a linear program solver may be low enough for use at run time as an acceptance test.
- The problem is considerably more complex when the error functions of the tasks are nonlinear, especially if the tasks have different error functions, as these functions typically are in practice.
- In particular,when the error functions of some tasks are concave, we cannot no longer confine our search for an optimal static schedule among schedules that allocate the same amount of processor time to all jobs in each task.
- When the error functions are concave, the problem of finding an optimal static schedule is NP hard, just like the problem of scheduling jobs with 0/1 constraints is.

<u>General constrained optimization formulation:</u>

- To describe the constrained optimization formulation in general , we suppose that the given system consists of n jobs j_i , for i= 1, 2, 3,…n (For periodic tasks , n is the number of jobs in each hyper period) the known parameters of each job j_i are its release time r_i , deadline d_i , mandatory execution time $e_{m,i}$, is allocated x_i units of processor time by the job's deadline .
- The release times and deadlines of all the jobs partition the time interval from the earliest release time to the latest deadline of all jobs into disjoint intervals , each of which does not contain any release time or deadline .
- Let I_k , for k= 1, 2…k ($k \leq 2n-1$), denote these time intervals, (For eg: suppose there are two jobs and their release times and deadlines are 0, 2, 5 and 6. These time instants partition the time interval from 0 to 6 into three intervals (0,2], (2,4], and (5,6].)
- For the sake of convenience, we let $t_k andt_{k+1}$ denote the beginning and the end of the interval I_k ,respectively .
- In other words, $I_k = (t_k , t_{k+1}]$.
- Each of these instants is either a release time or a deadline of some job in the system.
- Let a_i ,k denote the amount of processor time in interval I_k that is allocated to job j_i .
- The problem of finding an optimal static schedule of the n jobs can be stated as follows ; We want to find allocations a_i ,k , for i=1,2,…k Subjected to the constraints

a_i ,k≥0 for i=1,2,…n; k=1,2,…k (a)

a_i ,k=0 for i=1,2,…n; k=1,2,…k (b)

$$0 \leq \sum_{i=1}^{n} a_{i,k} \leq t_{k+1} - t_k \quad \text{for k=1,2,….k} \quad \text{(c)}$$

And $e_{m,i} \leq \sum_{k=1}^{k} a_{i,k} \leq e$ if the job j_i is monotone (d)

Or $\sum_{k=1}^{k} a_{i,k} = e_{m,i} \; or \; e$ if the job j_i has 0/1 constraint (f)

Equation(a) states the obvious fact that $a_{i,k}$ must be nonnegative.

The constraint of Eq. b follows from the fact that a job can be schedule only in the intervals that are within its feasible interval $(r_i, d_i]$.

Equation (c) ensures that the total amount of processor allocations of each monotone job must satisfy eq(d),which makes sure that the total amount of processor time in all intervals that is allocated to the job is no less than its mandatory execution time and no greater than its execution time .

The processor allocations of each job with 0/1 constraint must satisfy eq. 10.5e.

The optimal static schedule is given by the allocation $a_{i,k}$'s that satisfy the above constrains and minimize the objective function , which is either the total error E_{total} or the maximum error E_{max} given by Eq(2) or is the average error

$$E_{AVE} = \sum_{I=1}^{N} Wt_i, \epsilon_i(x_i) \text{(g)}$$

Amount x_i of processor time allocated to the optional component of j_i is given by

$$x_i = \sum_{k=1}^{k} a_{i,k} - e_{m,i} \text{ (h)}$$

Again, the complexity of this optimization problem depends on the error functions and weights of jobs.

The simplest case is when jobs have identical weights.

When jobs have identical weights and error functions are linear, an optimal static schedule that minimizes the total (average) error of n monotone jobs can be found by a simple O(n log n) algorithm.

When jobs have convex error functions and identical weights, the total error of all jobs can be kept small by making the ratio xi/eo,i of the processor time allocation xi of each optional job to its execution time equal to this ratio of every other optional job as much as possible.

In other words, we want to minimize max1≤i≤n(1-xi/e0,i).

An optimal schedule can be found in O(n2) time when optional jobs have identical execution times and in O(n3) time when the jobs have arbitrary execution times.

When jobs have arbitrary weights, we have a linear program to solve in the simple case when all the error functions are linear.

In general, the constrained optimization problem ranges from a quadratic programming problem to an NP-hard problem.

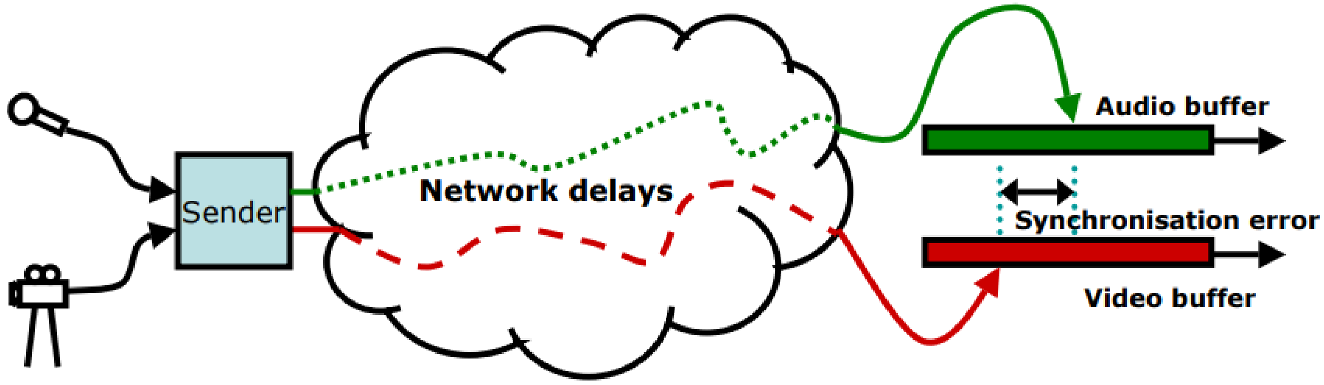
The constrained optimization problem can be generalized in a straightforward manner to take into account the dependency of result qualities on the allocations of other resources in addition to processor time.

