CONTENTS

NECS 082 : REAL TIME SYSTEM

1

UNIT

Introduction

PART-1

Introduction : Definition, Typical Real Time Applications : Digital Control, High Level Controls, Signal Processing etc., Release Times, Deadlines, and Timing Constraints.

CONCEPT OUTLINE : PART-1

- A system is called a real time system, when we need quantitative expression of time (*i.e.*, real time) to describe the behaviour of the system.
- **Applications of real time systems :** Real time systems found applications in wide ranging areas such as :
  1. Industrial applications
  2. Defense applications
  3. Medical applications
  4. Automotive and transportation applications
  5. Internet and multimedia applications
- **Release time :** The release time of a job is the instant of time at which the job becomes available for execution.
- **Deadline :** The deadline of a job is the instant of time by which its execution is required to be completed.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. What are real time systems ?

OR

What are the characteristics of 'Real Time System' ? Explain its multimedia applications. **UPTU 2012-13, Marks 05**

Answer

1. A real time system is one that must process information and produce a response within a specified time else risk severe consequences, including failure.
2. That is, in a system with a real time constraint it is not good to have the correct action or the correct answer after a certain deadline.

3. Real time system can also be defined as any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.
4. For example, ABS, aircraft control, ticket reservation system, over temperature monitor in nuclear power station, mobile phone, over temperature controller, Doppler blood flow monitor, ECG/arrhythmia monitor.

Characteristics of real time system :

1. **Time constraints :**
   a. Every real time task is associated with some time constraints. One very common form of time constraint is deadlines associated with tasks.
   b. A task deadline specifies the time before which the task must complete and produce the result. Other types of timing constraints are delay and duration.
2. **New correctness criterion :**
   a. In real time system, correctness implies not only logical correctness of the result but the time at which the results are produced is also important.
   b. A logically correct result produced after the deadline would be considered an incorrect result.
3. **Embedded :** A vast majority of real time system are embedded in nature. An embedded computer system is physically embedded in its environment and often controls it.
4. **Safety criticality :**
   a. In many real time systems, safety and reliability are intricately bonded together making them safety critical. A safe system is one that does not cause any damage even when it fails.
   b. A reliable system is one that can operate for long durations of time without exhibiting any failures.
   c. A safety critical system is required to be highly reliable since any failure of the system can cause extensive damages.
5. **Concurrency :**
   a. A real time system usually needs to respond to several independent events within very short and strict time bounds.
   b. For example, a chemical plant automation system, which monitors the progress of a chemical reaction and controls the rate of reaction by changing the different parameters of reaction such as pressure, temperature and chemical concentration.
6. **Distributed and feedback structure :**
   a. In real time systems, the different components of the system are naturally distributed across widely spread geographic locations.

b. In these systems, the different events of interest are done at the geographically separate locations.

c. Therefore, these events may often have to be handled locally and responses produced to them to prevent overloading of the underlying communication network.

7. **Task criticality :**

a. Task criticality is a measure of the cost of failure of a task. Task criticality is determined by examining how critical are the results produced by the task to the proper functioning of the system.

b. A real time system may have task of very different criticalities.

c. It is therefore, natural to expect that the criticalities of the different tasks must be taken into consideration while designing for fault tolerance.

8. **Custom hardware :**

a. A real time system is implemented on custom hardware that is specifically designed and developed for the particular purpose.

b. For example, cell phones use processors which are tiny, supporting only those processing capabilities that are really necessary for cell phone operation and specifically designed to be power efficient to conserve battery life.

c. The capabilities of the processor used in a cell phone are substantially different from that of a general purpose processor.

9. **Reactive :** A reactive system is one in which an ongoing interaction between the computer and environment is maintained.

10. **Stability :**

a. Under overload conditions, real time systems need to continue to meet the deadline of the most critical tasks, though the deadline of non critical tasks may not be met.

b. This is in contrast to the requirement of fairness for traditional system even under overload conditions.

11. **Exception handling :**

a. Many real time systems work round-the-clock and often operate without human operators. For example, consider a small automated chemical plant that is set up to work nonstop.

b. When there are no human operators, taking corrective actions on a failure becomes difficult.

c. Even if no corrective action can be immediately taken, it is desirable that a failure does not result in catastrophic situations.

Multimedia applications :

1. One of the most frequently encountered real time applications : multimedia.

2. A multimedia application may process, store, transmit, and display any number of video streams, audio streams, images, graphics, and text.
3. A video stream is a sequence of data frames which encodes a video. An audio stream encodes a voice, sound, or music.
4. Without compression, the storage space and transmission bandwidth required by a video are enormous. Therefore, a video stream, as well as the associated audio stream, is invariably compressed as soon as it is captured.
5. **MPEG Compression/ Decompression :**
   a. A video compression standard is MPEG-2. The standard makes use of three techniques.
   b. They are motion compensation for reducing temporal redundancy, discrete cosine transform for reducing spatial redundancy, and entropy encoding for reducing the number of bits required to encode all the information.
6. **Decompression :**
   a. During decompression, the decoder first produces a close approximation of the original matrix (*i.e.*, an 8 × 8 pixel block) by performing an inverse transform on each stored transform matrix.
   b. It then reconstructs the images in all the frames from the major blocks in I-frames and difference blocks in P-and B-frames.

Que 1.2. **Explain various issues related to real time system.**

OR

Explain architecture and operating system issues in context of real time system.

Answer

A real time system must be much more reliable than its individual hardware and software components. It must be capable of working in harsh environments, rich in electromagnetic noise and elementary particle radiation and in the face of rapidly changing computing loads. Issues in real time systems are :

Architecture issues :

1. **Processor architecture :** For reason of economy, generally off-the-shelf processors are preferred. Real time system designer design their own processor. We need to estimate execution time of task and it is important to know how long a particular architecture of processor takes to execute this task.
2. **Network architecture :** To make system reliable which provide sufficient processing capacity, most real time systems are multiple processor machine which are connected in a network.

3. **Architecture for clock synchronization :** In order to facilitate the interaction between the multiple units of a real time system, the clocks of these units must be tightly synchronized.
4. **Fault tolerance and reliability evaluation :** There can be a major catastrophe when real time system fails, such systems must therefore be highly fault tolerant.

Operating system issues :

1. **Task assignment and scheduling :** The scheduling of tasks ensures that real time deadlines are met which is the most important aspect of a real time operating system.
2. **Communication protocols :** It is important to have interprocessor communication among the tasks so that the controller can do the specific action at right time. There should be a communication protocol so that there is less delay in performing task and the system is controllable.
3. **Failure management and recovery :** When a processor or software module fails, the system must limit such failure and recover from it.
4. **Clock synchronization algorithm :** Hardware synchronization architecture are built out of phase locked clocks. There are also software implementations of fault tolerant clock synchronization. In this, we have a hardware clock and a software based correction. The clock time is the sum of the hardware time and the correction. A new correction is calculated at regular resynchronization intervals.

Other issues :

1. **Programming languages :**
   a. Real time engineers need much greater control over timing and need to interface to special purpose device.
   b. In real time system there are additional requirements in the programming language.
   c. They require that deadlines must be met and they must specify absolute time intervals and enforce timing constraints.
2. **Databases :**
   a. There are many real time database applications, such as the stock market, airline reservations and artificial intelligence.
   b. So, these databases should be updated at right time whenever there is any change in the data.
3. **Performance measures :**
   a. Commonly used performance measures such as conventional reliability and throughput are useless for real time system.
   b. Performance measures like performability and hard deadlines are used in real time system.
   c. Performability is defined as the probability that the system will allow each accomplishment level to be met.

Que 1.3. **State and explain the issues involved in real time computing.**

UPTU 2013-14, Marks 05

Answer

Issues in real time computing :

1. A real time computer must be much more reliable than its individual hardware and software components.
2. It must be capable of working in harsh environments, rich in electromagnetic noise and elementary-particle radiation, and in the face of rapidly changing computation loads.
3. The field of real time computing is especially rich in research problems because all problems in computer architecture, fault tolerant computing, and operating systems are also problems in real time computing, with the added complexity that real time constraints must be met.
4. For example, take task scheduling.
5. The purpose of task scheduling in a general purpose system is fairness, by which we mean that the computer's resources must be shared out equitably among the users.
6. This end is usually achieved by using round-robin scheduling.
7. Each process is associated with a time-slice.
8. The computer executes a process until one of the following happens : it is done, it has to stop for something like a disk access or interrupt, or its allotted time-slice has expired.
9. The computer then switches context to another process.
10. There may be variations on this basic theme. For example, the time-slice could be varied according to how much execution time has already been spent on that task.
11. Round-robin scheduling ensures that one user does not get a disproportionate share of the computer's services. Such an approach does not work in a real time application.

Que 1.4. **Explain the issues involved in designing caches for real time systems.**

UPTU 2013-14, Marks 05

Answer

1. Caches are common place in computers, they reduce the effective memory-access time and increase the average throughput. There are two ways to allocate cache space.
2. One is to allow the process currently executing the right to use the entire cache area.

3. This keeps the miss rate (effective memory-access time) low and is the approach adopted by general purpose systems.

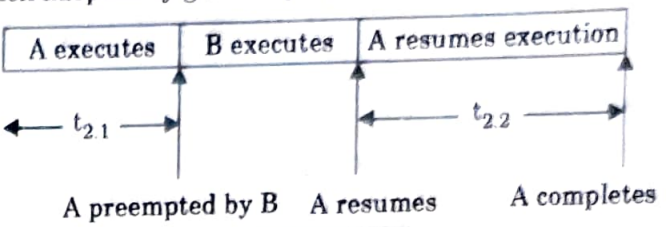


Fig. 1.4.1.

4. However, from a real time engineer's standpoint, this approach has the unpleasant side effect of working task runtime less predictable.

Example :

a. Suppose process A takes t_1 seconds to execute when it is not interrupted by any other process.

b. If A is preempted by another process B and then takes up from where it left off when B gets done.

c. It is not necessary that $t_{2.1} + t_{2.2} = t_1$ because when B was executing it might have displaced some of A's lines from the cache to make room for its own working set.

d. When A resumes and accesses on these displaced lines, a cache miss results and access time is greater than it would have been if B had not come along and disrupted things. As a result, $t_{2.1} + t_{2.2}$ is very likely to be greater than t_1.

e. The execution time of A will depend on the number of times A is preempted and on what happens to the cache upon each preemption.

f. It is important that the task execution time be predictable to allow the designer to figure out if all critical tasks will meet their deadlines.

g. If the effect of the cache is to make task runtimes very unpredictable, it may actually do more harm than good from the designer's point of view.

h. A real time system is typically designed for a specific application.

i. Since the consequences of failure are more severe in real time systems than in their general purpose counter-parts, such systems need to be specified more carefully and their performance measures that are capable of expressing timing requirements.

j. We require, among other things, means to predict the execution times of programs, to model the reliability of software and hardware, to assign tasks to processors and schedule them so that deadlines are met and to develop mechanisms by which the system can quickly recover from the failure of an individual component.

Que 1.5. **Explain structure of real time system.**

OR

Describe the architecture of real time system.

OR

Explain various components of a typical real time system with block diagram. **UPTU 2014-15, Marks 05**

Answer

Structure of a real time system :

Fig. 1.5.1 shows a schematic block diagram of real time system in control of some process.

1. **Controlled process :**
   a. The state of the controlled process and of the operating environment (for example, pressure, temperature, speed, and altitude) is acquired by sensors, which provide inputs to the controller, the real time computer.
2. **Sensor :**
   a. A sensor converts some physical characteristics of its environment into electrical signals.
   b. The data rate from each sensor depends on how quickly the measured parameters can change, it is usually less than 1 kilobyte/second (kb/s.)
3. **Job list :**
   a. There is a fixed set of application tasks or jobs, the "job list" is shown in Fig. 1.5.1.
   b. The software for these tasks is preloaded into the computer.
   c. If the computer has a shared main memory, then the entire software is loaded into that.
   d. If, on the other hand, it consists of a set of private memories belonging to individual processors, so as to which memories each job should be loaded into.

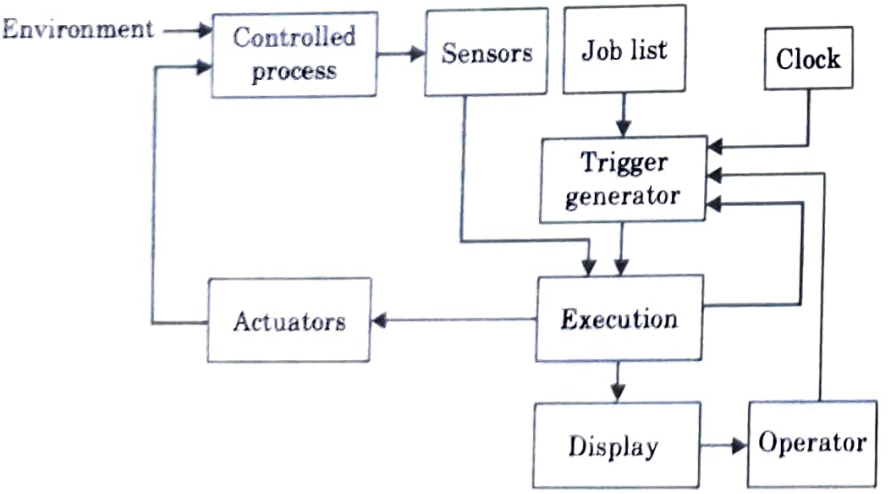


Fig. 1.5.1. A schematic block diagram of a real time system.

4. **Trigger generator :**
   a. The "trigger generator" is a representation of the mechanism used to trigger the execution of individual jobs.
   b. It is not really a separate hardware unit; typically it is part of the executive software.
   c. Many of the jobs are periodic (*i.e.*, they execute regularly).
5. **Execution :**
   a. The schedule for these jobs can be obtained offline and loaded as a lookup table to be used by the scheduler.
   b. Jobs can also be initiated depending on the state of the controlled process or on the operating environment.
   c. For example, when the pressure of a boiler is greater than the preset threshold in a chemical plant or the altitude is less than the threshold in an aircraft, it may be necessary to run some task every x ms.
   d. Finally, jobs can be initiated on command from the operator input panel.
6. **Actuator :**
   a. An actuator is any device that takes its inputs from the output interface of a computer and converts these electrical signals into some physical actions on its environment.
   b. The output of the computer is fed to the actuators.
   c. Fault tolerant techniques ensure that, despite a small number of erroneous outputs from the computer, the actuators are set correctly.
   d. The actuators typically have a mechanical or a hydraulic component, and so their time constants are quite high.
   e. As a result, the data rates to the actuators are quite low; one command per 25 ms, on average, is not uncommon.
7. **Display :**
   a. The output of the computer is fed to the displays.

Que 1.6. **What is an embedded real time system ? Enumerate its types.**

OR

Write the features and characteristics of embedded systems. Also, mention its applications.

Answer

1. An embedded system can be defined as one which has computer hardware with software embedded in it as one of its most important component.

2. An embedded system is a combination of computer hardware and software, either fixed in capability or programmable *i.e.*, specifically designed for a particular kind of application device.
3. Embedded systems are programmable which are provided with a programming interface.
4. Typically an embedded system consists of a microcomputer with software in ROM/FLASH memory, which starts running a dedicated application as soon as power is turned ON and does not stop until power is turned OFF.
5. A general purpose definition of embedded systems is that they are devices used to control, monitor or assist the operation of equipment machinery or plant.
6. Embedded reflects the fact that they are an integral part of the system that includes hardware and mechanical parts.

Characteristics of embedded system :

1. Dedicated functions or tasks.
2. Real time response to task.
3. Generally not reprogrammable by the end user.
4. Complex algorithms.
5. Complex Graphic User Interface (GUI).

Features of embedded system :

a. It consists of hardware and embedded software which is capable of performing any specific task.
b. It may or may not contain an operating system for functioning.
c. The software is pre-programmed and it is non-alterable by the end user.
d. For certain category of systems like mission critical system, response time requirement is highly critical.

Classification of embedded system :

1. **Small scale embedded system :**
   a. These systems are designed with a single 8 or 16 bit microcontroller.
   b. They have little hardware and software complexities.
   c. The main tools are : editor, assembler, Integrated Development Environment (IDE).
   d. 'C' language is used for programming.
   e. The software has to fit within the available memory.
2. **Medium scale embedded system :**
   a. These systems are designed with a single or a few 16 or 32 bit microcontrollers.
   b. They have both software and hardware complexities.

c. The main tools are : RTOS, simulator, debugger and an Integrated Development Environment (IDE).

d. C, C++, Visual C++ or Java language can be used for programming.

3. **Sophisticated embedded system :**

a. These systems may need scalable processors or configurable processors and programmable logic arrays.

b. They have enormous hardware and software complexities.

c. They are constrained by the processing speeds available in their hardware.

d. Certain functions used to obtain additional speed are : encryption and decryption algorithm, discrete cosine transformation algorithm, TCP/IP protocol stacking and network driver functions.

e. Tools for these systems are not readily available at reasonable cost.

Que 1.7. **Elaborate the misconception that real time computing is fast computing.**

Answer

1. A real time system is one which must produce a response within a specified time.
2. If the process is not done before or within the deadline then the result is not considered to be useful.
3. So for these types of systems to be successful we should have a prior knowledge of the parameters of a particular process.
4. For example, if we have fixed release times and known resource request times then the system can compute accurately the occurrence times of future events.
5. The tasks with fixed release times and known request times are said to be predictable.
6. All the tasks are triggered by clock interrupts which occur at predicted known time instants.
7. Predictability is often achieved by either static or dynamic scheduling of real time tasks to meet their deadlines.
8. Static scheduling makes scheduling decisions at compile time and is off-line.
9. Dynamic scheduling is online and uses schedulability test to determine whether a set of tasks can meet their deadlines.
10. Hence, if we have the predicted parameters of the particular task in real time system then the task can be completed in the specified time accurately with minimum response time.

Que 1.8. **What are the traditional performance measures used for real time systems ?** **UPTU 2014-15, Marks 05**

OR

Discuss that traditional performance measures are not suitable for real time systems.

Answer

1. Reliability, availability, and throughput, together with related measures, are the traditional performance measure.
2. These measures are widely used for general-purpose systems :
   a. Reliability is the probability that the system will not undergo failure over any part of a prescribed interval.
   b. Availability is the fraction of time for which the system is up.
   c. Throughput is the average number of instructions per unit time that the system can process.
3. These measures require us to define what is meant by the system being up or down.
4. For simple systems, this is obvious.
5. When the system is gracefully degradable, however, we have to define a set of failed system states.
6. The system will be up whenever it is outside this set of failure states.
7. The set of failure states depends on the application, since different applications require different capabilities on the part of the computer.
8. The probability of the system being outside the failure states throughout some interval of time is the capacity reliability over that period.

Drawback of traditional performance measures :

1. The drawback of these measures is that they do not provide a mechanism to study the interplay among the hardware, the system software, and the applications software.
2. Reliability typically focuses on either the hardware or the software, independently of each other.
3. Availability is practically useless as a measure of the ability to meet execution deadlines; the long-term fraction of time for which the system is up tells us nothing about the length of the individual down times.
4. Throughput is an average measure; it conveys the raw computing capacity of the computer, but tells us nothing about the response time of the various control jobs.
5. These traditional measures work well so long as the set of failed states can be unambiguously defined in terms of just the hardware.

6. For complex systems, this set depends not only on the state of the hardware but also on the current allocation of tasks to the processors, and the way in which these are scheduled.
7. There is no clean way to incorporate these effects in traditional measures.
8. For these reasons, traditional reliability, availability, and throughput are not suitable performance measures for real time computers.

Que 1.9. What system considerations are required in designing real time systems ? **UPTU 2014-15, Marks 05**

Answer

Following are the various system considerations required in designing real time systems :

1. **Memory requirements :**
   a. During initial prototyping, we should plan on more memory on the target than during the final stages.
   b. This is because we will often be running debugging versions of software, which may be larger.
   c. Also, we will want to include diagnostics and utility programs, which again will consume more memory than expected.
   d. Once our prototype system is up and running, we can then start thinking about how much memory we "really" need.
2. **Peripherals :**
   a. This includes such items as disk controllers, network cards, PC-Card controllers, flash memory chips and graphics controllers.
   b. Graphics controllers are one of the particularly delicate areas in the design of an embedded system, often because a chip may be very new when it is selected and we may not yet have a driver for it.
3. **Debugging :**
   a. In many cases, especially in cost-sensitive designs, we would not want to provide any additional functionality beyond that absolutely required for the project at hand.
   b. But since the project is usually a brand new design, we will need to ensure that the hardware actually works per se and then actually works with the software.
   c. We recommend that we install some form of easy-to-get-at hardware debugging port, so that the software can output diagnostics as it is booting. This debug port can be left off for final assembly or a slightly modified "final" version of the board can be created.

d. The cost savings in terms of software development time generally pay for the hardware modifications many times over.

4. **Processor speed :**
   a. Although Neutrino is a real time operating system, this fact alone does not necessarily mean that any given application will run quickly.
   b. Graphical user interface applications can consume a reasonable amount of CPU and are particularly sensitive to the end-user's perception of speed.

5. **Other design considerations :** There are other design considerations that relate to both the hardware and software development process. They are :
   a. Hard real time system design :
      i. Provision of asynchronous IOs.
      ii. Provision of locks or spin locks.
      iii. Predictions of interrupt latencies and context switching latencies of the tasks.
      iv. Predictability is achieved by writing all functions which execute always take the same predefined time intervals in case of varying rates of occurrences of the events.
   b. Soft real time system design :
      i. One in which deadlines are mostly met.
      ii. Soft real time means that only the precedence and sequence for the task operations are defined, interrupt latencies and context switching latencies are small but there can be few deviations between expected latencies of the tasks and observed time constraints and a few deadline misses are accepted.
      iii. The preemption period for the soft real time task in worst case may be about a few ms.
      iv. Mobile phone, digital cameras and orchestra playing robots are examples of soft real time systems.

Que 1.10. **Explain about feasibility, optimality and performance measures of schedules.**

OR

Define and explain performability. **UPTU 2013-14, Marks 05**

Answer

1. A valid schedule is a feasible schedule if every job completes within its deadline.
2. A set of jobs is schedulable according to a scheduling algorithm if when using the algorithm the schedules always produces a feasible schedule.

3. The criterion to measure the performance of scheduling algorithms for hard real time applications is their ability to find feasible schedules of the given application system whenever such schedules exist.
4. Hence, a hard real time scheduling algorithm is optimal if the algorithm always produces a feasible schedule if the given set of jobs has feasible schedules.
5. Conversely if an optimal algorithm fails to find a feasible schedule, then the given set of jobs cannot feasibly be scheduled by any algorithm.
6. Other commonly used performance measures are the maximum and average tardiness, lateness and response time and the miss, loss and invalid rates.
7. The right choice of performance measure depends on the objective of scheduling.
8. For example, when a set of jobs is not schedulable by any algorithm, then a schedule can be produced according to which the number of jobs failing to complete in time is the smallest.
9. Hence, an algorithm performs better if it can produce a schedule with a smaller number of late jobs than others.
10. Alternatively, it is not considered that how many jobs are late but their tardiness is kept small. In this case, the algorithms that give small, maximum or average tardiness is used.
11. The lateness of a job is the difference between its completion time and its deadline.
12. Unlike the tardiness of a job which never has negative values, the lateness of a job which completes early is negative, while the lateness of a job which completes late is positive.
13. Sometimes, jitters in the completion times are kept small; it is done by using scheduling algorithms that try to minimize the average absolute lateness of job.

Performability :

1. Performability improves upon the traditional measures by explicitly and formally accounting for the fact that the performance of a real time computer should be tied to the consequent performance of the process that it controls.
2. The controlled process is defined as having several accomplishment levels which are different levels of performance as seen by the user. Each such level of performance is associated with the execution of a certain set of control tasks.
3. To accomplish each control task requires the real time computer to run a set of control algorithms.
4. The performability of the real time computer is defined as the probability that the computer system will allow each accomplishment level to be met.

5. Fig. 1.10.1 shows four views of a hierarchical view of performability :
   a. Each view is driven by the requirements of the one above it and receives input from the one below it.
   b. Each view is more detailed than the one above it.

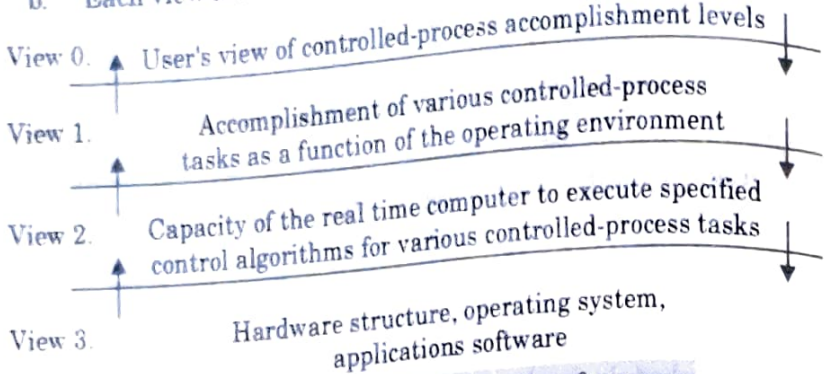


Fig. 1.10.1. Hierarchical view of performance.

   c. View 0 specifies in terms of the state variables of the controlled process just what enables the user to distinguish one grade of performance from another.
   d. View 1 is more detailed in specifying the controlled-process tasks that must be run to meet each such grade of performance, and view 2 is even more detailed in specifying which algorithms must be run to meet the controlled-process tasks in view 1.
   e. Finally, view 3 considers the hardware structure, the operating system, and the applications software attributes needed to meet the requirements of view 2.
6. Performability takes the operating environment into account in views 1 and 2.
7. The controls that must be applied and the algorithms that must be run in order to achieve certain user-defined grades of performance are functions of the operating environment.
8. However, performability's advantage lies not in this, but in its ability to express performance in the only measure that truly matters—the user' perception of it.

Que 1.11. Explain about typical real time applications.

Answer

1. These systems may include digital control, command and control, signal processing and telecommunication systems.
2. These systems are used to control the engines and brakes of car and regulate traffic lights to schedule and monitor the take-off and landing of plane, to monitor and regulate the blood pressure and heart beats.
3. Following are the typical real time applications :

i. Digital control :

a. Many real time systems are embedded in sensors and actuators and function as digital controllers.

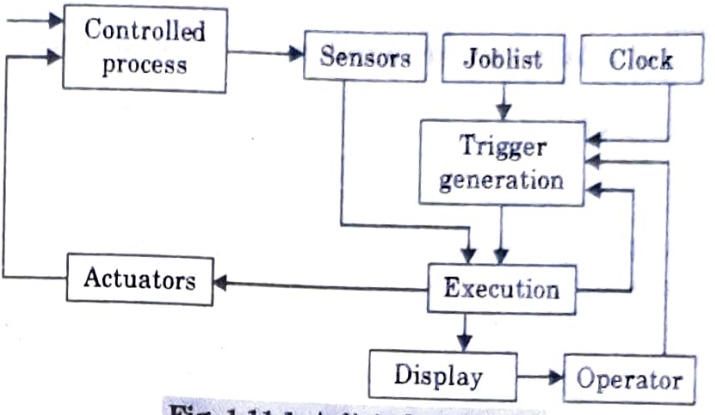


Fig. 1.11.1. A digital controller.

b. The term plant in the block diagram referred to a controlled system, for example, engine or brake.

c. The state of the plant is monitored by sensors and can be changed by actuators.

d. The 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. This computation is called the control-law computation.

e. Thus, the generated output activates the actuators which bring the plant closer to the desired state.

ii. High level controls :

a. In complex monitor and control system, controllers are typically organized hierarchically. Digital controllers at the lowest level directly control the physical plant.

b. Each output of higher level controller is a reference input of one or more lower level controllers.

c. For example, a patient care system may consist of microprocessor based controllers that monitor and control the patient's blood pressure, respiration and glucose.

d. These may be high level controller which interacts with the operator.

e. According to the command, high level controller do the computation and its output can be transmitted which is reference input for lower level controllers.

iii. Signal processing :

a. Most signal processing applications have real time requirements.

b. So, for real time systems the response time must be under a few milliseconds to a few seconds. For example, in digital filtering, voice compression etc.

Que 1.12. Explain various application areas where real time systems are useful. **UPTU 2014-15, Marks 05**

Answer

1. **Industrial applications :** Industrial applications constitute a major usage area of real time systems. A few examples of industrial applications of real time systems are : process control systems, industrial automation systems, SCADA applications, test and measurement equipment, and robotic equipment.
2. **Medical :** A few examples of medical applications of real time systems are : robots, MRI scanners, radiation therapy equipment, bedside monitors, and Computerized Axial Tomography (CAT).
3. **Peripheral equipment :** A few examples of peripheral equipment that contain embedded real time systems are : laser printers, digital copiers, fax machines, digital cameras, and scanners.
4. **Automotive and transportation :** A few examples of automotive and transportation applications of real time systems are : automotive engine control systems, road traffic signal control, air-traffic control, high-speed train control, car navigation systems, and MPFI engine control systems.
5. **Telecommunication applications :** A few example uses of real time systems in telecommunication applications are : cellular systems, video conferencing, and cable modems.
6. **Aerospace :** A few important uses of real time systems in aerospace applications are : avionics, flight simulation, airline cabin management systems, satellite tracking systems and computer on-board an aircraft.
7. **Consumer electronics :** Consumer electronics area is replete with numerous applications of real time systems. A few sample applications of real time systems in consumer electronics are : set-top boxes, audio equipment, internet telephony, microwave ovens, intelligent washing machines, home security systems, air conditioning and refrigeration, toys and cell phones.
8. **Defence applications :** Typical defence applications of real time systems include : missile guidance systems, anti-missile systems, satellite-based surveillance systems.
9. **Miscellaneous applications :** Real time systems have found numerous other applications in our everyday life. An example of such an application is a railway reservation system.

Que 1.13. Explain the air traffic control system.

UPTU 2012-13, Marks 05

Answer

1. The controller at the highest level of a control hierarchy is a command and control system.
2. An air traffic control (ATC) system is an excellent example.
3. Fig. 1.13.1 shows a possible architecture.

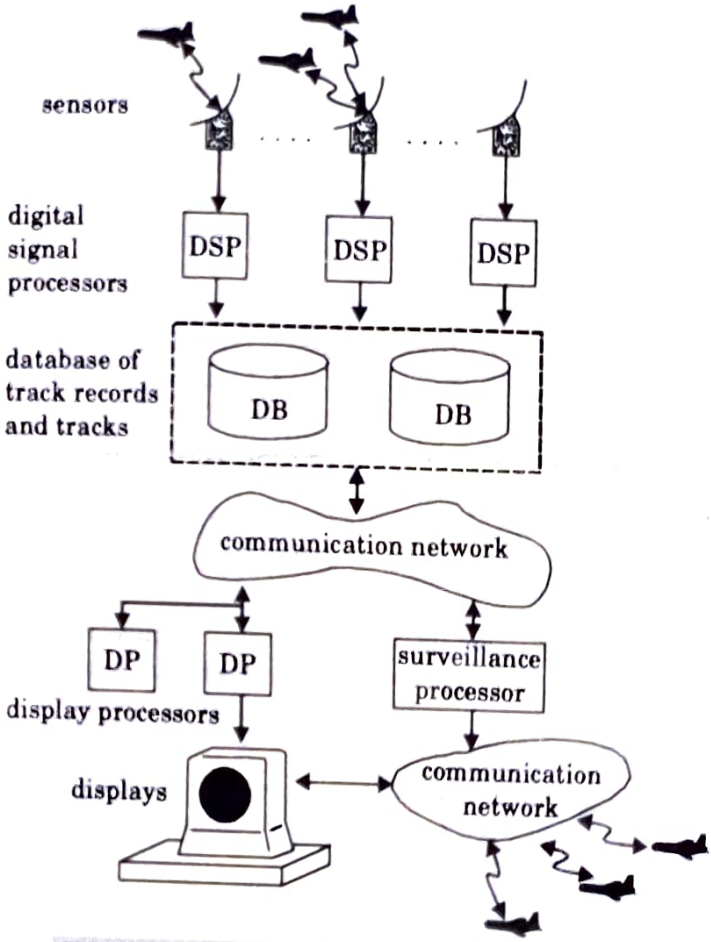


Fig. 1.13.1. An architecture of air traffic control system.

