LOVELY PROFESSIONAL UNIVERSITY

REAL TIME SYSTEMS

Edited By

Gurwinder Kaur

Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

CONTENT

Unit 1: Concept of Real-time System 1
Gurwinder Kaur, Lovely Professional University

Unit 2: Introduction to Real-time Applications 12
Gurwinder Kaur, Lovely Professional University

Unit 3: Hard Versus Soft Real-time System 30
Gurwinder Kaur, Lovely Professional University

Unit 4: A Reference Model of Real-time Systems 40
Gurwinder Kaur, Lovely Professional University

Unit 5: Real-time System Dependencies 49
Gurwinder Kaur, Lovely Professional University

Unit 6: Commonly used Approaches to Real-time Scheduling 60
Sandeep Kumar, Lovely Professional University

Unit 7: Commonly used Algorithm to Real-time Scheduling 68
Sandeep Kumar, Lovely Professional University

Unit 8; Working of Real-time Scheduling 81
Sandeep Kumar, Lovely Professional University

Unit 9: Concept of Clock-driven Scheduling 88
Sandeep Kumar, Lovely Professional University

Unit 10: Working of Clock-driven Scheduling 96
Sandeep Kumar, Lovely Professional University

Unit 11: Application of Clock-driven Scheduling 104
Sandeep Kumar, Lovely Professional University

Unit 12: Priority-driven Scheduling of Periodic Tasks 115
Gurwinder Kaur, Lovely Professional University

Unit 13: Working of Priority-driven Scheduling of Periodic Tasks 125
Gurwinder Kaur, Lovely Professional University

Unit 14: Advance Priority-driven Scheduling of Periodic Tasks 133
Gurwinder Kaur, Lovely Professional University

SYLLABUS

Real Time Systems

Objectives: To enable the students to understand the technicalities of Real Time Systems. Student can identify real time tasks and their criticalness. Student will learn various real time task scheduling techniques.

S. No.	Description
1.	**Introduction to Real Time Applications:** Digital Control, High Levels Control, Signal Processing, Other Real Time Applications.
2.	**Hard Versus Soft Real-Time System:** Jobs and Processors, Release Time, Deadline and Timing Constraints, Hard and Soft Timing Constraints, Hard Real Time Systems, Soft Real Time Systems.
3.	**A Reference Model of Real Time System:** Processors and Resources, Temporal Parameters of Real Time Model, Precedence Constraints and Data Dependencies.
4.	Other Types of Dependences, Functional Parameters, Resource Parameters of Jobs and Parameters of Resources, Scheduling Hierarchy.
5.	**Commonly used Approaches to Real Time Scheduling:** Clock-Driven Approach, Weight Round-Robin Approach, Priority-Driven Approach, Dynamic versus Static system, Effective Release Times and Deadlines.
6.	**Commonly used Approaches to Real Time Scheduling:** Optimality of the EDF and LST Algorithm, Nonoptimility of the EDF and the LST Algorithm, Challenges in Validating Timing Constraints in Priority-Driven System, Off-Line versus On –Line Scheduling.
7.	**Clock-Driven Scheduling:** Notations and Assumptions, Static, Timer-Driven Scheduler, General Structure of Cyclic Scheduler, Cyclic Scheduling.
8.	**Clock-Driven Scheduling:** Improving the Average Response Time of Aperiodic jobs, Scheduling Sporadic Jobs, Practical Consideration and Generalizations, Algorithm for Constructing Static Schedules, Pros and Cons of Clock Driven Scheduling.
9.	**Priority Driven Scheduling of Periodic Tasks:** Static Assumptions, Fixed Priority versus Dynamic Priority Algorithms, Maximum Schedulable Utilization, Optimality of the RM and DM Algorithms, A Schedulability Test for Fixed-Priority Tasks with Short Response Time.
10.	**Priority Driven Scheduling of Periodic Tasks:** Schedulability Test for Fixed--Priority Tasks with Arbitrary Response Time, Sufficient Schedulability conditions for the RM and DM Algorithm, Practical Factors

Notes

Unit 1: Concept of Real-time System

CONTENTS

Objectives

Introduction

1.1 Real-time Systems

1.2 Hard Real-time Systems

1.3 Soft Real-time Systems

1.4 Hard Versus Soft Real-time System

1.5 Summary

1.6 Keywords

1.7 Review Questions

1.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe the Utility of an Operating System
- Define the Term Real-time System
- Understand the Structure and Components of Real-time System
- Explain Hard and Soft Real-time System
- Describe the Differences between Hard and Soft Real-time System

Introduction

An Operating System (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard. To be considered "real-time", an operating system must have a known maximum time for each of the critical operations that it performs (or at least be able to guarantee that maximum most of the time). Some of these operations include OS calls and interrupt handling. Operating systems that can absolutely guarantee a maximum time for these operations are commonly referred to as "hard real-time", while operating systems that can only guarantee a maximum most of the time are referred to as "soft real-time". In practice, these strict categories have limited usefulness – each RTOS solution demonstrates unique performance characteristics and the user should carefully investigate these characteristics.

1.1 Real-time Systems

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer *after* a certain

Notes

deadline: it is either by the deadline or it is useless! Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some event in the physical world, and the output has to relate to that same event.

Caution The lag from input time to output time must be sufficiently small for acceptable timeliness.

Real-Time systems span several domains of computer science. They are defence and space systems, networked multimedia systems, embedded automotive electronics etc. In a real-time system the correctness of the system behaviour builds upon not only the logical results of the computations, but also on the physical instant at which these results are produced. A real-time system changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped. Based on this a real-time system can be decomposed into a set of subsystems i.e., the controlled object, the real-time computer system and the human operator. A real-time computer system must react to stimuli from the controlled object (or the operator) within time intervals dictated by its environment. The instant at which a result is produced is called a deadline. If the result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm. If a catastrophe could result if a firm deadline is missed, the deadline is hard.

Real-Time systems are becoming pervasive. Typical examples of real-time systems include Air Traffic Control Systems, Networked Multimedia Systems, and Command Control Systems etc. In a Real-Time System the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. Real-Time systems are classified from a number of viewpoints i.e. on factors outside the computer system and factors inside the computer system. Special emphasis is placed on hard and soft real-time systems. A missed deadline in hard real-time systems is catastrophic and in soft real-time systems it can lead to a significant loss. Hence predictability of the system behaviour is the most important concern in these systems. Predictability is often achieved by either static or dynamic scheduling of real-time tasks to meet their deadlines. Static scheduling makes scheduling decisions at compile time and is off-line. Dynamic scheduling is online and uses schedulability test to determine whether a set of tasks can meet their deadlines. The present paper talks about static and dynamic scheduling algorithms and operating systems support for these mechanisms.

Example: ABS, aircraft control, ticket reservation system at airport, over-temperature monitor in nuclear power station, mobile phone, oven temperature controller, Doppler blood-flow monitor, ECG/arrhythmia monitor.

Although there is no clear dividing line between real-time and non-real-time systems, there are a set of distinguishing features (listed below) which can assist with an outline classification schema to identify real-time applications.

Timing: The most common definition of a real-time system involves a statement similar to 'Real-time systems are required to compute and deliver correct results within a specified period of time.' Does this mean that a non-real-time system such as a payroll program, could print salary cheques two years late, and be forgiven because it was not a real-time system? Hardly so! Obviously there are time constraints on non-real-time systems too. There are even circumstances in which the early delivery of a result could generate more problems than lateness of delivery. A premature newspaper obituary could sometimes create as much havoc as an early green on a traffic light controller.