The added dimensions in the resource versus quality trade-off increase the number of constraints the solution must meet.

The complexity of the problem can be kept tractable only by keeping the error functions simple (e.g., linear) and the constraints linear.

5.2 Tasks with temporal distance constraints:

- This section describes algorithms for scheduling tasks that are required to meet temporal distance constraints. An example of tasks with temporal distance constraints is Lip Synchronization.
- Lip synchronization is a common operation in multimedia applications.
- Audio and video decoding must complete within a short period of time, or the lip-sync is broken



Temporal Distance Model: To define the term distance constraint precisely, consider a task Ti comprises a chain of jobs Ji,k for k = 1, 2, …n. The first job, Ji,1, is released at time φi . This means first job Ji,1 in Ti is ready for execution at φi, and each subsequent job, Ji,k+1 where (k≥1), becomes ready when its predecessor, Ji,k, completes .Ti is an end-to-end task, and φi is its phase.

- Let *f*i,k completion time/finish time of kth job Ji,k .
- The *temporal distance*(or simply distance) between this job $J_{i,k}$ and next job $J_{i,k+1}$ is $f_{i,k+1}$-$f_{i,k}$ is difference between their completion time.
- The task Ti has a *Temporal distance constraint* Ci, if :

  fi,1 – φi ≤ Ci (Initial job)

  fi,k+1 – fi,k ≤ Ci for k = 1, 2, … (Later job)

If the completion time of all the jobs in Ti according to a schedule satisfy these inequalities then Ti meets its distance constraint Ci (i.e Jobs must complete within time Ci of their predecessor).

- A schedule of a system T consisting of tasks T_1, T_2, …….T_n with distance constraints C_1,C_2,….C_n is feasible if every task in T meets the task's distance constraint. A system is schedulable according to an algorithm if the algorithm surely produces a feasible schedule.
- We index the tasks in order of their distanceconstraints i < j, then Ci < Cj.

- We said that the distance constraints $C_1,C_2,\ldots.C_n$ for n tasks are harmonic if Ci divides Cj for every pair of i < j.
- Density of a task, Ti, with execution time ei and temporal distance constraint Ci is $\delta_i = \frac{e_i}{C_i}$
- Density Δ of the system $\Delta = \sum_{i=1}^{n} \delta_i$
- Just like periodic and sporadic tasks, tasks with temporal distance constraints are preemptable. However , it is reasonable to disallow arbitrary preemption. In particular, the scheduler is not allowed to preempt a job $J_{i,k}$,just before its completion, leaving an infinitesimally small portion to be complete later.
- if the scheduler is allowed this preemption, then the problem of scheduling T_i to meet the distance constraint Ci is trivial: The scheduler simply schedules the last infinitesimally small portion of the job in Ti periodically Ci units apart and then schedules the reaming portions of the jobs as if they were released periodically with period Ci.

5.2.1 <u>Distance Constraint Monotonic(DCM) Algorithm</u> :

- Distance Constraint Monotonic (DCM) Algorithm does not preempt jobs arbitrarily.
- DCM can schedule tasks to explicitly meet distance constraints if appropriate
  - If you care about inter-job timing, as well as each job meeting its deadline.
  - Jobs not only meet deadlines, they occur with Ci of the actual completion time of an earlier task.
- Use a fixed priority scheduling algorithm, similar to deadline monotonic: Distance Constraint Monotonic scheduling.
- The algorithm has two elements: priority assignment and job separation constraints.

i). Assign ***task priorities*** monotonically according to distance constraint

- ➢ Smaller the distance constraint Ci of task Ti, higher the task's priority. Therefore, tasks with indices i-1 or less have higher priorities than Ti, for all i=1,2,…,n.
- ➢ Jobs run with the fixed priority of the task to which they belong

ii). Provide ***separation*** between jobs to allow low priority tasks to run and meet their constraints

- ➢ The scheduler were to let each successor job on $J_{i,k+1}$ (k>=1) in a task Ti with distance constraints be ready for execution as tasks. This is why the scheduler imposes a separation constraint between consecutive jobs in each task Ti.
- ➢ The Separation constraint between consecutive two jobs $J_{i,k}$ and $J_{i,k+1}$ (k>=1) in Ti is the minimum length of time between the completion time of $f_{i,k}$ of a job $J_{i,k}$ and the ready time $r_{i,k+1}$ of its immediate successor $J_{i,k+1}$. i.e "When a job completes, delay it's successor as long as possible, to allow jobs from lower priority tasks to run".
- ➢ It improves schedulability.