4. The ATC system monitors the aircraft in its coverage area and the environment (for example, weather condition) and generates and presents the information needed by the operators (*i.e.*, the air traffic controllers).
5. 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.

6. Thus, the ATC system indirectly controls the embedded components in low levels of the control hierarchy.
7. In addition, the ATC system provides voice and telemetry links to on-board avionics.
8. Thus, it supports the communication among the operators at both levels (*i.e.*, the pilots and air traffic controllers).
9. The ATC system gathers information on the "state" of each aircraft via one or more active radars.
10. Such a radar interrogates each aircraft periodically.
11. When interrogated, an aircraft responds by sending to the ATC system its "state variables" : identifier, position, altitude, heading, and so on.
12. The ATC system processes messages from aircraft and stores the state information thus obtained in a database.
13. This information is picked up and processed by display processors.

Que 1.14. Write short note on :

a. Release time
b. Deadline

Answer

a. Release time :

1. The release time of a job is the instant of time at which the job becomes available for execution.
2. The job can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met.
3. For example, a system which monitors and controls several furnaces, samples and reads each temperature sensor every 100 msec and places the sampled readings in memory, after its initialization and execution (say at time 0).
4. It also computes the control law of each furnace every 100 msec in order to process the temperature readings and determine flow rates of fuel, air and coolant.
5. Suppose that the system begins the first control law computation at time 20 msec.
6. The control law is computed periodically in terms of release times of the control law computation jobs : $J_0, J_1 \ldots\ldots, J_k \ldots\ldots$
7. The release time of the job J_k in this job stream is $20 + k \times 100$ msec for $k = 0, 1, \ldots$.
8. Jobs will have no release time if all the jobs are released when the system begins execution.

b. Deadline :

1. The deadline of a job is the instant of time by which its execution is required to be completed.
2. Suppose that in the previous example, each control law computation job must complete by the release time of the subsequent job.
3. Then their deadlines are 120 msec, 220 msec and so on respectively.
4. If the control law computation jobs must complete sooner, their deadlines may be 70 msec, 170 msec and so on. A job has no deadline if its deadline is at infinity.
5. Response time is the length of time from the release time of the job to the instant when it completes. The maximum allowable response time of a job is called relative deadline.
6. Hence, the relative deadline of every control law computation job mentioned above is 100 or 50 msec.
7. The deadline of a job, sometimes called its absolute deadlines is equal to its release time plus its relative deadlines.

Que 1.15. What do you mean by timing constraints ? What are durational timing constraints ? What are minimum and maximum timing constraints and how are they different from durational constraints ? **UPTU 2013-14, Marks 05**

OR

What is the difference between a performance constraint and a behavioural constraint in a real time system ? Explain with the help of one example of each. **UPTU 2014-15, Marks 05**

Answer

1. The correctness of real time tasks depend both on the logical correctness of the result as well as on the satisfaction of the corresponding timing constraints that apply to certain events in a system.
2. These events may be generated by the tasks themselves or the environment of the system. For example, activation of a motor is an event.
3. Events are classified into two types :

i. Stimulus events :

a. These are generated by the environment and act on the system. These events can be produced asynchronously.

b. For example, a user pressing a button on a telephone generates a stimulus event to act on the telephone system. These can also be produced synchronously.

c. For example, periodic sensing of the temperature of the reactor in a nuclear plant.

ii. Response events :

a. Response events act on the environment and are usually produced by the system in response to some stimulus events.

b. For example, consider a chemical plant where the temperature exceeds 100°C, the system responds by switching off the heater.

c. Here, the event of temperature exceeding is the stimulus and switching off of the heater is the response.

d. Response events can either be periodic or aperiodic.

4. Timing constraints can broadly be classified into performance and behavioural constraints.

5. Performance constraints ensure that the computer system performs satisfactorily whereas behavioural constraints ensure that the environment of a system is well behaved.

6. These constraints can further be classified into following three types :

a. Delay constraints :

i. It captures the minimum time (delay) that must elapse between the occurrence of two arbitrary events e_1 and e_2.

ii. After e_1 occurs, if e_2 occurs earlier than the minimum delay, then there is a delay violation.

iii. It can be expressed as : $t(e_2) - t(e_1) \geq d$, where $t(e_2)$ and $t(e_1)$ are the time stamps on the events e_2 and e_1 respectively and d is the minimum delay specified from e_2.

b. Deadline constraints :

i. It captures the permissible maximum separation between any two arbitrary events e_1 and e_2.

ii. The second event must follow the first event within the permissible maximum separation time.

iii. It can be expressed as : $t(e_2) - t(e_1) \leq d$ where $t(e_2)$ and $t(e_1)$ are the time stamps on the occurrence of events e_1 and e_2 respectively and d is the deadline.

c. Duration constraints :

i. A duration constraint on an event specifies the time period over which the event acts.

Difference between performance and behaviourial constraints :

S.No	Performance constraint	Behaviourial constraint
1.	It ensures that the computer system performs satisfactorily.	It ensures that the environment of a system is well behaved.
2.	Performance constraints are the constraints that are imposed on the response of the system.	Behaviourial constraints are the constraints that are imposed on the stimuli generated by the environment.
3.	Deadline constraint is further divided in performance constraint as : **a. Stimulus-Response (SR) :** The deadline is defined on the response event, measured from the occurrence of the corresponding stimulus event. Constraint is imposed on response event. For example, once the receiver of the handset is lifted, the dial tone must be produced by the system within 2 sec, otherwise a beeping sound is produced until the handset is replaced. **b. Response-Response (RR) :** Constraint is defined on two response events. In this case, once the first response event occurs, the second response event must occur before a certain deadline. The constraint has been defined on a response event hence, it is a performance constraint. For example, once the ring tone is given to the callee, the corresponding ring back tone must be given to the caller within 2 sec, otherwise the call is terminated.	Deadline constraint is further divided in behaviourial constraint as : **a. Response-Stimulus (RS) :** Here the deadline is on the production of response counted from the corresponding stimulus. The constraint is imposed on the stimulus event. For example, once the dial tone appears, first digit must be dialled within 30 sec, otherwise the system enters an idle state and an idle tone is produced. **b. Stimulus-Stimulus (SS) :** The deadline is defined between two stimuli. The constraint is imposed on the second event which is stimulus hence, it is a behaviourial constraint. For example, once a user completes dialling a digit, he must dial the next digit within the next 5 sec. Otherwise, an idle tone is produced.

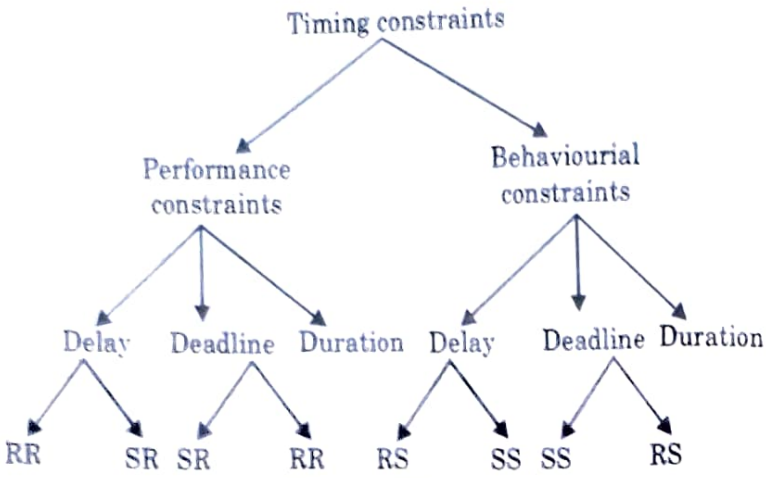


Fig. 1.15.1.

Duration constraint is further divided into two types :

1. **Maximum durational constraint :**
   a. In this type of constraint, the requirement is that once the event starts, the event must end before a certain maximum duration elapses.
   b. It uses the maximum duration which defines the maximum time for a particular event and that event should complete before this time.
2. **Minimum durational constraint :**
   a. In this type of constraint, the requirement is that once the event starts, the event must not end before a certain minimum duration.
   b. It uses the minimum duration that defines the smallest time which should be completed for a particular event.

Que 1.16. Discuss under what circumstances it is necessary to meet time constraints of time critical jobs for benefit of system.

OR

List the possible task timing constraints.

OR

Explain hard and soft timing constraints.

Answer

1. The constraint which is imposed on the timing behaviour of a job is called timing constraint.
2. A timing constraint of a job can be specified in terms of its release time and relative or absolute deadlines.
3. Some complex timing constraints cannot be specified conveniently in terms of release times and deadlines.

i. **Hard timing constraints :**
   a. A timing constraint or deadline is hard if the failure to meet is considered to be fatal fault.

b. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences.

c. For example, a late command to stop a train may cause a collision and a bomb dropped too late may hit a civilian population instead of the intended military target.

ii. Soft timing constraints :

a. It is timing constraint in which there is no severe loss when deadline is over.

b. These are not expressed as absolute values. Instead, these are expressed in terms of the average response times required.

c. In this the deadline overruns are tolerable but not desired.

d. For example, in web browsing after clicking the URL, corresponding web page is fetched and displayed within a couple of seconds on an average.

e. However, when it takes several minutes to display a requested page, we still do not consider the system to have failed, but merely express that the performance of the system has degraded.

Que 1.17. What do you understand by temporal constraints ? List the possible task timing constraints. **UPTU 2014-15, Marks 05**

Answer

1. Temporal constraints may or may not be met, however a specific class of applications, called in safety critical real time applications, requires the temporal constraint must to be always met.
2. Such safety-critical real time applications are systems whose failure or malfunction may lead to human losses. Since temporal behaviour is critical, such applications have to be certified to always meet the temporal constraints before they can start operating.
3. Temporal constraints exist in the forms of soft temporal constraints, hard temporal constraints, and firm temporal constraints.
4. Soft temporal constraints are found in soft real time systems. A soft real time system is one in which the response time (*i.e.*, a temporal constrain) is specified as an average value and the system does a best effort approach to meet these requirements.
5. A soft temporal constraint specifies an average bound between two events, and a single violation of this constraint does not significantly matter, whereas many violations and high value deviations do.
6. An example system with soft constraints is an airline reservation system. It is irrelevant if the reservation sometimes takes a little more time to book a reservation, as long as the booking operation's execution time is bounded reasonably.
7. Hard temporal constraints are found in mission-critical real time systems. A hard real time system is a system where the response time is specified as an absolute value that is typically dictated by the environment.

8. A hard temporal constraint specifies an absolute upper bound between two events. A single violation of such a constraint does matter as hard real time applications are often safety-critical applications.
9. An example of such a system is the electronic braking system mentioned above; if the braking system does not meet the constraint just once, this might have been the critical instant where meeting the constraint could have saved lives.
10. Firm temporal constraints are found in firm real time systems. Each constraint has a soft and hard part consisting of an average bound (soft) and an absolute bound (hard), respectively.
11. An example is the mentioned patient's ventilation system. Specifying the bound between two ventilation cycles, this timing constraint has a soft constraint and a hard constraint. The soft constraint is fourteen ventilations per minute, and the hard constraint is twelve ventilations.
12. Temporal constraints can be absolute or relative. Absolute temporal constraints are measured with respect to a global clock, usually our wall clock.
13. Relative temporal constraints are measures with respect to the occurrence times of specific events on the local clock.

Types of task timing constraints :

1. Different timing constraints associated with a real time system can broadly be classified into performance and behavioural.
2. Behavioural constraints ensure that the environment of a system is well behaved, whereas performance constraints ensure that the computer system performs satisfactorily.
3. Each of the performance and behavioural constraints can further be classified into the following three types :
   i. Delay constraint
   ii. Deadline constraint
   iii. Duration constraint

Delay :

1. A delay constraint captures the minimum time (delay) that must elapse between the occurrence of two arbitrary events e_1 and e_2. After e_1 occurs, if e_2 occurs earlier than the minimum delay, then a delay violation is said to occur.
2. A delay constraint on the event e_2 can be expressed more formally as follows :

$$t(e_2) - t(e_1) \geq d$$

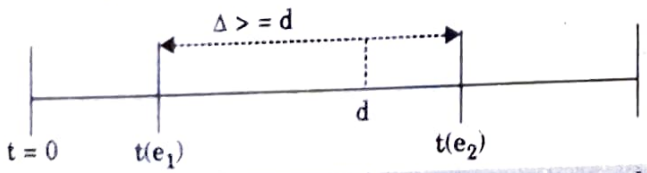


Fig. 1.17.1. Delay constraint between two events e_1 and e_2.

3. Where $t(e_2)$ and $t(e_1)$ are the time stamps on the events e_2 and e_1, respectively, and d is the minimum delay specified from e_2.
4. A delay constraint on the events e_2 with respect to the event e_1 is shown in Fig. 1.17.1. In Fig. 1.17.1, Δ denotes the actual separation in time between the occurrence of the two events e_1 and e_2 and d is the required minimum separation between the two events (delay).
5. It is easy to see that e_2 must occur after at least d time units have elapsed since the occurrence of e_1, otherwise we shall have a delay violation.

Deadline :

1. A deadline constraint captures the permissible maximum separation between any two arbitrary events e_1 and e_2.
2. In other words, the second event (*i.e.*, e_2) must follow the first event (*i.e.*, e_1) within the permissible maximum separation time.
3. Consider that $t(e_1)$ and $t(e_2)$ are the time stamps on the occurrence of the events e_1, and e_2, respectively, and d is the deadline as shown in Fig. 1.17.2.
4. In Fig. 1.17.2, Δ denotes the actual separation between the time of occurrence of the two events e_1 and e_2, and d is the deadline.
5. A deadline constraint implies that e_2 must occur within d time units of e_1's occurrence.
6. We can alternatively state that $t(e_1)$ and $t(e_2)$ must satisfy the constraint :

$$t(e_2) - t(e_1) \leq d$$

7. The deadline and delay constraints can further be classified into two types, each based on whether the constraint is imposed on the stimulus or on the response event.

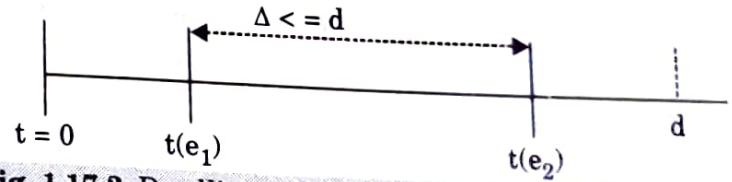


Fig. 1.17.2. Deadline constraint between two events e_1 and e_2.

Duration :

1. A duration constraint on an event specifies the time period over which the event acts.
2. A duration constraint can either be minimum type or maximum type.
3. The minimum type duration constraint requires that once the event starts, the event must not end before a certain minimum duration; whereas a maximum type duration constraint requires that once the event starts, the event must end before a certain maximum duration elapses.

PART-2

Hard Real Time System and Soft Real Time Systems, Reference Models for Real Time Systems : Processors and Resources.

CONCEPT OUTLINE : PART-2

- **Hard real time systems :** Embedded system that control things like aircraft, nuclear reactors, chemical power plants, jet engines where "something bad" will happen if the computer do not deliver its output in time are called hard real time system.
- **Soft real time systems :** A system in which jobs have soft deadlines is called soft real time system. For example, online transaction system, telephone switches etc.
- **Reference model :** According to this model, each system is characterized by three elements :
  1. A workload model
  2. A resource model
  3. Algorithms

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.18. **Differentiate between hard and soft real time system. Also explain the timing constraints of real time system.**

UPTU 2012-13, Marks 05

OR

With suitable examples explain the difference between soft and hard real time systems.

UPTU 2013-14, Marks 05

OR

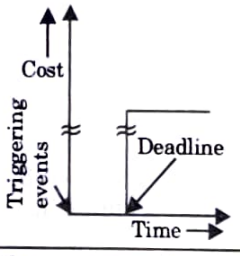
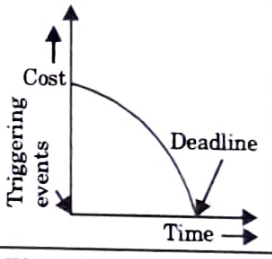
Differentiate between hard and soft real time systems.

UPTU 2014-15, Marks 05

OR

Differentiate between soft, firm and hard real time systems and give one suitable example of each.

Answer

S. No.	Hard real time system	Firm real time system	Soft real time system
1.	Hard real time system is constrained to produce its results within certain predefined time bounds.	Firm real time system is associated with some predefined deadline before which it is required to produce its results.	Soft real time systems also have time bounds associated with it. The timing constraints on soft real time systems are not expressed as absolute values, instead expressed in terms of the average response time required.
2.			
3.	The cost function in hard real time system is associated with the system.	The cost may be interpreted as loss of revenue in firm real time system.	The cost is associated to overrunning, but this cost may be abstract in case of soft real time system.
4.	Overrun in response time leads to potential loss of life or big financial damage.	If it does not complete within its deadline, the system does not fail. Late results are merely discarded.	Deadline overruns are tolerable, but not desired.
5.	Time bound usually range from several microseconds to a few milliseconds.	Associated time bound typically range from a few milliseconds to several hundreds of milliseconds.	Time bounds associated are expressed as absolute values.
6.	For example : ROBOT	For example : Video conferencing	For example : Web browsing

Timing constraints : Refer Q. 1.15, Page 25C, Unit-1.

Que 1.19. Write the difference between real time system and other systems. **UPTU 2012-13, Marks 05**

Answer

1. Differences between real time systems and time sharing systems :

S.No.	Real time systems	Time sharing systems
1	Real time systems works under fixed time constraints.	This system works for the same or less time on each process.
2	This system uses priority scheduling.	This system uses FCFS (First Come First Served) but do not complete a process at once, but a portion of it.
3	Resources remain dedicated for a user for a fixed amount of time and can be reassigned to another user after that time.	Users can share resources.
4	Example : Real time systems are required in airlines ticket booking systems.	Example : Time sharing systems are required in online file systems.

2. Difference between distributed systems and real time system :

a. A distributed OS provides the essential services and functionality required of an OS, adding attributes and particular configurations to allow it to support additional requirements such as increased scale and availability.

b. To a user, a distributed OS works in a manner similar to a single-node, monolithic operating system. That is, although it consists of multiple nodes, it appears to users and applications as a single-node.

c. Separating minimal system-level functionality from additional user-level modular services provides a "separation of mechanism and policy."

d. Mechanism and policy can be simply interpreted as "how something is done" versus "why something is done," respectively. This separation increases flexibility and scalability.

e. A real time operating system (RTOS) is an operating system (OS) intended to serve real time application process data as it comes in, typically without buffering delays. Processing time requirements

(including any OS delay) are measured in tenths of seconds or shorter.

f. A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task; the variability is jitter.

3. Real time OS differ from general-purpose OS :

a. Setting priorities :

i. When programming an application, most operating systems (of any type) allow the programmer to specify a priority for the overall application and even for different tasks within the application (threads).

ii. These priorities serve as a signal to the OS, dictating which operations the designer feels are most important.

iii. The goal is that if two or more tasks are ready to run at the same time, the OS will run the task with the higher priority.

iv. In practice, general-purpose operating systems do not always follow these programmed priorities strictly.

v. Because general-purpose operating systems are optimized to run a variety of applications and processes simultaneously, they typically work to make sure that all tasks receive at least some processing time.

vi. As a result, low-priority tasks may in some cases have their priority boosted above other higher priority tasks.

vii. This ensures some amount of run time for each task, but means that the designer's wishes are not always followed.

viii. In contrast, real time operating systems follow the programmer's priorities **much** more **strictly**.

ix. On most real time operating systems, if a high priority task is using 100 % of the processor, no other lower priority tasks will run until the high priority task finishes.

x. Therefore, real time system designers must program their applications carefully with priorities in mind.

xi. In a typical real time application, a designer will place time-critical code (example, event response or control code) in one section with a very high priority.

xii. Other less-important code such as logging to disk or network communication may be combined in a section with a lower priority.

b. Interrupt latency :

i. Interrupt latency is measured as the amount of time between when a device generates an interrupt and when that device is serviced.

ii. While general-purpose operating systems may take a variable amount of time to respond to a given interrupt, real time operating systems must guarantee that all interrupts will be serviced within a certain maximum amount of time.

iii. In other words, the interrupt latency of real time operating systems must be bounded.

c. **Performance :**

i. One common misconception is that real time operating systems have better performance than other general-purpose operating systems.

ii. While real time operating systems may provide better performance in some cases due to less multitasking between applications and services, this is not a rule.

iii. Actual application performance will depend on CPU speed, memory architecture, program characteristics, and more.

iv. Though real time operating systems may or may not increase the speed of execution, they can provide much more precise and predictable timing characteristics than general-purpose operating systems.

Que 1.20. Explain reference model of real time systems.

Answer

1. A good model abstracts the irrelevant details of real time application like system resources (for example, whether the processor is by Intel or Motorola or whether the transmission medium is cable or fiber).
2. A good model allows us to focus on the timing properties and resource requirements of system components and the way the operating system allocates the available system resources among them.
3. By focusing the relevant characteristics of the system, we can get better timing behaviour of each component and the overall system.
4. One of the model of real time systems is reference model. According to this model, each system is characterized by three elements :
   a. A workload model that describes the applications supported by the system.
   b. A resource model that describes the system resources available to the applications.
   c. Algorithms that define how the application system uses the resources at all times.
5. This model has a set of features in terms of which we can describe the relevant characteristics of real time applications and the properties of the underlying platform.

6. Through this model, we can analyze, simulate and even emulate the system based on its description.

Elements of the reference model :

a. Processors and resources

b. Temporal parameters of real time workload

c. Periodic task model

d. Precedence constraint and data dependency

Que 1.21. Explain the term processors and resources in context of real time systems.

Answer

All the system resources are divided into two major types : processors and resources.

1. **Processors :**

   a. Processors are often called servers and active resources.

   b. Computers, transmission links, disks and database server are examples of processors.

   c. They carry out machine instructions, move data from one place to another, retrieve files, process queries and so on.

   d. Every job must have one or more processors in order to execute and make progress toward completion.

   Types of processors :

   a. Two processors are of the same type if they are functionally identical and can be used interchangeably. Hence, two transmission links with the same transmission rate between a pair of sender and receiver are processors of the same type.

   b. Processors that are functionally different, or for some other reason cannot be used interchangeably, are of different types. CPUs, transmission links and disks are of different types because they are functionally different. To denote processor(s), letter P is used. For m processors in the system, we use $P_1, P_2, \ldots\ldots, P_m$.

2. **Resources :**

   a. These are called passive resources. For example, resources are memory, sequence numbers, mutexes and database locks.

   b. A job may need some resources in addition to the processor in order to make progress.

   c. One of the attributes of a processor is its speed.

   d. For example, we usually model transactions that query and update a database as jobs; these jobs execute on a database server.

e. If the database server uses a locking mechanism to ensure data integrity, then a transaction also needs the locks on the data objects it reads or writes in order to proceed.

f. The locks on the data objects are resources.

g. Letter R is used to denote resources.

h. The resources are reusable if they are not consumed during use.

i. If the system contains p resources means that there are p types of serially reusable resources, each resource may have one or more units and each unit is used in a mutually exclusive manner.

j. A resource is plentiful if no job is ever prevented from execution by the lack of this resource.

k. A resource that can be shared by an infinite number of jobs need not be explicitly modeled and hence never appears in this model.

PART-3

Temporal Parameters of Real Time Workload, Periodic Task Model, Precedence Constraints and Data Dependency.

CONCEPT OUTLINE : PART-3

- **Temporal parameters of real time workload :** The workload on processors consists of jobs, each of which is a unit of work to be allocated processor time and other resources.
- **Execution time :** Another temporal parameter of a job, J_i, is its execution time *i.e.*, the amount of time required to complete the execution of J_i when it executes alone and has all the resources.
- The jobs are said to have precedence constraints if they are constrained to execute in some order.
- If the jobs can execute in any order, they are said to be independent.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.22. What is the use of temporal parameters of real time workload in reference model ?

OR

What are the parameters of real time workload ? Explain in brief.

UPTU 2012-13, Marks 05

Answer

Use of temporal parameters of real time workload :

1. The workload on processors consists of jobs, each of which is a unit of work to be allocated processor time and other resources.
2. A set of related jobs that execute to support a function of the system is called a task.
3. Many parameters of hard real time jobs and tasks are known at all times otherwise, it would not be possible to ensure that the system meet its hard real time requirements.
4. The number of tasks (or jobs) in the system is one such parameter.
5. The number of tasks may change when the system operation mode changes and the number of tasks in the new mode is also known.
6. Moreover, these numbers are known a priori before the system begins execution.
7. For example, in flight control system, during cruise mode, the system has 12 tasks.
8. If the system triply replicates all control law computations during landing, the number of tasks increases to 24 when it operates in the landing mode.
9. Each job J_i is characterized by its temporal parameters, functional parameters, resource parameters and interconnection parameters.
10. Its temporal parameters tell us its timing constraints and behaviour.
11. Release time, absolute deadline and relative deadline of a job J_i, are temporal parameters.

Parameters of real time workload :

A. Execution time :

1. It is the amount of time required to complete the execution of J_i when it executes alone and has all the resources it requires.
2. Hence, the value of this parameter depends mainly on the complexity of the job and the speed of the processor used to execute the job, not on how the job is scheduled.
3. Execution time is represented by e_i.
4. The actual amount of time required by a job to complete its execution may vary for many reasons.
5. For example, a computation may contain conditional branches and these conditional branches may take different amounts of time to complete.
6. If the underlying system has performance enhancing features (for example, cache memory and pipeline), the amount of time a computation takes to complete may vary each time it executes even when it has no conditional branches.

7. For these reasons, the actual execution time of a computation job is unknown until it completes.
8. The execution time e_i of the job J_i is in the range $[e_i^-, e_i^+]$ where e_i^- and e_i^+ are the minimum execution time and the maximum execution time of J_i respectively.
9. The maximum execution time of each job is adequate to determine whether each job can always complete by its deadline or not.
10. For this reason, in most deterministic models used to characterize hard real time applications, the term execution time e_i of each job J_i specifically means its maximum execution time.
11. There are two good reasons for the common use of the deterministic approach.
12. Many hard real time systems are safety-critical. These systems are typically designed and implemented in such a way that the variations in job execution times are kept as small as possible.
13. The need to have relatively deterministic execution times places many restrictions on implementation choices.
14. By working with these restrictions and making the execution times of jobs almost deterministic, the designer can model more accurately the application system deterministically.
15. The other reason for the common use of the deterministic approach is that hard real time portion of the system is often small.
16. The timing requirements of the rest of the system are soft.
17. An option is to design the hard real time subsystem based on its worst case processor time and resource requirements even though their actual requirement may be much smaller.
18. Then we can use the methods and tools supported by the deterministic models to ensure that the hard timing constraints will surely be met at all times.

B. Fixed, jittered and sporadic release times :

1. In many systems, we do not know exactly when each job will be released.
2. In other words, we do not know the actual release time r_i of each job J_i; only that r_i is in a range $[r_i^-, r_i^+]$. r_i can be as early as the earliest release time r_i^- and as late as the latest release time r_i^+.
3. Indeed, some models assume that only the range of r_i is known and call this range the *jitter* in r_i, or *release-time jitter*.
4. Sometimes, the jitter is negligibly small as compared with the values of other temporal parameters.
5. Almost every real time system is required to respond to external events which occur at random instants of time.

6. When such an event occurs, the system executes a set of jobs in response.
7. The release times of these jobs are not known until the event triggering in them occurs.
8. These jobs are called sporadic jobs or aperiodic jobs because they are released at random time instants.

Que 1.23. Discuss the factors that are to be analyzed for estimating execution time for real time system.

Answer

Refer Q. 1.22, Page 38C, Unit-1.

Que 1.24. How do pipelining and interrupts affect execution time estimation ? **UPTU 2013-14, Marks 05**

Answer

Analysis of source code :

1. Let us begin by considering a very simple stretch of code.

```
L1 : a : = b * c;
L2 : b : = d + e;
L3 : d : = a – f;
```

2. This is straight-line code.
3. Once control is transferred to the first of these statements, execution will continue sequentially until the last statement has been completed.
4. Straight-line code is thus a stretch of code with exactly one entry point and exactly one exit point.
5. Let us consider how to make a timing estimate for this stretch of code.
6. The total execution time is given by,

$$\sum_{i=1}^{3} T_{exec}(Li) \qquad ...(1.24.1)$$

where $T_{exec}(Li)$ is time needed to execute Li.

7. The time needed to execute $L1$ will depend on the code that the compiler generates for it. For example, $L1$ could be translated into the following sequence :

```
L1 . 1 Get the address of c
L1 . 2 Load c
L1 . 3 Get the address of b
L1 . 4 Load b
L1 . 5 Multiply
L1 . 6 Store into a
```

8. The time needed to execute these instructions will depend on the machine architecture.
9. If the machine is very simple, does not use pipelining, and has only one I/O port to the memory, then the execution time is given by the sum $\sum_{i=1}^{6} T_{exec}(L1.i)$. This assumes that the machine is devoted during that time to just these instructions; for example, that there are no interrupts.
10. However, there are two factors that could make this bound rather loose.
11. First, we are implicitly assuming that the variables b and c are not already in the CPU registers and have to be fetched from the cache or the main memory.
12. This overlooks the possibility that some preceding code might have already loaded these variables into the registers and that they are still there.
13. Second, the bounds on the execution times of individual instructions could be loose because they are data-dependent.
14. For example, the multiply operation does not take a deterministic time on most machines the time it takes to multiply two numbers depends on the numbers themselves.
15. If we do not know in advance what these numbers are, we can only write loose bounds on the multiplication time.
16. Suppose that we have the following loop :
    L4 . while (P) do
    L5 . Q1 ;
    L6 . Q2 ;
    L7 . Q3 ;
    L8 . end while ;
    where P is a logical statement and $Q1$, $Q2$, and $Q3$ are instructions.
17. It is obvious that unless we know how many times this loop is going to be executed, we have no way of estimating the execution time.
18. So, at the very least, we need upper and lower bounds on the number of loop iterations.
19. These bounds may be derived either from an analysis of P, $Q1$, $Q2$, and $Q3$, or from some other user-derived input.
20. The difficulty of deriving such bounds is why while loops are forbidden in the Euclid real time programming language.
21. Consider the following if-then-else construct :
    L9 . if B1 then
    S1 ;
    else if B2 then
    S2 ;
    else if B3 then

```
        S3 ;
    else
S4 ;
    end if ;
```

22. The execution time will depend on which of the conditions B1, B2, B3 are true. In the case where B1 is true, the execution time is

$$T(B1) + T(S1) + T(JMP) \quad ...(1.24.2)$$

where T(JMP) is the time it takes to jump to the end of the if-then-else construct.