Interrupt driven: After the requirement for maximum response delay times, the next characteristic of real-time systems is their involvement with events. These often manifest themselves in terms

Notes

of interrupt signals arising from the arrival of data at an input port, or the ticking of a hardware clock, or an error status alarm. Because real-time systems are often closely coupled with special equipment (a situation that is termed 'embedded') the programmer has also to gain a reasonable understanding of the hardware if the project is to be a thorough success. Once again, however, the demarcation between traditional data processing and real-time systems is not easy to draw because event-driven GUI interfaces are so widely used within all desktop applications.

Low-level programming: The C language is still most popular for writing device drivers for new hardware. But because high-level languages, including C, do not generally have the necessary instructions to handle interrupt processing, it has been common for programmers to drop down to assembler level to carry out this type of coding. Because ASM and C are classified as low-level languages by many programmers, who may be more familiar with database systems and windowing interfaces, it has been suggested as a distinguishing characteristic of real-time programmers that they prefer to use low-level languages. This can be seen as somewhat misleading, when the real-time high-level languages Modula-2 and ADA are taken into consideration.

Specialized hardware: Most real-time systems work within; or at least close beside, specialized electronic and mechanical devices. Unfortunately, to make matters more difficult, during development these are often only prototype models, with some doubt surrounding their functionality and reliability. This is especially true for small embedded microcontrollers which may even be required to perform as critical component parts within a feedback control loop. The oven power controller illustrated below could employ an integrated microcontroller to monitor the oven temperature and adjust the electrical power accordingly. Such applications place a heavy responsibility on the programmer to fully understand the functional role of the software and its contribution to the feedback delay which governs the system response. Code may have to run synchronously with the hardware or other software systems, such as when telephone transmissions are sequenced 8000 times a second to maintain acceptable voice quality. Very often this leads the programmer into other disciplines: electrical theory, mechanics, acoustics, physiology or optics. Real-time programmers rarely have a routine day.

Volatile data: I/O Another special issue for real-time software concerns 'volatile data'. These are variables which change their value from moment to moment, due to the action of external devices or agents, through interrupts or DMA. This is distinguished from the situation where input data is obtained from a disk file, or from the keyboard under program control. The most common example encountered by real-time programmers involves input channels which operate autonomously to bring in new values for memory variables when data arrives at an input port. The software must then be structured to check for changes at the correct rate, so as not to miss a data update.

Multi-tasking: Real-time systems are often expected to involve multitasking. In this situation, several processes or tasks cooperate to carry out the overall job. When considering this arrangement, there should be a clear distinction drawn between the static aggregation of groups of instructions into functions for compilation, and the dynamic sequencing of tasks which takes place at run-time. It has already been suggested that full multi-tasking is not always necessary, but it can be positively advantageous to programmers in simplifying their work. It is also widely accepted that many computer systems have become so complex that it has become necessary to decompose them into components to help people to understand and build them. In the traditional data processing field, for example, the production of invoices from monthly accounts requires several distinct operations to be carried out. These can be sequenced, one after the other, in separate phases of processing. With real-time systems this is rarely possible; the only way to partition the work is to run components in parallel, or concurrently. Multi-tasking provides one technique which can assist programmers to partition their systems into manageable components which have delegated responsibility to carry out some part of the complete activity.

Notes

Thus, multi-tasking, although generally seen as an implementation strategy, can also offer an intellectual tool to aid the designer.

Run-time scheduling: The separation of an activity into several distinct, semi-autonomous tasks leads to the question of task sequencing. In traditional DP applications the sequence planning is largely done by the programmer. Functions are called in the correct order and the activity is completed. But for real-time systems this is only half the story. The major part of sequencing takes place at run-time, and is accomplished by the operating system through the action of the scheduler. It is as if the sequencing decisions have been deferred, it is a kind of 'late sequencing', to draw a parallel with the established term 'late binding', used with regard to code linking. This is perhaps the most interesting feature of real-time systems. The manner in which the various activities are evoked in the correct order is quite different from that of a traditional DP system which normally relies on the arrival of data records from an input file to sequence the functions, and so it is predetermined and fixed.

Task Prepare a chart to show major parts of sequencing.

Unpredictability: Being event driven, real-time systems are at the mercy of unpredictable changes in their environments. It is just not feasible to anticipate with 100 per cent certainty all the permutations of situations which may arise. In my experience, the worst offenders are actually the human users, who seem totally unable, or unwilling, to understand what the designer intended. Any choice offered by a menu or sequence of YES/NO alternatives will soon reveal unexpected outcomes during field trials. The exact ordering or sequencing of all the functions which deals with these interactions has to be decided at run-time by the scheduler, giving much more flexibility in response. Considerable effort is now put into extensive simulation testing in order to trap as many of these bugs as possible, even before the designs are released.

Notes **Life-critical Code**

Although not always the case, real-time systems can involve serious risk. A failure to run correctly may result in death or at least injury to the user and others. Such applications are becoming more and more common, with the aircraft and automobile industries converting their products to 'fly by wire' processor technology. This removes from the driver/pilot all direct, muscular control over the physical mechanism, relying entirely on digital control systems to carry out their commands. The burden of developing real-time, life-critical software, with all the extra checking, documentation and acceptance trials required, may raise the cost beyond normal commercial projects, of similar code complexity, by an astonishing factor of 30. Most real-time applications are intended to run continuously, or at least until the user turns off the power. Telephone exchanges, for example, contain millions of lines of real-time code, and are expected to run non-stop for 20 years.

The increasing use of embedded microprocessors within medical monitoring and life-support equipment, such as radiological scanners and drug infusion pumps, makes consideration of software reliability and systems integrity even more urgent. Some research effort has been expended in devising a method to formally prove correct a computer program, much in the same way that mathematicians deal with algebraic proofs. So far, the products resulting from this work have not generated much commercial interest.

Notes

1.2 Hard Real-time Systems

A hard real-time task is one that is constrained to produce its results within certain predefined time bounds. The system is considered to have failed whenever any of its hard real-time tasks does not produce its required results before the specified time bound.

When a process is considered hard real-time, it must complete its operation by a specific time. If it fails to meet its deadline, its operation is without value and the system for which it is a component could face failure. When a system is considered soft real-time, however, there is some room for lateness. For example, in such a system, a delayed process may not cause the entire system to fail. Instead, it may lead to a decrease in the usual quality of the process or system.

Hard real-time systems: These are often used in embedded systems. Consider, for example, a car engine control system. Such a system is considered hard because a late process could cause the engine to fail.

An example of a system having hard real-time tasks is a robot. The robot cyclically carries out a number of activities including communication with the host system, logging all completed activities, sensing the environment to detect any obstacles present, tracking the objects of interest, path planning, effecting next move, etc. Now consider that the robot suddenly encounters an obstacle. The robot must detect it and as soon as possible try to escape colliding with it. If it fails to respond to it quickly (i.e. the concerned tasks are not completed before the required time bound) then it would collide with the obstacle and the robot would be considered to have failed. Therefore detecting obstacles and reacting to it are hard real-time tasks.

Another application having hard real-time tasks is an anti-missile system. An anti-missile system consists of the following critical activities (tasks). An anti-missile system must first detect all incoming missiles, properly position the anti-missile gun, and then fire to destroy the incoming missile before the incoming missile can do any damage. All these tasks are hard real-time in nature and the anti-missile system would be considered to have failed, if any of its tasks fails to complete before the corresponding deadlines.

Applications having hard real-time tasks are typically safety-critical. This means that any failure of a real-time task, including its failure to meet the associated deadlines, would result in severe consequences. This makes hard real-time tasks extremely critical. Criticality of a task can range from extremely critical to not so critical. Task criticality therefore is a different dimension than hard or soft characterization of a task. Criticality of a task is a measure of the cost of a failure – the higher the cost of failure, the more critical is the task.