- According to DCM algorithm, the delay Ci – Wi is the separation constraint on job in each task Ti , chosen to ensure that jobs complete and just meet their temporal distance constraint, where Wi is maximum response time of job in Ti.
- The scheduler computes the ready times of jobs in Ti, according to

$$r_{i,1} = \varphi$$

$$r_{i,k+1} = f_{i,k} + Ci - Wi \qquad \text{for } k<=1$$

- The delay Ci–Wi is the separation constraint, chosen to ensure that jobs complete and just meet their temporal distance constraint.
- An implicit assumption here is Wi ≤ Ci., this condition must hold; otherwise it is impossible for Ti to meet its distance constraint Ci.
- Once released, job is scheduled according to task priority
  –Might not execute immediately…
- A system T that is schedulable according to the DCM algorithm, the maximum response Wi of a jobs in each task Ti can be obtained as follows:
  - ➢ Find maximum response time W1of highest priority task T1
  - ➢ For each Ti for i>1, find Wi after deriving a DCM schedule for all higher priority tasks, assuming all tasks are released at time 0(worst case response time, when all tasks start at once).
- A distance constraint task is similar to an end-to-end task in the sense that it consists chain of jobs. The DCM scheduler gives each task a fixed prority; according to DCM algorithm , atask with a smale distance constraint has a higher priority.

i) **Scheduling Tasks with Harmonic Distance Constraints:** The temporal distance between a job $J_{i,k+1}$ and its immediate predecessor is Ci- Wi + y .,When the response time of $J_{i,k+1}$ is y. By definition y ≤ Wi, this distance is no greater than Ci.

- ➢ Again the critical assumption is that Wi ≤ Ci., this condition must hold; otherwise it is impossible for Ti to meet its distance constraint Ci.
- ➢ Distance constraints are harmonic: longer constraints are always integer multiples of shorter–separation constraint as previously described

Theorem: The system is schedulable and meets temporal distance constraints if Δ≤1
Proof: The system devolves into a rate monotonic schedule with period Ci.

ii) **Scheduling Tasks with Arbitrary Distance Constraints:** If we relax the system definition to have arbitrary temporal distance constraints, it becomes difficult to prove schedule correct. Can transform such a system into one with harmonic temporal distance constraints through the **specialization** operation.

Specialization: A way to improve the schedulability of tasks with arbitrary distance constraints is to first transform them into tasks with harmonic distance constraints. The operation carried out for this purpose is called specialization

Parameters of Accelerated Tasks: Given a system of tasks *T1*, *T2*, …, *Tn* with distance constraints *C1*, *C2*, … *Cn*the specialization operation transforms it into a set of **accelerated tasks.** The accelerated tasks *T1'*, *T2'*, … *Tn'* have distance constraints *C1', C2'*, …, *Cn'*.Where:

1. The execution time of *Ti'*equals the execution time of *Ti*
2. The distance constraint *Ci'≤Ci*
3. The new distance constraints are harmonic

Tighten the distance constraints, reducing the schedulable utilisation of the system, but allowing proof of schedulability.

****

Unit – VI

Objectives:

To gain the knowledge on model of real-time communication architecture, Medium Access Control protocol in DQDB networks and operating system function, open system architecture.

Syllabus:

Real-Time Communications

- Model of real time communication: architecture, real-time connections and service disciplines-packet switched networks.
- Medium access-control protocols of broadcast networks: medium access control protocol in DQDB networks: DQDB architecture

Operating Systems

- Threads and tasks, Kernel- structure of microkernel, interrupts
- Memory management
- I/O and networking
- Open system architecture: objectives, two-level scheduler.

Outcomes:

Students will be able to

- describe model of real time communication.
- specify service disciplines of real time communication.
- design packet switched networks.
- describe medium access control protocol in DQDB networks.
- describe structure of microkernel and interrupts handling.
- explain open source architecture.

6.1 <u>Model of real time communication</u>: It is a well known model of distributed system. The hosts are connected by a communication networks or several interconnected networks. The top layers are simplified and all the entities above all the transport layer applications are called.

6.1.1 <u>Architecture Overview :</u> It mainly focuses on messages exchanged among applications on different hosts. **Real-time communications** (RTC) is a term used to refer to any live telecommunications that occur without transmission delays. RTC is nearly instant with minimal latency. RTC data and messages are not stored between transmission and reception.

- The source and destinations of every message are application tasks residing on different hosts.
- The network interface of each host contains an input queue and an output queue which can also be referred to as input/output buffers or simply buffers.

- For the sake of correctness, it is assumed that these queues are jointly maintained by two local servers: the transport (TP) handler and Network Access Control (NAC) handler.
  - ➢ The TP handler interfaces with local applications and provides them with message transport services.
  - ➢ The NAC handler interfaces with the network below and provides network access and messages transmission services to the TP handler.

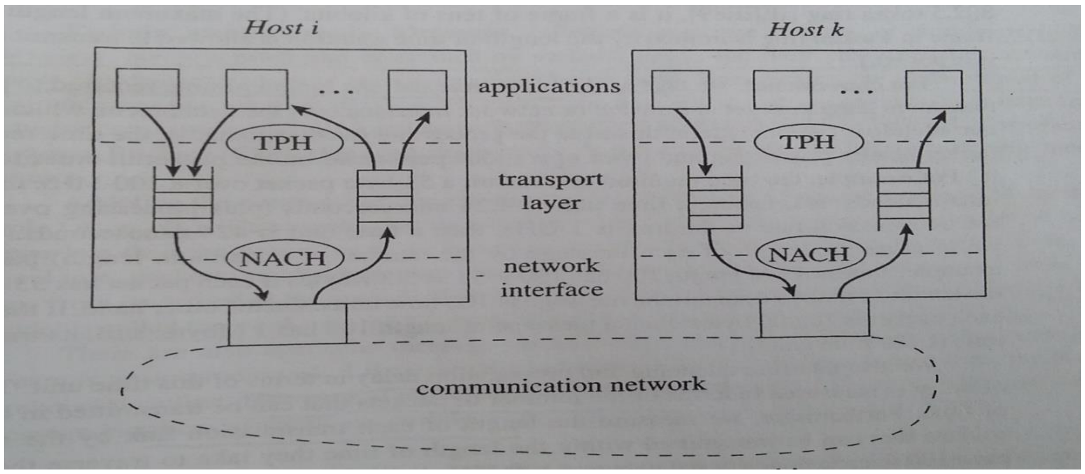


Figure : Real-Time Communication

- Figure shows the data paths indicated by heavy arrows traversed by messages in and out of two hosts. The circles marked TPH and NACH are TP and NAC handlers.
- When requested to send a message by a local application task, the source TP handler places the message in that output queue. From there, each outgoing message is delivered to the network under the control of the source NAC handler.
- After the message has traversed to the network, the destination NAC handler places the message in the input queue and notices the destination TP handler. Then, the destination TP handler moves the message to the address space of the destination application task and notifies the application of the arrival of the message.
- **Packets:** Messages are fragmented into segments before transmission through communication network. Each segment is handled by the network as a basic transmission unit called a frame, a packet or a cell. The transmission of the unit is non preemptable. Example, a packet is a 53-byte cell in an ATM network