23. In the case where B1 is false but B2 is true, the execution time is

$$T(B1) + T(B2) + T(S2) + T(JMP) \quad ...(1.24.3)$$

24. The equation for the other two cases is written similarly.

25. If $t_{lower}(i)$ and $t_{upper}(i)$ are the lower and upper bounds of the estimates of case i, then (since there are four cases in all) the lower and upper execution time bounds for this construct are given by

$$\min_{i \in (1, 2, 3, 4)} t_{lower}(i) \text{ and } \text{Mix}_{i \in (1, 2, 3, 4)} t_{upper}(i) \text{ respectively.} \quad ...(1.24.4)$$

26. This becomes considerably more complex if we allow interrupts.

37. The execution-time bound is then a function of the rates at which the interrupts occur and on the bounds on the time taken to service each interrupt.

28. Figure 1.24.1 shows the schematic of an experimental estimation system developed at the University of Washington for programs written in C.

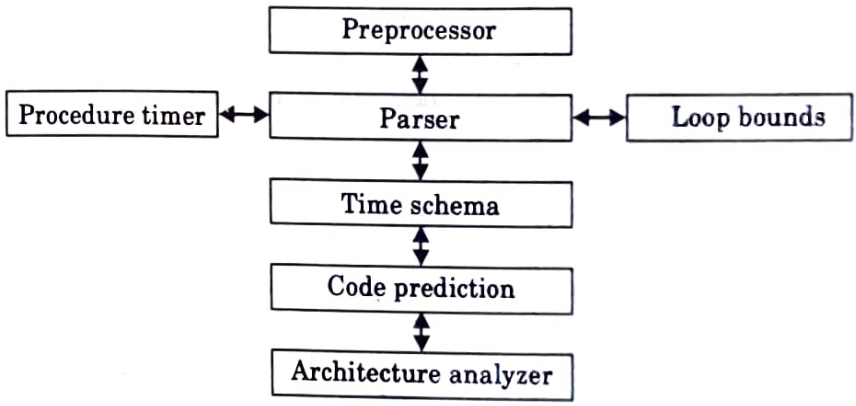


Fig. 1.24.1.

29. The preprocessor produces compiled assembly-language code and marks off blocks of code to be analyzed.

30. For example, it might mark off the code associated with a single source level instruction or a straight-line stretch of code as a block.

31. The parser analyzes the input source program. The procedure timer maintains a table of procedures and their execution times.

32. The loop bounds module obtains bounds on the number of iterations for the various loops in the system.

33. The time schema is independent of the system; it depends only on the language.
34. It computes the execution times of each block using the execution time estimates computed by the code prediction module.
35. The code prediction module does this by using the code generated by the preprocessor and using the architecture analyzer to include the influence of the architecture.

Accounting for pipelining :

1. The traditional Von Neumann model of computers assumes sequential execution.
2. An instruction is fetched, decoded, and executed.
3. Only after that has been done does work begin on the next instruction.
4. Under such conditions, computing the execution time of a straight-line stretch of code (in the absence of interrupts) consists of adding up the execution times of the individual instructions.
5. The instruction execution-time is itself computed by simply adding up the times taken to fetch the instruction, decode it, fetch any operands that are required, execute the instruction, and, finally, carry out any store operations that are called for.
6. The time taken for each of these steps can be obtained by examination of the machine architecture.
7. Most modern computers do not follow the von Neumann model exactly.
8. They use pipelining, which involves the simultaneous handling of different parts of different instructions.
9. To make programming easier, the machine still looks to the programmer as if it were following the von Neumann model.
10. Much of the complexity in dealing with pipelined machines arises from dependencies between instructions simultaneously active in the computer.
11. For example, if instruction I_i requires the output of I_j, I_i must wait for I_j to produce that output before it can execute. Two other sources of complexity are conditional branches and interrupts.
12. The condition of a conditional branch has to be evaluated before the processor can tell whether the branch will be taken. Such an evaluation typically takes place in the execute stage.
13. After the unconditional branch has been uncovered, but before the system knows whether or not it will be taken, the fetch unit has the following options :
    a. Stop prefetching instructions until the condition has been evaluated.
    b. Guess whether or not the branch will be taken and fetch instructions according to this guess.
14. The first alternative degrades performance, and so systems usually implement the second.

15. In the worst case, there will be an incorrect guess.
16. In such an event, the incorrectly prefetched instructions must be discarded and fetching must recommence, from the correct point,
17. Interrupts are another source of complexity. Suppose an interrupt occurs and transfers control to an interrupt-handling routine.
18. The worst-case execution time must then take into account the interrupt-handling time.

Que 1.25. Explain periodic task model in context of real time system.

OR

Define aperiodic and sporadic jobs. Explain the general strategy to handle sporadic jobs.

OR

Differentiate between periodic, aperiodic and sporadic jobs with suitable examples.

Answer

Periodic task model : The periodic task model is a well-known deterministic workload model. With its various extensions the model characterizes accurately many traditional hard real time applications, such as digital control, real time monitoring and constant bit-rate voice/video transmission. Many scheduling algorithms based on this model have good performance and well understood behaviour. There are many methods and tools to support the design, analysis and validation of real time system that can be accurately characterized by the model.

A. Periodic tasks :

1. In the periodic task model, each computation or data transmission that is executed repeatedly at regular or semiregular time intervals in order to provide a function of the system on a continuing basis is modeled as a period task.
2. Specifically, each periodic task, denoted by T_i, is a sequence of jobs.
3. The period p_i of the periodic task T_i is the minimum length of all time intervals between release times of consecutive jobs in T_i.
4. Its execution time is the maximum execution time of all the jobs in it.
5. At all times, the period and execution time of every periodic task in the system are known.
6. The accuracy of the periodic task model decreases with increasing jitter in release times and variations in execution times.
7. So, a periodic task is an inaccurate model of the transmission of a variable bit-rate video because of the large variation in the execution times of jobs.
8. The release time $r_{i,1}$ of the first job $J_{i,1}$ in each task T_i is called the phase of T_i.

9. We use ϕ_i to denote the phase of T_i, that is, $\phi_i = r_{i,1}$.
10. In general, different tasks may have different phases.
11. Some tasks are in phase, meaning that they have the same phase.
12. We use H to denote the least common multiple of p_i for $i = 1, 2, \ldots n$. A time interval of length H is called a hyperperiod of the periodic tasks.
13. The number N of jobs in each hyperperiod is equal to $\Sigma_{i=1}^{n} H/p_i$.
14. The length of a hyperperiod of three periodic tasks with periods 3, 4 and 10 is 60. The total number N of jobs in the hyperperiod is 41.
15. A job in T_i that is released at t must complete D_i units of time after t; D_i is the (relative) deadline of the task T_i.
16. For every task, a job is released and becomes ready at the beginning of each period and must complete by the end of the period.
17. In other words, D_i is equal to p_i for all n.
18. This requirement is consistent with the throughput requirement that the system can keep up with all the work demanded of it at all times.

B. Sporadic tasks :

1. These are the tasks in which the jobs are released at random time instants and have hard deadlines.
2. These are the hard real time tasks.
3. In these tasks, the primary concern is to ensure that the deadlines are always met and the secondary concern is to minimize their response times.
4. A sporadic task T_i can be represented by a three tuple : $T_i = (e_i, g_i, d_i)$, where e_i is the worst case execution time of an instance of the task, g_i denotes the minimum separation between the consecutive instances of the task, d_i is the relative deadline.
5. Here g_i implies that once an instance of a sporadic task occurs, the next instance cannot occur before (g_i) time units have elapsed *i.e.*, g_i restricts the rate at which sporadic tasks can arise.
6. Many sporadic tasks such as emergency message arrivals are highly critical in nature.
7. For example, in a robot a task that gets generated to handle an obstacle that suddenly appears, the tasks that handle fire conditions in factory are the sporadic tasks.

C. Aperiodic tasks :

1. These are the tasks in which the jobs have either soft deadline or no deadline.
2. These are the soft real time tasks. In these tasks, the primary concern is to optimize the responsiveness of the system.
3. In these tasks, the minimum separation g_i between two consecutive instances can be 0 *i.e.*, two or more instances of an aperiodic task might occur at the same time instant.

4. These tasks can tolerate the few deadline misses.
5. All the interactive commands issued by users are handled by aperiodic tasks.
6. For example, operator requests, keyboard presses, mouse movements etc.

Que 1.26. Compare and contrast critical and non-critical tasks.

UPTU 2013-14, Marks 05

Answer

Critical and non-critical tasks :

1. Real time tasks can also be classified according to the consequences of their not being executed on time.
2. Critical tasks are those whose timely execution is critical; if deadlines are missed, catastrophes occur.
3. Examples include life-support systems and the stability control of aircraft.
4. Despite their critical nature, it is not always essential that every iteration of the critical tasks be executed successfully and within a given deadline.
5. Critical tasks are often executed at a higher frequency than is absolutely necessary.
6. This constitutes time redundancy and ensures one successful computation every n_i iterations of critical periodic task i, which is sufficient to keep the system alive.
7. Of course, the actual value of n_i will depend on the application, the nature of the task itself, and the number of times the task is invoked over a given interval.
8. Non-critical real time (or soft real time) tasks are, as the name implies, not critical to the application.
9. However, they do deal with time-varying data and hence they are useless if not completed within a deadline.
10. The goal in scheduling these tasks is thus to maximize the percentage of jobs successfully executed within their deadlines.

Que 1.27. What do you understand by precedence constraints and data dependency ?

OR

What do you mean by precedence constraints and dependencies among them ?

UPTU 2012-13, Marks 05

Answer

Precedence constraints and data dependency :

1. Data and control dependencies among jobs may constrain the order in which they can execute.
2. The jobs are said to have precedence constraints if they are constrained to execute in some order.

3. For example, in queries to an information server, the authorization to access the requested information is first checked before each query is processed and the requested information is retrieved.
4. The retrieval job cannot begin execution until the authentication job completes.
5. The communication job that forwards the information to the requester cannot begin until the retrieval job completes.

Precedence graph and task graph :

1. A partial order relation $<$, called a precedence relation, is used over the set of jobs to specify the precedence constraints among jobs.
2. A job J_i is a predecessor of another job J_k if J_k cannot begin execution until the execution of J_i completes. J_i is an immediate predecessor of J_k if $J_i < J_k$ and there is no other job J_j such that $J_i < J_j < J_k$.
3. Two jobs J_i and J_k are independent when neither $J_i < J_k$ nor $J_k < J_i$.
4. A job with predecessors is ready for execution when the time is at or after its release time and all of its predecessors is completed.
5. To represent the precedence constraints among jobs in a set J, we use a directed graph $G = (J, <)$.
6. Each vertex in this graph represents a job in J.
7. There is a directed edge from the vertex J_i to the vertex J_k when the job J_i is an immediate predecessor of the job J_k.
8. This graph is called a precedence graph.
9. A task graph, which gives a general way to describe the application system, is an extended precedence graph.
10. In a precedence graph, the vertices in a task graph represent jobs.
11. They are shown as circles and squares in Fig. 1.27.1.
12. The numbers in the bracket above each job give its feasible interval.
13. The edges in the graph represent dependencies among jobs.
14. If all the edges are precedence edges, representing precedence constraints, then the graph is a precedence graph.
15. Fig. 1.27.1 includes two periodic tasks. The task whose jobs are represented by the vertices in the top row has phase 0, period 2 and relative deadline 7.
16. The jobs in it are independent; there are no edges to or from these jobs.
17. In other words, the jobs released in later periods are ready for execution as soon as they are released even though some job released earlier is not yet complete.
18. This is the usual assumption about periodic tasks.
19. The vertices in the second row of figure represent jobs in a periodic task with phase 2, period 3 and relative deadline 3.
20. The jobs in it are dependent; the first job is the immediate predecessor of the second job, the second job is the immediate predecessor of the third job and so on. The precedence graph of this task is a chain as shown in Fig. 1.27.1.

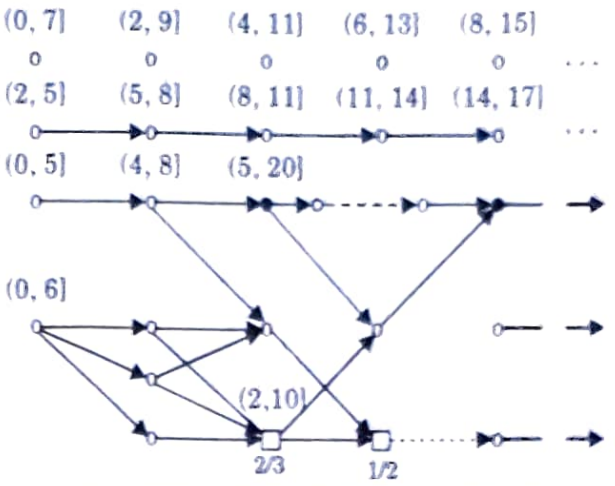


Fig. 1.27.1. Example of task graphs.

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

Data dependency :

1. Whenever a process is generating the data, it stores that data in the shared address space and then this data can be used by any other process at any time.
2. Hence, there is dependency among the data which are used for performing the tasks.
3. For example, in an avionics system, the navigation job updates the location of the air plane periodically.
4. These data are placed in shared space.
5. Whenever the flight management job needs navigation data, it reads the most current data produced by navigation job.
6. There is no any precedence constraint between the navigation job and the flight management job.

Que 1.28. What do you mean by preemptivity of jobs ? What are the differences in non-preemptable and preemptable jobs ?

OR

What do you understand by function parameters ? Explain their types.

Answer

In addition to scheduling and resource access control decisions which are made disregarding most functional characteristics of jobs, there are several functional properties which can affect these decisions. The workload model must explicitly describe these relevant properties and this is done by the values of functional parameters.

Following are the types of function parameters :

1. **Preemptivity of jobs :**
   a. The scheduler may suspend the execution of less urgent job and give the processor to a more urgent job.

b. When the more urgent job completes, the scheduler returns the processor to the less urgent job so the job can resume execution.
c. This interruption of job execution is called preemption.
d. Two types of preemption :
   i. **Preemptable jobs :** A job is preemptable if the execution can be suspended at any time to allow the execution of other jobs and further it can be resumed from the point of suspension. For example, computation jobs which are executed on CPUs.
   ii. **Non-preemptable jobs :** A job is non-preemptable if it must be executed from start to completion without interruption. This constraint may be imposed because if the job is suspended then it must be executed again from the beginning. For example, in transmissions of data frames in a token ring if the transmission of frame is interrupted before it completes, the partially transmitted frame is discarded by the receiver. The entire frame must be retransmitted from the start.

2. **Criticality of jobs :**
   a. In any system, all the jobs are not equally important.
   b. The criticality (or importance) of a job is a positive number that indicates how critical the job is with respect to other jobs in the system.
   c. The job with more criticality is more important than other jobs.
   d. During an overload when it is not possible to schedule all the jobs to meet their deadlines then the less critical jobs can be discarded so that the more critical jobs can meet their deadlines.
   e. For example, in a flight control and management system, the job that controls the flight is more critical than the navigation job that determines the current position.

3. **Optional executions :**
   a. In a system, there can be some jobs or portions of jobs which are optional.
   b. If an optional job or optional portion of job completes after its deadline or is not executed then the performance may degrade but the result is satisfactory.
   c. During an overload it is not possible to complete all the jobs in same time, we may choose to discard optional jobs so that the mandatory jobs can complete in time.
   d. For example, in a collision avoidance system, the job that computes the correct evasive action and informs the operator of this action is optional.

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

2 UNIT

Real Time Scheduling

PART-1

Common Approaches to Real Time Scheduling : Clock Driven Approach, Round-Robin Approach, Priority-Driven Approach, Dynamic Versus Static System.

CONCEPT OUTLINE : PART-1

- **Real time task scheduling :** It refers to determining the order in which the various tasks are to be taken up for execution by the operating system.
- **Classification of scheduling algorithms :**

1. Clock driven :
   a. Table-driven
   b. Cyclic
2. Event driven :
   a. Simple priority-based
   b. Rate monotonic analysis (RMA)
   c. Earliest -Deadline-First (EDF)
3. Hybrid :
   a. Round-robin

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. Explain task scheduling and its important concepts and terminologies.

OR

Write the important terms used in real time task scheduling.

Answer

1. Real time task scheduling essentially refers to determining the order in which the various tasks are to be taken up for execution by the operating system.
2. Every operating system relies on one or more task scheduler to prepare the scheduling algorithm.
3. Following are few important concepts and terminologies :
   a. **Valid schedule :** A valid schedule for a set of tasks is one where at most one task is assigned to a processor at a time, no task is scheduled

before its arrival time and precedence and resource constraints of all tasks are satisfied.

b. **Feasible schedule :** A valid schedule is called a feasible schedule, only if all tasks meet their respective time constraints in the schedule.

c. **Proficient scheduler :**

i. A task scheduler S1 is said to be more proficient than another scheduler S2, if S1 can feasibly schedule all task sets that S2 can feasibly schedule, but not vice versa.

ii. That is, S1 can feasibly schedule all task sets that S2 can, but there exists at least one task set that S2 cannot feasibly schedule, whereas S1 can.

iii. If S1 can feasibly schedule all task sets that S2 can feasibly schedule and vice versa, then S1 and S2 are called equally proficient schedulers.

d. **Optimal scheduler :**

i. A real time task scheduler is called optimal, if it can feasibly schedule any task set that can be feasibly scheduled by any other scheduler.

ii. It is not possible to find a more proficient scheduling algorithm than an optimal scheduler.

iii. If an optimal scheduler cannot schedule some task set, then no other scheduler is able to produce a feasible schedule for that task set.

e. **Scheduling points :**

i. The scheduling points of a scheduler are the points on timeline at which the scheduler makes decisions regarding which task is to be run next.

ii. A task scheduler does not need to run continuously, it is activated by the operating system only at the scheduling points to make the scheduling decision as to which task to be run next.

f. **Preemptive scheduler :**

i. A preemptive scheduler is one which when a higher priority task arrives, suspends any lower priority task that may be executing and takes up the higher priority task for execution.

ii. Thus, in a preemptive scheduler, it cannot be the case that a higher priority task is ready and waiting for execution, and the lower priority task is executing.

g. **Utilization :**

i. The processor utilization of a task is the average time for which it executes per unit time interval.

ii. For a periodic task T_i, the utilization is $u_i = e_i/p_i$, where e_i is the execution time and p_i is the period of T_i. For a set of periodic

PART-1

Common Approaches to Real Time Scheduling : Clock Driven Approach, Round-Robin Approach, Priority-Driven Approach, Dynamic Versus Static System.

CONCEPT OUTLINE : PART-1

- **Real time task scheduling :** It refers to determining the order in which the various tasks are to be taken up for execution by the operating system.
- **Classification of scheduling algorithms :**

1. Clock driven :
   a. Table-driven
   b. Cyclic
2. Event driven :
   a. Simple priority-based
   b. Rate monotonic analysis (RMA)
   c. Earliest -Deadline-First (EDF)
3. Hybrid :
   a. Round-robin

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. **Explain task scheduling and its important concepts and terminologies.**

OR

Write the important terms used in real time task scheduling.

Answer

1. Real time task scheduling essentially refers to determining the order in which the various tasks are to be taken up for execution by the operating system.
2. Every operating system relies on one or more task scheduler to prepare the scheduling algorithm.
3. Following are few important concepts and terminologies :
   a. **Valid schedule :** A valid schedule for a set of tasks is one where at most one task is assigned to a processor at a time, no task is scheduled

before its arrival time and precedence and resource constraints of all tasks are satisfied.

b. **Feasible schedule :** A valid schedule is called a feasible schedule, only if all tasks meet their respective time constraints in the schedule.

c. **Proficient scheduler :**

i. A task scheduler S1 is said to be more proficient than another scheduler S2, if S1 can feasibly schedule all task sets that S2 can feasibly schedule, but not vice versa.

ii. That is, S1 can feasibly schedule all task sets that S2 can, but there exists at least one task set that S2 cannot feasibly schedule, whereas S1 can.

iii. If S1 can feasibly schedule all task sets that S2 can feasibly schedule and vice versa, then S1 and S2 are called equally proficient schedulers.

d. **Optimal scheduler :**

i. A real time task scheduler is called optimal, if it can feasibly schedule any task set that can be feasibly scheduled by any other scheduler.

ii. It is not possible to find a more proficient scheduling algorithm than an optimal scheduler.

iii. If an optimal scheduler cannot schedule some task set, then no other scheduler is able to produce a feasible schedule for that task set.

e. **Scheduling points :**

i. The scheduling points of a scheduler are the points on timeline at which the scheduler makes decisions regarding which task is to be run next.

ii. A task scheduler does not need to run continuously, it is activated by the operating system only at the scheduling points to make the scheduling decision as to which task to be run next.

f. **Preemptive scheduler :**

i. A preemptive scheduler is one which when a higher priority task arrives, suspends any lower priority task that may be executing and takes up the higher priority task for execution.

ii. Thus, in a preemptive scheduler, it cannot be the case that a higher priority task is ready and waiting for execution, and the lower priority task is executing.

g. **Utilization :**

i. The processor utilization of a task is the average time for which it executes per unit time interval.

ii. For a periodic task T_i, the utilization is $u_i = e_i/p_i$, where e_i is the execution time and p_i is the period of T_i. For a set of periodic

tasks $\{T_i\}$, the total utilization due to all tasks $U = \sum_{i=1}^{n} \frac{e_i}{p_i}$. The objective of any good scheduling algorithm is to feasibly schedule even those task sets that have very high utilization.

h. Jitter :

i. Jitter is the deviation of a periodic task from its strict periodic behaviour.

ii. The arrived time jitter is the deviation of the task from arriving at the precise periodic time of arrival.

iii. It may be caused by imprecise clocks, or other factors such as network congestions.

iv. Similarly, completion time jitter is the deviation of the completion of a task from precise periodic points.

Que 2.2. Explain different types of real time scheduling algorithms.

OR

Explain classification of real time task scheduling algorithms.

Answer

Classification of real time task scheduling algorithms :

1. Several schemes of classification of real time task scheduling algorithms exist.
2. A popular scheme classifies the real time task scheduling algorithms based on how the scheduling points are defined.
3. The three main types of schedulers according to the classification scheme are clock driven, event driven and hybrid.

a. Clock driven : The clock driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock.

i Table driven

ii. Cyclic

b. Event driven : In event driven, the scheduling points are defined by certain events which prevents clock interrupts to occur.

i. Simple priority-based

ii. Rate Monotonic Analysis (RMA)

iii. Earliest-Deadline-First (EDF)

c. Hybrid : The hybrid use both clock interrupts as well as event occurrences to define their scheduling points.

i. Round-robin

4. Event driven schedulers are more sophisticated than clock driven schedulers and usually more proficient and flexible than clock driven schedulers.
5. These are more proficient because they can feasibly schedule some task sets which clock driven schedulers cannot.
6. These are more flexible because they can feasibly schedule sporadic and aperiodic tasks in addition to periodic tasks, whereas clock driven schedulers can satisfactorily handle only periodic tasks.
7. Another classification of real time task scheduling algorithms can be made based upon the type of task acceptance test that a scheduler carries out before it takes up a task for scheduling.
8. The acceptance test is used to decide whether a newly arrived task would be taken up for scheduling or rejected.
9. Based on the task acceptance test used, there are two broad categories of task schedulers :
   - **a. Planning based :**
     - i. In planning based schedulers, when a task arrives, the scheduler first determines whether the task can meet its deadlines if it is taken up for execution.
     - ii. If not, it is rejected.
     - iii. If the task can meet its deadline and does not cause other already scheduled tasks to miss their respective deadlines, then the task is accepted for scheduling.
     - iv. Otherwise, it is rejected.
   - **b. Best effort :**
     - i. In best effort schedulers, no acceptance test is applied.
     - ii. All tasks that arrive are taken up for scheduling and best effort is made to meet its deadlines.
     - iii. But, no guarantee is given as to whether a task's deadline would be met or not.
10. A third type of classification of real time tasks is based on the target platform on which the tasks are to be run.
11. The different classes of algorithms according to this scheme are :
    - a. Uniprocessor
    - b. Multiprocessor
    - c. Distributed
12. Uniprocessor scheduling algorithms are simplest of the three classes of algorithms.
13. In contrast to uniprocessor algorithms, in multiprocessor and distributed scheduling algorithms first a decision has to be made regarding which task need to run on which processor and then these tasks are scheduled.

14. In contrast to multiprocessors, the processors in distributed system do not possess shared memory.
15. Also in contrast to multiprocessors, there is no global up-to-date information available in distributed systems.
16. This makes uniprocessor scheduling algorithms (that assume a central state information of all tasks and processors to exist) unsuitable for use in distributed systems.

Que 2.5. What do you mean by clock driven approach of job scheduling ? Also, explain the basic features of clock driven schedulers.

Answer

1. When scheduling is clock driven, decisions on what jobs execute at what times are made at specific time instants.
2. These instants are chosen a priori before the system begins execution.
3. Typically in a system that uses clock driven scheduling, all the parameters of hard real time jobs are fixed and known.
4. A schedule of the jobs is computed offline and is stored for use at runtime.
5. The scheduler schedules the jobs according to this schedule at each scheduling decision time.
6. In this way, scheduling overhead during runtime can be mimimized.
7. A frequently adopted choice is to make scheduling decisions at regularly spaced instants.
8. One way to implement a scheduler that makes scheduling decisions periodically is to use a hardware timer.
9. The timer is set to expire periodically without the intervention of the scheduler.
10. When the system is initialized, the scheduler selects and schedules the jobs that will execute until the next scheduling decision time and then blocks itself waiting for the expiration of the timer.
11. When the timer expires, the scheduler awakes and repeats these actions.

Following are the basic features of two important clock driven schedulers :

1. **Table driven scheduling :**
   a. Table driven schedulers usually precompute which task would run when and store this schedule in a table at the time the system is designed or configured.
   b. Rather than automatic computation of the schedule by the scheduler, schedule for the set of tasks in the application can be

designed and stored in a table (called schedule table) to be used by the scheduler at run time which is shown in Table 2.3.1.

Table 2.3.1

Task	Start time in milliseconds
T_1	0
T_2	3
T_3	11
T_4	13
T_5	18

c. Table 2.3.1 shows that task T_1 would be taken up for execution at time instance 0, T_2 would start execution 3 msec afterwards and so on.

2. **Cyclic schedulers :**
   a. A cyclic scheduler repeats a precomputed schedule.
   b. The precomputed schedule needs to be stored only for one major cycle.
   c. Each task in the task set to be scheduled repeats identically in every major cycle.
   d. The major cycle is divided into one or more minor cycles.
   e. Each minor cycle is also called a frame.
   f. Cyclic schedulers are very popular and are being extensively used in the industry.
   g. A large majority of all small embedded applications are based on cyclic schedulers that are simple, efficient, and are easy to program.
   h. For example, a temperature controller which is used as cyclic scheduler, periodically sample the temperature of a room and maintains it at a preset value.

Que 2.4. **Explain scheduling point of a task scheduling algorithm. How the scheduling points are determined in (i) clock driven system (ii) event driven system ?** **UPTU 2014-15, Marks 05**

Answer

Scheduling points :

1. The scheduling points of a scheduler are the points on timeline at which the scheduler makes decisions regarding which task is to be run next.
2. It is important to note that a task scheduler does not need to run continuously, it is activated by the operating system only at the

scheduling points to make the scheduling decision as to which task to be run next.

3. In a clock driven scheduler, the scheduling points are defined at the time instants marked by interrupts generated by a periodic timer.
4. The scheduling points in an event driven scheduler are determined by occurrence of certain events.

Que 2.5. Explain weighted round-robin approach for job scheduling.

OR

"Weighted round-robin scheduling is special form of round-robin scheduling". Do you agree with this statement ? Justify your answer. **UPTU 2012-13, Marks 05**

Answer

1. The round-robin approach is commonly used for scheduling time-shared applications.
2. When jobs are scheduled on a round-robin basis, every job joins a First-In-First-Out (FIFO) queue when it becomes ready for execution.
3. The job at the head of the queue executes for at most one time slice (a time slice is typically in the order of tens of milliseconds).
4. If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue to wait for its next turn.
5. When there are n ready jobs in the queue, each job gets one time slice in every m time slice, *i.e.*, every round.
6. Because the length of the time slice is relatively short, the execution of every job begins almost immediately after it becomes ready.

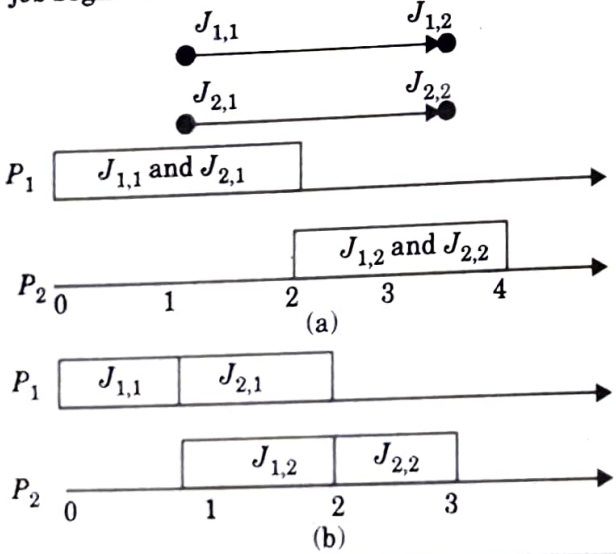


Fig. 2.5.1. Round-robin scheduling of precedence-constrained job.

7. Each job gets $1/n^{th}$ share of the processor when there are n jobs ready for execution. This is why, the round robin algorithm is also called the processor sharing algorithm.
8. The weighted round-robin algorithm has been used for scheduling real time traffic in high speed switched networks.
9. 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.
10. The weight of a job refers to the fraction of processor time allocated to the job.
11. Specifically, a job with weight ωt gets ωt time slice every round, and the length of a round is equal to the sum of the weights of all the ready jobs.
12. By adjusting the weight of jobs, we can speed up or retard the progress of each job toward its completion.
13. By giving each job a fraction of the processor, round-robin scheduler delay the completion of every job.
14. If it is used to schedule precedence constrained jobs, the response time of a chain of jobs can be unduly large. For this reason, the weighted round-robin approach is not suitable for scheduling such jobs.
15. On the other hand, a successor job may be able to incrementally consume what is produced by a predecessor.
16. In this case, weighted round-robin scheduling is a reasonable approach, since a job and its successors can execute concurrently in a pipelined fashion.
17. For example, we consider the two sets of jobs, $J_1 = \{J_{1,1}, J_{1,2}\}$ and $J_2 = \{J_{2,1}, J_{2,2}\}$ as shown in Fig. 2.5.1. The release times of all jobs are 0 and their execution times are 1.
18. $J_{1,1}$ and $J_{2,1}$ execute on processor P_1 and $J_{1,2}$ and $J_{2,2}$ execute on processor P_2. Suppose that $J_{1,1}$ is the predecessor of $J_{1,2}$ and $J_{2,1}$ is the predecessor of $J_{2,2}$.
19. Fig. 2.5.1(a) shows that both sets of jobs complete approximately at time 4 if the jobs are scheduled in a weighted round-robin manner.
20. In contrast, the schedule in Fig. 2.5.1(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.
21. 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.
22. 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.