Did u know? For hard real-time tasks in practical systems, the time bounds usually range from several micro seconds to a few milliseconds. It may be noted that a hard real-time task does not need to be completed within the shortest time possible, but it is merely required that the task must complete within the specified time bound. In other words, there is no reward in completing a hard real-time task much ahead of its deadline.

Self Assessment

Fill in the blanks:

1. An …… system is responsible for managing the hardware resources of a computer and hosting applications that run on the computer.

Notes

2. An operating system must have a known maximum …… for each of the critical operations that it performs.
3. A …… system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure.
4. Real-time systems are often expected to involve ……
5. The separation of an activity into several distinct, semi-autonomous tasks leads to the question of task ……
6. Hard real-time systems are often used in …… systems.

1.3 Soft Real-time Systems

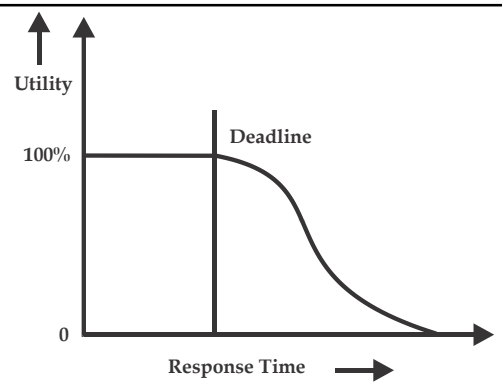
Soft real-time systems also have time bounds associated with them. However, unlike hard and firm real-time tasks, the timing constraints on soft real-time tasks are not expressed as absolute values. Instead, the constraints are expressed either in terms of the average response times required.

Soft real-time systems are usually employed when there are multiple, connected systems that must be maintained despite shifting events and circumstances. These systems are also used when concurrent access requirements are present. For example, the software used to maintain travel schedules for major transportation companies are often soft real-time. It is necessary for such software to update schedules with little delay. However, a delay of a few seconds is not likely to cause mayhem.

An example of a soft real-time task is web browsing. Normally, after an URL (Uniform Resource Locater) is clicked, the corresponding web page is fetched and displayed within a couple of seconds on the average. 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.

Another example of a soft real-time task is a task handling a request for a seat reservation in a railway reservation application. Once a request for reservation is made, the response should occur within 20 seconds on the average. The response may either be in the form of a printed ticket or an apology message on account of unavailability of seats. Alternatively, we might state the constraint on the ticketing task as: At least in case of 95% of reservation requests, the ticket should be processed and printed in less than 20 seconds.

Figure 1.1: Utility of the Results Produced by a Soft Real-Time Task as a Function of Time



Source: http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-28.pdf

Notes

Let us now analyse the impact of the failure of a soft real-time task to meet its deadline, by taking the example of the railway reservation task. If the ticket is printed in about 20 seconds, we feel that the system is working fine and get a feel of having obtained instant results. As already stated, missed deadlines of soft real-time tasks do not result in system failures. However, the utility of the results produced by a soft real-time task falls continuously with time after the expiry of the deadline as shown in Fig. 1.1. In Fig. 1.1, the utility of the results produced are 100% if produced before the deadline, and after the deadline is passed the utility of the results slowly falls off with time. For soft real-time tasks that typically occur in practical applications, the time bounds usually range from a fraction of a second to a few seconds.

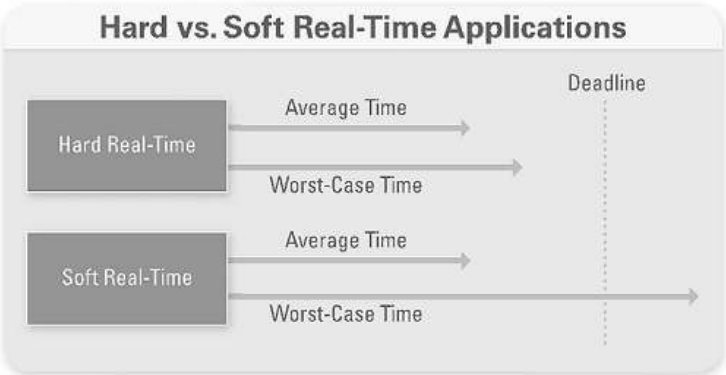
1.4 Hard Versus Soft Real-time System

An OS that can absolutely guarantee a maximum time for the operations it performs is referred to as hard real-time. In contrast, an OS that can usually perform operations in a certain time is referred to as soft real-time.

Lab Exercise Go to Visit http://www.dauniv.ac.in/downloads/EmbsysRevEd_PPTs/Chap_8Lesson13EmsysNewHardSoftRTConsideration.pdf and collect more information on Hard and soft real-time systems.

Hard real-time systems are designed to absolutely guarantee that a task will execute within a certain worst-case timeframe. Therefore, for projects involving safety or systems that could result in a large investment in the event of failure, hard real-time is often a requirement. On the other hand, soft real-time systems are designed to satisfy your timing requirements most of the time but without absolute certainty. This can be acceptable for operations like video processing, where a lost data frame is not good but may not necessarily be a critical problem.

Figure 1.2: Hard vs Soft Real-time Applications



Hard real-time systems guarantee (when programmed correctly) that a deadline will be consistently met, while soft real-time systems may periodically exceed the deadline.

Self Assessment

State whether the following statements are True or False:

7. Hard real-time systems are usually employed when there are multiple, connected systems that must be maintained despite shifting events and circumstances.

Notes

8. An OS that can absolutely guarantee a maximum time for the operations it performs is referred to as hard real-time.
9. Soft real-time systems are designed to absolutely guarantee that a task will execute within a certain worst-case timeframe.
10. For projects involving safety or systems that could result in a large investment in the event of failure, hard real-time is often a requirement.
11. Soft real-time systems are designed to satisfy your timing requirements most of the time but without absolute certainty.
12. Hard real-time systems guarantee (when programmed correctly) that a deadline will be consistently met, while soft real-time systems may periodically exceed the deadline.

Case Study

Correct Development of Real-time Embedded Systems in UML

OMEGA will develop a methodology and tools for the development of real-time and embedded systems using UML, based on a clean semantics of the different architectural viewpoints and their relations. The aim of the project is to increase the efficiency and competitiveness of the European software industry by providing tools improving the quality of software while reducing the expense of the validation phase. The OMEGA approach to software quality is to use UML for the description of a unique reference model, from which are derived semantically related models for functional, validation, performance analysis, as well as implementations; all evolutions are reported in the reference model for tracking of its influence. A semantically sound component based development plays an important role, which makes sure that interfaces are sufficient to guarantee the requirements.

Objectives

OMEGA aims at the definition of a development methodology in UML for embedded and real-time systems based on formal techniques and used to improve commercially available UML tools. For this purpose we will identify reasonable and effective subsets of UML for real-time, as well as necessary extensions. Provide formal foundations, methods and tools for compositional verification of real-time systems within UML. Construct a development methodology based on the UML modelling and specification capabilities and the verification methods and tools developed in the project. Apply industrial case studies for evaluating the proposed methodology and verification tools. Work description:

To achieve our aim, we will develop results in the following interdependent directions:

1. ***Modelling and Specification Language:*** We select a small subset of UML notations that allow the design of reactive and real-time systems. If needed, we also propose small extensions. The resulting language contains notations to model the system under development including both functional and non-functional aspects, and specify the requirements to be met by the system.
2. ***Verification and Synthesis:*** We will adapt and extend existing formal verification technologies to UML, identify the new needs in verification techniques raised by the powerful structuring features of UML and develop compositional verification

Contd...

Notes

methods, allowing deriving properties of systems from properties of components. The techniques are connected to two industrial CASE tools, leading to two verification tool-sets. We will also develop tools that in certain cases directly synthesize systems satisfying required properties.

3. ***Development Methodology:*** We will develop a methodology, providing guidelines about the use and the combination of the different notations. In particular, the methodology will be based on refinement and property preservation rules, relating the different abstraction levels.