<u>Real Time Traffic Models :</u>

- In real time communication, the term ‘real time traffic’ means isochronous of synchronous traffic consisting of message streams which are generated by their sources on a continuing basis and delivered to their respective destinations on a continuing basis.
- Such traffic includes periodic and sporadic messages that require some guarantee for on time delivery. There are also aperiodic(asynchronous) messages.
- Each of these type of message is referred to as a message stream and is denoted by M_i for some index i to distinguish it from other messages.

<u>Performance Objectives and Constraints</u> :

We want to measure the performance of scheduling, synchronization, and flow control algorithms used for real time communication and the performance of the resultant communication system along two dimension: from the points of view of the user and the system. The user is concerned with the on time delivery of periodic and sporadic messages and the average response time of aperiodic messages.

- **Miss rate**: The fraction of all message instances or packets that are delivered to their destinations too late.
- **Loss rate:** Gives the fraction of all message instances in the stream that are dropped en route for flow and congestion control reasons.
- **Invalid rate:** combination of miss and loss rate (sum of miss and loss rate)
- **Delay Jitter:** The variation in the delays suffered by different message instances or packets in the stream.
- **Buffer Requirement:** A packet that arrives too early to be processed by the destination must be buffered. So, a larger delay jitter of a message stream means that more buffers must be provided by the stream.
- **Throughput:** The rate of each message stream measures the throughput of the stream.

6.1.2 <u>Service Discipline</u>:

The combination of an acceptance test and admission control protocol, and synchronization protocol and a scheduling algorithm used for the purpose of rate control, jitter control and scheduling of packets transmission is called a service discipline. Rate control and jitter control serve the purpose of flow control for real time. Service discipline are of two types:

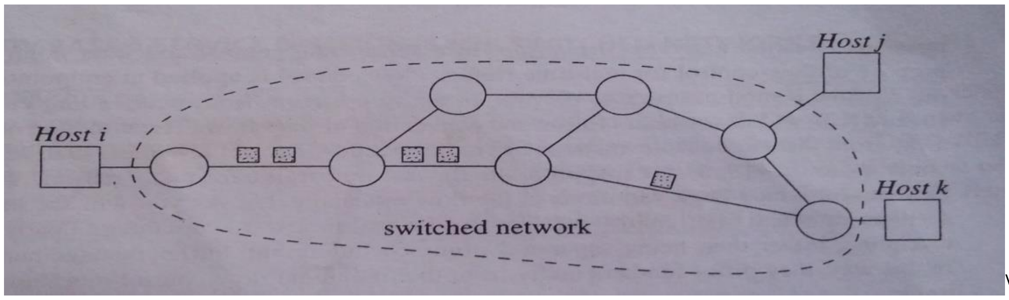
1. Rate allocating
2. Rate controlled

- **Rate allocating** discipline allows packets on each connection to be transmitted at higher rates than the guaranteed rate provided the switch can still meet the guarantees to all other connections.
- A service discipline is **rate controlled** if it ensures each connection the guaranteed rate but never allows packets on any connection to be sent above the guaranteed rate.

6.1.3 <u>Packet-Switched Networks</u> :

- They make several assumptions, which are valid for most switched, multipop networks. The diagram in figure 6-2(a) illustrates such a network. The circles in the diagram represent switches.
- Figure 6-2(b) shows a m x m switch; it has m input links and m output links, both called links 1, 2…m. the switch routes packets on its input links to its output links.

- We can represent every switching (i.e…, routing) pattern by a permutation of the m-tuple (1, 2…m). A number I at position K means that the switch is configured to route a packet coming on the input link I to (the queue of) the output link K at the same time when it is routing packets on the other input links to other output links as specified by the permutation.
- As an example, for a 4 x 4 switch, the 4-tuple (2,4,1,3) means that a packet on input link 2 goes to the queue of output link 1, a packet on input link 4 goes to the queue of output link 2, and so on. This is the switching pattern depicted by figure 6-2(b). The switch is nonblocking, meaning that every permutation represents a possible switching pattern.



\

Figure 6-2(a)

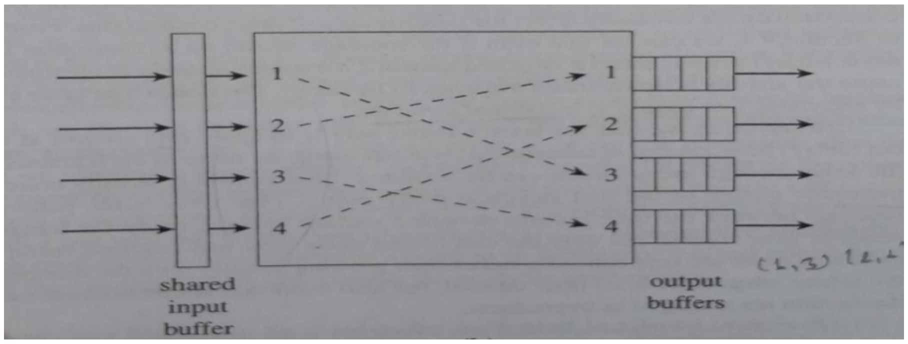


Figure 6-2(b)

- The end-to-end delay of each packet through a switched network is equal to the sum of the per hop delays it suffers passing through all the switches en route plus the total time it takes to propagate along all the links between the switches.