Que 2.6. **What do you mean by priority-driven scheduling of jobs ?**

Answer

1. The term priority-driven algorithms refer to a large class of scheduling algorithms that never leave any resource idle intentionally.
2. A resource idles only when no job requiring the resource is ready for execution.
3. Scheduling decisions are made when events such as releases and completions of jobs occur. Hence, priority-driven algorithms are event driven.
4. Other commonly used names for this approach are greedy scheduling, list scheduling and work-conserving scheduling. A priority-driven algorithm is greedy because it tries to make locally optimal decisions.
5. Leaving a resource idle while some job is ready to use the resource is not locally optimal.
6. So, when a processor or resource is available and some job can use it to make progress, such an algorithm never makes the job wait.
7. Sometimes, it is better to have some jobs wait even when they are ready to execute and the resources they require are available.
8. The term list scheduling is also descriptive because any priority-driven algorithm can be implemented by assigning priorities of jobs.
9. Jobs ready for execution are placed in one or more queues ordered by the priorities of the jobs.
10. At any scheduling decision time, the jobs with the highest priorities are scheduled and executed on the available processors.
11. Hence, a priority-driven scheduling algorithm is defined to a great extent by the list of priorities it assigns to jobs, the priority list and other rules, such as whether preemption is allowed, define the scheduling algorithm completely.
12. Round-robin scheduling can be thought of as priority-driven : the priority of the executing job is lowered to the minimum among all jobs waiting for execution after the job has executed for a time slice.
13. Other examples of priority driven are : FIFO and LIFO, SETF (Shortest-Execution-Time-First) and LETF (Longest-Execution-Time-First) algorithms.
14. The task graph as shown in Fig. 2.6.1, is a classical precedence graph, all its edges represent precedence constraints. The number next to the name of each job is its execution time.
15. J_5 is released at time 4. All the other jobs are released at time 0. We want to schedule and execute the jobs on two processors P_1 and P_2.
16. They communicate via a shared memory. The schedulers of the processors keep one common priority queue of ready jobs.

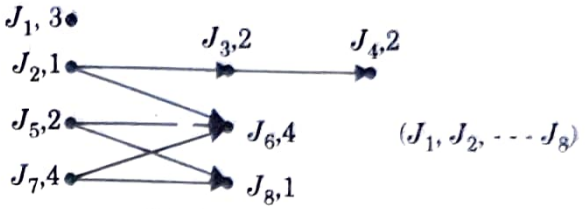


Fig. 2.6.1. Task graph.

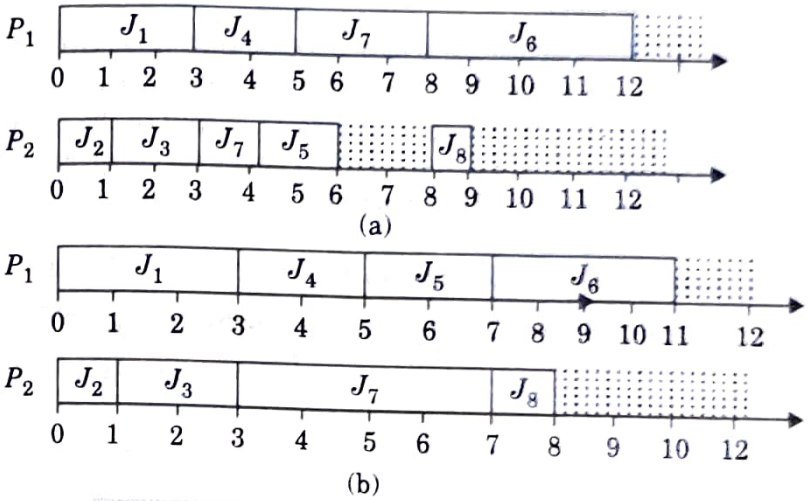


Fig. 2.6.2. Example of priority-driven scheduling (a) preemptive (b) non-preemptive.

17. **The priority list is given next to the graph :** J_i has a higher priority than J_k if $i < k$. All the jobs are preemptable, scheduling decisions are made whenever some job becomes ready for execution or some job completes.

18. Fig. 2.6.2(a) shows the schedule of the jobs on the two processors generated by the priority-driven algorithm following this priority assignment. At time 0, jobs J_1, J_2 and J_7 are ready for execution.

19. Since J_1 and J_2 have higher priorities than J_7, they are ahead of J_7 in the queue and hence are scheduled. Processor makes new scheduling decisions when the following events occur :

    a. At time 1, J_2 completes and, hence, J_3 becomes ready. J_3 is placed in the priority queue ahead of J_7 and is scheduled on P_2, the processor freed by J_2.

    b. At time 3, both J_1 and J_3 complete. J_5 is still not released. J_4 and J_7 are scheduled.

    c. At time 4, J_5 is released. Now, there are three ready jobs. J_7 has the lowest priority among them. Consequently, it is preempted. J_4 and J_5 have the processors.

    d. At time 5, J_4 completes. J_7 resumes on processor P_1.

e. At time 6, J_5 completes. Because J_7 is not yet completed, both J_6 and J_8 are not ready for execution. Consequently, processor P_2 becomes idle.

f. J_7 finally completes at time 8. J_6 and J_8 can now be scheduled and they are.

20. Fig. 2.6.2(b) shows a nonpreemptive schedule according to the same priority assignment. Before time 4, this schedule is the same as the preemptive schedule.

21. However, at time 4 when J_5 is released, both processors are busy. It has to wait until J_4 completes before it can begin execution.

22. It turns out that for this system this postponement of the higher priority job benefits the set of jobs as a whole. The entire set completes 1 unit of time earlier according to the nonpreemptive schedule.

Que 2.7. **Define task and explain different types of task classes. State the issues involved in the allocation scheduling problem.**

UPTU 2013-14, Marks 05

Answer

Task :

1. A task is a basic unit of programming that an operating system controls.
2. Depending on how the operating system defines a task in its design, this unit of programming may be an entire program or each successive invocation of a program.
3. Since one program may make requests of other utility programs, the utility programs may also be considered tasks (or subtasks).
4. All of today's widely used operating systems support multitasking, which allows multiple tasks to run concurrently, taking turns using the resources of the computer.

Different types task classes :

1. A task may be periodic, sporadic, or aperiodic.
2. A task T_i is periodic if it is released periodically, say every P_i seconds. P_i is called the period of task T_i.
3. The periodicity constraint requires the task to run exactly once every period; it does not require that the tasks be run exactly one period apart.
4. Quite commonly, the period of a task is also its deadline.
5. The task is sporadic if it is not periodic, but may be involved at irregular intervals.
6. Sporadic tasks are characterized by an upper bound on the rate at which they may be invoked.

7. This is commonly specified by requiring that successive invocations of a sporadic task T_i be separated in time by at least $t(i)$ seconds. Sporadic tasks are sometimes also called aperiodic.
8. However, some people define aperiodic tasks to be those tasks which are not periodic and which also have no upper bound on their invocation rate.

Following are the issues involved in the allocation scheduling problem :

1. An inability to direct map timing or importance constraints into priority values.
2. The problem of dealing with tasks whose execution time is either unknown or may vary over time.
3. The problem of dealing with tasks whose execution time or execution rates deviates significantly at runtime from the behavior expected at design time.
4. The problem of degrading performance gracefully in times of overload.
5. The problem of ensuring full utilization of the processor or other system resources in tightly resource constrained systems.
6. Issues involved in allocation of time slots, constraints and optimization.

Que 2.8. Differentiate between static and dynamic systems.

Answer

1. Jobs that are ready for execution are placed in a priority queue common to all processors.
2. When a processor is available, the job at the head of the queue executes on the processor.
3. Such a multiprocessor system is a dynamic system because jobs are dynamically dispatched to processors. In Fig. 2.6.1, we allowed each preempted job to resume on any processor and hence, jobs are migratable.
4. The job migrates if it starts execution on a processor, preempted, and later resumes on a different processor.
5. Another approach to scheduling in multiprocessor and distributed systems is to partition the jobs in the system into subsystems and assign and bind the subsystems statically to the processors.
6. Jobs are moved among processors only when the system must be reconfigured *i.e.*, when the operation mode of the system changes or some processor fails.
7. Such a system is called a static system, because the system is statically configured.

8. If jobs on different processors are dependent, the schedulers on the processors must synchronize the jobs according to some synchronization and resource access control protocol.
9. Except for the constraints thus imposed, the jobs on each processor are scheduled by themselves.
10. The response of the static system is just as good as that of the dynamic system.
11. While dynamic system may be more responsive on the average, their worst case real time performance may be poorer than static system.
12. We can get better average responses by dynamically dispatching and executing jobs.
13. There is no reliable technique to validate the timing constraints of dynamic systems while such techniques exist for static systems.
14. Hence, most hard real time systems are static.

PART-2

Optimality of Effective-Deadline-First (EDF) and Least-Slack-Time-First (LST) Algorithms, Rate Monotonic Algorithms.

CONCEPT OUTLINE : PART-2

- Priority-driven algorithms are divided into :
  1. Fixed-priority algorithm
  2. Dynamic priority algorithm
- Examples of fixed priority algorithms :
  1. Rate monotonic (RM) algorithm
  2. Deadline monotonic (DM) algorithm
- Examples of dynamic priority algorithms :
  1. Earliest- Deadline- First (EDF) algorithm
  2. Least- Slack- Time (LST) algorithm
- **Rate Monotonic Algorithm (RMA) :** It assigns priorities to tasks based on their periods *i.e.*, the shorter the period , the higher the priority.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.9. **Explain the preemptive earliest-deadline-first (EDF) scheduling. Also, write the drawbacks of EDF.**

Answer

Preemptive Earliest-Deadline-First (EDF) Algorithm :

1. A processor following the EDF algorithm always executes the task whose absolute deadline is the earliest.
2. EDF is a dynamic-priority scheduling algorithm; the task priorities are not fixed but change depending on the closeness of their absolute deadline.
3. EDF is also called the deadline-monotonic scheduling algorithm.
4. In treatment of the EDF algorithm, we will make all the assumptions we made for the RM algorithm, except that the tasks do not have to be periodic.
5. EDF is an optimal uniprocessor scheduling algorithm.
6. That is, if EDF cannot feasibly schedule a task set on a uniprocessor, there is no other scheduling algorithm that can do so.
7. If all the tasks are periodic and have relative deadlines equal to their periods, the test for task-set schedulability is particularly simple.
8. If the total utilization of the task set is no greater than 1, the task set can be feasibly scheduled on a single processor by the EDF algorithm.
9. There is no simple schedulability test corresponding to the case where the relative deadlines do not all equal the periods; in such a case, we actually have to develop a schedule using the EDF algorithm to see if all deadlines are met over a given interval of time.
10. The following is a schedulability test for EDF under this case.
11. Define $u = \sum_{i=1}^{n} (e_i / P_i)$, $d_{max} = \max_{1 \le i \le n} \{d_i\}$ and $P = \text{lcm}\,(P_1, \ldots P_n)$. (Here "lcm" stands for least common multiple.) Define $h_T(t)$ to be the sum of the execution times of all tasks in set T whose absolute deadlines are less than t. A task set of n tasks is not EDF- feasible iff,
    a. $u > 1$ or
    b. there exists

$$t < \min \left\{ P + d_{max}, \frac{u}{1-u} \max_{1 \le i \le n} (P_i - d_i) \right\}$$

    such that $h_T(t) > t$.
12. EDF is an optimal uniprocessor scheduling algorithm (*i.e.*, if a set of tasks cannot be feasibly scheduled under EDF), there is no other uniprocessor algorithm that can feasibly schedule them.

Drawbacks of EDF : Some of the important shortcomings of EDF when used for scheduling real time tasks in practical applications are :

1. **Transient overload problem :**
   a. Transient overload denotes the overload of a system for a very short time. Transient overload occurs when some task takes more time to complete than what was originally planned during the design time.
   b. For example, it might enter an infinite loop or encounter an unusual condition and enter a rarely used branch due to some abnormal input values.
   c. When EDF is used to schedule a set of periodic real time tasks, a task overshooting its completion time can cause some other task(s) to miss their deadlines.
   d. Even the most critical task might miss its deadline due to a very low priority task overshooting its planned completion time.
   e. So, it should be clear that under EDF any amount of careful design will not guarantee that the most critical task would not miss its deadline under transient overload. This is a serious drawback of the EDF scheduling algorithm.
2. **Resource sharing problem :** When EDF is used to schedule a set of real time tasks, unacceptably high overheads might have to be incurred to support resource sharing among the tasks without making tasks to miss their respective deadlines.
3. **Efficient implementation problem :**
   a. The efficient implementation is often not feasible as it is difficult to restrict the number of tasks with distinct deadlines to a reasonable number.
   b. The efficient implementation that achieves O(1) overhead assumes that the number of relative deadlines is restricted. This may be unacceptable in some situations.

Que 2.10. **Can we consider EDF (Earliest-Deadline-First) as a dynamic priority scheduling algorithm for real time task ? Justify your answer.** **UPTU 2014-15, Marks 05**

Answer

1. No, we cannot consider EDF as a dynamic priority scheduling algorithm.
2. If EDF were to be consider a dynamic priority-scheduling, we should be able determine the precise priority value of a task at any point of time and also how it change with time.
3. EDF scheduling does not require any priority value for a task.
4. In fact, EDF has no notion of a priority value of a task.
5. Tasks are scheduled solely on the proximity to their deadline.
6. However, the longer a task waits in a ready queue, the higher is the chance (probability) of being taken up for scheduling.

7. So, we can imagine that a virtual priority value associated with a task keeps increasing with time until the task is taken up for scheduling.
8. In EDF the tasks neither have any priority value associated with them, nor does the scheduler perform any priority computations to determine the schedulability of a task at either runtime or compile time.

Que 2.11. Explain the Least-Slack-Time-First (LST) algorithm. Also, explain its drawbacks and non optimality.

Answer

1. Another algorithm that is optimal for scheduling preemptive jobs on one processor is the Least-Slack-Time-First (LST) algorithm (also called the Minimum-Laxity-First (MLF) algorithm).
2. At any time t, the slack (or laxity) of a job with deadline at d is equal to $d - t$ minus the time required to complete the remaining portion of the job.
3. For example, take the job J_1 as shown in Fig. 2.11.1. It is released at time 0, its deadline is 6, and its execution time is 3. Hence, its slack is equal to 3 at time 0. The job starts to execute at time 0.
4. As long as it executes, its slack remains at 3, because at any time t before its completion, its slack is $6 - t - (3 - t)$.
5. Now suppose that it is preempted at time 2 by J_3, which executes from time 2 to 4.
6. During this interval, the slack of J_1 decreases from 3 to 1 (At time 4, the remaining execution time of J_1 is 1, so its slack is $6 - 4 - 1 = 1$).

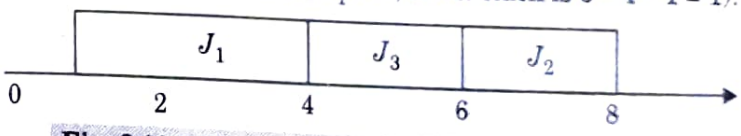


Fig. 2.11.1. Example illustrating the LST algorithm.

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

Drawbacks of LST algorithm :

1. While the EDF algorithm does not require any knowledge of the execution times of jobs, the LST algorithm does. This is a serious disadvantage.
2. The actual execution times of jobs are often not known until the jobs complete.
3. Obviously, it is impossible for us to calculate the actual amounts of slack under this circumstance.
4. We typically calculate the slack of each job based on its maximum execution time e_i^+ when the range $[e_i^-, e_i^+]$ of execution time e_i of every job is relatively small.
5. Furthermore, we require that the maximum (and sometimes even the actual) execution time of each sporadic or aperiodic job become known upon its arrival since this knowledge is needed for slack computation.

Non-optimality of LST :

1. Consider three independent non-preemptable jobs, J_1, J_2 and J_3 with release times 0, 2 and 4, indicated by the arrows above the schedules.
2. The execution times are 3, 6 and 4 and deadlines are 10, 14 and 12 respectively.

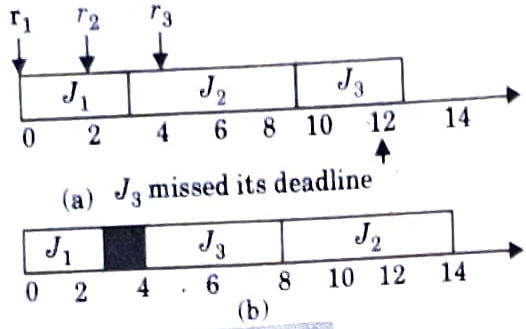


Fig. 2.11.2.

3. LST would produce the infeasible schedule as shown in Fig. 2.11.2(a) whereas a feasible schedule is possible as shown in Fig. 2.11.2(b).
4. Now, let us consider that we have two processors and three jobs J_1, J_2 and J_3 with execution times 1, 1 and 5 and deadlines 1, 2 and 5 respectively. All with release time 0.

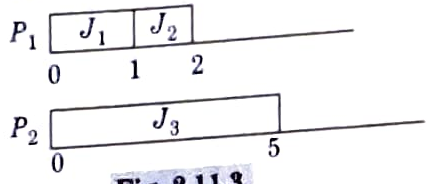


Fig. 2.11.3.

5. LST gives a feasible schedule as shown in Fig. 2.11.3. But in general, LST is non-optimal for multiprocessors.

Que 2.12. Prove that when all the jobs have the same release time or when they have same deadline the LST algorithm is optimal.

UPTU 2012-13, Marks 05

Answer

1. We will use the "Exchange argument" to show that the algorithm is Optimal. Let A be a schedule generated by the earliest deadline first algorithm and let V be the optimal schedule. We will modify the solution V to match it to A while not increasing the maximum lateness. It will thus prove that A = V *i.e.* A is the optimal solution.
2. We can define "Idle Time" in a schedule as the time when no jobs are scheduled. We can observe that there can be an optimal schedule which doesn't have any Idle Time. We can easily convert a schedule having some idle time into one which doesn't have any idle time. This factor will only reduce the maximum lateness.

a. Given a schedule S, call a pair of jobs (i, j) as an inversion, if d(i) > d(j), but s(i) < s(j). In other words, job i is scheduled earlier than job j, but has a later deadline.

b. The schedule produced by our algorithm has no inversions.

c. Several jobs may have the same deadline.

d. All such jobs can be scheduled in any manner without inversions.

3. Now let's show how all schedules that have no inversions has the same maximum lateness. Consider jobs with same deadline say d. Since the jobs have no inversion then we can schedule them consecutively. The last scheduled job among them will have the highest lateness. However, the maximum lateness doesn't depend only upon the last job. All jobs having the same deadline together contribute to a maximum lateness.
4. Now, that we have shown the above, we can say that there is an optimal solution with no inversion and no idle time.
5. We will now modify O carefully to O′ so that the schedule has
   a. No inversions
   b. The maximum lateness of O′ is at most the maximum lateness of O.
6. Three steps in the proof.
   a. If O has an inversion, then there is a pair of jobs i and j that are scheduled consecutively, and dj < di.
   b. After swapping the schedules of i and j in O, the resulting schedule has one less inversion, and
   c. The new schedule (from b) has a maximum lateness that is at most that of O.
7. Now, we know that the solution produced by our algorithm has no inversions and no idle time. There is an optimal solution with no inversions and no idle time.
8. All solutions with no inversions and no idle time have the same maximum lateness. Therefore, our solution has optimal maximum lateness.

Que 2.13. What is rate monotonic (RM) scheduling algorithm ? If there are two tasks, T_1 and T_2 and $\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq 2(\sqrt{2}-1)$ then show that the tasks are RM- schedulable.

Answer

1. A well known fixed priority algorithm is rate monotonic algorithm.
2. This algorithm assigns priorities to tasks based on their periods, the shorter the period, the higher is the priority.
3. The rate (of job releases) of a task is the inverse of its period.
4. Hence, the higher its rate, the higher its priority.
5. The schedule produced by the algorithm is referred to as an RM schedule.

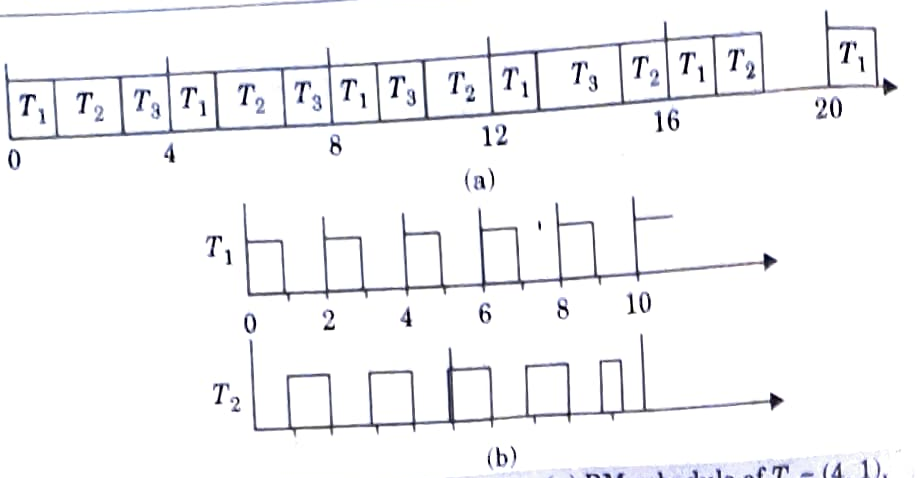


Fig. 2.13.1. Examples of RM schedules. (a) RM schedule of $T_1 = (4, 1)$, $T_2 = (5, 2)$, and $T_3 = (20, 5)$ (b) RM schedule of $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$.

6. For example, Fig. 2.13.1(a) shows the RM schedule of the system.
7. This system contains three tasks : $T_1 = (4, 1)$, $T_2 = (5, 2)$ and $T_3 = (20, 5)$.
8. The priority of T_1 is highest because its rate is highest (or its period is the shortest).
9. Each job in this task is placed at the head of the priority queue and is executed as soon as the job is released.
10. T_2 has the next highest priority. Its jobs execute in the background of T_1.
11. For this reason, the execution of the first job in T_2 is delayed until the first job in T_1 completes and the fourth job in T_2 is preempted at time 16 when the fifth job in T_1 is released.
12. Similarly, T_3 executes in the background of T_1 and T_2, the jobs in T_3 execute only when there is no job in the higher priority tasks ready for execution.
13. Since, there is always at least one job ready for execution until time 18, the processor never become idle until that time.
14. The schedule as shown in Fig. 2.13.1(b) is for the task $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$. The tasks are in phase.
15. Here, the schedule is represented in a different form.
16. Instead of using one timeline, called a Gantt chart, to represent a schedule on a processor, timeline for each task is used.
17. Each timeline is labeled at the left by the name of a task, the timeline shows the intervals during which the task executes.
18. According to the RM algorithm, task T_1 has a higher priority than task T_2.
19. Consequently, every job in T_1 is scheduled and executed as soon as it is released. The jobs in T_2 are executed in the background of T_1.

Schedulability test for RMA : It can be determined from a knowledge of the worst case execution times and periods of the tasks. The worst case execution times are usually determined experimentally or through simulation studies.

Following are the two important criteria for testing the schedulability :

1. **Necessary condition :** A set of periodic real time tasks would be RMA schedulable if they satisfy the following condition :

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum_{i=1}^{n} u_i \leq 1$$

where e_i is the worst case execution time and p_i is the period of the task T_i, n is the number of tasks to be scheduled, and u_i is the utilization due to all tasks in the task set.

2. **Sufficient condition :**

   a. This condition was obtained by Liu and Layland in 1973.

   b. A set of n real time periodic tasks are schedulable under RMA, if

$$\sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$$

   where u_i is the utilization due to task T_i.

   c. If the set of tasks satisfies the sufficient condition, then the set of tasks would be RMA schedulable.

   d. There are two tasks T_1 and T_2, e_1 and p_1 are the worst case execution time and period of the task T_1.

   e. Similarly, e_2 and p_2 are the worst case execution time and period of the task T_2.

   f. Let us compute the total utilization due to given two tasks.

$$\sum_{i=1}^{2} \frac{e_i}{p_i} = \frac{e_1}{p_1} + \frac{e_2}{p_2}$$

   Now given, $\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq 2(\sqrt{2} - 1)$

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq 2(1.414 - 1)$$

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq 0.824$$

   which satisfies both the necessary and sufficient conditions.

   g. Its utilization of tasks is already less than 0.824. Hence, it is less than 1 which satisfies the necessary condition and it satisfies the sufficient condition because it is given that utilization factor is less than the Liu and Layland's condition *i.e.*,

$$\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq n(2^{1/n} - 1)$$

where $n = 2$, because there are two tasks.

Que 2.14. **A system contains the following four periodic tasks. These tasks are scheduled by the rate monotonic algorithm.**

$$T_1 = (e_1 = 0.75, p_1 = 3)$$
$$T_2 = (e_2 = 1.5, p_2 = 3.5)$$
$$T_3 = (e_3 = 0.6, p_3 = 6)$$
$$T_4 = (e_4 = 1, p_4 = 10)$$

Are the tasks schedulable ?

Answer

Let us first compute the total CPU utilization due to the task set :

$$\sum_{i=1}^{4} u_i = \frac{0.75}{3} + \frac{1.5}{3.5} + \frac{0.6}{6} + \frac{1}{10}$$
$$= 0.25 + 0.42 + 0.1 + 0.1 = 0.878.$$

This is less than 1, therefore the necessary condition for schedulability of the tasks is satisfied.

Now, checking for the sufficient condition, the maximum achievable utilization is given by :

$$n(2^{1/n} - 1) = 4(2^{1/4} - 1)$$
$$= 4(1.189 - 1) = 4(0.189)$$
$$= 0.756.$$

$$\sum_{i=1}^{4} u_i \not\leq 0.756 \text{ ; since } 0.878 \not\leq 0.756,$$

Hence, the tasks are not RMA schedulable.

Que 2.15. **When DM algorithm fails RM always fails and when DM finds a feasible schedule then sometimes RMA fails. Explain this with example.** **UPTU 2012-13, Marks 05**

Answer

Let us assume $T_1 = (e_1 = 10$ mSec, $p_1 = 50$ mSec, $d_1 = 35$ mSec), $T_2 = (e_2 = 15$ mSec, $p_2 = 100$ mSec, $d_2 = 20$ mSec), $T_3 = (e_3 = 20$ mSec, $p_3 = 200$ mSec, $d_3 = 200$ mSec).

1. First, let us check RMA schedulability of the given set of tasks, by checking the Lehoczkey's criterion :
   a. The tasks are already ordered in descending order of their priorities.
   b. Checking for T_1 : 10 mSec < 35 mSec. Therefore, T_1 would meet its first deadline.

c. Checking for T_2 : 10 + 15 $\nleq$ 20. Therefore, T_2 will miss its first deadline.

d. Hence, the given task set cannot be feasibly scheduled under RMA.

2. Now let us check the schedulability using DMA :

a. Under DMA, the priority ordering of the tasks is as follows : $Pr\ (T_2) > Pr\ (T_1) > Pr\ (T_3)$

b. Checking for T_2 : 10 mSec < 35 mSec. Hence, T_2 will meet its first deadline.

c. Checking for T_1 : (15 + 20) mSec ≤ 20 mSec. Hence, T_1 will meet its first deadline.

d. Checking for T_3 : (70 + 30 + 20) mSec < 200 mSec. Therefore, T_3 will meet its deadline.

Therefore, the given task set is schedulable under DMA but not under RMA.

Que 2.16. **What are the differences between fixed priority and dynamic priority scheduling approach ? Explain which one is more suitable for periodic tasks.** **UPTU 2012-13, Marks 05**

OR

Differentiate between fixed and dynamic priority driven preemptive scheduling.

Answer

S. No.	Fixed priority driven scheduling algorithm	Dynamic priority driven scheduling algorithm
1.	**Optimality :** Scheduling algorithm x is said to be optimal in the sense that no other fixed priority assignment can lead to a valid schedule which cannot be obtained by x.	At any time, EDF executes among those tasks that have been released and not yet fully serviced (pending tasks), one whose absolute deadline is earliest. If no task is pending, the processor is idle.
2.	**Feasibility condition** For a given synchronous periodic task set, the RM schedule is feasible if $u \leq n\ (2^{1/n} - 1)$.	For a given synchronous periodic task set, the EDF schedule is feasible if and only if $u \leq 1$.

3.	**Worst case response time** The worst case response time r_i of a task T_i of non-concrete periodic or sporadic task set is found in a scenario in which all tasks are at their maximum rate and released synchronously at a critical instant $t = 0$.	The worst case response time of a task T_i is found in a deadline busy period for T_i in which all tasks but T_i are released synchronously from the beginning of the deadline busy period and at their maximum rate.
4.	**Shared resources and release jitter** Shared resources and release jitter in presence of fixed priority leads to the same reasoning than in presence of dynamic driven scheduling.	For reasons such as tick scheduling or distributed contexts for fixed priority scheduling that were extended for dynamic priority scheduling, tasks may be allowed to have a release jitter (*i.e.*, a task T_i may be delayed for a maximum time J_i before being actually released).

Que 2.17. **What do you mean by static scheduling and dynamic scheduling ? Explain with examples. Give the advantages and disadvantages of static and dynamic scheduling. List different types of multiprocessor and uniprocessor scheduling algorithms.**

UPTU 2013-14, Marks 05

Answer

Static and Dynamic scheduling :

1. Hard real time scheduling can be broadly classified into two types : static and dynamic.
2. In static scheduling, the scheduling decisions are made at compile time.
3. A runtime schedule is generated offline based on the prior knowledge of task-set parameters, for example, maximum execution times, precedence constraints, mutual exclusion constraints, and deadlines. So runtime overhead is small.
4. On the other hand, dynamic scheduling makes its scheduling decisions at runtime, selecting one out of the current set of ready tasks.
5. Dynamic schedulers are flexible and adaptive.
6. But they can incur significant overheads because of runtime processing.
7. Preemptive or non-preemptive scheduling of tasks is possible with static and dynamic scheduling.

8. In preemptive scheduling, the currently executing task will be preempted upon arrival of a higher priority task.
9. In non-preemptive scheduling, the currently executing task will not be preempted until completion.

Advantages of dynamic scheduling :

1. **High flexibility :** Schedule can easily adapt to changes in the system, for example, new tasks can be added dynamically.
2. **External events are handled efficiently :** I/O units handled via interrupt which activates a task efficient for different types of tasks :
   a. Sporadic tasks can be easily supported (via suitable priority assignment).
   b. Scheduling algorithms are often optimal.

Disadvantages of dynamic scheduling :

1. Complicates communication between tasks :
   a. Exact time of data availability is not known in advance, which requires extra synchronization between tasks.
   b. Task execution is difficult to adapt to existing time-slot-based network protocols (but works very well with many priority-based network protocols, for example, CAN and Token Ring).
2. Task execution becomes indeterministic : Temporary deviations ("jitter") in task periodicity may occur.
3. Exact feasibility tests often have high time complexity : Low observability (difficult to debug).

Advantages of static scheduling :

Static scheduling has the significant advantage that the order of execution of tasks is determined offline (before the execution of the program), so the runtime scheduling overheads can be very small.

Disadvantages of static scheduling :

Static scheduling has drawbacks when applied to real-world systems. They require significant a priori knowledge of the system tasks, both execution times and ordering. Therefore, they are quite rigid and inflexible.

Types of multiprocessor scheduling algorithms :

1. Utilization balancing algorithm
2. Next-fit algorithm
3. Bin-packing algorithm
4. Myopic offline scheduling algorithm
5. Buddy strategy
6. Assignment with precedence constraints.