4. ***Technology Transfer:*** We will show how the developed results - theory, methods and tools – can be applied to real-time systems development by using appropriate extensions of commercially available tools. Our approach will be evaluated and adapted on the basis of four industrial case studies.

Milestones

1. Definition of a UML kernel model (KM): a minimal subset of UML for the development of real-time and embedded systems;
2. Semantic foundations of the KM;
3. Adaption of existing model-checking techniques to the KM for component verification;
4. Two integrated tool-sets for system verification based on compositional methods and synthesis; and
5. A development methodology based on semantic preserving notions of refinement.

Questions:

1. Describe the aims and objectives of development methodology in UML for embedded and real-time systems.
2. What additional can you comment about real-time embedded systems in UML?

Source: http://cordis.europa.eu/search/index.cfm?fuseaction=proj.document&PJ_RCN=5407653

1.5 Summary

- An operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer.
- An operating system must have a known maximum time for each of the critical operations that it performs.
- A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure.
- Real-time systems are often expected to involve multitasking.
- The separation of an activity into several distinct, semi-autonomous tasks leads to the question of task sequencing.
- Hard real-time systems are often used in embedded systems.
- Soft real-time systems are usually employed when there are multiple, connected systems that must be maintained despite shifting events and circumstances.

Notes

- An OS that can absolutely guarantee a maximum time for the operations it performs is referred to as hard real-time.
- Hard real-time systems are designed to absolutely guarantee that a task will execute within a certain worst-case timeframe.
- For projects involving safety or systems that could result in a large investment in the event of failure, hard real-time is often a requirement.
- Soft real-time systems are designed to satisfy your timing requirements most of the time but without absolute certainty.
- Hard real-time systems guarantee (when programmed correctly) that a deadline will be consistently met, while soft real-time systems may periodically exceed the deadline.

1.6 Keywords

Hard Real-time System: It is hardware or software that must operate within the confines of a stringent deadline.

Predictability: It is the condition of real-time system has to react to all possible events in a predictable way.

Real Time: The term is used to describe a number of different computer features. For example, real-time operating systems are systems that respond to input immediately.

Real-time System: It is a real-time system when it can support the execution of applications with time constraints on that execution.

Soft Real-time System: It can be a vending machine rising cost for lateness of results as it will take longer to treat a customer when the performance of the vending machine is degrading, less customers pay at this machine which results in less profits for the shop owner.

Timeliness: It is a condition of real-time system that to meet deadlines it is required that the application has to finish certain tasks within the time boundaries.

1.7 Review Questions

1. What do you understand by the term "real-time"?
2. Describe the structure and importance of real-time system.
3. Explain the concept of real-time operating system.
4. How is the concept of real-time different from the traditional notion of time? Explain your answer using a suitable example.
5. Discuss in brief hard and soft real-time system.
6. Identify the key differences between hard real-time, soft real time, and firm real-time systems.
7. Give an example of a soft real-time task and a non-real-time task.
8. Explain the key difference between the characteristics of soft real-time task and a non-real-time task.
9. Define timeline in real-time system.
10. What are timing constraints?

Notes

Answers: Self Assessment

1.	operating	2.	time
3.	real-time	4.	multitasking
5.	sequencing	6.	embedded
7.	False	8.	True
9.	False	10.	True
11.	True	12.	True

1.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real Time Systems*, McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben–Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

www.ece.cmu.edu/~koopman/des_s99/real_time/

www.csie.ntu.edu.tw/~d6526009/ucOS2/Chapter-2.pdf

www.algonet.se/~staffann/developer/realtimeintro.htm

www.microchip.com/stellent/groups/SiteComm_sg/.../en543053.pdf

Notes

Unit 2: Introduction to Real-time Applications

CONTENTS

Objectives
Introduction
2.1 Digital Control
2.2 High Level Controls
2.3 Signal Processing
2.3.1 Discrete-time Signals
2.3.2 Streams and Channels
2.3.3 Functions and Systems
2.4 Other Real-time Applications
2.4.1 Industrial Applications
2.4.2 Medical
2.4.3 Peripheral Equipment
2.4.4 Automotive and Transportation
2.4.5 Telecommunication Applications
2.4.6 Aerospace
2.4.7 Consumer Electronics
2.4.8 Defence Applications
2.4.9 Miscellaneous Applications
2.5 Summary
2.6 Keywords
2.7 Review Questions
2.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Digital Control
- Enumerate High Level Controls
- Explain Signal Processing
- Analyse Other Real-time Applications

Introduction

Real-time systems have of late, found applications in wide ranging areas. In the following, we list some of the prominent areas of application of real-time systems and in each identified case;

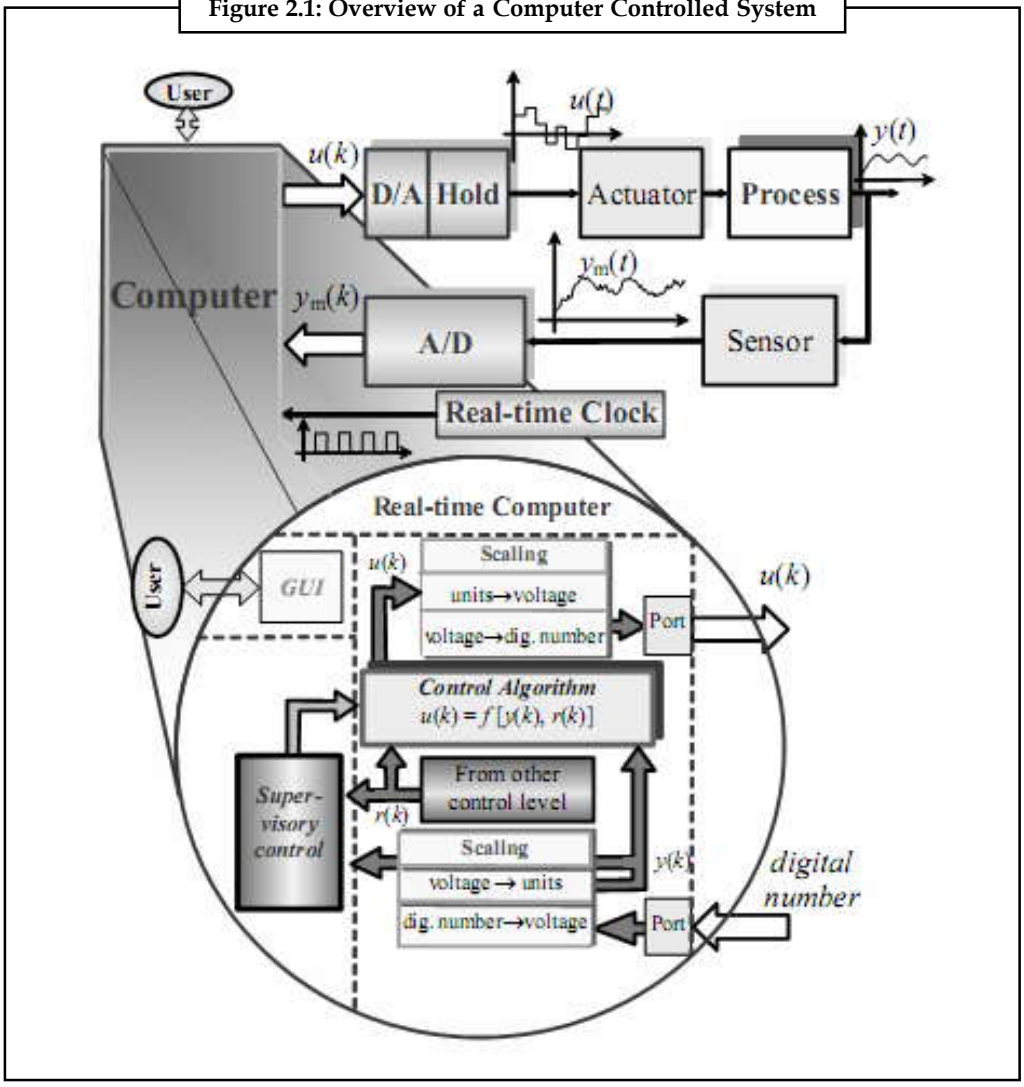
Notes

we discuss a few example applications in some detail. As we can imagine, the list would become very vast if we try to exhaustively list all areas of applications of real-time systems. We have therefore restricted our list to only a handful of areas, and out of these we have explained only a few selected applications to conserve space. In this unit, we will discuss some applications of real-time systems.