6.2 <u>Medium access-control protocols of broadcast networks:</u>

- 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 transmission medium is done distributed by network interfaces of hosts in the networks.
- Some MAC protocols are designed for distributed scheduling by loosely coupled schedulers, and they are fundamentally different from scheduling algorithms, based on distributed scheduling model.
- Two of them are timed-token MAC protocol and priority-based reservation scheme.
- **Timed-token MAC protocol** used in several network standards and survivable Adaptable Fiber Optic Embedded Networks.
- **Priority-based reservation scheme** was proposed for IEEE 806.6 Distributed Queue Dual Bus (DQDB) metropolitan area network standard.

6.2.1 <u>Medium-Access Control in DQDB Networks</u>: As shown in below figure Distributed-Queue Dual Bus (DQDB) network, the type of the network for which the IEEE 806.6 metropolitan area network standard [IEEE90a] was developed.

- As its name implies, a DQDB network has two unidirectional buses, called bus A and bus B.
- The slot generator at the head end of each bus generates 53 octet slots on the bus; a station places its outgoing data for stations downstream using these slots. Together, the buses provide the stations with full-duplex capability.
- There are two types of slots on the network: preallocated (PA) slots and Queue-Arbitrated (QA) slots. PA slots are for isochronous traffic. Allocation to bandwidth provided by PA slots is done centrally by the bandwidth manager and virtual circuit server (BMVS).
- QA slots are for asynchronous traffic. As an example, during connection establishment and tear down, stations and the BMVS communicate via QA slots. Access to these slots is scheduled distributed by individual stations and the BMVS.

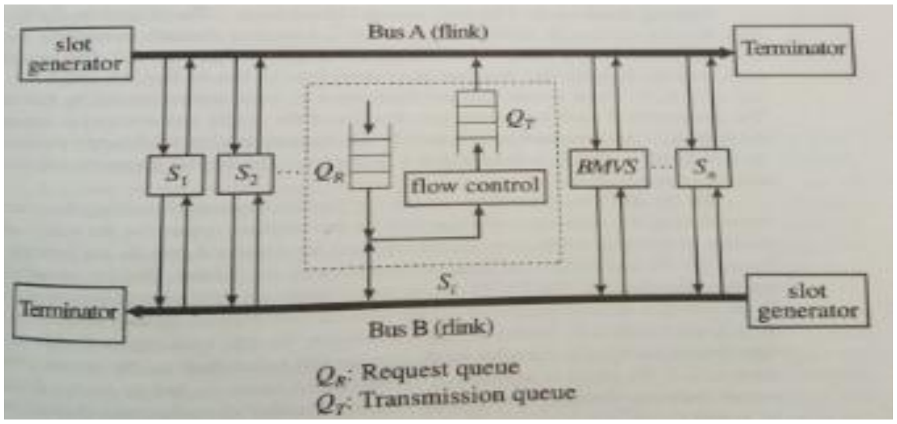


Figure: DQDB Network Architecture

Centralized Allocation of Isochronous Bandwidth: We now focus on how PA slots on one bus are allocated to isochronous connections using the bus.

- These slots are divide into groups; the slot generator generates a slot every 125 milliseconds periodically for each group. Each slot group is identified by its Virtual Circuit Identifier (VCI).
- The header of every PA slot contains the VCI of the group to which the slot belongs.

Connection Establishment: The basic unit of the bandwidth allocation is a 64-kbits/sec channel. The 64-kbits/sec bandwidth of a channel is provided by one octet in every slot of a PA slot group. Since the data field of each PA slot contains 48 octets, each PA slot group provides 48 isochronous channels. Each of these channels identified by a 2-tuple of group number and offset; the latter specifies the octet within the slot used by the channel. As examples, channels (33,7) and (177,42) are the seventh octet of the slots in group 33 and the forty-second octet of the slots in group 177, respectively.

- A connection consists of one or more isochronous channels. A source station requests connection by sending the BMVS a request containing the source and the destination IDs of the new connection and the requested number of channels.
- The BMVS accepts the request when there are sufficient free isochronous channels to meet the request; otherwise, it rejects the request. When it accepts a request, the BVMS first finds and allocates to the new connection the octets to be used by the connection.
- The BMVS then sends to the source station the VCI(s) and offset(s) which identify the corresponding isochronous channels(s) it has just allocated to the connection. The source station passes this information to the destination(s).
- Hereafter, the source sends data on the connection via the channel(s), and the destination copies the data carried on the channel(s). The channel(s) used by a connection is freed when the connection is torn down.

6.3 Operating System:

- A good real-time operating system not only provides efficient mechanisms and services to carry out good real-time scheduling and resource management policies, but also keeps its own time and resource consumptions predictable and accountable.
- More than general-purpose operating systems, a real-time operating system should be modular and extensible.
- A real-time operating system may have a microkernel that provides only essential services, such as scheduling, synchronization, and interrupt handling.

6.3.1 Threads and Tasks

- A thread implements a computation job and is the basic unit of work handled by the scheduler.
- Admission of a job (or a task) into the system after an acceptance test; this step encompasses the creation of a thread that implements the job.

- When the kernel creates a thread, it allocates memory space to the thread and brings the code to be executed by the thread into memory.
- In addition, it instantiates a data structure called the **Thread Control Block (TCB)** and uses the structure to keep all the information it will need to manage and schedule the thread.