Types of uniprocessor scheduling algorithms :

1. Traditional rate monotonic (RM) algorithm
2. Rate monotonic deferred server (DS) algorithm
3. Earliest deadline first (EDF) algorithm
4. IRIS tasks algorithm

Que 2.18. Define IRIS tasks. Give a scheduling algorithm for tasks with identical linear reward functions. **UPTU 2013-14, Marks 05**

Answer

IRIS tasks :

1. We have assumed that to obtain an acceptable output, a task has to be run to completion.
2. If the task is not run to completion, we get zero reward from it (*i.e.*, it may as well not have been run). However, there are large number of tasks for which this is not true.
3. These are iterative algorithms. The longer they run, the higher the quality of their output (upto some maximum runtime).
4. Search algorithms for finding the minimum of some complicated function are also examples of iterative tasks.
5. The longer we search the parameter space, the greater is the chance that we will obtain the optimum value or something close to it.
6. Tasks of this type are known as increased reward with increased service (IRIS) tasks.
7. The reward function associated with an IRIS task increases with the amount of service given to it.
8. Typically, the reward function is of the form :

$$R(x) = \begin{cases} 0 & \text{if } x < m \\ r(x) & \text{if } m \le x \le 0 + m \\ r(0 + m) & \text{if } x > 0 + m \end{cases}$$

   where $r(x)$ is monotonically non decreasing in x.
9. The reward is 0 upto some time m; if the task is not executed upto that point, it produces no useful output.
10. Tasks with this reward function can be regarded as having a mandatory and an optional component.

11. The mandatory portion (with execution time m) must be completed by the deadline if the task is critical; the optional portion is done if time permits.

12. The optional portion requires a total of o time to complete. In each case, the execution of a task must be stopped by its deadline d.

Scheduling algorithm :

1. Run the EDF algorithm on the task set T to generate a schedule S_t. If this is feasible,

   An optimal schedule has been found : STOP.

   Else,

   go to step 2.

   end if

2. Run the EDF algorithm on the task set M, to generate a schedule S_m. If this set is not feasible,

   T cannot be feasibly scheduled : STOP.

   Else,

   Define a_i as the i^{th} instant in S_m when either the scheduled task changes, or the processor becomes idle, $i = 1, 2, \ldots$.

   Let k be the total number of these instants.

   Define a_0 as when the first task that begins executing in S_m.

   Define $\tau(j)$ as the task that executes in S_m in $[a_j, a_{j+1}]$,

   Define $L_t(j)$ and $L_m(j)$ as the total execution time given to task $\tau(j)$ in $S_t(j)$ and $S_m(j)$ respectively, after time a_j.

   go to step 3.

   end if

3. $j = k - 1$

   do while $(0 \le j \le k - 1)$

   if $(L_m(j) > L_t(j))$ then

   Modify S_t by

   (a) assigning $L_m(j) - L_t(j)$ of processor time in $[a_j, a_{j+1}]$ to $\tau(j)$, and

   (b) reducing the processor time assigned to other tasks in $[a_j, a_{j+1}]$ by $L_m(j) - L_t(j)$.

   Update $L_t(1), \ldots, L_t(j)$ appropriately.

end if

$j = j - 1$

end do

end

Que 2.19. Discuss fixed priority assignment algorithm used for scheduling of tasks in real time systems.

Answer

1. Let us consider the additional time demand in fixed priority system by using the following modified task parameters in the computation of the time demand function of task T_i.

2. Following are the important rules used for scheduling of tasks in real time system :

   Rule 1 : Include the task $T_o = (p_o, e_o)$ in the set of higher priority tasks.

   Rule 2 : Add $(K_k + 1) CS_o$ to the execution time e_k of every higher priority tasks T_k (*i.e.*, for $k = 1, 2 \ldots, i$) where K_k is the number of times it may self-suspend.

   Rule 3 : For every lower priority task T_k, $k = i + 1, \ldots, n$, add a task (p_k, CS_o) in the set of higher priority tasks.

   Rule 4 : Make the blocking time $b_i(np)$ due to non preemptability of T_i equal to

   $$\left(\left\lceil \max_{i+1 \le k \le n} \frac{\theta_k}{p_o} \right\rceil + 1 \right) p_o ,$$

   where θ_k is the maximum execution time of non preemptable sections of the lower priority task T_k.

3. Rule 1 is obvious. It takes into account the clock interrupt service overhead of the scheduler.

4. Each time a job in T_i or a task with a higher or equal priority than T_i becomes ready, the scheduler spends CS_o units of time to move the job to the ready queue.

5. This time is taken care of by rule 2. Similarly, because of rule 3, the time the scheduler takes to move lower priority jobs from the pending queue to the ready queue is added to the time-demand function of T_i.

6. Because lower priority jobs never execute in a level π_i busy interval, we need not to be concerned with whether they self suspend.

7. To see the rationale behind rule 4, we note that a job in T_i may suffer upto p_o units of delay waiting in the pending queue each time when it becomes ready, and we treat this delay as a factor of its blocking time.

8. This is why the blocking term $b_i(np)$ is equal to p_o even when lower priority tasks do not have non preemptable sections (*i.e.*, θ_k is 0 for all $k > i$).

9. If some lower priority tasks have non preemptable sections, a job may wait in the pending queue longer than the maximum execution time of the non preemptable sections.

10. In the worst case, a job may be released shortly after the $(x-1)^{th}$ clock interrupt for some x, a lower priority job enters a non-preemptable section shortly before the x^{th} clock interrupt, and exits the section shortly after the y^{th} clock interrupt ($y \geq x$).

11. The job waits in pending queue for approximately p_o units before the x^{th} clock interrupt, $(y-x)\,p_o$ units between clock interrupts x and y, and p_o units after the y^{th} clock interrupt. This is the reason for the expression of the blocking time.

Que 2.20. Explain rate monotonic scheduling algorithm ? Also, write its advantages and disadvantages.

Answer

RMA algorithm : Refer Q. 2.13, Page 69C, Unit-2.

Advantages of RMA :

1. RMA is very commonly used for scheduling real time tasks in practical applications.
2. Basic support is available in almost all commercial real time operating system for developing applications using RMA.
3. RMA is simple and efficient and is also the optimal static priority task scheduling algorithm.
4. Unlike EDF, it requires very few special data structures.
5. Most commercial real time operating system support real time (static) priority levels for tasks.
6. Tasks having real time priority levels are arranged in multilevel feedback queues.

Disadvantages of RMA :

1. It is very difficult to support scheduling of aperiodic and sporadic tasks under RMA.
2. Further, RMA is not optimal when task periods and deadlines differ.

Que 2.21. **Determine whether the following set of periodic real time tasks is scheduling on a unprocessed using RMA (Rate Monotonic Algorithm).**

Task	Start Time (MS)	Processing Time (MS)	Period (MS)	Deadline (MS)
T_1	20	25	150	100
T_2	40	7	40	40
T_3	60	10	60	50
T_4	25	10	30	20

UPTU 2014-15, Marks 05

Answer

1. Let us first compute the total CPU utilization achieved due to the given tasks.

$$U = \sum_{i=1}^{4} u_i = \frac{25}{150} + \frac{7}{40} + \frac{10}{60} + \frac{10}{30} = 0.84 \leq 1$$

2. Therefore, the necessary condition is satisfied.
3. The sufficiency condition is given by

$$\sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$$

4. Therefore, $0.84 \leq 4(2^{1/4} - 1)$

$= 0.84 \leq 0.76$

$\Rightarrow$ Not satisfied.

5. Although, the given set of tasks fails the Liu and Layland's test which is pessimistic in nature, we need to carry out Lehoczky's test.
6. We need to reorder the tasks according to their decreasing priorities.

Task	Start time (ms)	Processing Time (ms)	Period (ms)	Deadline (ms)
T_4	25	10	30	20
T_2	40	7	40	40
T_3	60	10	60	50
T_1	20	25	150	100

7. Testing for task T_4 : Since $e_4 <= d_4$, therefore, T_4 would meet its first deadline.

8. Testing for task $T_2 : 7 + \left\lceil \frac{40}{30} \right\rceil * 10 \leq 40$

   $\Rightarrow$ Satisfied

   $\Rightarrow$ Task T_2 would meet its first deadline.

9. Testing for task $T_3 : 10 + \left\lceil \frac{60}{40} \right\rceil * 7 + \left\lceil \frac{60}{30} \right\rceil * 10 \leq 50$

   $\Rightarrow$ Satisfied

   $\Rightarrow$ Task T_3 would meet its first deadline.

10. Testing for task $T_1 : 25 + \left\lceil \frac{150}{60} \right\rceil * 10 + \left\lceil \frac{150}{40} \right\rceil * 7 + \left\lceil \frac{150}{30} \right\rceil * 10 \leq 100$

    $\Rightarrow$ Satisfied

    $\Rightarrow$ Therefore, task T_1 would fail to meet its first deadline.

11. Hence, the given task set is not RMA schedulable.

PART-3

Offline Versus Online Scheduling, Scheduling Aperiodic and Sporadic Jobs in Priority Driven and Clock Driven Systems.

CONCEPT OUTLINE : PART-3

- A schedule may be precomputed (offline scheduling), or obtained dynamically (online scheduling).
- **Offline scheduling :** It involves scheduling in advance of this operation.
- **Online scheduling :** In this, the tasks are scheduled as they arrive in the system.
- A task can be periodic, sporadic, or aperiodic.
- The task (or job) is sporadic if it is not periodic, but may be invoked at irregular intervals.
- Sporadic tasks are sometimes called aperiodic. However, they also have no upper bound on the invocation rate.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.22. **Explain how online scheduling of jobs is done.**

OR

What are the advantages/ disadvantages of online scheduling and offline scheduling ?

Answer

Offline scheduling :

1. An offline algorithm takes complete information about the system activities, which reflects the knowledge about anticipated environmental situations and requirements and creates a single table, representing a feasible solution to the given requirements.
2. As the algorithm is performed offline, fairly complex task sets can be handled, for example, precedence constraints, distribution and communication over networks, task allocation, mutual exclusion, separation of tasks etc.
3. If a feasible solution is found, then retries are possible, for example, by changing the parameterization of the algorithm or the properties of the task set.
4. At runtime, a very simple runtime dispatcher executes the decisions represented in the task *i.e.*, a portion of a task to execute next.
5. A minimum granularity of time is assumed for the invocations of the runtime scheduler, called slots.

Advantages of offline scheduling :

a. For deterministic systems, the timing behaviour is one of the advantage because the computation of the schedules is done offline, the complexity of the scheduling algorithm used for this purpose is not important.

b. Offline schedules can make nearly full use of the resources because schedule is computed offline before the execution and there is prior knowledge of release times and processor/resource requirements of all the jobs.

Disadvantages of offline scheduling :

1. Offline schedule is inflexible because this approach can be used only in the deterministic systems.
2. There should be fixed set of functions, release times and processor/ resource requirements for computing the schedule.

Online scheduling :

1. In event triggered systems, events invoke an online scheduler, which takes a decision based on a set of predefined parameters, for example, represented as priorities.
2. An online schedulability test can be used to show that if a set of rules is applied to a given task set at runtime, all tasks will meet their deadlines.
3. Major representative lines of such algorithms are based on properties of fixed priorities, for example, rate monotonic or dynamic.

Advantages of online scheduling :

1. An online scheduler can accommodate dynamic variations in user demands and resource availability.

2. The price of the flexibility and adaptability is reduced ability for making the best use of system resources.
3. When system is not overloaded, online scheduling always produces a feasible schedule of all offered jobs.

Disadvantages of online scheduling :

1. The performance of online scheduling algorithms is poor when the system is so overloaded.
2. It is important to keep the system from being overloaded using some overload management or load shedding algorithms.

Que 2.23. Differentiate online and offline scheduling and static versus dynamic system. **UPTU 2012-13, Marks 05**

Answer

S. No	Online scheduling algorithm	Offline scheduling algorithm
1.	Online scheduling algorithms do not need to have the complete knowledge of the task set or its timing constraints.	In offline scheduling, scheduler has complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints etc.
2.	Scheduling decisions are based on dynamic parameters that may change during system evolution.	Scheduling decisions are based on fixed parameters, assigned to task before their activation.
3.	Online schedules are flexible and adaptive, but they can incur significant overheads because of runtime processing.	Offline guaranteed schedule is stored and dispatched late during runtime of the system.
4.	This algorithm is referred to as dynamic or runtime scheduling.	This is often referred to as static or pre-runtime scheduling.
5.	Usage of competitive analysis on scheduling problems. Online algorithm is characterized by making decision "online" which means a point in the time a second.	No conception of "point of time". We have knowledge of problem before we make decision. Even we use simplest method, such as enumeration, then we can obtain optimal solution.

Static versus dynamic system : Refer Q. 2.8, Page 63C, Unit-2.

Que 2.24. What are the distinguishing characteristics of periodic, aperiodic and sporadic real time tasks ? **UPTU 2014-15, Marks 05**

Answer

Characteristics of periodic real time tasks :

1. A periodic task is one that repeats after a certain fixed time interval.
2. The precise time instants at which periodic task recur are usually demarcated by clock interrupts.
3. For this reason, periodic task are sometimes referred to as clock driven tasks.
4. The fixed time interval after which a task repeats is called the period of the task.
5. If T_i is a periodic task, then the time from 0 till the occurrence of the first instance of T_i (*i.e.*, $T_i(1)$)is denoted by ϕ_i; and is called the phase of the task.
6. The second instance (*i.e.*, $T_i(2)$) occurs at $\phi_i + p_i$.
7. The third instance (*i.e.*, $T_i(3)$) occurs at $\phi_i + 2 * p_i$ and so on.
8. Formally a periodic task T_i can be represented by a four tuple (ϕ_i, p_i, e_i, d_i) where p_i is the period of task, e_i is the worst case execution time of the task, and d_i is the relative deadline of the task.
9. To represent real time periodic task, consider the track correction task typically found in rocket control software.
10. However, periodic tasks can also come into existence dynamically
11. The computation that occurs in air traffic monitors, once a flight is detected by the radar till the radar exits the radar signal zone is an example of a dynamically created periodic task.

Characteristics of aperiodic real time tasks :

1. An aperiodic task is in many ways similar to a sporadic task.
2. An aperiodic task can arise at random instants.
3. However, in case of an aperiodic task, the minimum separation g_i between two consecutive instances can be 0. That is, two or more instance of an aperiodic task might occur at the same time instant.
4. Also, the deadline for an aperiodic task is expressed as either an average value or is expressed statistically.
5. Aperiodic task are generally soft real time tasks.
6. Aperiodic tasks can recur in quick succession. It, therefore, becomes very difficult to meet the deadlines of all instances of an aperiodic task.

Characteristics of sporadic real time task :

1. A sporadic task is one that recurs at random instants.
2. A sporadic task T_i can be represented by a three tuple :

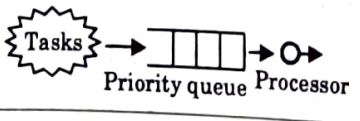
$$T_i = (e_i, g_i, d_i)$$

where e_i is the worst case execution time of an instance of the task, g_i denotes the minimum separation between two consecutive instances of the task, d_i is the relative deadline.

3. The minimum separation between two consecutive instances of the task implies that once an instance of a sporadic task occurs, the next instance cannot occur before g_i time units have elapsed.
4. The first instance of a sporadic task T_i is denoted by $T_i(1)$ and the successive instance by $T_i(2)$, $T_i(3)$, etc.
5. Many sporadic tasks such as emergency message arrivals are highly critical in nature. For example, in a robot a task that handles an obstacle that suddenly appears is a sporadic task.
6. The criticality of sporadic task varies from highly critical to moderately critical.
7. For example, an I/O device interrupt, or a DMA interrupt is moderately critical.

Que 2.25. Differentiate between priority-driven and clock driven systems.

Answer

S.No.	Priority-driven system	Clock driven system
1.		
2.	These are also called event driven because each scheduling decision is made when events, such as release and completions of job occurs.	These are also called time driven because each scheduling decision is made at a specific time, independent of events, such as job releases or completions in the system.
3.	This approach is used in non real time scheduling algorithms like FIFO (First-In-First-Out) and SETF (Shortest-Execution-Time-First).	This approach is applicable only when the system is deterministic, except for a few aperiodic jobs.
4.	When the scheduler is invoked, the job with the highest priority is dispatched.	A template schedule is a priori (off line) computed and recorded for run time use.
5.	Jobs are assigned priorities, often determined by an algorithm.	They schedule jobs at decision times.

Que 2.26. Discuss the general structure of cyclic schedules. How is average response time of aperiodic jobs improved ?

Answer

1. The scheduling decision times partition the timeline into intervals called frames. Every frame has length f called the frame size.
2. Because scheduling decisions are made only at the beginning of every frame, there is no preemption within each frame.
3. The phase of each periodic task is a non-negative integer multiple of the frame size. The first job of every task is released at the beginning of some frame.

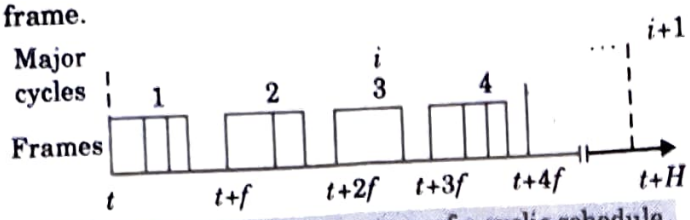


Fig. 2.26.1. General structure of a cyclic schedule.

4. Scheduler checks whether every job scheduled in the frame has been released and is ready for execution. It also checks whether there is any overrun and takes the necessary error handling action whenever it finds any erroneous condition.

Frame size constraints :

1. The frames are kept sufficiently long so that every job can start and complete its execution within a frame. In this way, no job will be preempted.
2. This can be done if we make the frame size f is larger than the execution time e_i of every task T_i. In other words,

$$f \geq \max_{1 \leq i \leq n}(e_i)$$

3. To keep the length of the cyclic schedule as short as possible, the frame size f should be chosen so that it divides H, the length of the hyperperiod of the system. This condition is met when f divides the period p_i of at least one task T_i, that is,

$$\left\lfloor \frac{p_i}{f} \right\rfloor - \frac{p_i}{f} = 0 \text{ for at least one } i.$$

4. When this condition is met, there is an integer number of frames in each hyperperiod. We let F denote this number and call a hyperperiod that begins at the beginning of the $(kF + 1)^{st}$ frames, for any $k = 0, 1, \ldots$ a major cycle.

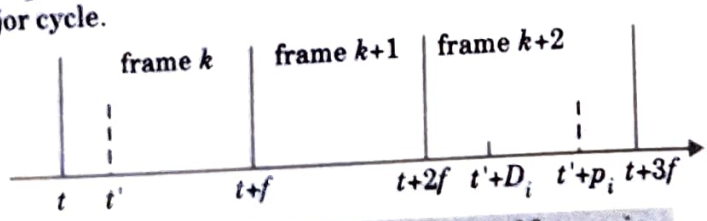


Fig. 2.26.2. A constraint on the value of frame size.

5. To complete every job by its deadline, frame size is kept sufficiently small so that between the release time and deadline of every job, there is at least one frame.
6. Fig. 2.26.2 shows the suitable range of f for a task $T_i = (p_i, e_i, D_i)$. When f is in this range, there is at least one frame between the release time and deadline of every job in the task.
7. In Fig. 2.26.2, t denotes the beginning of a frame (called the k^{th} frame) in which a job in T_i is released, and t' denotes the release time of this job.
8. We need to consider two cases : $t' > t$ and $t' = t$. If t' is greater than t, as shown in Fig. 2.26.2, we want the $(k+1)^{st}$ frame to be in the interval between the release time t' and the deadline $t' + D_i$ of this job.
9. We must have $t + 2f$ equal to or earlier than $t' + D_i$, i.e., $2f - (t' - t) \leq D_i$. Because the difference $t' - t$ is at least equal to the greatest common divisor $gcd(p_i, f)$ of p_i and f, this condition is met if the following inequality holds :

$$2f - gcd(p_i, f) \leq D_i$$

9. This equation holds for all $i = 1, 2, \ldots\ldots n$. In the special case, when t' is equal to t, it suffices to choose a frame size that is equal to or smaller than D_i. The condition $f \leq D_i$ is satisfied for all values of f.
10. In the process of constructing a cyclic schedule, we have to make three kinds of design decisions : choosing a frame size, partitioning jobs into slices, and placing slices in the frames.
11. These decisions cannot be made independently. The more slices a job is partitioned into, the higher is the context switch and communication overhead.

Que 2.27. What is deferrable server ? Explain the time-demand analysis method.

UPTU 2012-13, Marks 05

Answer

Deferrable server :

1. A deferrable server is the simplest of bandwidth preserving servers.
2. Like a poller, the execution budget of a deferrable server with period p_s and execution budget e_s is replenished periodically with period p_s.
3. Unlike a poller, however, 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 (p_s, e_s) are as follows :

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

Time-demand analysis method :

1. We can use the time-demand method to determine whether all the periodic tasks remain schedulable in the presence of deferrable server (p_s, e_s).

2. For a job $J_{i,c}$ in T_i that is released at a critical instant t_0, we add the maximum amount $e_s + \lceil (t - e_s)/p_s \rceil e_s$ of processor time demanded by the server at time t units after t_0.
3. Hence, the time-demand function of the task T_i is given by :

$$w_i(t) = e_i + b_i + e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k$$

for $0 < t \le p_i$

when the deferrable server (p_s, e_s) has the highest priority.

4. To determine whether the response time of T_i ever exceeds its relative deadline D_i in the case of $D_k \le p_k$ for all $k = 1, 2, \ldots, i$, we check whether $w_i(t) \le t$ is satisfied at any of the values of t that are less than or equal to D_i.
5. Again, we only need to check at values of t that are integer multiples of p_k for $k = 1, 2, \ldots, i-1$ and at D_i.
6. In addition, since $w_i(t)$ also increases at the replenishment times of the server at or before the deadline of $J_{i,c}$ (i.e., at $e_s, e_s + p_s, e_s + 2p_s, \ldots e_s + \lfloor (D_i - e_s)/p_s \rfloor p_s$), we also need to check whether $w_i(t) \le t$ at these values of t.
7. The response time of T_i is always equal to or less than D_i if the time supply t ever becomes equal to or larger than the time demand $w_i(t)$ at any of these values of t.
8. It is also easy to modify the time-demand analysis method for periodic tasks whose response times are longer than the respective periods to account for the effect of a deferrable server.
9. When the server has the highest priority, we add the time demand of the server (which is equal to $e_s + \lceil (t - e_s)/p_s \rceil e_s$) to the expression of $w_i(t)$.
10. Fig. 2.27.1 shows the time-demand functions of T_1 and T_2 in the system.
11. To determine whether T_2 is schedulable, we must check whether $w_i(t) \le t$ at 1 and 4, which are the replenishment times of the server before the deadline 6.5 of the first job in T_2, in addition to 3.5, which is the period of T_1, and 6.5.
12. As expected, both tasks are schedulable according to this test.

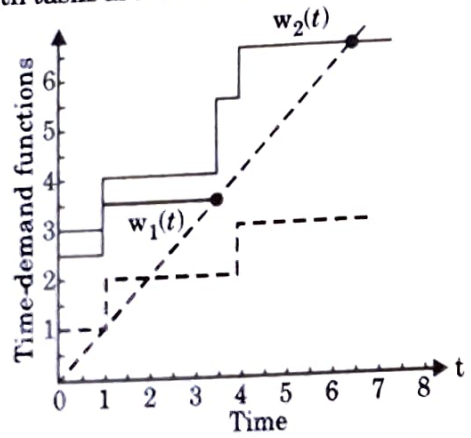


Fig. 2.27.1. The time-demand functions of $T_1 = (2, 3.5, 1.5)$ and $T_2 = (6.5, 0.5)$ with a deferrable server (3, 1.0).

Que 2.28. **What do you mean by scheduling of aperiodic and sporadic jobs ?**

Answer

1. Most of the scheduling algorithm does not require the knowledge of their execution times after they are released. The parameters of each sporadic job become known after it is released.
2. Sporadic jobs may arrive at any instant, even immediately after each other. Their execution times may vary widely and their deadlines are arbitrary.
3. For scheduling the jobs, the operating system maintains the priority queue.
4. The ready periodic jobs are placed in the periodic task queue, ordered by their priorities that are assigned according to the given periodic task scheduling algorithm.
5. Similarly, each sporadic job is assigned a priority and is placed in priority queue, which may or may not be the same as the periodic task queue. Each new aperiodic job is placed in the aperiodic job queue.

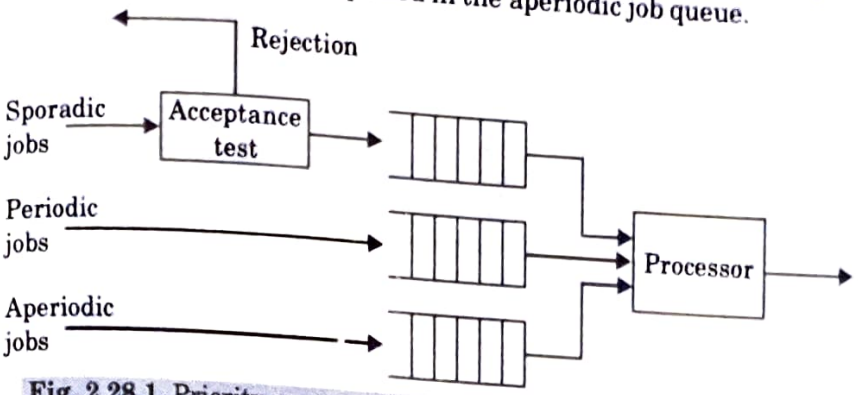


Fig. 2.28.1. Priority queue maintained by the operating system.

6. Aperiodic jobs are inserted in the aperiodic job queue and new sporadic jobs are inserted into a waiting queue to await acceptance without the intervention of the scheduler.

a. Based on the execution time and deadline of each newly arrived sporadic jobs, the scheduler decides whether to accept or reject the job.

b. If it accepts the job, it schedules the job so that the job completes in time without causing periodic tasks and previously accepted sporadic jobs to miss their deadlines.

c. The scheduler tries to complete each aperiodic job as soon as possible.

3

UNIT

Resource Sharing

PART-1

Effect of Resource Contention and Resource Access Control (RAC), Non-Preemptive Critical Sections, Basic Priority-Inheritance and Priority-Ceiling Protocols, Stack Based Priority-Ceiling Protocol.

CONCEPT OUTLINE : PART-1

- **Resource sharing :** In this real time, tasks need to share some resources among themselves.
- **Resource contention :** Two jobs conflict with one another, or have a resource contention, if some of the resources they require are of the same type.
- **Non-preemptive critical section (NPCS) :** When a job (or task) holds any resource, it executes at a priority higher than the priorities of all jobs. This protocol is called the NPCS.
- **Basic priority-inheritance :** It is defined by following rules :

1. Scheduling rule
2. Allocation rule
3. Priority - inheritance rule

- **Priority-ceiling protocol :** It extends the priority-inheritance protocol to prevent deadlocks and to further reduce the blocking time.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. **What do you understand by resource sharing among real time tasks ?**

Answer

1. In many applications, real time tasks need to share some resources among themselves.
2. Often these shared resources need to be used by the individual tasks in exclusive mode.
3. This means that a task that is using a resource, cannot immediately hand over the resource to another task that requests the resource at any arbitrary point in time; but it can do so only after it completes its use of the resource.

4. If a task is preempted from using a resource before it completes using the resource, then the resource can become corrupted.
5. Examples of such resources are files, devices, and certain data structures.
6. These resources are also called non-preemptable resources, or critical resources.
7. Some authors loosely refer to non-preemptable resources as critical sections, though the standard operating system literature uses this term to refer to sections of code in which some non-preemptable resources are used in exclusive mode.
8. Sharing of critical resources among tasks requires a different set of rules, compared to the rules used for sharing resources such as a CPU among tasks.
9. Resources such as CPU can be shared among tasks with cyclic scheduling, EDF, and RMA being the popular methodologies.
10. We must understand that CPU is an example of a serially reusable resource.
11. Another important feature of a serially reusable resource is that a task executing on a CPU can be preempted and restarted at a later time without any problem.
12. A non-preemptable resource is also used in the exclusive mode.
13. However, a task using a non-preemptable resource cannot be preempted from the resource usage, otherwise the resource would become inconsistent and can lead to system failure.
14. Therefore, when a lower priority task has already gained access to a non-preemptable resource and is using it, even a higher priority task would have to wait until the lower priority task using the resource completes.
15. For this reason, algorithms such as EDF and RMA that are popularly used for sharing a set of serially reusable resources (for example, CPU) cannot satisfactorily be used to share non-preemptable resources among a set of real time tasks.

Que 3.2. Explain resource contention/conflicts.

Answer

1. Two jobs conflict with one another, or have a resource contention, if some of the resources they require are of the same type.
2. The jobs contend for a resource when one job requests a resource that the other job already has.
3. The scheduler always denies a request if there are not enough free units of the resource to satisfy the request.
4. A scheduler may deny a request even when the requested resource units are free in order to prevent some undesirable execution behaviour.

5. The example in Fig. 3.2.1 illustrates the effect of resource contentions.
6. In this example, there are three jobs, J_1, J_2 and J_3, whose feasible intervals are (6, 14], (2, 17] and (0, 18], respectively.
7. The release time and deadline of each job are marked by the vertical bar on each of the timelines.
8. The jobs are scheduled on the processor on the earliest-deadline-first basis.
9. Hence, J_1 has the highest priority and J_3 the lowest.
10. All three jobs require the resource R, which has only 1 unit.
11. In particular, the critical sections in these jobs are [R; 2], [R; 4], and [R; 4], respectively.
12. The black boxes in Fig. 3.2.1 show when the jobs are in their critical sections.
13. At time 0, only J_3 is ready. It executes.
14. At time 1, J_3 is granted the resource R when it executes $L(R)$.
15. J_2 is released at time 2, preempts J_3, and begins to execute.

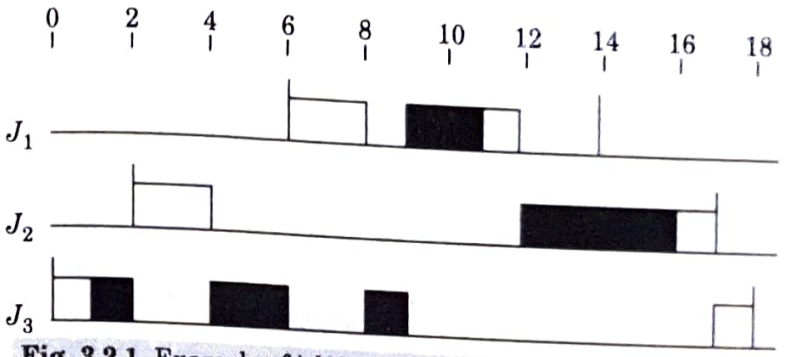


Fig. 3.2.1. Example of job interaction due to resource contention.

16. At time 4, J_2 tries to lock R. Because R is in use by J_3, this lock request fails. J_2 becomes blocked, and J_3 regains the processor and begins to execute.
17. At time 6, J_1 becomes ready, preempts J_3 and begins to execute.
18. J_1 executes until time 8 when it executes a $L(R)$ to request R. J_3 still has the resource. Consequently, J_1 becomes blocked. Only J_3 is ready for execution, and it again has processor and executes.
19. The critical section of J_3 completes at time 9. The resource R becomes free when J_3 executes $U(R)$. Both J_1 and J_2 are waiting for it. The priority of the former is higher. Therefore, the resource and the processor are allocated to J_1, allowing it to resume execution.
20. J_1 releases the resource R at time 11. J_2 is unblocked. Since J_1 has the highest priority it continues to execute.

21. J_1 completes at time 12. Since J_2 is no longer blocked and has a higher priority than J_3, it has the processor, holds the resource, and begins to execute. When it completes, at time 17, J_3 resumes and executes until completion at 18.