2.1 Digital Control

The introduction of digital computers in the control loop has allowed developing more flexible control systems including higher-level functions and advanced algorithms. Furthermore, most current complex control systems could not be implemented without the application of digital hardware. However, the simple sequence sensing–control–actuation for the classical feedback control becomes more complex as well. Nowadays, this sequence can be supplemented as follow: sensing–data acquisition–control law calculation–actuation–data base update. Figure 2.1 shows an overview of such control systems.

Figure 2.1: Overview of a Computer Controlled System



Source: http://nguyendangbinh.org/TinHocNangCao/RealTimeControlSystems_A_Tutorial.pdf

Notes

Thus, the control system now contains not only wired components but also algorithms, which must be programmed, i.e. software is now included in the control loop. This leads to new aspects to take into account by designing control systems:

1. Errors due to A/D and D/A conversion as well as due to limited length word calculations. This subject is well-treated in the literature. It is not meaningful to talk about guarantying real-time performance. It is true that occasionally time constraints can be relaxed without introducing additional problems in the control loop. This particularly applies to nice designed laboratory experiments. However, this actually depends on the application, and the time criticality should be proved for each individual case. On the other hand, real-time performance cannot be 100% guaranteed while hardware and software failures cannot be avoided at all.

2. Software developing is prone to errors. Thus, a new concept has to be introduced to consider this aspect, the verification, i.e. a mechanism to test if the software is doing exactly what it is expected. Here it is necessary to remark that in general a high per cent of errors in digital control systems are caused by programming mistakes. Hence, digital control projects need not only control engineers but also engineers with skills in software engineering and computer programming. We do not care about real-time in our digital control system and even though it works. This statement is similar to the previous one. The problem here is that you are not able to know when your system can fail.

3. Standard textbooks on digital control systems normally assume that sampling is uniform, periodic and synchronous. This leads to a case of "zero-time-execution" for the control law. However, that is not realistic since the control algorithm also consumes some time producing a control or feedback delay (control or feedback latency), i.e. a delay between a sampling instant and the instant at which a control-signal value is applied to the actuator.

   Did u know? Real-time programming is assembly coding, priority interrupt programming and device driver writing.

   It is true that some code is still writing in assembler. However, high programming languages like C, Ada 95, Modula 2 and Real-time Java are normally used to develop real-time software. Device driver programming is necessary for real-time as well as non-real-time systems but they should be provided by the operating system or by the device manufacturer. Interrupt programming should be in principle avoided as much as possible.

   If the controller design is based on a model and the delay is constant and known, it could be helpful to use a tool for the description of the inter-sample behaviour (e.g. modified z-transform) in order to obtain a discrete time model more approximated to the real case.

4. The computational time of control algorithms can change from one sampling instant to other (e.g. hybrid controller with controller switching mechanism, event based controllers, adaptive controllers with on-line parameter update, etc.). This variation in the delay is called control jitter (according to the IEEE, jitter is "the time-related abrupt, spurious variation in the duration of any specified related interval"). Moreover, value calculation for the control signal is usually carried out using multitasking defining a set of control tasks with respective priorities. Thus, a task can be pre-empted by higher priority tasks. In general, it can be said that the control system is also affected by several kind of jitter depending on context: sampling jitter, control latency jitter, input jitter, output jitter, etc.

Finally, real-time issues are often ignored in the implementation of digital control systems. This is in part a consequence of erroneous definitions and false interpretations. Popular misconceptions from the control engineering community about real-time systems are for example:

Notes

The computer was connected to the plant by mean of A/D and D/A converters in order to obtain the real-time system.

Caution Analog plants should be connected to the computer through A/D and D/A converters. This link with the "real world" does not lead to a real-time system. On the other hand, it is possible to find real-time systems in complete digital contexts.

Our plant is so slow that real-time is actually no problem. A slow control system, which does not need a fast computer, can require critical time constraints. It is also possible that a control system does no need any hard real-time requirements but it is not necessarily a consequence of the slow plant.

It is not meaningful to talk about guarantying real-time performance. It is true that occasionally time constraints can be relaxed without introducing additional problems in the control loop. This particularly applies to nice designed laboratory experiments. However, this actually depends on the application, and the time criticality should be proved for each individual case.

Caution Real-time performance cannot be 100% guaranteed while hardware and software failures cannot be avoided at all.

We do not care about real-time in our digital control system and even though it works. This statement is similar to the previous one. The problem here is that you are not able to know when your system can fail.

Real-time programming is assembly coding, priority interrupt programming and device driver writing. It is true that some code is still writing in assembler. However, high programming languages like C, Ada 95, Modula 2 and Real-time Java are normally used to develop real-time software. Device driver programming is necessary for real-time as well as non-real-time systems but they should be provided by the operating system or by the device manufacturer. Interrupt programming should be in principle avoided as much as possible.

Digital controllers make three assumptions:

- Sensor data give accurate estimates of the state-variables being monitored and controlled – noiseless
- The sensor data gives the state of the plant – usually must compute plant state from measured values
- All parameters representing the dynamics of the plant are known.

If any of these assumptions are not valid, a digital controller must include a model of the correct system behaviour

- Estimate actual state based on noisy measurement of each iteration of the control loop
- Use estimated plant state instead of measured state to derive control output
- Often requires complex calculation, modelling

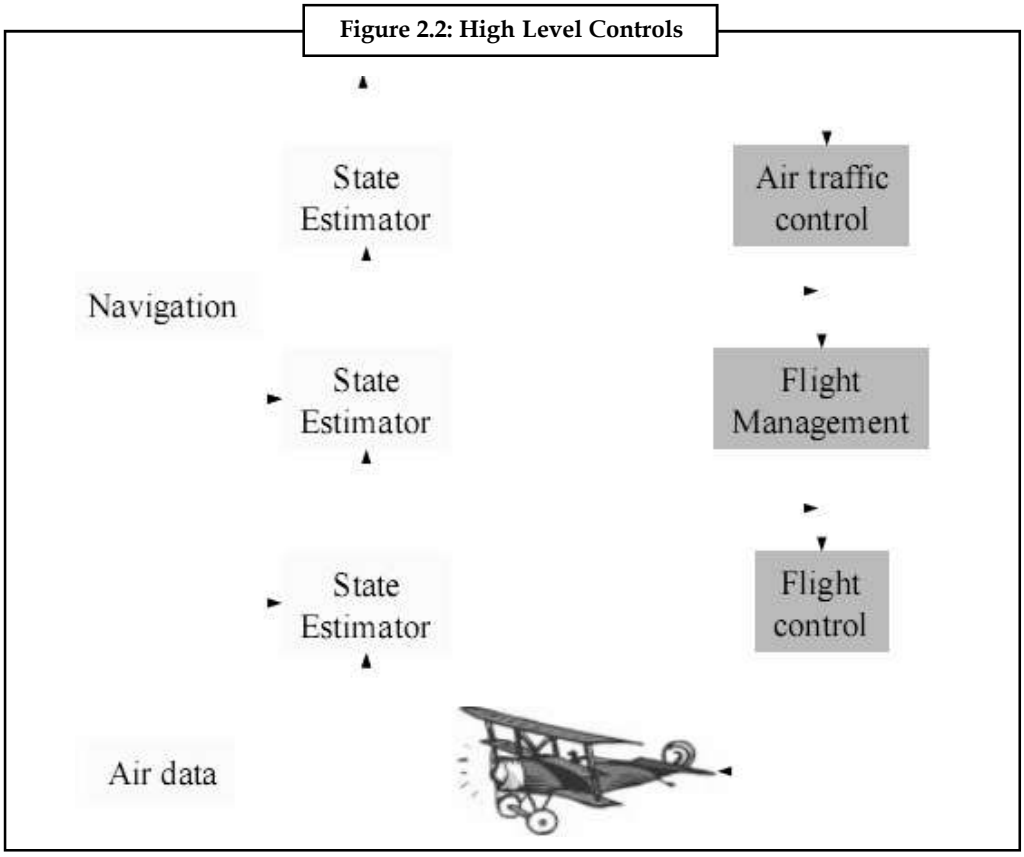
Task Apply various assumptions to a digital controller and note down the observations you have.