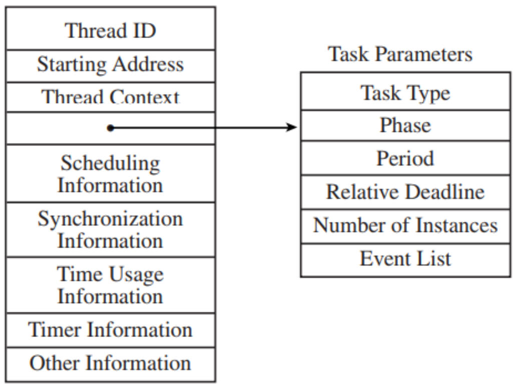


Fig: Thread Control Block

- The information kept in the TCB of a thread includes the ID of the thread and the starting address of thread's code.
- The context of a thread refers to the values of registers (e.g., program counter and status register) and other volatile data that define the state and environment of the thread.
- When a thread is executing, its context changes continuously.
- When the thread stops executing, the kernel keeps its context at the time in the thread's TCB.
- If the operating system inserts a thread in a queue (e.g., the ready or suspend queue), we mean that it inserts (a pointer to) the TCB of the thread into a linked list of TCBs of other threads in that queue. The kernel destroys a thread by deleting its TCB and deallocating its memory space.

Periodic Threads:

- A periodic (computation) task is a thread that executes periodically.
- It is clearly inefficient if the thread is created and destroyed repeatedly every period.
- In an operating system that supports periodic tasks the kernel reinitializes such a thread and puts it to sleep when the thread completes.
- The kernel keeps track of the passage of time and releases (i.e., moves to the ready queue) the thread again at the beginning of the next period. We call such a thread a periodic thread.
- The parameters of a periodic thread include its phase (i.e., the interval between its creation and its first release time), period, relative deadline, and the number of instances.
- A periodic thread with a finite number of instances terminates and may be destroyed by the kernel after it has executed for the specified number of times.

- These parameters are given by the application when its requests the creation of the periodic thread. They are kept in the TCB of the thread.
- Most commercial operating systems do not support periodic threads. We can implement a periodic task at the user level as a thread that alternately executes the code of the task and sleeps until the beginning of the next period.

Aperiodic, Sporadic, and Server Threads:

- We can implement a sporadic or aperiodic task as a sporadic thread or aperiodic thread that is released in response to the occurrence of the specified types of events.
- The events that cause the releases of these threads occur sporadically and may be triggered by external interrupts.Upon its completion, a sporadic thread or aperiodic thread is also reinitialized and suspended.
- We call a thread that implements a bandwidth-preserving server or a slack stealer a server thread.
- Server queue is such a queue, is simply a list of pointers which give the starting addresses of functions to be executed by the server thread.
- Each aperiodic (or sporadic) job is the execution of one of these functions. Upon the occurrence of an event that triggers the release of an aperiodic job, the event handler inserts into this list a pointer to the corresponding function.
- Thus, the aperiodic job is “released” and queued. When the server is scheduled, it executes these functions in turn.

Major States: There are five major states of a thread.

- ➢ *Sleeping*:
  - ▪ A periodic, aperiodic, or server thread is put in the *sleeping* state immediately after it is created and initialized.
  - ▪ It is released and leaves the state upon the occurrence of an event of the specified types. Upon the completion of a thread that is to execute again, it is reinitialized and put in the sleeping state.
  - ▪ A thread in this state is not eligible for execution.
- ➢ *Ready*: A thread enters the ready state after it is released or when it is preempted. A thread in this state is in the ready queue and eligible for execution.
- ➢ *Executing*: A thread is the *executing* state when it executes.
- ➢ *Suspended (or Blocked):* A thread that has been released and is yet to complete enters the suspended (or blocked) state when its execution cannot proceed for some reason. The kernel puts a suspended thread in the suspended queue.
- ➢ *Terminated:* A thread that will not execute again enters the terminated state when it completes. A terminated thread may be destroyed.

- ➢ A job can be suspended or blocked for many reasons.
  - ▪ For example, it may be blocked due to resource-access control;
  - ▪ It may be waiting to synchronize its execution with some other thread(s);

    - It may be held waiting for some reason (e.g., I/O completion and jitter control), and so on.

- A bandwidth preserving server thread enters the suspended state when it has no budget or no aperiodic job to execute. The operating system typically keeps separate queues for threads suspended or blocked for different reasons (e.g., a queue for threads waiting for each resource).
- For the sake of simplicity, we call them collectively the suspended queue.
- Similarly, the kernel usually keeps a number of ready queues.

6.3.2 <u>The Kernel</u>

- A real-time operating system consists of a microkernel that provides the basic operating system functions.
- There are three reasons for the kernel to take control from the executing thread and execute itself:
    - to respond to a system call
    - do scheduling and service timers
    - handle external interrupts
- The kernel also deals with recovery from hardware and software exceptions.

<u>System Calls</u>

- The kernel provides many functions which, when called, do some work on behalf of the calling thread.
- An application can access kernel data and code via these functions. They are called Application Program Interface (API) functions.