Que 3.3. Explain effects of resource contention and Resource Access Control (RAC) in detail.

Answer

Effects of resource contention and resource access control :

A resource access control protocol or an access control protocol is a set of rules that govern when and under what condition each request for resource is granted and how requiring resources are scheduled.

Priority inversion, timing anomalies and deadlock :

1. Priority inversion can occur when the execution of some jobs or portions of jobs is non-preemptable.
2. Resource contentions among jobs can also cause priority inversion.
3. Because resources are allocated to job on a non-preemptive basis, a higher priority job can be blocked by a lower priority job if the job conflict, even when the execution of both jobs is preemptable.
4. In the Fig. 3.3.1, the lowest priority job J_3 first blocks J_2 and then blocks J_1 while it holds the resource R.
5. As a result priority inversion occurs in intervals (4, 6] and (8, 9].

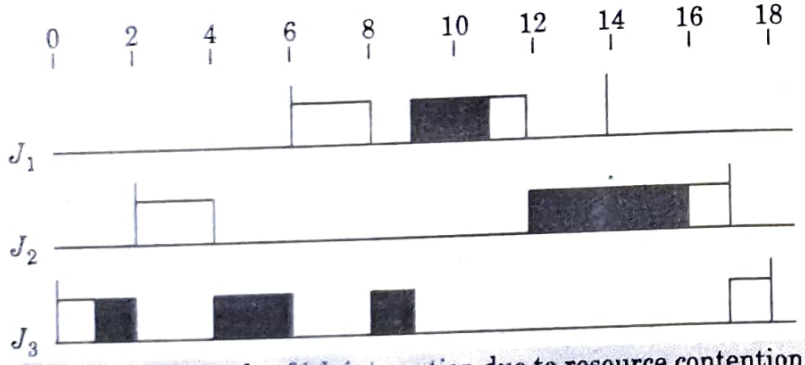


Fig. 3.3.1. Example of job interaction due to resource contention.

6. When priority inversion occurs, timing anomalies invariably follow.
7. Fig. 3.3.2 gives an example.
8. The three jobs are the same as those shown in Fig. 3.3.1, except that the critical section in J_3 is $[R; 2.5]$.
9. In other words, the execution time of the critical section in J_3 is shortened by 1.5.
10. Reduction allows jobs J_2 and J_3 to complete sooner.
11. Unfortunately, rather than meeting its deadline at 14, J_1 misses its deadline because it does not complete until 14.5.

Fig. 3.3.2. Example illustrating timing anomaly.

12. Without good resource access control, the duration of a priority inversion can be unbounded.
13. For example, in Fig. 3.3.3 the jobs J_1 and J_3 have the highest priority and lowest priority respectively.
14. At time 0, J_3 becomes ready and executes.
15. It acquires the resource R shortly afterwards and continues to execute.
16. After R is allocated to J_3, J_1 becomes ready. It preempts J_3 and executes until it requests resource R at time 3.
17. Because the resource is in use, J_1 becomes blocked and a priority inversion begins.
18. While J_3 is holding the resource and executes a job J_2 with a priority higher than J_3 but lower than J_1 is released. Moreover, J_2 does not require the resource R.
19. This job preempts J_3 and executes to completion. Thus, J_2 lengthens the duration of this priority inversion.
20. In this situation, the priority inversion is said to be uncontrolled.
21. There can be an arbitrary number of jobs with priorities lower than J_1 and higher than J_3 released in the meantime.
22. They can further lengthen the duration of the priority inversion.
23. When priority inversion is uncontrolled, a job can be blocked for an infinitely long time.
24. Non-preemptivity of resource allocation can also lead to deadlocks.
25. For example, when there are two jobs and both require resources X and Y.
26. The jobs are in deadlock when one of them holds X and requests for Y, while the other holds Y and requests for X.
27. Hence, no resource access control protocol can eliminate the priority inversion and anomalous behaviour caused by resource contention.

21. J_1 completes at time 12. Since J_2 is no longer blocked and has a higher priority than J_3, it has the processor, holds the resource, and begins to execute. When it completes, at time 17, J_3 resumes and executes until completion at 18.

Que 3.3. Explain effects of resource contention and Resource Access Control (RAC) in detail.

Answer

Effects of resource contention and resource access control :

A resource access control protocol or an access control protocol is a set of rules that govern when and under what condition each request for resource is granted and how requiring resources are scheduled.

Priority inversion, timing anomalies and deadlock :

1. Priority inversion can occur when the execution of some jobs or portions of jobs is non-preemptable.
2. Resource contentions among jobs can also cause priority inversion.
3. Because resources are allocated to job on a non-preemptive basis, a higher priority job can be blocked by a lower priority job if the job conflict, even when the execution of both jobs is preemptable.
4. In the Fig. 3.3.1, the lowest priority job J_3 first blocks J_2 and then blocks J_1 while it holds the resource R.
5. As a result priority inversion occurs in intervals (4, 6] and (8, 9].

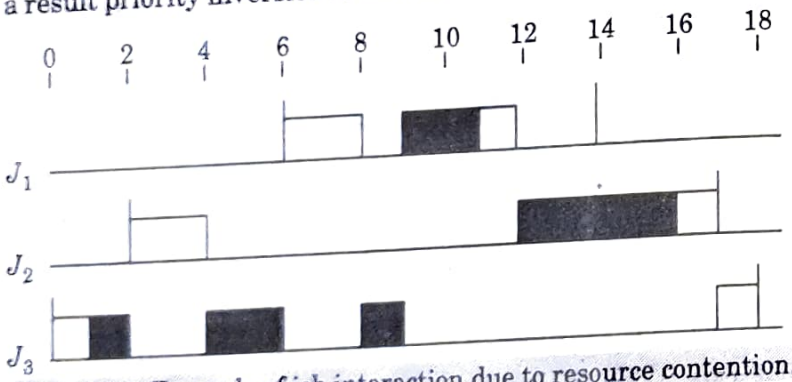


Fig. 3.3.1. Example of job interaction due to resource contention.

6. When priority inversion occurs, timing anomalies invariably follow.
7. Fig. 3.3.2 gives an example.
8. The three jobs are the same as those shown in Fig. 3.3.1, except that the critical section in J_3 is $[R; 2.5]$.
9. In other words, the execution time of the critical section in J_3 is shortened by 1.5.
10. Reduction allows jobs J_2 and J_3 to complete sooner.
11. Unfortunately, rather than meeting its deadline at 14, J_1 misses its deadline because it does not complete until 14.5.

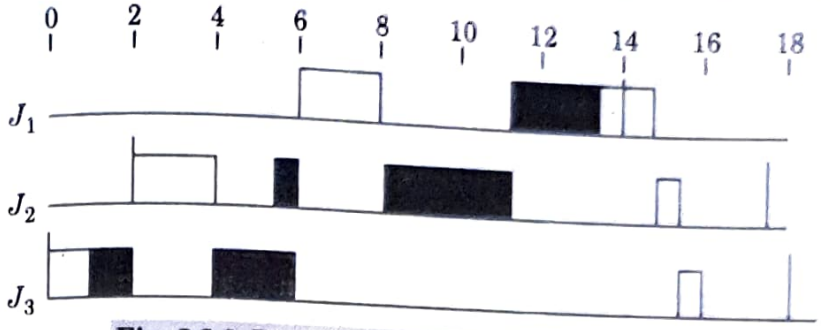


Fig. 3.3.2. Example illustrating timing anomaly.

12. Without good resource access control, the duration of a priority inversion can be unbounded.
13. For example, in Fig. 3.3.3 the jobs J_1 and J_3 have the highest priority and lowest priority respectively.
14. At time 0, J_3 becomes ready and executes.
15. It acquires the resource R shortly afterwards and continues to execute.
16. After R is allocated to J_3, J_1 becomes ready. It preempts J_3 and executes until it requests resource R at time 3.
17. Because the resource is in use, J_1 becomes blocked and a priority inversion begins.
18. While J_3 is holding the resource and executes a job J_2 with a priority higher than J_3 but lower than J_1 is released. Moreover, J_2 does not require the resource R.
19. This job preempts J_3 and executes to completion. Thus, J_2 lengthens the duration of this priority inversion.
20. In this situation, the priority inversion is said to be uncontrolled.
21. There can be an arbitrary number of jobs with priorities lower than J_1 and higher than J_3 released in the meantime.
22. They can further lengthen the duration of the priority inversion.
23. When priority inversion is uncontrolled, a job can be blocked for an infinitely long time.
24. Non-preemptivity of resource allocation can also lead to deadlocks.
25. For example, when there are two jobs and both require resources X and Y.
26. The jobs are in deadlock when one of them holds X and requests for Y, while the other holds Y and requests for X.
27. Hence, no resource access control protocol can eliminate the priority inversion and anomalous behaviour caused by resource contention.

Fig. 3.3.3. Uncontrolled priority inversion.

Que 3.4. What is non-preemptive critical section ?

Answer

1. The simple way to control access of resources is to schedule all critical sections on the processor non-preemptively.
2. When a job requests a resource, it is always allocated the resource.
3. When a job holds any resource, it executes at a priority higher than the priorities of all jobs.
4. This protocol is called Non-Preemptive Critical Section (NPCS) protocol.
5. Because no job is ever preempted when it holds any resource, deadlock can never occur.
6. Fig. 3.4.1 shows the schedule of these jobs when their critical sections are scheduled non-preemptively on the processor.
7. According to this schedule, J_1 is forced to wait for J_3 when J_3 holds the resource.
8. However as soon as J_3 releases the resource, J_1 becomes unblocked and executes to completion.
9. Because J_1 is not delayed by J_2, it completes at time 10, rather than 15 according to the schedule in Fig. 3.3.3.
10. In general, uncontrolled priority inversion in Fig. 3.3.3, can never occur.
11. The reason is that a job J_h can be blocked only if it is released when some lower priority job is in a critical section.
12. If it is blocked, once the blocking critical section completes, all resources are free.
13. No lower priority job can get the processor and acquire any resource until J_h completes.
14. Hence, J_h can be blocked only once and its blocking time due to resource conflict is at most equal to the maximum execution time of the critical sections of all lower priority jobs.

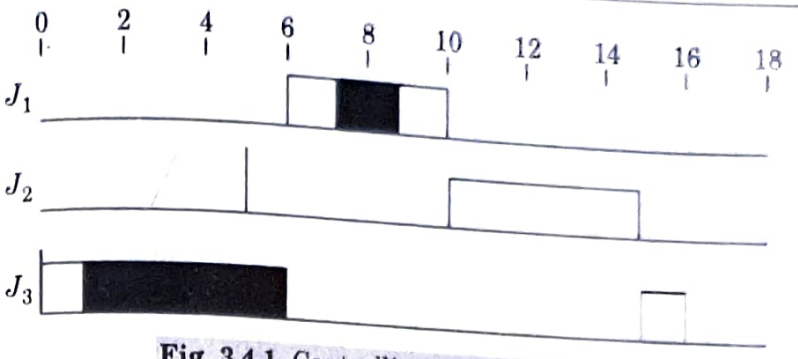


Fig. 3.4.1. Controlling priority inversion by disallowing preemption of critical section.

15. The most important advantage of the NPCS protocol is its simplicity, especially when the number of resource units are arbitrary.
16. The protocol does not need any prior knowledge about resource requirements of jobs.
17. It is simple to implement and can be used in both fixed priority and dynamic priority systems.
18. It is good protocol when all the critical sections are short and when most of the jobs conflict with each other.
19. The disadvantage of this protocol is that every job can be blocked by every lower priority job that requires some resource even when there is no resource conflict between them.

Que 3.5. Explain basic priority-inheritance protocol with its working. Also, explain its rules and properties.

OR

Define priority-inheritance protocol (PIP) with its properties and show that PIP works as greedy algorithm.

UPTU 2014-15, Marks 10

Answer

Basic priority-inheritance protocol :

1. The priority-inheritance protocol is also a simple protocol.
2. It works with any preemptive, priority-driven scheduling algorithm.
3. Like the NPCS protocol it does not require prior knowledge of resource requirement of jobs.
4. The priority-inheritance protocol does not prevent deadlock.
5. When there is no deadlock, the protocol ensures that no job is ever blocked for an indefinitely long time because uncontrolled priority inversion cannot occur.
6. The priority that is assigned to a job according to the scheduling algorithm is called its assigned priority.

7. 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.
8. In particular, the current priority $\pi_l(t)$ of a job J_l may be raised to the higher priority $\pi_h(t)$ of another job J_h.
9. When this happens, we say that the lower priority job J_l inherits the priority of the higher priority job J_h and J_l executes at its inherited priority $\pi_h(t)$.
10. The priority-inheritance protocol is defined by the following rules.
11. These rules govern the ways current priorities of jobs are set and jobs are scheduled when some of them contend for resources.
12. This protocol assumes that every resource has only 1 unit.

Rules of the basic priority-inheritance protocol :

1. **Scheduling rule :**
   a. Ready jobs are scheduled on the processor preemptively in a priority-driven manner according to their current priorities.
   b. At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority.
   c. 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.
   b. If R is not free, the request is denied and J is blocked.
3. **Priority-inheritance rule :**
   a. When the requesting job J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J.
   b. The job J_l executes at its inherited priority $\pi(t)$ until it releases R; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it acquires the resource R.

Example :

a. There are five jobs and two resources Black and Shaded.
b. The parameters of the jobs and their critical sections are listed in Table 3.5.1.
c. Jobs are indexed in decreasing order of their priorities.
d. The priority π_i of J_i is i, and the smaller the integer, the higher the priority.
e. In the schedule in Fig. 3.5.1, black boxes show the critical sections when the jobs are holding Black.
f. Shaded boxes show the critical sections when the jobs are holding Shaded.

Table : 3.5.1. Parameters of jobs.

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]

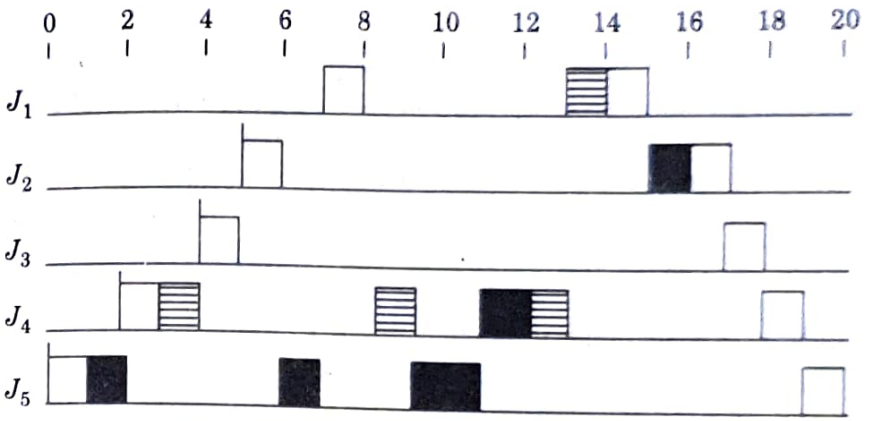


Fig. 3.5.1. Schedule under priority inheritance.

i. At time 0, job J_5 becomes ready and executes at its assigned priority 5. At time 1, it is granted the resource Black.

ii. At time 2, J_4 is released. It preempts J_5 and starts to execute.

iii. At time 3, J_4 requests Shaded. Shaded, being free, is granted to the job. The job continues to execute.

iv. At time 4, J_3 is released and preempts J_4. At time 5, J_2 is released and preempts J_3.

v. 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.

vi. J_1 is released at time 7. Having the highest priority 1, it preempts J_5 and starts to execute.

vii. 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.

viii. 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.

ix. 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.

x. 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, acquired Shaded and begins to execute.

xi. 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.

xii. At time 17, J_2 completes. Afterwards, jobs J_3, J_4 and J_5 execute in turn to completion.

Properties of the priority-inheritance protocol :

1. When resource accesses are controlled by the priority-inheritance protocol, there are two types of blocking, direct blocking and priority-inheritance blocking (or simply inheritance blocking).
2. J_2 is directly blocked by J_5 in (6, 11] and by J_4 in (11, 12.5] and J_1 is directly blocked by J_4 in (8, 13].
3. In addition, J_3 is blocked by J_5 in (6, 7] because the latter inherits a higher priority in this interval.
4. Later at time 8, when J_4 inherits priority 1 from J_1, J_3 becomes blocked by J_4 as a consequence.
5. Fig. 3.5.1 shows that jobs can transitively block each other.
6. At time 9, J_5 blocks J_4 and J_4 blocks J_1 so, priority-inheritance is transitive.
7. In the time interval (9, 11), J_5 inherits J_4's priority, which J_4 inherited from J_1.
8. As a consequence J_5 indirectly inherits the J_1's priority.
   a. **The priority inheritance protocol does not prevent deadlock :**
      i. Let us suppose that J_5 in this example were to request the resource Shaded sometime after Shaded has been granted to J_4.
      ii. These two jobs would be deadlocked.
   b. **The priority-inheritance protocol does not reduce the blocking times suffered by jobs as small as possible :**
      i. It is true that in the absence of a deadlock, a job can be blocked directly by any lower priority job for at most once for the duration of one outermost critical section.

PIP works as a greedy algorithm :

1. Priority inheritance is difficult to implement, with many complicated scenarios arising when two or more tasks attempt to access the same resources.
2. The algorithm for resolving a long chain of nested resource locks is complex.

3. It is possible to incur a lot of overhead as hoisting one task results in hoisting another task, and another, until finally some task is hoisted that has the resources needed to run.
4. After executing its critical section, each hoisted task must then return to its original priority.
5. Fig. 3.5.2 shows the simplest case of the priority inheritance protocol in which a low-priority task acquires a resource that's then requested by a higher priority task.
6. Fig. 3.5.3 shows a slightly more complex case, with a low-priority task owning a resource that is requested by two higher-priority tasks.
7. Fig. 3.5.4 demonstrates the potential for complexity when three tasks compete for two resources.

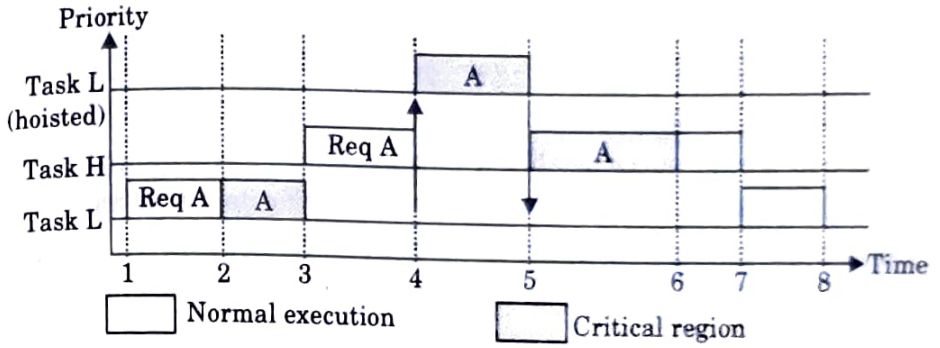


Fig. 3.5.2. A Simple priority inheritance.

1. Task L receives control of the processor and begins executing.
1. The task makes a request for Resource A.

   Task L is granted ownership of Resource A and enters its critical region.

   Task L is preempted by Task H, a higher-priority task.
0. Task H begins executing and requests ownership of Resource A, which is owned by Task L.

   Task L is hoisted to a priority above Task H and resumes executing its critical region.

   Task L releases Resource A and is lowered back to its original priority.
0. Task H acquires ownership of Resource A and begins executing its critical region.

   Task H releases Resource A and continues executing normally.

   Task H finishes executing and Task L continues executing normally.

   Task L finishes executing.

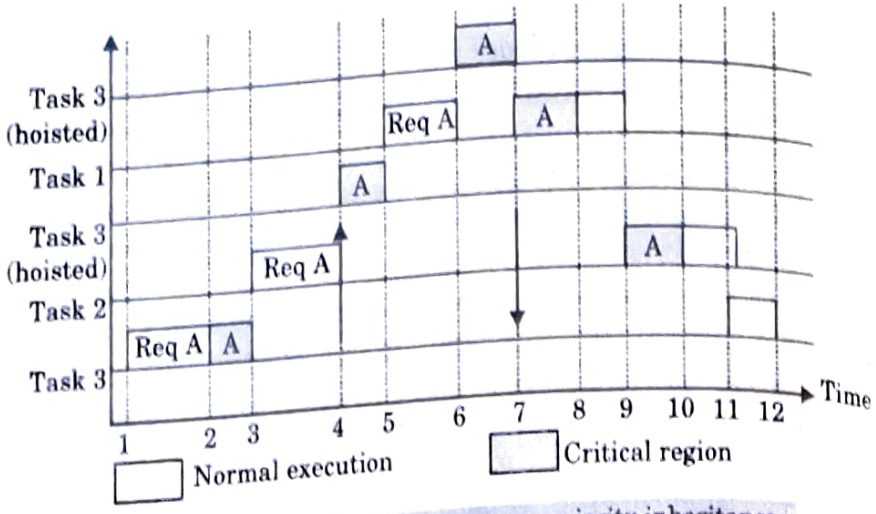


Fig. 3.5.3. Three-task, one-resource priority inheritance.

1. Task 3 gets control of the processor and begins executing.
1. The task requests ownership of Resource A.

   Task 3 acquires Resource A and begins executing its critical region.

   Task 3 is preempted by Task 2, a higher-priority task.
0. Task 2 begins executing normally and requests Resource A, which is owned by Task 3.

   Task 3 is hoisted to a priority above Task 2 and resumes executing its critical region.

   Task 3 is preempted by Task 1, a higher-priority task.
0. Task 1 begins executing and requests Resource A, which is owned by Task 3.

   Task 3 is hoisted to a priority above Task 1.
0. Task 3 resumes executing its critical region.

   Task 3 releases Resource A and is lowered back to its original priority.
0. Task 1 acquires ownership of Resource A and begins executing its critical region.

   Task 1 releases Resource A and continues executing normally.

   Task 1 finishes executing. Task 2 acquires Resource A and begins executing its critical region.

   Task 2 releases Resource A and continues executing normally.

   Task 2 finishes executing. Task 3 resumes and continues executing normally.

   Task 3 finishes executing.

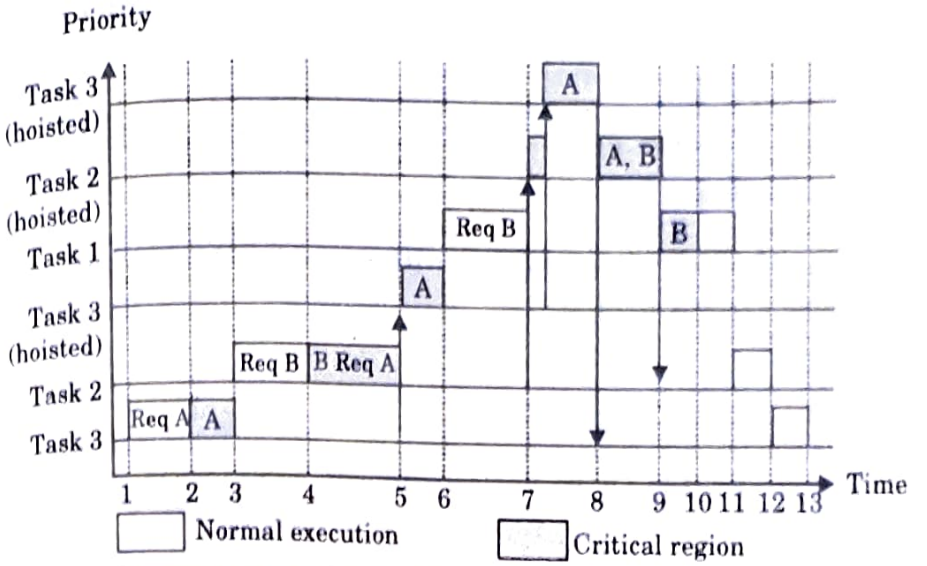


Fig. 3.5.4. Three-task, two-resource priority inheritance.

1. Task 3 is given control of the processor and begins executing. The task requests Resource A.
2. Task 3 acquires ownership of Resource A and begins executing its critical region.
3. Task 3 is preempted by Task 2, a higher-priority task. Task 2 requests ownership of Resource B.
4. Task 2 is granted ownership of Resource B and begins executing its critical region.
1. The task requests ownership of Resource A, which is owned by Task 3.

   Task 3 is hoisted to a priority above Task 2 and resumes executing its critical region.

   Task 3 is preempted by Task 1, a higher-priority task.
0. Task 1 requests Resource B, which is owned by Task 2.

   Task 2 is hoisted to a priority above Task 1. However, Task 2 still cannot execute because it must wait for Resource A, which is owned by Task 3.
0. Task 3 is hoisted to a priority above Task 2 and continues executing its critical region.

   Task 3 releases Resource A and is lowered back to its original priority.
0. Task 2 acquires ownership of Resource A and resumes executing its critical region.

   Task 2 releases Resource A and then releases Resource B. The task is lowered back to its original priority.
0. Task 1 acquires ownership of Resource B and begins executing its critical region.

   Task 1 releases Resource B and continues executing normally.

   Task 1 finishes executing. Task 2 resumes and continues executing normally.

Task 2 finishes executing. Task 3 resumes and continues executing normally.
Task 3 finishes executing.

Que 3.6. **Discuss priority-ceiling protocol and explain how it avoids deadlock.**

OR

Explain priority-ceiling protocol with example. Also, explain how avoidance is done by it.

Answer

Basic priority-ceiling protocol :

1. The priority-ceiling protocol extends the priority-inheritance protocol to prevent deadlocks and to further reduce the blocking time.
2. This protocol makes two key assumptions :
   i. The assigned priorities of all jobs are fixed.
   ii. The resources required by all jobs are known a priori before the execution of any job begins.
3. To define the protocol, we need two additional terms.
4. The protocol makes use of a parameter, called priority-ceiling, of every resource.
5. The priority-ceiling of any resource R_i is the highest priority of all the jobs that require R_i and is denoted by $\Pi(R_i)$.
6. For example, the priority-ceiling Π(Black) of the resource Black in the example in Table 3.6.1 and Fig. 3.6.1 is 2 because J_2 is the highest priority job among the jobs requiring it.
7. At any time t, the current priority-ceiling $\hat{\Pi}(t)$ of the system is equal to the highest priority-ceiling of the resources that are in use at the time, if some resources are in use.
8. If all the resources are free at the time, the current ceiling $\hat{\Pi}(t)$ is equal to Ω, a non-existing priority level that is lower than the lowest priority of all jobs.
9. Now, the priority-ceiling protocol is defined for the case when there is only 1 unit of every resource.

Rules of basic priority-ceiling protocol :

1. **Scheduling rule :**
   a. At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.
   b. Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$.

2. **Allocation rule :**
Whenever a job J requests a resource R at time t, one of the following two conditions occurs :
   a. R is held by another job. J's request fails and J becomes blocked.
   b. R is free.
      i. If priority $\pi(t)$ is higher than the current priority-ceiling $\hat{\Pi}(t)$, R is allocated to J.
      ii. If J's priority $\pi(t)$ is not higher than the ceiling $\hat{\Pi}(t)$ of the system, R is allocated to J only if J is the job holding the resource(s) whose priority-ceiling equal to $\hat{\Pi}(t)$; otherwise J's request is denied and J becomes blocked.

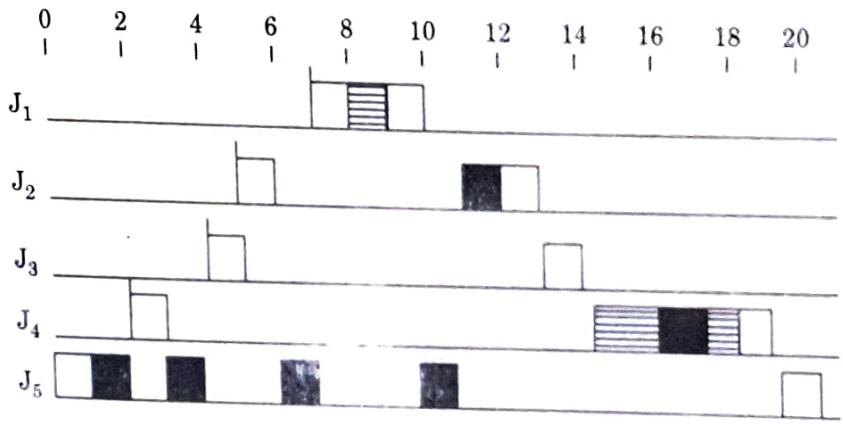
3. **Priority-inheritance rule :**
When J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J. J_l executes at its inherited priority until the time when it releases every resource whose priority-ceiling is equal to or higher than $\pi(t)$; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it was granted the resource(s).

Fig. 3.6.1 shows the schedule of the system of jobs whose parameters are listed in Table 3.6.1 when their accesses to resources are controlled by the priority-ceiling protocol.

The priority-ceilings of the resources Black and Shaded are 2 and 1 respectively.

Table : 3.6.1. Parameters of jobs

Job	r_i	e_i	π_i	Critical section
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]



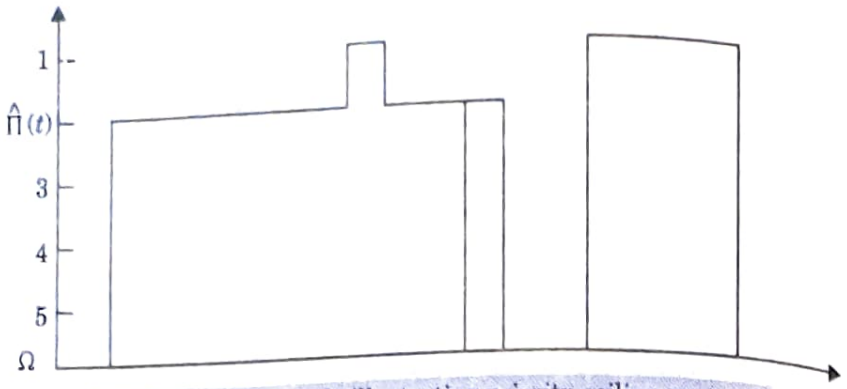


Fig. 3.6.1. A schedule illustrating priority-ceiling protocol.

i. In the interval (0, 3], this schedule is the same as the schedule shown in Fig. 3.5.1, which is produced under the basic priority-inheritance protocol. In particular, the ceiling of the system at time 1 is Ω. When J_5 requests Black, it is allocated the resource according to (i) in part (b) of rule 2. After Black is allocated, the ceiling of the system is raised to 2, the priority-ceiling of Black.

ii. At time 3, J_4 requests Shaded. Shaded is free; however because the ceiling $\hat{\Pi}(3) (= 2)$ of the system is higher than the priority of J_4, J_4's request is denied according to (ii) in part (b) of rule 2. J_4 is blocked and J_5 inherits J_4's priority and executes at priority 4.

iii. At time 4, J_3 preempts J_5 and at time 5, J_2 preempts J_3. At time 6, J_2 requests Black and becomes directly blocked by J_5. Consequently, J_5 inherits the priority 2; it executes until J_1 becomes ready and preempts it. During all this time, the ceiling of the system remains at 2.

iv. When J_1 requests Shaded at time 8, its priority is higher than the ceiling of the system. Hence, its request is granted according to (i) in part (b) of rule 2, allowing it to enter its critical section and complete by the time 10. At time 10, J_3 and J_5 are ready. The latter has higher priority (*i.e.* 2); it resumes.

v. At time 11, when J_5 releases Black, its priority returns to 5 and the ceiling of the system drops to Ω. J_2 becomes unblocked, is allocated Black [according to (i) in part (b) of rule 2], and starts to execute.

vi. At time 14, after J_2 and J_3 complete, J_4 has the processor and is granted the resource Shaded because its priority is higher than Ω, the ceiling of the system at the time. It starts to execute. The ceiling of the system is raised to 1, the priority ceiling of Shaded.

vii. At time 16, J_4 requests Black, which is free. The priority of J_4 is lower than $\hat{\Pi}(16)$, but J_4 is the job holding the resource (*i.e.* Shaded) whose priority-ceiling is equal to $\hat{\Pi}(16)$. Hence, according to (ii) of part (b) rule 2, J_4 is granted Black. It continues to execute. The rest of the schedule is self-explanatory.