Notes

2.2 High Level Controls

Controllers are usually systematized in a hierarchy. Higher level controllers and multiple control loops supervise the behaviour of low-level controllers. Decision making complexity and time-scale increases as go up hierarchy. Move from control to planning. Higher level planning must still be done in real-time, although deadlines are less tight.

Figure 2.2: High Level Controls



Self Assessment

Fill in the blanks:

1. Real-time systems have found applications in …… ranging areas.
2. Errors arises due to A/D and D/A conversion as well as due to ……
3. It is not meaningful to talk about guarantying real-time ……
4. Software developing is …… to errors.
5. Controllers are usually …… in a hierarchy.
6. Higher level controllers and multiple control loops supervise the behaviour of …… controllers.

2.3 Signal Processing

The term real-time "refers to systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced".

Notes

Examples of real-time systems include command and control systems, process control systems, flight control systems, and so on. Digital signal processing systems analyse, produce, and transform discrete-time signals.

2.3.1 Discrete-time Signals

A discrete-time signal is a function that is defined only at a particular set of values of time. In the common case called uniform sampling, a discrete-time signal x(n) is related to a continuous analog signal xa(t) by

$$x(n) = x_a(nT), -\infty < n < \infty$$

where T is the sampling period, and $f_s = 1/T$ is the sampling frequency.

Notes **Internal and External Signals**

Signals can be internal or external, in the latter case; they constitute an interface between the signal processing system and an external environment. Many signal processing systems are "offline", that is, the external signals are not processed and produced simultaneously with their occurrence in the external environment. In real-time signal processing, however, they are.

Although discrete-time signals are defined over all time, computer implementations find it more convenient to consider only non-negative time indices, where time zero is the nominal time at which the real-time program begins operating. Thus:

$$x(n) = x_a(nT), n \geq 0$$

From here on, quantification of time indexes over positive n is implicit. Let t_x denote the clock of signal x, that is, the sequence of times at which x is defined. Each element of a clock is called a "tick." The clock of a uniformly- sampled signal x with period T is

$$t_x = \{nT\} \mid n \geq 0\}$$

A non-uniformly-sampled signal is not characterised quite so easily. Let $\tilde{x}$ be a non-uniformly-sampled signal with clock $\tilde{t}_x$. Then:

$$\tilde{x}(n) = x_a(\tilde{t}_x(n)),\ n \geq 0$$

In general, a discrete-time signal does not represent samples of an analog signal. The sequence of values representing changes of the state of my refrigerator door – open, closed, or almost-closed is an example. A sequence of messages between processes in a multi-tasking operating system is another. The term uniformly-clocked is used to mean a signal with clock {nT | n >= 0} for constant T, and non-uniformly-clocked for any other signal.

2.3.2 Streams and Channels

A signal is represented in a computer by a register or data structure that updates in time to contain zero, one, or more values of the signal. This register or data structure will call a channel. It could be implemented in software or hardware; in either case, a program (or hardware) can write to and read from the channel. Define a stream as the sequence of values that passes through

Notes

a channel. A uniformly-clocked signal x is implemented by the synchronous stream x. Streams are thus equivalent to signals:

$x(n) = x(n)$

$\tilde{y}(n) = \tilde{y}(n)$

Unlike a signal, a stream has not one but two clocks. The write clock x, w_x, is the sequence of times at which stream values become defined, in other words, the times at which they are written to the stream's channel. The read clock of x, r_x, is the sequence of times at which stream values are consume, that is, read from the channel. Read and write clocks are in general non-uniformly spaced: they represent actual times at which certain events (reading from or writing to a stream) occur, not idealised times at which signal values occur.

Did u know? While signal clocks are understood by the program, read and write clocks are solely for our own use in analysing and describing programs.

Assume that the action of writing a value to or reading a value from a channel consumes an infinitesimally small amount of time. The read clock of a stream x must therefore be later than its write clock:

$r_x > w_x$

The relation between a signal's clock and its stream's clocks depend on whether the signal is internal or external. Consider an input signal x. Because its stream, x, is produced by the external environment, the signal clock is equal to the stream's write clock:

$w_x > t_x$

For an output signal y, the stream's read clock is equal to the signal clock:

$r_y > t_y$

Correct real-time operation hinges on the program responding appropriately to external signals. From equations, we have, for input stream x and output stream y,

$r_x > t_x$

and:

$w_y > t_y$

In other words, the program cannot attempt to read from an input stream too early, and must write to an output stream sufficiently early. The non- zero time difference between the read or write clock and the signal clock can be attributed to the time taken by the program to transfer a value between a signal and a stream (for example, the interrupt response time and the time to copy a value from a real-time input-output port into the channel).

2.3.3 Functions and Systems

A signal processing system with one input signal x and one output signal y is a function from x to y:

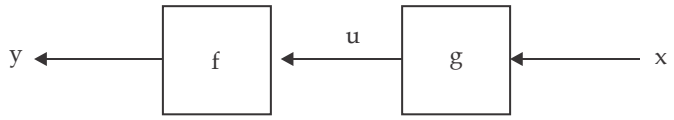
$y = f(x)$

A system is composed of functions, each with its own input and output signals. For example, the system consisting of two series-connected functions f and g is:

$y = f(g(x))$

Notes

Figure 2.3: A Simple Block Diagram



Source: http://ptolemy.eecs.berkeley.edu/~johnr/papers/pdf/thesis.pdf

Giving a name to the internal stream between g and f, this can also be written as the system of equations

u = g(x)

y = f(u)

When implemented in a block diagram system, f and g become blocks, and x, u, and y become streams. This simple system is shown in Figure 2.3. A very important function in signal processing systems is the "delay" operator.

$$y = z^{-k}x \Rightarrow y(n) = \begin{cases} 0, & n < k \\ x(n-k) & n \geq k \end{cases}$$

Did u know? The delay operator is not usually implemented as a process, but by inserting k initial zero values into a FIFO buffer contained in the relevant channel.

2.4 Other Real-time Applications

Let us now discuss some other real-time applications.

2.4.1 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.

Example: **Chemical Plant Control**

Chemical plant control systems are essentially a type of process control application. In an automated chemical plant, a real-time computer periodically monitors plant conditions. The plant conditions are determined based on current readings of pressure, temperature, and chemical concentration of the reaction chamber. These parameters are sampled periodically. Based on the values sampled at any time, the automation system decides on the corrective actions necessary at that instant to maintain the chemical reaction at a certain rate.