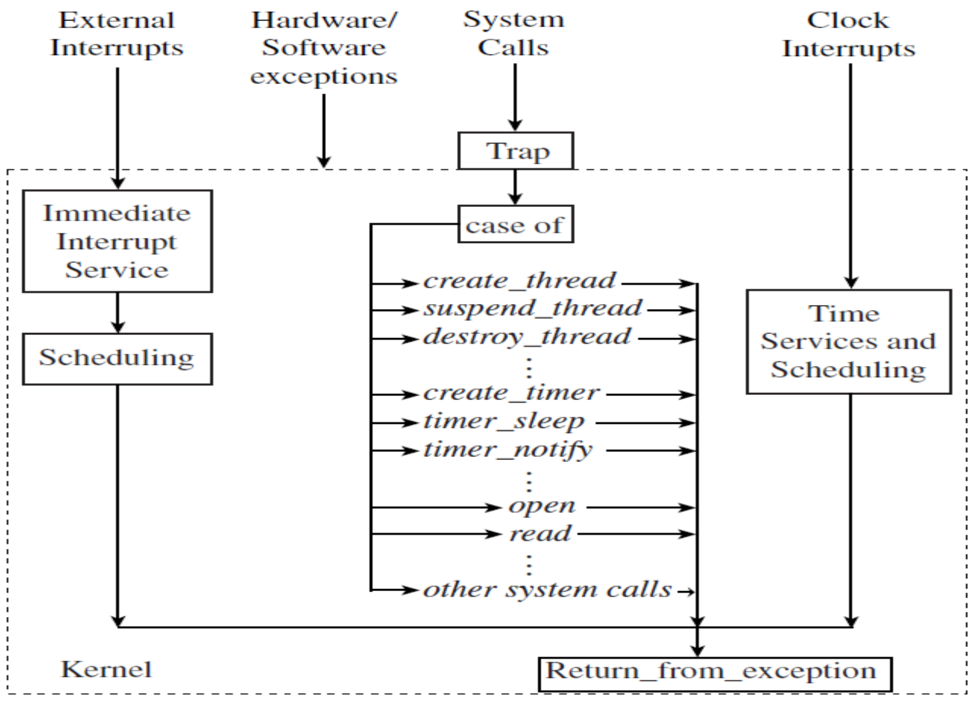


FIGURE 12–2 Structure of a microkernel.

- A system call is a call to one of the API functions. In a system that provides memoryprotection, user and kernel threads execute in separate memory spaces.
- Upon receiving a system call, the kernel saves the context of the calling thread and switches from the user mode to the kernel mode.
- It then picks up the function name and arguments of the call from the thread's stack and executes the function on behalf of the thread. When the system call completes, the kernel executes a return from exception.
- As a result, the system returns to the user mode. The calling thread resumes if it still has the highest priority. If the system call causes some other thread to have the highest priority, then that thread executes.
- **Synchronous system call:** when a thread makes a synchronous system call. The calling thread is blocked until the kernel completes the called function.
- **Asynchronous system call**: When the call is asynchronous (e.g., in the case of an asynchronous I/O request), the calling thread continues to execute after making the call. The kernel provides a separate thread to execute the called function.
- **Thread management functions:**
    - create thread
    - suspend thread
    - resume thread
    - destroy thread.
- A (software) timer is an object used to keep track of time. In addition to system wide timers, most operating systems allow threads (or processes) to have their own timers.
- A per thread timer is created by the kernel on behalf of a thread when the thread calls the create timer function. The timer will keep track of time according to that clock.
- A clock is a hardware device that contains a counter. At any time, the content of the counter gives a representation of the current time.
- A set-timer function call specifies the ID of the timer to be set and an expiration time.
    - ➢ By calling this function, a thread asks the kernel to carry out an action at the timer expiration time.
    - ➢ The action may be the execution of a specified function, or the waking up of a suspended thread, or the placement of a message in a message queue, and so on.
    - ➢ We say that a timer event occurs at the timer's expiration time; when a timer event occurs, the kernel carries out the specified action.

Memory Management

- A task is not admitted into the system if there is not enough memory to meet its peak memory space demand.
- Aspects of memory management: virtual memory mapping, paging, and memory protection.
- All general-purpose operating systems support virtual memory and memory protection, not all real-time operating systems do, and those that do typically provide the user with the choice of protection or no protection.

Virtual Memory Mapping:

- We can divide real-time operating systems into three categories depending on whether they support virtual memory mapping (i.e., virtual contiguity) and paging (i.e., demand paging or swapping).
- Real-time operating systems designed primarily for embedded real-time applications such as data acquisition, signal processing, and monitoring, may not support virtual memory mapping.
- Upon request, the system creates physically contiguous blocks of memory for the application.
- The application may request variable size segments from its memory block and define a memory partition consisting of physically contiguous, fixed-size buffers.
- Memory fragmentation is a potential problem for a system that does not support virtual mapping.
- After allocating variable-size segments, large fractions of individual blocks may be unused.
- The available space may not be contiguous and contiguous areas in memory may not be big enough to meet the application's buffer space demand.
- The solution is to provide virtual memory mapping from physical addresses, which may not be contiguous, to a contiguous, linear virtual address space seen by the application.
- The penalty of virtual address mapping is the address translation table, which must be maintained and hence contribute to the size of the operating system.

Memory Locking:

- A real-time operating system may support paging so that non real-time, memory demanding applications (e.g., editors, debuggers and performance profilers) needed during development can run together with target real-time applications.
- Such an operating system must provide applications with some means to control paging.
- All operating systems, including general-purpose ones, offer some control, with different granularities.
- An examples, Real-Time POSIX-compliant systems allow an application to pin down in memory all of its pages.
- In some operating systems (e.g., Windows NT), the user may specify in the create thread system call that all pages belonging to the new thread are to be pinned down in memory.
- The LynxOS operating system controls paging according to the demand-paging priority.
- Memory pages of applications whose priorities are equal to or higher than this priority are pinned down in memory while memory pages of applications whose priorities are lower than the demand-paging priority may be paged out.

Memory Protection:

- Many real-time operating systems do not provide protected address spaces to the operating system kernel and user processes.
- Argument for having only a single address space includes simplicity and the light weight of system calls and interrupts handling. For small embedded applications, the overhead space of a few kilobytes per process is more serious.