Deadlock avoidance by priority-ceiling protocol :

1. Principles of operating system that one way to avoid deadlock is the use of ordered-resource technique.
2. The set of priority-ceiling of resources impose a linear order on all the resources.
3. It may not surprise you that deadlock can never occur under the priority-ceiling protocol.
4. In order to gain a deeper insight into how the protocol works to prevent deadlock, we pause to look at a more complicated example in Fig. 3.6.2, there are three jobs : J_1, J_2, and J_3 with priorities 1, 2, and 3, respectively.
5. Their release times are 3.5, 1, and 0 and their critical sections are [Dotted; 1.5], [Black; 2 [Shaded; 07]], and [Shaded; 4.2 [Black; 2.3]], respectively.
6. In this schedule, the intervals during which the jobs are their critical sections are shown as the dotted box (the critical section associated with resources Dotted), shaded boxes (critical sections associated with resource Shaded), and black boxes (critical sections associated with resource Black)

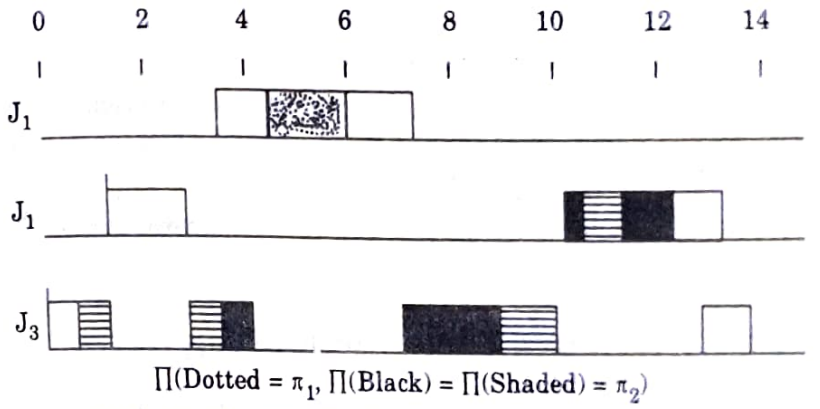


Fig. 3.6.2. Example illustrating how priority-celling protocol prevents deadlock.

a. When J_3 requests Shaded at time 0.5, no resource is allocated at the time, J_3's request is granted. When job J_2 become ready at time 1, it preempts J_3.

b. At time 2.5, J_2 request Black. Because Shaded is already allocated to J_3 and has priority ceiling 2, the current ceiling of the system is 2. J_2's priority is 2. According to (ii) or part (b) of rule 2, J_2 is denied Black, even though the resource is free. Since J_2 is blocked, J_3 inherits the priority 2 (rule 3), resumes, and starts to execute.

c. When J_3 request Black at time 3, it is holding the resource whose priority ceiling is the current ceiling of the system. According to (ii) to part (b) or rule 2, J_3 is granted the resource Black, and it continues to execute.

d. J_3 is preempted again at time 3.5 when J_1 becomes ready. When J_1 requests Dotted at time 4.5, the resources is free and the priority of J_1 is higher than the ceiling of the system. (i) or part (b) of rule 2 applies, and Dotted is allocated to J_1, allowing the job to enter into its critical section and proceed to complete at 7.3.

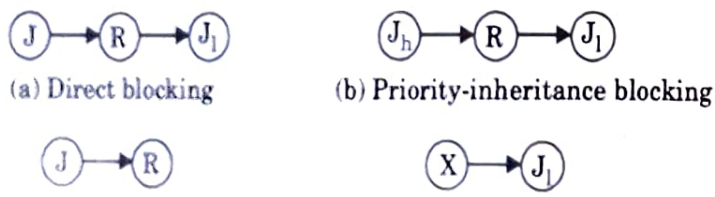
Que 3.7. What are the properties of priority-inheritance protocol ? Differentiate between the priority-inheritance protocol and priority-ceiling protocol. **UPTU 2012-13, Marks 10**

Answer

Properties of priority-inheritance protocol : Refer Q.3.5, Page 97C, Unit-3.

Difference between priority-inheritance protocol and priority-ceiling protocol :

1. A fundamental difference between the priority-inheritance and priority-ceiling protocol is that the former is greedy while the latter is not.
2. The allocation rule of the priority-inheritance protocol lets the requesting job have a resource whenever the resource is free.
3. In contrast, according to the allocation rule of the priority-ceiling protocol, a job may be denied its requested resource even when the resource is free at the time.
4. The wait for graph in Fig. 3.7.1 illustrates the three ways in which a job J can be blocked by a lower priority job when resource accesses are controlled by the priority-ceiling protocol.
5. J can be directly blocked by a lower priority job J_l, as shown by the wait for graph in Fig. 3.7.1(a).
6. As a consequence of the priority-inheritance rule, a job J can also be blocked by a lower priority job J_l which has inherited the priority of a higher priority job J_h.
7. The wait for graph in Fig. 3.7.1(b) shows this situation.
8. The allocation rule may cause a job J to suffer priority-ceiling blocking, which is represented by the graph in Fig. 3.7.1(c).



(a) Direct blocking (b) Priority-inheritance blocking

Π (X) is equal to or higher than π(t)

(c) Avoidance blocking

Fig. 3.7.1. Ways for a job to block another job.

9. The requesting job J is blocked by a lower priority job J_l when J requests a resource R that is free at the time.
10. The reason is that J_l holds another resource X whose priority-ceiling is equal to or higher than J's priority $\pi(t)$.
11. Rule 3 says that a lower priority job is directly blocking or priority-ceiling blocking the requesting job J inherits J's priority $\pi(t)$.

Que 3.8. **Why the term avoidance blocking is given to priority-ceiling protocol ? How do you compute the blocking time ? Explain with example.** **UPTU 2012-13, Marks 10**

Answer

Avoidance blocking : Refer Q. 3.6, Page 104C, Unit-3.

Computation of blocking time :

1. To illustrate how to do this computation, let us consider the Fig. 3.8.2.
2. Even for this small system, it is error prone if we compute the blocking times of all jobs by inspection. Table 3.8.1, give us a systematic way.
3. There is a row for each job that can be blocked.
4. The tables list only the non-zero entries; all the other entries are zero.
5. Since jobs are not blocked by higher-priority jobs, the entries; at and below "*" in each column are zero.
6. The leftmost part is the direct-blocking table.
7. It lists for each job, the duration for which it can be directly blocked by each of the lower priority jobs.
8. The entries in this table come directly from the resource requirement graph of the system.
9. Indeed, for the purpose of calculating the blocking times of the jobs, this table gives a complete specification of the resource requirements of the jobs.

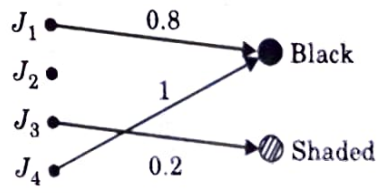


Fig. 3.8.1. Example on duration of blocking.

10. The middle part of Table. 3.8.1, is the priority-inheritance blocking table.
11. It lists the maximum duration for which each job can be priority-inheritance blocked by each of the lower priority jobs.
12. For example, J_6 can inherit priority π_1 of J_1 for 2 units of time when it directly blocks J_1.

13. Hence, it can block all the other jobs for 2 units for time.
14. In the table, we show 2 units of inheritance blocking time of J_2 and J_3 by J_6.
15. However, because J_6 can also inherit π_3 for 4 units of time, it can block J_4 and J_5 for 4 units of time.
16. This is the reason that the entries in the fourth and fifth rows of column 6 are 4.

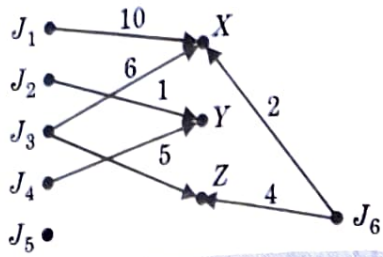


Fig. 3.8.2. Example illustrating the computation of blocking times.

17. In general, a systematic way to get the entries in each column of this table from the entries in the corresponding column of the direct-blocking table is as follows.
18. The entry at column k and row i of the inheritance blocking table is the maximum of all the entries in column k and row 1, 2,...., $i - 1$ of the direct-blocking table.

Table 3.8.1.

	Directly blocked by					Priority-inheritance blocked by					Priority-ceiling blocked by				
	J_2	J_3	J_4	J_5	J_6	J_2	J_3	J_4	J_5	J_6	J_2	J_3	J_4	J_5	J_6
J_1		6			2										
J_2	*		5			*	6			2	*	6			2
J_3		*			4		*	5		2		*	5		2
J_4			*					*		4			*		4
J_5				*					*	4				*	

19. The rightmost table in Table 3.8.1 is the avoidance (priority-ceiling) blocking table.
20. It lists the maximum duration for which each job can be avoidance blocked by each lower priority job.
21. In general, when the priorities of all the jobs are distinct, the entries in the avoidance blocking table are equal to corresponding entries in the priority-inheritance blocking table, except for jobs which do not require any resources.
22. Jobs which do not require any resource are never avoidance blocked, just as they are never directly blocked.

23. The blocking time $b_i(rc)$ of each job J_i is equal to the maximum value of all the entries in the i^{th} row of the three tables.
24. From Table 3.8.1, we have $b_i(rc)$ is equal to 6, 6, 5, 4, 4, and 0 for i = 1, 2,.... 6, respectively.

Que 3.9. **What do you understand by stack-based priority-ceiling protocol ?**

OR

Write the rules of stack-based priority-ceiling protocol.

Answer

Stack-based priority-ceiling (ceiling-priority) protocol :

1. A resource in the system is the runtime stack.
2. Till now, we have assumed that each job has its own runtime stack.
3. Sometimes, especially in systems where the number of jobs is large, it may be necessary for the jobs to share a common runtime stack, in order to reduce overall memory demand.
4. Space in the (shared) stack is allocated to jobs contiguously in the last-in-first-out manner.
5. When a job J executes, its stack space is on the top of the stack.
6. The space is free when a job completes. When J is preempted, the preempting job has the stack space above J's.
7. J can resume execution only after all the jobs holding stack space above its space complete, free their stack spaces, and leave J's stack space on the top of the stack again.
8. To ensure deadlock free sharing of the runtime stack among jobs, it must be ensured that no job is ever blocked because it is denied some resource once its execution begins.
9. This protocol allows jobs to share the runtime stack if they never self suspend.
10. In the statement of the rules of the stack based priority-ceiling protocol, the term (current) ceiling $\hat{\Pi}(t)$ of the system is used, which is the highest priority ceiling of all the resources that are in use at time t.
11. Ω is a non-existing priority level that is lower than the lowest priority of all jobs.
12. The current ceiling is Ω when all resources are free.

Rules defining basic stack-based priority-ceiling protocol :

0. Update of the current ceiling :

a. Whenever all the resources are free, the ceiling of the system is Ω.

b. The ceiling $\hat{\Pi}(t)$ is updated each time a resource is allocated or freed.

1. **Scheduling rule :**
   a. After a job is released, it is blocked from starting execution until its assigned priority is higher than the current ceiling $\hat{\Pi}(t)$ of the system.
   b. At all times, jobs that are not blocked are scheduled on the processor in a priority-driven, preemptive manner according to their assigned priorities.

2. **Allocation rule :**
   a. Whenever a job requests a resource, it is allocated the resource.
   b. According to the scheduling rule, when a job begins to execute, all the resources it will ever need during its execution are free.
   c. This is why the allocation rule is as simple as stated above. More importantly, no job is ever blocked once its execution begins.
   d. Likewise, when a job J is preempted, all the resources the preempting job will require are free, ensuring that the preempting job can always complete so J can resume.
   e. Consequently, deadlock can never occur.

Example :

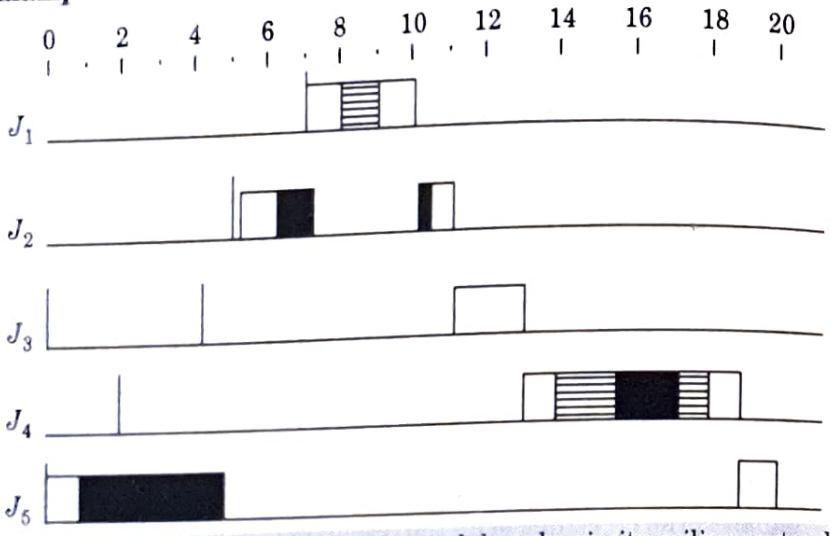


Fig. 3.9.1. Schedule illustrating the stack-based, priority-ceiling protocol.

1. The schedule in Fig. 3.9.1 shows how the system of jobs in Fig. 3.7.1 would be scheduled if the stack-based, priority-ceiling protocol were used instead of the basic priority-ceiling protocol.
2. To illustrate the stack based protocol, let J_2 be released at 4.8 and the execution time of the critical section of J_2 be 1.2.
3. At time 2 when J_4 is released, it is blocked from starting because its priority is not higher than the ceiling of the system, which is equal to 2 at the time.

4. This allows J_5 to continue execution.
5. For the same reason, J_3 does not start execution when it is released.
6. When J_2 is released at time 4.8, it cannot start execution because the ceiling of the system is 2.
7. At time 5, the resource held by J_5 becomes free and the ceiling of the system is at Ω.
8. Consequently, J_2 starts to execute since it has the highest priority among all the jobs ready at the time.
9. As expected when it requests the resource Black at time 6, the resource is free.
10. It acquires the resource and continues to execute.
11. At time 7 when J_1 is released, its priority is higher than the ceiling of the system, which is 2 at the time.
12. J_1, therefore, preempts J_2 and holds the space on the top of the stack until it completes at time 10.
13. J_2 then resumes and completes at 11.
14. Afterwards, J_3, J_4 and J_5 complete in the order of their priorities.
15. From this example, we see that the scheduling rule of the stack-based priority-ceiling protocol achieves the same objective as the more complicated priority-inheritance rule of the basic priority-ceiling protocol.
16. When we compare the schedule in Fig. 3.9.1 with the schedule in Fig. 3.7.1 which is produced by the basic priority-ceiling protocol, the higher priority jobs J_1, J_2 and J_3 either complete earlier than or at the same time as when they are scheduled according to the basic priority ceiling protocol.

PART-2

Use of Priority-Ceiling Protocol in Dynamic Priority Systems, Preemption-Ceiling Protocol, Access Control in Multiple-Unit Resources, Controlling Concurrent Accesses to Data Objects.

CONCEPT OUTLINE : PART-2

- **Dynamic priority system :** In this system, the priorities of the periodic tasks change with time while the resources required by each task remain constant.
- Like the priority-ceiling protocol, the preemption-ceiling protocol has :
  1. A basic version
  2. A stack- based version

- **Data objects :** They are a special type of shared resources.
- When jobs are scheduled preemptively, their accesses to data objects may be interleaved.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.10. Define dynamic priority systems with suitable example and discuss the implementation of priority-ceiling protocol in such system.

OR

Explain use of priority-ceiling protocol in dynamic work for the real time application. **UPTU 2014-15, Marks 10**

Answer

Dynamic priority system :

1. In a dynamic priority system, the priorities of the periodic tasks change with time while the resources required by each task remain constant.
2. As a consequence, the priority ceilings of the resources may change with time.
3. For some dynamic systems, we can still use the priority-ceiling protocol to control resource accesses provided. We update the priority-ceiling of each resource and the ceiling of the system each time when task priorities change, except for this update, the priority-ceiling protocol can be applied without modification in job level fixed-priority systems.
4. In such a system, the priorities of jobs, once assigned, remain fixed with respect to each other.
5. In particular, the order in which jobs in the ready job queue are sorted among themselves does not alter each time a newly released job is inserted in the queue.

Implementation of priority-ceiling protocol in dynamic priority system :

1. To implement the basic priority-ceiling protocol in a job-level fixed-priority system, we have to update the priority-ceilings of all resources whenever a new job is released.
2. Specifically when a new job is released, its priority relative to all the jobs in the ready queue is assigned according to the given dynamic-priority algorithm.
3. Then the priority-ceilings of all the resources are updated based on the new priorities of the tasks and the ceiling of the system is updated based on the new priority-ceilings of the resources.

Example :

1. The example in Fig. 3.10.1 illustrates the use of this protocol in an EDF system.
2. The system shown here has three tasks : $T_1 = (0.5, 2.0, 0.2; [Black; 0.2])$, $T_2 = (3.0, 1.5; [Shaded; 0.7])$ and $T_3 = (5.0, 1.2; [Black; 1.0 [Shaded; 0.4]])$.
3. The priority-ceiling of the two resources Black and Shaded are updated at times 0, 0.5, 2.5, 3, 4.5, 5, 6 and so on.
4. Consecutive positive integers are used to denote the priorities of all the ready jobs, the highest priority 1.
5. The working of the priority-ceiling protocol to produce the schedule segment is shown in Fig. 3.10.1.
6. To emphasize that the priority-ceiling of a resource R_i may change with time we denote it by $\Pi_t(R_i)$.

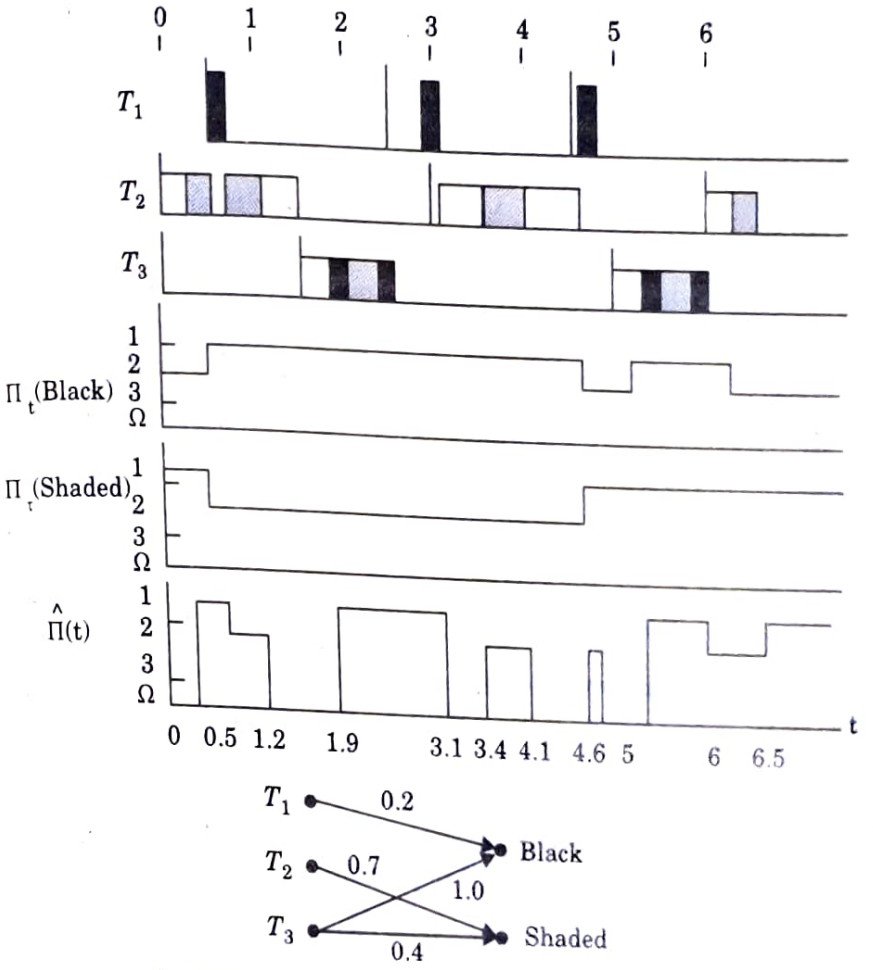


Fig. 3.10.1. Example illustrating the use of the basic priority-ceiling protocol.

i. At time 0, there are only two ready jobs $J_{2,1}$ and $J_{3,1}$. $J_{2,1}$ (and hence T_2) has priority 1 while T_3 has priority 2, the priority of $J_{3,1}$. The priority-

ceilings of Black and Shaded are 2 and 1 respectively. Since $J_{2,1}$ has a higher priority, it begins to execute. Because no resource is in use, the ceiling of the system is Ω. At time 0.3, $J_{2,1}$ acquires Shaded and the ceiling of the system rises from Ω to 1, the priority-ceiling of Shaded.

ii. At time 0.5, $J_{1,1}$ is released and it has a higher priority than $J_{2,1}$ and $J_{3,1}$. Now the priorities of T_1, T_2 and T_3 become 1, 2 and 3, respectively. The priority-ceiling Π_t(Black) of Black is 1, the priority of $J_{1,1}$ and T_1. The priority-ceiling Π_t(Shaded) of Shaded becomes 2 because the priority of $J_{2,1}$ and T_2 is now 2. The ceiling of the system based on these updated values is 2. For this reason, $J_{1,1}$ is granted the resource Black. The ceiling of the system is 1 until $J_{1,1}$ releases Black and completes at time 0.7. Afterwards, $J_{2,1}$ continues to execute and the ceiling of the system is again 2. When $J_{2,1}$ completes at time 1.7, $J_{3,1}$ commences to execute and later acquires the resources as shown.

iii. At time 2.5, $J_{1,2}$ is released. It has priority 1, while $J_{3,1}$ has priority 2. This update of task priorities leads to no change in priority-ceilings of the resources. Since the ceiling of the system is at 1, $J_{1,2}$ becomes blocked at 2.5. At time 2.9, $J_{3,1}$ releases Black, and $J_{1,2}$ commences execution.

iv. At time 3.0, only T_1 and T_2 have jobs ready for execution. Their priorities are 1 and 2 respectively. The priority-ceiling of the resources remain unchanged until time 4.5.

v. At time 4.5, the new job $J_{1,3}$ of T_1 has a later deadline than $J_{2,2}$ (Again, T_3 has no ready job) T_2 becomes 1. This change in task priorities causes the priority ceiling of Black and Shaded to change to 2 and 1, respectively.

vi. At time 5, when $J_{3,2}$ is released, it is the only job ready for execution at the time and hence has the highest priority. The priority-ceiling of both resources are 1. These values remain until time 6.

vii. At time 6, both $J_{2,3}$ and $J_{3,2}$ are ready and the former has an earlier deadline. We now have the same condition as at time 0.

Que 3.11. **Discuss basic features and governing rules of preemption-ceiling protocol and mention its relative merits over priority-ceiling protocol.** **UPTU 2012-13, Marks 10**

OR

Explain

i. Preemption-ceiling protocol.

ii. Stack-base preemption-ceiling protocol.

Answer

i. Preemption-ceiling protocol :

1. A preemption-ceiling protocol makes decision on whether to grant a free resource to any job based on the preemption level of the job in a manner similar to the priority-ceiling protocol.

2. This protocol also assumes that the resource requirements of all the jobs are known a priori.
3. After assigning preemption levels to all the jobs, we determine the preemption-ceiling of each resource.
4. Specifically when there is only 1 unit of each resource, which we assume is the case here the preemption-ceiling $\Psi(R)$ of resource R is the highest preemption level of all the jobs that require the resource.
5. The (preemption) ceiling of the system $\hat{\Psi}(t)$ at any time t is the highest preemption-ceiling of all the resources that are in use at t.
6. When the context is clear and there is no chance of confusion, we will refer to $\hat{\Psi}(t)$ as the ceiling of the system.
7. Ω is used to denote a preemption level that is lower than the lowest preemption level among all jobs since there is no possibility of confusion.
8. When all the resources are free, the ceiling of the system is Ω.
9. Like the priority-ceiling protocol, the preemption-ceiling protocol also has a basic version and a stack-based version.
10. The former assumes that each job has its own stack and the latter allows the jobs to share a common stack.
11. Basic versions of priority-ceiling and preemption-ceiling protocols differ mainly in their allocation rules.
12. For this reason, only the allocation rule of the basic preemption-ceiling protocol is given.
13. The principle of this rule for both protocols is the same, the only difference being the parameters used by the rule.

Rules of basic preemption-ceiling protocol :

1 and 3 : The scheduling rule (*i.e.*, rule 1) and priority-inheritance rule (*i.e.*, rule 3) are the same as the corresponding rules of the priority-ceiling protocol.

2. **Allocation rule :** Whenever a job J requests resource R at time t, one of the following two conditions occurs :
   a. R is held by another job. J's request fails and J becomes blocked.
   b. R is free.
      i. If J's preemption level $\psi(t)$ is higher than the current preemption-ceiling $\hat{\Psi}(t)$ of the system, R allocated to J.
      ii. If J's preemption level $\psi(t)$ is not higher than the ceiling $\hat{\Psi}(t)$ of the system, R is allocated to J only if J is the job holding the resource(s) whose preemption-ceiling is equal to $\hat{\Psi}(t)$; otherwise, J's request is denied and J becomes blocked.

ii. Stack-based, preemption-ceiling protocol :

1. The stack-based preemption-ceiling protocol is called the Stack-Based Protocol (SBP).

2. It is defined by the following rules : Rules 0, 1 and 2 are essentially the same as the corresponding rules of the stack-based priority-ceiling protocol; again the difference is that priority levels/ceilings are replaced by preemption levels/ceilings.
3. In addition, the stack-based, preemption-ceiling protocol has an inheritance rule.

Rules of basic stack-based, preemption-ceiling protocol :

0. Update of the current ceiling :

a. Whenever all the resources are free, the preemption-ceiling of the system is Ω.

b. The preemption-ceiling $\hat{\Psi}(t)$ is updated each time a resource is allocated or freed.

1. Scheduling rule :

a. After a job is released, it is blocked from starting execution until its preemption level is higher than the current ceiling $\hat{\Psi}(t)$ of the system and the preemption level of the executing job.

b. At any time t, jobs that are not blocked are scheduled on the processor in a priority driven, preemptive manner according to their assigned priorities.

2. Allocation rule : Whenever a job J requests for a resource R, it is allocated by the resource.

3. Priority-inheritance rule : When some job is blocked from starting, the blocking job inherits the highest priority of all the blocked jobs.

Merits over priority-ceiling protocol :

1. Less time consuming.
2. Less storage overhead.

Que 3.12. Explain priority-ceiling of multiple unit resources with suitable example.

Answer

Priority (preemption)-ceiling of multiple unit resources :

1. The first step in extending the priority-ceiling protocol is to modify the definition of the priority-ceilings of resources.
2. Let $\Pi(R_i, k)$, for $k \leq v_i$, denote the priority-ceiling of a resource R_i when k out of the v_i (≥ 1) units of R_i are free.
3. If one or more job in the system require more than k units of R_i, $\Pi(R_i, k)$ is the highest priority of all these jobs.
4. If no job requires more than k units of R_i, $\Pi(R_i, k)$ is equal to Ω, the non existing lowest priority.
5. The priority-ceiling $\Pi(R_j)$ of a resource R_j that has only 1 unit is $\Pi(R_j, 0)$.
6. Let $k_i(t)$ denote the number of units of R_i that are free at time t.

7. Because this number changes with time, the priority-ceiling of R_i changes with time.
8. The current priority-ceiling of the system at time t is equal to the highest priority-ceiling of all the resources at the time.
9. For example, in Fig. 3.12.1, the resource requirement graph gives the numbers of units of the resources X, Y and Z required by the five jobs that are indexed in decreasing order of their priorities.
10. Table 3.12.1 gives the priority-ceilings of each resource for different numbers of free resource units.

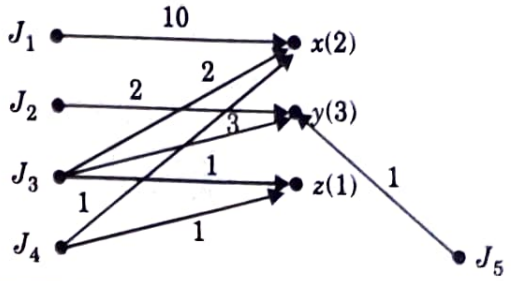


Fig. 3.12.1. Example of priority-ceiling of multiple unit.

Table 3.12.1

Resources		$x(2)$	$y(3)$	$z(1)$
Units Required by	J_1	1	0	0
	J_2	0	2	0
	J_3	2	3	1
	J_4	1	0	1
	J_5	0	1	0
$\Pi(*, 0)$		π_1	π_2	π_3
$\Pi(*, 1)$		π_3	π_2	Ω
$\Pi(*, 2)$		Ω	π_3	Ω
$\Pi(*, 3)$		Ω	Ω	Ω

11. For example, there are 2 units of X. When 1 unit of X is used, only J_3 is directly blocked. Therefore, $\Pi(X, 1)$ is π_3.
12. J_1 is also directly blocked when both units of X are in use. For this reason, $\Pi(X, 0)$ is π_1, the higher priority between π_1 and π_3.
13. When both units of X are free, the ceiling of the resource is Ω.
14. Similarly, since J_2, J_3 and J_5 require 2, 3 and 1 unit of Y, which has 3 units, $\Pi(Y, 0)$, $\Pi(Y, 1)$ and $\Pi(Y, 2)$ are equal to π_2, π_2 and π_3 respectively.
15. Suppose that at time t, 1 unit of each of X, Y and Z is free.

16. The priority-ceilings of the resources are π_3, π_2 and Ω respectively and the priority-ceiling of the system is π_2.

Modified rules :

1. It is straightforward to modify the ceiling priority protocol so it can deal with multiple unit resources.
2. In essence, the scheduling and allocation rules remain unchanged except for the new definition of priority-ceiling of resources.
3. However, since more than one job can hold a resource, scheduling rule 1b needs to be rephrased for clarity. It should read as follows :

Scheduling rule of multiple unit ceiling priority protocol : Upon acquiring a resource R and leaving $k \geq 0$ free units of R, a job executes at the higher of its priority and the priority-ceiling $\Pi(R, k)$ of R.

Allocation rule of multiple unit priority (preemption) ceiling protocol : Whenever a job J requests k unit of resources R at time t, one of the following two conditions occurs :

a. Less than k units of R are free. J's request fails and J becomes directly blocked.

b. k or more units of R are free.

   i. If J's priority $\pi(t)$ [preemption level $\psi(t)$] is higher than the current priority-ceiling $\hat{\Pi}(t)$ [preemption-ceiling $\hat{\Psi}(t)$] of the system at the time, k units of R are allocated to J until it releases them.

   ii. If J's priority $\pi(t)$ [preemption level $\psi(t)$] is not higher than the system ceiling $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$], k units of R are allocated to J only if J holds the resources whose priority-ceiling (preemption ceiling) is equal to $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$]; otherwise J's request is denied and J becomes blocked.