Each time the plant conditions are sampled, the automation system should decide on the exact instantaneous corrective actions required such as changing the pressure, temperature, or chemical concentration and carry out these actions within certain predefined time bounds. Typically, the time bounds in such a chemical plant control application range from a few micro seconds to several milliseconds.

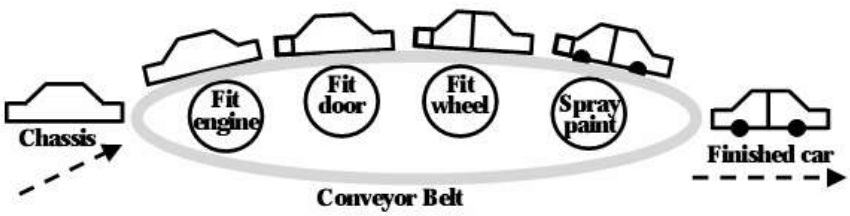
Notes

Example: **Automated Car Assembly Plant**

An automated car assembly plant is an example of a plant automation system. In an automated car assembly plant, the work product (partially assembled car) moves on a conveyor belt (see Fig. 2.4). By the side of the conveyor belt, several workstations are placed. Each workstation performs some specific work on the work product such as fitting engine, fitting door, fitting wheel, and spray painting the car, etc. as it moves on the conveyor belt. An empty chassis is introduced near the first workstation on the conveyor belt.

A fully assembled car comes out after the work product goes past all the workstations. At each workstation, a sensor senses the arrival of the next partially assembled product. As soon as the partially assembled product is sensed, the workstation begins to perform its work on the work product. The time constraint imposed on the workstation computer is that the workstation must complete its work before the work product moves away to the next workstation. The time bounds involved here are typically of the order of a few hundreds of milliseconds.

Figure 2.4: Schematic Representation of an Automated Car Assembly Plant



Source: http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-28.pdf

Example: **Supervisory Control and Data Acquisition (SCADA)**

SCADA are a category of distributed control systems being used in many industries. A SCADA system helps monitor and control a large number of distributed events of interest. In SCADA systems, sensors are scattered at various geographic locations to collect raw data (called events of interest). These data are then processed and stored in a real-time database. The database models (or reflects) the current state of the environment. The database is updated frequently to make it a realistic model of the up-to-date state of the environment. An example of a SCADA application is an Energy Management System (EMS). An EMS helps to carry out load balancing in an electrical energy distribution network. The EMS senses the energy consumption at the distribution points and computes the load across different phases of power supply. It also helps dynamically balance the load. Another example of a SCADA system is a system that monitors and controls traffic in a computer network. Depending on the sensed load in different segments of the network, the SCADA system makes the router change its traffic routing policy dynamically. The time constraint in such a SCADA application is that the sensors must sense the system state at regular intervals (say every few milliseconds) and the same must be processed before the next state is sensed.

Task Organize a debate on the topic "SCADA application is an Energy Management System".

Notes

2.4.2 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).

Example: **Robot Used in Recovery of Displaced Radioactive Material**

Robots have become very popular nowadays and are being used in a wide variety of medical applications. An application that we discuss here is a robot used in retrieving displaced radioactive materials. Radioactive materials such as Cobalt and Radium are used for treatment of cancer. At times during treatment, the radioactive Cobalt (or Radium) gets dislocated and falls down. Since human beings cannot come near a radioactive material, a robot is used to restore the radioactive material to its proper position. The robot walks into the room containing the radioactive material, picks it up, and restores it to its proper position. The robot has to sense its environment frequently and based on this information, plan its path. The real-time constraint on the path planning task of the robot is that unless it plans the path fast enough after an obstacle is detected, it may collide with it. The time constraints involved here are of the order of a few milliseconds.

2.4.3 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.

Example: **Laser Printer**

Most laser printers have powerful microprocessors embedded in them to control different activities associated with printing. The important activities that a microprocessor embedded in a laser printer performs include the following: getting data from the communication port(s), typesetting fonts, sensing paper jams, noticing when the printer runs out of paper, sensing when the user presses a button on the control panel, and displaying various messages to the user. The most complex activity that the microprocessor performs is driving the laser engine. The basic command that a laser engine supports is to put a black dot on the paper. However, the laser engine has no idea about the exact shapes of different fonts, font sizes, italic, underlining, boldface, etc. that it may be asked to print. The embedded microprocessor receives print commands on its input port and determines how the dots can be composed to achieve the desired document and manages printing the exact shapes through a series of dot commands issued to the laser engine. The time constraints involved here are of the order of a few milliseconds.

Lab Exercise Go to URL http://www.fujitsu.com/us/services/computing/peripherals/scanners/workgroup/ and collect more information on fi-60F A6 High-Speed Flatbed Scanner.

2.4.4 Automotive and Transportation

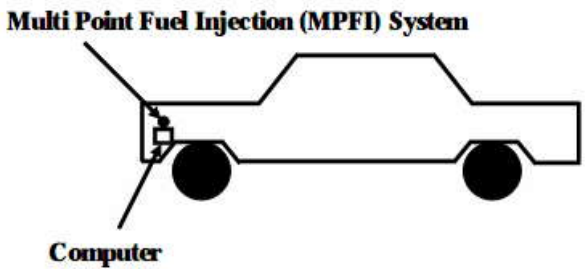
A few examples of automotive and transportation applications of real-time systems are: automotive engine control systems, road traffic signal control, air-traffic control, high-speed train control, car navigation systems, and MPFI engine control systems.

Notes

Example: **Multi-Point Fuel Injection (MPFI) System**

An MPFI system is an automotive engine control system. A conceptual diagram of a car embedding an MPFI system is shown in Fig.2.5. An MPFI is a real-time system that controls the rate of fuel injection and allows the engine to operate at its optimal efficiency. In older models of cars, a mechanical device called the carburettor was used to control the fuel injection rate to the engine. It was the responsibility of the carburettor to vary the fuel injection rate depending on the current speed of the vehicle and the desired acceleration. Careful experiments have suggested that for optimal energy output, the required fuel injection rate is highly nonlinear with respect to the vehicle speed and acceleration. Also, experimental results show that the precise fuel injection through multiple points is more effective than single point injection. In MPFI engines, the precise fuel injection rate at each injection point is determined by a computer. An MPFI system injects fuel into individual cylinders resulting in better 'power balance' among the cylinders as well as higher output from each one along with faster throttle response. The processor primarily controls the ignition timing and the quantity of fuel to be injected. The latter is achieved by controlling the duration for which the injector valve is open — popularly known as pulse width. The actions of the processor are determined by the data gleaned from sensors located all over the engine. These sensors constantly monitor the ambient temperature, the engine coolant temperature, exhaust temperature, emission gas contents, engine rpm (speed), vehicle road speed, crankshaft position, camshaft position, etc. An MPFI engine with even an 8-bit computer does a much better job of determining an accurate fuel injection rate for given values of speed and acceleration compared to a carburettor-based system. An MPFI system not only makes a vehicle more fuel efficient, it also minimizes pollution by reducing partial combustion.

Figure 2.5: A Real-Time System Embedded in an MPFI Car



Source: http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-28.pdf

2.4.5 Telecommunication Applications

A few example uses of real-time systems in telecommunication applications are: cellular systems, video conferencing, and cable modems.

Example: **A Cellular System**

Cellular systems have become a very popular means of mobile communication. A cellular system usually maps a city into cells. In each cell, a base station monitors the mobile handsets present in the cell. Besides, the base station performs several tasks such as locating a user, sending and receiving control messages to a handset, keeping track of call details for billing purposes, and hand-off of calls as the mobile moves. Call hand-off is required when a mobile moves away from a base station. As a mobile moves away, its received signal strength (RSS)

Notes

falls at the base station. The base station monitors this and as soon as the RSS falls below a certain threshold value, it hands-off the details of the on-going call of the mobile to the base station of the cell to which the mobile has moved. The hand-off must be completed within a sufficiently small predefined time interval so that the user does not feel any temporary disruption of service during the hand-off. Typically call hand-off is required to be achieved within a few milliseconds.