- Critics points out a change in any module may require retesting the entire system. This can significantly increase the cost of developing all but the simplest embedded systems.
- For this reason, many real-time operating systems support memory protection.
- A good alternative is to provide the application with the choices in memory management. such as the choices in virtual memory configuration offered by VxWorks and QNX.
- In VxWorks, we can choose to have only virtual address mapping, to have text segments and exception vector tables write protected, and to give each task a private virtual memory when the task requests for it.

I/O and Networking: Three modern features of file system and networking software are

(1) Multithreaded-server architecture
(2) Early demultiplexing
(3) Lightweight protocol stack.
These features were developed to improve the performance of time-shared applications, high-performance applications and network appliances.

Multithreaded Server Architecture:

- Servers in modern operating systems are typically multithreaded.
- In response to a request, such a server activates (or creates) a work thread to service the request.
- By properly prioritizing the work thread, we can minimize the duration of priority inversion and better account for the CPU time consumed by the server while serving the client. As a example, we consider a client thread that does a read.
- Suppose that the file server's request queue is prioritized, the request message has the same priority as the client, and the work thread for each request inherits the priority of the request message.
- As a consequence, the length of time the client may be blocked is at most equal to the time the server takes to activate a work thread. I/O requests are sent to the disk controller in priority order.
- For the purpose of schedulability analysis, we can model the client thread as an end-to-end job consisting of a CPU sub job, which is followed by a disk access sub job, and the disk-access job executes on the disk system and is in turn followed by a CPU sub job.
- Both CPU sub jobs have the same priority as the client thread, and their execution times include the lengths of time the work thread executes.
- Since the time taken by the server to activate a work thread is small, the possible blocking suffered by the client is small.
- In contrast, if the server were single-threaded, a client might be blocked for the entire duration when the server executes on behalf of another client, and the blocking time can be orders of magnitude larger.

Early Demultiplexing:

- Traditional protocol handlers are based on layered architectures. They can introduce a large blocking time.

- For example, when packets arrive over a TCP/IP connection, the protocol module in the network layer acknowledges the receipts of the packets, strips away their headers, reassembles them into IP data grams, and hands off the datagram to the IP module after each datagram is reassembled.
- Similarly, the TCP and IP modules reassemble IP data grams into messages, put the messages in order and then deliver the messages to the receiver.
- Because much of this work is done before the identity of the receiver becomes known, it cannot be correctly prioritized.
- Typically, the protocol modules execute at a higher priority than all user threads and block high-priority threads when they process packets of low-priority clients.
- The duration of priority inversion can be reduced and controlled only by identifying the receiver of each message as soon as possible.
- Once the receiver is known, the execution of the protocol modules can be at the priority of the receiver.Traditionally, incoming messages are first moved to the buffer space of the protocol modules.
- They are then copied to the address space of the receiver. Early demultiplexing also makes it possible to eliminate the extra copying.Some communication mechanisms for high-speed local networks take this approach.
- The fact that messages are copied directly between the network interface card and the application is a major reason that these mechanisms can achieve an end-to-end (i.e., application-to-application) latency in the order of 10 to 20 microseconds.

Lightweight Protocol Stack:
Protocol processing can introduce large overhead and long latency. This is especially true when the protocol stack has the client/server structure:

  - Each higher-level protocol module uses the services provided by a protocol module at a layer below.
  - This overhead can be reduced by combining the protocol modules of different layers into a single module and optimizing the module whenever possible.
  - A challenge is how to provide lightweight, low overhead protocol processing and still retain the advantages of the layer architecture.

An example of operating systems designed to meet this challenge is the Scout operating system. Scout is based on the Path abstraction.
Specifically, Scout uses paths as a means to speed up the data delivery between the network and applications.
At configuration time, a path consisting multiple stages is created one stage at a time starting from one end of the path.
A stage is created by invocating the function *pathCreate( )* on the router specified by one of the arguments of the function.
As the result of the invocation of *pathCreate( )* and the subsequent invocation of *createStage* function, the router creates a stage of the path and identifies the next router, if any, on the path.
Similarly, the next stage is created by the next router, and if the next router is to be followed by yet another router, that router identified. This process is repeated until the entire sequence of stages is created.

The stages are then combined into a single path object and initialized. Whenever possible, Scout applies path transformation rules to optimize the path object.
At run time, each path is executed by a thread. Scout allows these threads to be scheduled according to multiple arbitrary scheduling policies and allocates a fraction of CPU time to threads scheduled according to each policy.

Open system architecture:

Objectives: A real-time operating system should create an open environment.

1. Independent Design Choice:

- ❖ The developer of a real-time application that is to run in an open environment can use a scheduling discipline best suited to the application to schedule the threads in the application and control their contention for resources used only by the application.

2. Independent Validation:

- ❖ To determine the schedulability of an application, its developer can assume that the application runs alone on a virtual processor that is the same as the target processor but has a speed that is only a fraction $0 < s < 1$ of the speed of the target processor.
- ❖ In other words, if the maximum execution time of a thread is e on the target processor, then the execution time used in the schedulability analysis is e/s.
- ❖ The minimum speed at which the application is schedulable is the required capacity of the application.

3. Admission and Timing Guarantee:

- ❖ The open system always admits non real-time applications.
- ❖ Each real-time application starts in the non real-time mode.
- ❖ After initialization, a real-time application requests to execute in the real-time mode by sending an admission request to the operating system.
- ❖ In this request, the application informs the operating system of its required capacity, plus a few overall timing parameters of the application.
- ❖ The operating system subjects the requesting application to a simple but accurate acceptance test.
- ❖ If the application passes the test, the operating system switches the application to run in real-time mode.
- ❖ Once in real-time mode, the operating system guarantees the schedulability of the application, regardless of the behavior of the other applications in the system.