Que 3.13. What is priority-inheritance rule ? Explain with suitable example.

Answer

The following priority-inheritance rule is proposed that each job is blocked at most once for the duration of one critical section.

Priority-inheritance rule :

1. When the requesting job J becomes blocked at t, the job with the highest priority among all the jobs holding the resource R that has the highest priority-ceiling among all the resources inherits J's priority until it releases its unit of R.
2. A job may request and hold arbitrary numbers of resources.
3. The example as shown in Fig. 3.13.1, illustrates that a straightforward generalization of the priority-ceiling protocol and the priority-inheritance rule ensures that each job is blocked at most once.

4. The system in this example has five jobs indexed in decreasing order of their priorities (In the description, the priorities are 1, 2, 3, 4 and 5).
5. There are two resources Black and Shaded.
6. The numbers of units are 5 and 1, respectively.
   i. At time 0, J_5 starts to execute. When it requests 1 unit of Black at time 0.5, the ceiling of the system is Ω; therefore, it is granted 1 unit of Black and continues to execute. The ceiling of the system stays at Ω because there are still sufficient units of Black to meet the demand of every job.
   ii. At time 1, J_4 becomes ready. It preempts J_5 and later, requests and is granted 1 unit of Black. Now J_2 would be directly blocked if it requests Black and the ceiling of Black and consequently of the system, becomes 2, the priority of J_2.

Resource	No. of	Units required					$\Pi(*, k), k =$					
	units	J_1	J_2	J_3	J_4	J_5	0	1	2	3	4	5
Black	5	2	4	0	1	1	1	1	2	2	Ω	Ω
Shaded	1	1	1	0	0	1	1	Ω	Ω	Ω	Ω	Ω

(a)

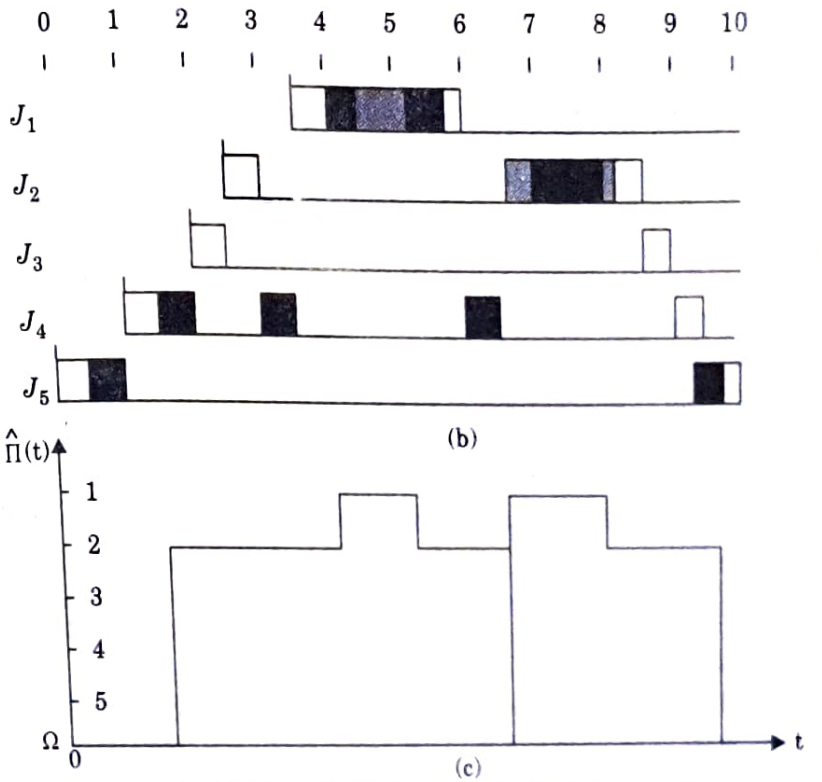


Fig. 3.13.1. Example of multiple unit resource.

iii. At time 2, J_3 preempts J_4 and at time 2.5, J_2 preempts J_3. J_2 becomes blocked when it requests Shaded at time 3 because its priority is not higher than the ceiling of the system. J_4 now inherits priority 2 and executes.

iv. At time 3.5, J_1 preempts J_4. Since its priority is higher than the ceiling of the system, J_1 is allocated both resources when it requests them.

v. At time 6, J_1 completes and J_4 continues to execute until it releases its 1 unit of Black at time 6.5. The ceiling of the system returning to Ω, J_2 is allocated Shaded. After Shaded is allocated, the ceiling of the system becomes 1.

vi. At time 7, when J_2 requests 4 units of Black, the units are available. The ceiling of the system is 1, but J_2 holds the resource with this priority-ceiling. Hence, it is allocated 4 units of Black.

vii. When J_2 completes, J_3 resumes. When J_3 completes, J_4 resumes and it is followed by J_5. The system becomes idle at time 10.

Que 3.14. Explain controlling concurrent accesses to data objects in detail.

OR

What is convex-ceiling protocol ? Also, write the rules of convex-ceiling protocol.

Answer

Controlling concurrent accesses to data objects :

1. Data objects are a special type of shared resources.
2. When jobs are scheduled preemptively, their accesses (*i.e.*, reads and writes) to data objects may be interleaved.
3. To ensure data integrity, it is common to require that the reads and writes be serializable.
4. A sequence of reads and writes by a set of jobs is serializable if the effect produced by the sequence on all the data objects shared by the jobs is the same as the effect produced by a serial sequence.

Convex-ceiling protocol :

1. The convex (priority)-ceiling protocol is extension of the priority-ceiling protocol.
2. It is an improvement over the PCP (Priority and Preemption Ceiling Protocol)-2PL (Two Phase Locking) protocol because it reduces the duration of blocking.
3. In the description below, assume that there is only one of each data objects.

Priority-ceiling function :

1. As with the PCP-2PL protocol, the convex-ceiling protocol assumes that the scheduler knows a priori the data objects require by each job and therefore, the priority-ceiling of each data object.
2. In addition, each job notifies the scheduler immediately after it accesses each of its required objects for the last time.
3. We call a notification sent by a job J_i after it accesses R_k for the last time the last access notification for R_k by J_i and the time of this notification the last access time of R_k by J_i.
4. For each job J_i in the system, the scheduler generates and maintains the following two functions; the remainder priority-ceiling, $RP(J_i, t)$ and the priority-ceiling function, $\Pi(J_i, t)$.
5. $RP(J_i, t)$ is the highest priority ceiling of all data objects that J_i will require after time t.
6. When J_i is released, $RP(J_i, t)$ is equal to the highest priority-ceiling of all data objects required by the job.
7. The scheduler updates this function each time when it receives a last access notification from J_i.
8. When the job no longer requires any object, its remainder priority-ceiling is Ω.
9. When each job J_i starts execution, its priority-ceiling function $\Pi(J_i, t)$ is equal to Ω.
10. When J_i is allowed to access an object R_k for the first time, $\Pi(J_i, t)$ is set to the priority-ceiling $\Pi(R_k)$ of R_k if the current value of $\Pi(J_i, t)$ is lower than $\Pi(R_k)$.
11. Upon receiving a last access notification from J_i, the scheduler first updates the function $RP(J_i, t)$.
12. It then sets the priority-ceiling function $\Pi(J_i, t)$ of the job to $RP(J_i, t)$ if the remainder priority-ceiling is lower.
13. As with the priority-ceiling protocol, at any time t when the scheduler receives a request to access an object R for the first time from any job J, it computes the system ceiling $\hat{\Pi}(t)$.
14. $\hat{\Pi}(t)$ is equal to the highest priority of the priority-ceiling functions of all the jobs in the system.

Rules of convex-ceiling protocol :

1. **Scheduling rule :**
   a. At any time, jobs that are not suspended are scheduled on the processor in preemptive, priority driven manner.

b. Upon its release, the current priority of every job J_i is its assigned priority π_i. It executes at this priority except when the inheritance rule is applied.

2. **Allocation rule :**

When a job J_i requests to access a data object R for the first time,

a. If J_i's priority is higher than a system ceiling $\hat{\Pi}(t)$. J is allowed to continue execution and access R.

b. If J_i's priority is not higher than $\hat{\Pi}(t)$,

i. If $\hat{\Pi}(t)$ is equal to $\Pi(J_i, t)$, J_i is allowed to access R;

ii. Otherwise, J is suspended.

3. **Priority-inheritance rule :**

When J_i becomes suspended, the job J_l whose priority-ceiling function is equal to the system ceiling at the time inherits the current priority $\pi_i(t)$ of J_i.

Que 3.15. Explain real time concurrency control schemes.

Answer

1. One way to improve the responsiveness of soft real time jobs that read and write multiple data objects is to abort and restart the lower priority job whenever it conflicts with a higher priority job.
2. A policy that governs which job proceeds at the expense of which job is called a conflict resolution policy.

Conflict resolution policies :

1. Conflict resolution policies have been defined for database transactions.
2. Each transaction typically keeps a copy of each object it reads and may write in its own memory space.
3. When it completes all the reads, writes and computations, it writes all data objects it has modified back to the global space. This last step is called commit.
4. So, until a transaction commits, no shared data object in the database is modified and it is safe to abort and restart a transaction and take back the data objects allocated to it.
5. 2PL-HP schemes perform well for soft real time transactions compared with Optimistic Concurrency Control (OCC) schemes.
6. According to the 2PL-HP scheme, all transactions follow a two phase policy in their acquisitions of (locks of) data objects.

7. Whenever two transactions conflict, the lower priority transaction is restarted immediately.
8. In essence, this scheme allocates data objects to transactions preemptively. Therefore, priority inversion cannot occur.

Optimistic concurrency control schemes :

1. Optimistic concurrency control is an alternative approach to two phase locking.
2. Under the control of an OCC scheme, whether a transaction conflict with other executing transactions is checked immediately before the transaction commits. This step is called validation.
3. If the transaction is found to conflict with other transaction at the time, one of them is allowed to proceed and commit, while the conflicting transactions are restarted.
4. A priority based OCC scheme allows a conflicting lower priority scheme to proceed until validation time and then restarted if it is found to conflict with some higher priority transaction at validation time.

Que 3.16. How are deadlocks, unbounded priority inversions, and chain blocking prevented using PCP ? **UPTU 2014-15, Marks 05**

Answer

How is deadlock avoided in PCP :

1. Deadlocks occur only when different (more than one) tasks hold parts of each other's required resources at the same time, and then they request for the resources being held by each other.
2. But under PCP, when one task is executing with some resources, any other task cannot hold a resource that may ever be needed by this task.
3. That is, when a task is granted one resource, all its required resources must be free.
4. This prevents the possibility of any deadlock.

How is unbounded priority inversion avoided :

1. A higher priority task suffers unbounded priority inversion, when it is waiting for a lower priority task to release some of the resources required by it, and meanwhile intermediate priority tasks preempt the low priority task from CPU usage.
2. But such a situation can never happen in PCP since whenever a higher priority task waits for some resources which is currently being used by a low priority task, then the executing lower priority task is made to inherit the priority of the high priority task.

3. So, the intermediate priority tasks cannot preempt lower priority task from CPU usage. Therefore, unbounded priority inversions cannot occur under PCP.

How is chain blocking avoided :

1. Consider that a task T_i needs a set of resources $SR = \{R_i\}$.
2. Obviously, the ceiling of each resource R_i in SR must be greater than or equal to pri (T_i).
3. Now, assume that when T_i acquires some resource R_i, another task T_j was already holding a resource R_j and that $R_i, R_j \in SR$.
4. Such a situation would lead T_i to block after acquiring a resource.
5. But, when T_j locked R_j CSC should have been set to at least pri(T_i) by the resource grant clause of PCP and T_i could not have been granted R_i.
6. This is a contradiction with the premise.
7. Therefore, when T_i acquires one resource, all resources required by it must be free.
8. In a similar way, we can show that once a task T_i acquires a resource, it cannot undergo any inheritance-related inversion.
9. Assume that a lower priority task T_k is holding some resource R_z and a higher priority task T_h is waiting for the resource.
10. But, this is not possible as Ceil (R_2) must be at least as much as pri (T_h).
11. The CSC should, therefore, have been set to a value that is at least as much as pri (T_h) and T_i would have been prevented from accessing the resource R_i, in the first place.
12. This is a contradiction with the premise we started with. Therefore, it is not possible that a task undergoes inheritance-related inversion after it acquires a resource.
13. Using a similar reasoning, we can show that a task cannot suffer any avoidance inversion after acquiring a resource.
14. Thus, once a task acquires a resource it cannot undergo any inversion.
15. It is, therefore, clear that tasks under PCP are single blocking.

Que 3.17. **Differentiate between :**

i. **Multiprocessor system and Distributed system**
ii. **Identical and Heterogeneous processors**
iii. **Local and Remote resources**
iv. **RMFF & RMST algorithms**

Answer

i. **Multiprocessor and Distributed system :**

S. No.	Multiprocessor system	Distributed system
1.	These are known as tightly coupled system.	These are known as loosely coupled system.
2.	These are the systems that share physical memory in the system.	Distributed systems are devoid of any shared physical memory in the system.
3.	The interprocess communication in tightly coupled system is inexpensive and can be ignored compared to task execution time.	The interprocess communication in loosely coupled system is expensive.
4.	Inter task communication is achieved through reads and writes to the shared memory and not comparable to execution times.	Inter task communication is comparable to task execution times.
5.	Multiprocessor systems may use a centralized dispatcher or scheduler and dispatches or schedules and leads to high communication overheads.	Distributed systems do not use a centralized dispatcher or scheduler and thus no such communicational overhead.

ii. **Identical and Heterogeneous processor :**

S. No.	Identical Processor	Heterogeneous Processor
1.	Identical processors are processors of the same type *i.e.*, the processor can be used interchangeably.	Heterogeneous processors are processors in which each job can execute on some types of processor but, in general not on all types.
2.	For example, each of the CPUs in a parallel machine can execute every computation job in the system, thus the CPUs are identical.	For example, the execution time of computation intensive job may be one second on CPU1 but is five seconds on a less powerful CPU2. On the other hand, because CPU2 has better interrupt handling and

		I/O capabilities, the execution time of an I/O intensive job is ten seconds on CPU1 but is only three seconds on CPU2. Thus, it is also called unrelated processor.
3.	No such characterization is done as all are identical.	The unrelated processor model helps us to characterize all the systems.

iii. Local and Remote resources :

S. No.	Local resources	Remote resources
1.	Resource that resides on the local processor is called a local resource.	Resource that does not reside on the local processor rather resides on another processor is called remote resource.
2.	In end-to-end resource model, all jobs in an end-to-end task requires only resources on its local processor.	Multiprocessor priority-ceiling protocol performs all jobs on both local as well as global processors.
3.	Each component job requires only resources on the processor on which the component job executes.	No such specific requirement of the component job on the processor, it can execute remotely too.

iv. RMFF and RMST algorithm :

S. No.	Rate Monotonic First Fit algorithm	Rate Monotonic Small Task algorithm
1.	In the RMFF algorithm, tasks are first sorted in non-decreasing order according to their periods.	In the RMST algorithm, periodic tasks are first sorted in non-decreasing order according to their parameters X_1.
2.	The task can be assigned to a processor if the total utilization of T_i and the x tasks already scheduled on that processor is equal or less than $U_{RM}(x + 1)$.	The following schedulability condition, which follows from theorem, is used : $u_i + u_k \leq \max(ln2, 1 - \xi_k ln2)$.
3.	RMFF algorithm does not exploit the fact that the schedulable utilization of	RMST algorithm exploits this fact and is based on the theorem which states that the schedulable

	tasks on a processor is higher, when the tasks are closer to being simply periodic.	utilization $U_{RM}(\xi, n)$ of the RM algorithm as a function of ξ and n is given by : $U_{RM}(n, \xi) = (n-1)(2^{\xi/n-1} - 1) + 2^{1-\xi} - 1$ for $\xi \leq 1 - 1/n$ $U_{RM}(n, \xi) = n(2^{1/n} - 1)$ for $\xi \geq 1 - 1/n$

Que 3.18. Describe multiprocessor priority-ceiling protocol with suitable example.

Answer

1. MPCP (Multiprocessor Priority-Ceiling Protocol) assumes that tasks and resources have been assigned and statically bound to processors and that the scheduler of every synchronization processor knows the priorities and resource requirements of all the tasks requiring the global resources managed by the processor.
2. According to MPCP, the scheduler of each processor schedules all the local tasks and global critical sections on the processor on a fixed priority basis and except for the modification, controls their resource accesses according to the basic priority-ceiling protocol.
3. According to the MPCP model, when a task uses a global resource, its global critical section executes on the synchronization processor of the resource.
4. If the global critical section of a remote task were to have a lower priority than some local tasks 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.
5. To prevent this, the MPCP schedules all global critical sections at higher priorities than all the local tasks on every synchronization processor.
6. This can be easily implemented in a system where the lowest priority π_{lowest} of all tasks is known.
7. The scheduler of each synchronization processor schedules the global critical sections of a task with priority π_i at priority $\pi_i - \pi_{lowest}$.
8. For example, in a system where tasks have priorities 1 through 5, π_{lowest} is 5.
9. A global critical section of a task with priority 5 is scheduled at priority 0, which is higher than priority 1.
10. Similarly, the priority of a global critical section of a task with priority 1 is –4, which is the highest priority in the system.

Que 3.19. Define task assignment problem in real time systems. Discuss any two assignment schemes for periodic tasks.

Answer

Task assignment problem :

1. In a real time system, the application system is partitioned into modules and then modules are assigned and bound to processors.
2. This partition of system and assignment of modules is known as task assignment. Task assignment is done offline.
3. At some stage, execution times, resource requirements, data and control dependencies and timing constraints of all the tasks become known.
4. Then task assignment is done to determine how many processors of each type are needed, how the tasks should be partitioned and on which processor each module executes.
5. The task assignment algorithm are based on the following formulations :
   a. Ignoring both the costs of communication and the placement of resources.
   b. Considering the communication costs.
   c. Considering both communication costs and resource-access costs.

Task assignment schemes :

1. **Task assignment based on execution time requirements :**
   a. In this we consider only the processing time requirements of the tasks and ignore the communication costs.
   b. For example, when all tasks communicate via shared memory, the communication costs of individual tasks are independent of where the tasks execute.
   c. For some applications, it is possible to provide sufficient number of memory modules and carefully layout the address space of tasks to minimize memory contention.
2. **Task assignment to minimize total communication cost :**
   a. When CPUs are connected in a network, the cost of communication between a pair of tasks is usually significantly lower when they are on the same CPU than when they are on different CPUs.
   b. The goal of task assignment is to partition all the tasks in the system into modules and assign each module to a processor in such a way that the number of required processors to schedule all the tasks is minimum and some function of communication costs among tasks in different modules is minimum.

Types of task assignment algorithms :

1. **Utilization balancing algorithm :**
   a. It maintains the tasks in a queue in increasing order of their utilizations. It removes tasks one by one from the head of the queue and assigns them to the least utilized processor each time.
   b. In this algorithm, there is balanced utilization of all the different processors.

c. In a perfectly balanced system, the utilization at each processor equals the overall utilization of the processors of the system.

d. This algorithm is suitable when the number of processors in a multiprocessor is fixed.

2. **Next-fit algorithm :**

a. In this, a task set is partitioned and each partition is scheduled on a uniprocessor.

b. Unlike the utilization balancing algorithm, this algorithm does not require the number of processors of the system to be predetermined.

c. It classifies the different tasks into a few classes based on the utilization of the task.

d. One or more processors are assigned to each class of tasks.

e. The tasks with similar utilization values are scheduled on the same processor.

f. If the tasks are to be divided into m classes, a task T_i belongs to class j, $0 \le j \le m$, if $(2^{1/j+1} - 1) < e_i / p_i \le (2^{1/j} - 1)$.

To partition the tasks into four classes, different classes can be formulated as :

Class 1 : $(2^{1/2} - 1) < C_1 \le (2^{1/1} - 1)$

Class 2 : $(2^{1/3} - 1) < C_2 \le (2^{1/2} - 1)$

Class 3 : $(2^{1/4} - 1) < C_3 \le (2^{1/3} - 1)$

Class 4 : $0 < C_4 \le (2^{1/4} - 1)$

3. **Bin packing algorithm :**

a. In this algorithm, tasks are assigned to the processors such that the total utilization of all the tasks on the individual processors is less than 1.

b. There are two bin packing algorithms; the first fit random algorithm and the first fit decreasing algorithm.

c. In the first fit random algorithm, tasks are selected randomly and assigned to processors in an arbitrary manner until the utilization of a processor does not exceed 1.

d. In the first fit decreasing algorithm, the tasks are sorted in decreasing order of their utilization in a list.

e. The tasks are selected one by one from the list and assigned to the bin (processor) to which it fits first.

Que 3.20. Allocate the following task set according to next fit and bin packing algorithms.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
e_i	5	7	3	1	10	16	1	3	9	17	21
p_i	10	21	22	24	30	40	50	55	70	90	95

Having $M = 4$ classes with bounds $C_1 = (0.48, 1]$, $C_2 = (0.32, 0.48]$, $C_3 = (0.22, 0.32]$ and $C_4 = (0.0, 0.22]$.

Answer

First, we have to determine the utilization of the different tasks.

$$\text{For } T_1, u_1 = \frac{e_1}{p_1} = \frac{5}{10} = 0.5$$

$$T_2, u_2 = \frac{7}{21} = 0.33$$

$$T_3, u_3 = \frac{3}{22} = 0.136$$

$$T_4, u_4 = \frac{1}{24} = 0.041$$

$$T_5, u_5 = \frac{10}{30} = 0.33$$

$$T_6, u_6 = \frac{16}{40} = 0.4$$

$$T_7, u_7 = \frac{1}{50} = 0.02$$

$$T_8, u_8 = \frac{3}{55} = 0.054$$

$$T_9, u_9 = \frac{9}{70} = 0.128$$

$$T_{10}, u_{10} = \frac{17}{90} = 0.19$$

$$T_{11}, u_{11} = \frac{21}{95} = 0.22$$

Next fit :

Now for $C_1 = (0.48, 1]$

$u_1 = 0.5$, hence T_1 will be allocated to class C_1

for $C_2 = (0.32, 0.48]$

$u_2 = 0.33$

$u_5 = 0.33$

$u_6 = 0.4$, hence T_2, T_5, T_6 will be allocated to class C_2

for $C_3 = (0.22, 0.32]$

None of the utilization belongs to this range. Hence, no any task will be allocated to this class.

for $C_4 = (0.0, 0.22]$

$u_3 = 0.136$

$u_4 = 0.041$

$u_7 = 0.02$

$u_8 = 0.054$

$u_9 = 0.128$
$u_{10} = 0.19$
$u_{11} = 0.22,$

Hence $T_3, T_4, T_7, T_8, T_9, T_{10}$ and T_{11} will be allocated to class C_4.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
e_i	5	7	3	1	10	16	1	3	9	17	21
p_i	10	21	22	24	30	40	50	55	70	90	95
$u(i)$	0.5	0.33	0.14	0.04	0.33	0.40	0.02	0.05	0.13	0.19	0.22
Class	C_1	C_2	C_4	C_4	C_2	C_2	C_4	C_4	C_4	C_4	C_4

a. Let processor P_i is reserved for tasks in class C_i, $1 \le i \le 4$. T_1 is assigned to P_1, T_2 to P_2 and T_3 to P_4.

b. $T_4 \in C_4$ and since $\{T_3, T_4\}$ is RM schedulable on the same processor, we assign T_4 also to P_4.

c. $T_5 \in C_2$ and since $\{T_2, T_5\}$ is RM schedulable on the same processor, we assign T_5 also to P_2. $T_6 \in C_2$, however $\{T_2, T_5, T_6\}$ is not RM schedulable on the same processor, so we assign additional processor P_3 ($\because$ no any task is in class C_3) to C_2 tasks and assign T_6 to P_3. $T_7 \in C_4$ and $\{T_3, T_4, T_7\}$ is RM schedulable on the same processor, so we assign it to P_4.

d. Similarly T_8, T_9, T_{10} and T_{11} are RM schedulable on processor P_4, so these tasks are also assigned to P_4.

Processor	Tasks
P_1	T_1
P_2	T_2, T_5
P_3	T_{11}
P_4	$T_3, T_4, T_7, T_8, T_9, T_{10}, T_{11}$

Bin packing :

We have to sort all the tasks according to their decreasing utilization.

T_1	T_6	T_2	T_5	T_{11}	T_{10}	T_3	T_9	T_8	T_4	T_7
0.5	0.4	0.33	0.33	0.22	0.19	0.14	0.13	0.05	0.04	0.02

The ordered list is $L = (T_1, T_6, T_2, T_5, T_{11}, T_{10}, T_3, T_9, T_8, T_4, T_7)$. The assignment process is done in Table 3.20.1 :

Table 3.20.1

S. No.	Task T_i	$u(i)$	Assigned to	(P_1, P_2, P_3)
1.	T_1	0.5	P_1	(0.50)
2.	T_6	0.4	P_1	(0.90)
3.	T_2	0.33	P_2	(0.90, 0.33)
4.	T_5	0.33	P_2	(0.90, 0.66)
5.	T_{11}	0.22	P_2	(0.90, 0.88)
6.	T_{10}	0.19	P_3	(0.90, 0.88, 0.19)
7.	T_3	0.14	P_3	(0.90, 0.88, 0.33)
8.	T_9	0.13	P_3	(0.90, 0.88, 0.46)
9.	T_8	0.05	P_1	(0.95, 0.88, 0.46)
10.	T_4	0.04	P_1	(0.99, 0.88, 0.46)
11.	T_7	0.02	P_2	(0.99, 0.90, 0.46)

Que 3.21. Define fixed priority end-to-end periodic tasks and further discuss the schedulability criterion of non-greedy synchronized tasks.

Answer

1. When there are many processors and each processor has its own scheduler then the type of the task is something different.
2. In a simple processor, jobs may have their individual response time, deadlines and precedence constraints.
3. But in multiprocessor, each task may have chain of subtasks in the precedence graph.
4. An end-to-end task T_i is periodic with period p_i if a chain of $n(i)$ jobs is released every p_i units of time and the jobs in the chain executes in turn on processors according to the visit sequence $(V_{i,1}, V_{i,2}, \ldots V_{i,n(i)})$ where $V_{i,k} = P_j$ means that the kth subtask, $T_{i,k}$ executes on processor P_j.
5. The first subtask $T_{i,1}$ is a periodic task with period p_i, the period of the parent task T_i.

Schedulability of fixed priority end-to-end periodic tasks :

1. In this, we consider that every subtask $T_{i,k}$ in the system is assigned a fixed priority $\pi_{i,k}$ and check whether every task can meet its end-to-end deadlines.
2. This is done by considering two cases; when tasks are synchronized non-greedily and greedily.

Schedulability of non-greedy synchronized tasks :

1. It is easier to determine the schedulability of tasks synchronized according to non-greedy protocol.
2. This is primary design objective of these protocols.

a. **Upper bounds to end-to-end response times :**

i. When sibling subtasks are synchronized according to any of the non-greedy synchronization protocols, we can treat each subtask $T_{i,k}$ like a periodic task when trying to find its maximum possible response time.

ii. The period of this subtask is equal to the period p_i of its parent task T_i and its execution time is $e_{i,k}$.

iii. An upper bound W_i to end-to-end response time of any periodic task T_i in a fixed priority system synchronized according to the MPM protocol and is given by :

$$W_i = \sum_{k=1}^{n(i)} W_{i,k}$$

where $n(i)$ is the number of subtasks in T_i and the $W_{i,k}$ is the upper bound to the response time of every subtask.

b. **Upper bounds to response times of subtasks :**

i. The problem of finding an upper bound to the end-to-end response time of a task T_i whose subtasks are synchronized non-greedily is just the finding upper bounds to the response times of its subtasks on all processors in its visit sequence.

ii. The most straightforward approach to bounding the response time of each subtask $T_{i,k}$ is to treat all the subtasks on the processor $V_{i,k}$ where $T_{i,k}$ execute as independent periodic tasks.

iii. Then, $W_{i,k}$ can be found by doing a time demand analysis on the subtask $T_{i,k}$.

Que 3.22. Discuss temporal distance model and hence, explain distance constraint monotonic algorithm.

Answer

Temporal distance model :

i. We use a task T_i that is a chain of jobs $J_{i,k}$, for $k = 1, 2, 3...$ Let ϕ_i denote the release time of T_i.

ii. This means that the first job $J_{i,1}$ in T_i is ready for execution at ϕ_i and each subsequent job $J_{i,k+1}$ $(k \geq 1)$ becomes ready when its immediate predecessor $J_{i,k}$ completes.

iii. Here, T_i is an end-to-end task and ϕ_i is its phase.

iv. Let $f_{i,k}$ denote the completion time of the kth job $J_{i,k}$ according to some schedule.

v. The temporal distance between this job $J_{i,k}$ and the next job $j_{i,k+1}$ in T_i is the difference $f_{i,k+1} - f_{i,k}$ between their completion times.

vi. The distance constraint of task T_i is C_i if the completion times of any two consecutive jobs in the task are required to satisfy the inequalities.

$$f_{i,1} - \phi_i \leq C_i$$
$$f_{i,k+1} - f_{i,k} \leq C_i \text{ for } k = 1, 2, ...$$

vii. If the completion times of all the jobs in T_i according to a schedule satisfy these inequalities then T_i meets its distance constraint C_i.

Distance constraint monotonic (DCM) algorithm :

1. This algorithm does not preempt jobs arbitrarily.
2. The algorithm has two elements, priority assignment and separation constraint.

a. Priority assignment :

i. The algorithm is priority driven and assigns fixed priorities to tasks on the basis of their temporal distance constraint.

ii. The smaller the distance constraint C_i of a task T_i, the higher the task's priority.

iii. Therefore, tasks with indices $i - 1$ or have less higher priorities than T_i for all $i = 1, 2, ..., n$.

b. Separation constraint :

i. If the successor job $J_{i,k+1}$ in a task T_i is ready as soon as its immediate predecessor job $J_{i,k}$ completes, it might leave no time for lower priority tasks.

ii. This is why the scheduler imposes a separation constraint between consecutive jobs in each task T_i.

iii. The separation constraint between two consecutive jobs $J_{i,k}$ and $J_{i,k+1}$ in T_i is the minimum length of time between the completion time $f_{i,k}$ of a job $J_{i,k}$ and the ready time $r_{i,k+1}$ of its immediate successor $J_{i,k+1}$.

3. According to DCM algorithm, the separation constraint imposed by the scheduler on jobs in each task T_i is equal to $C_i - W_i$ where W_i is the maximum response time of jobs in T_i.
4. The scheduler computes the ready times of jobs in T_i according to,

$$r_{i,1} = \phi;$$
$$r_{i,k+1} = f_{i,k} + C_i - W_i \text{ for } k \geq 1$$

5. There is an assumption that $W_i \leq C_i$.
6. This condition must hold, otherwise, it is impossible for T_i to meet its distance constraint C_i.
7. The maximum response time W_i of all jobs in each task T_i can be obtained as follows :

a. Find the maximum response time W_1 of the highest priority task T_1.

b. For each task T_i for $i > 1$, find W_i after the maximum response times of all higher priority tasks have been found by constructing a DCM schedule of i tasks $T_1, T_2, ... T_i$, assuming that every task was released at time 0, *i.e.*, $\phi_k = 0$ for all $1 \leq k \leq i$. W_i is equal to the response time of the first job in T_i according to this schedule.