2.4.6 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.

Example: **Computer On-board an Aircraft**

In many modern aircrafts, the pilot can select an "auto pilot" option. As soon as the pilot switches to the "auto pilot" mode, an on-board computer takes over all controls of the aircraft including navigation, take-off, and landing of the aircraft. In the "auto pilot" mode, the computer periodically samples velocity and acceleration of the aircraft. From the sampled data, the on-board computer computes X, Y, and Z co-ordinates of the current aircraft position and compares them with the pre-specified track data. Before the next sample values are obtained, it computes the deviation from the specified track values and takes any corrective actions that may be necessary. In this case, the sampling of the various parameters, and their processing need to be completed within a few micro seconds.

Internet and Multimedia Applications

Important use of real-time systems in multimedia and Internet applications include: video conferencing and multimedia multicast, Internet routers and switches.

Example: **Video Conferencing**

In a video conferencing application, video and audio signals are generated by cameras and microphones respectively. The data are sampled at a certain pre-specified frame rate. These are then compressed and sent as packets to the receiver over a network. At the receiver-end, packets are ordered, decompressed, and then played. The time constraint at the receiver-end is that the receiver must process and play the received frames at a predetermined constant rate. Thus if thirty frames are to be shown every minute, once a frame play-out is complete, the next frame must be played within two seconds.

2.4.7 Consumer Electronics

Consumer electronics area abounds 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.

Example: **Cell Phones**

Cell phones are possibly the fastest growing segment of consumer electronics. A cell phone at any point of time carries out a number of tasks simultaneously. These include: converting input voice to digital signals by deploying digital signal processing (DSP) techniques, converting electrical signals generated by the microphone to output voice signals, and sampling incoming

Notes

base station signals in the control channel. A cell phone responds to the communications received from the base station within certain specified time bounds. For example, a base station might command a cell phone to switch the on-going communication to a specific frequency. The cell phone must comply with such commands from the base station within a few milliseconds.

2.4.8 Defence Applications

Typical defines applications of real-time systems include: missile guidance systems, anti- missile systems, satellite-based surveillance systems.

Example : **Missile Guidance System**

A guided missile is one that is capable of sensing the target and homes onto it. Homing becomes easy when the target emits either electrical or thermal radiation. In a missile guidance system, missile guidance is achieved by a computer mounted on the missile. The mounted computer computes the deviation from the required trajectory and effects track changes of the missile to guide it onto the target. The time constraint on the computer-based guidance system is that the sensing and the track correction tasks must be activated frequently enough to keep the missile from diverging from the target. The target sensing and track correction tasks are typically required to be completed within a few hundreds of microseconds or even lesser time depending on the speed of the missile and the type of the target.

2.4.9 Miscellaneous Applications

Besides the areas of applications already discussed, real-time systems have found numerous other applications in our everyday life. An example of such an application is a railway reservation system.

Example: **Railway Reservation System**

In a railway reservation system, a central repository maintains the up-to-date data on booking status of various trains. Ticket booking counters are distributed across different geographic locations. Customers queue up at different booking counters and submit their reservation requests. After a reservation request is made at a counter, it normally takes only a few seconds for the system to confirm the reservation and print the ticket. A real-time constraint in this application is that once a request is made to the computer, it must print the ticket or display the seat unavailability message before the average human response time (about 20 seconds) expires, so that the customers do not notice any delay and get a feeling of having obtained instant results. However, as we discuss a little later, this application is an example of a category of applications that is in some aspects different from the other discussed applications. For example, even if the results are produced just after 20 seconds, nothing untoward is going to happen - this may not be the case with the other discussed applications.

Self Assessment

State whether the following statements are True or False:

7. The term real-time refers to systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.
8. Examples of real-time systems exclude command and control systems, process control systems, flight control systems, and so on.

Notes

9. Digital signal processing systems analyse, produce, and transform discrete-time signals.
10. A discrete-time signal is a function that is defined at any set of values of time.
11. Industrial applications constitute a major usage area of real-time systems.
12. A cellular system usually maps a city into cells.

Case Study

Developing High-performance Radar Applications Using the VSIPL++ API

The need for high-performance Signal- and Image-Processing (SIP) applications is driving interest in parallel and multicore hardware for military embedded systems. However, programing such complex architectures can increase development costs by reducing developer productivity and code reuse. Leveraging a library that implements the high-level VSIPL++ API provides a way for SIP software developers to take advantage of the performance potential of parallel and multicore hardware systems while satisfying schedule and cost constraints.

The problem: Getting from prototype to high-performance production code efficiently.

Consider the following challenge. A software developer has a Scalable Synthetic Aperture Radar (SSAR) benchmark expressed in 50 lines of MATLAB code. The developer needs to implement that benchmark to achieve good performance on two very different systems – a conventional x86 processor and a heterogeneous, multicore Cell Broadband Engine (Cell/B.E.) processor – in just one week.

For background, Synthetic Aperture Radar (SAR) is used for a variety of imaging and remote sensing applications, including reconnaissance, surveillance, and terrain mapping. To standardize benchmarking this common algorithm, MIT Lincoln Laboratory created a scalable synthetic SAR application as part of the High Performance Embedded Computing (HPEC) Challenge. The benchmark demonstrates the intense computational requirements found in actual systems using synthetic (and scalable) data. The raw radar returns are processed by means of a 2D Fourier matched filtering step, a spatial frequency interpolation step, and a transformation back to the spatial domain. Key mathematical operations, as in any SAR application, include FFTs, matrix multiplication, and interpolation. Thus, a software development technique that offers benefits for this SSAR's benchmark points to an approach that is likely to work for larger SIP applications as well.

The first step: Choosing a library-based solution

If high performance were the only goal, the developer could consider coding the algorithm in a low-level language such as C or assembly code. But the time constraints (only a week) combined with the need to develop for both an x86 and a Cell/B.E. processor, make this approach unworkable.

With a portable library, though, the same application code will run on more than one system. And a library allows a developer to program for an unfamiliar architecture, such as the Cell/B.E., in a familiar language using familiar development tools. The key is to find a library at the right level of abstraction – high enough to provide the necessary primitives for the application domain but low enough to allow for efficient implementations and thus high performance.

Contd...

Notes

For the SSAR benchmark, the VSIPL++ API provides the right level of abstraction. VSIPL++ is an open standard, high-level API for parallel high-performance signal and image processing. It is defined by the High Performance Embedded Computing Software Initiative (HPEC-SI – www.hpec-si.org), a consortium of industrial, academic, and governmental partners, with sponsorship from the Air Force Research Laboratory. Its goal is to simultaneously deliver productivity, portability, and performance. VSIPL++ defines a pure C++ interface for operations – including FFTs, filters, linear system solvers, and other mathematical functions – that allow SIP applications to be written at the problem domain level.

For the SSAR benchmark challenge, CodeSourcery used Sourcery VSIPL++, a library that provides an optimized implementation of the VSIPL++ API on x86, Power Architecture, and Cell/B.E. processors with useful extensions to the base VSIPL++ specification.

The next step: Implementing SSAR in VSIPL++

Using the VSIPL++ library, CodeSourcery implemented SSAR in C++ in just four days. To illustrate the relative advantage of the VSIPL++ API for developer productivity, There are three different implementations of the SSAR algorithm's fast time filter: (1) MATLAB, (2) simple, unoptimized C, and (3) VSIPL++. Mathematically, they all perform the same computation, but the VSIPL++ version, like the MATLAB version, is easy to understand because it is expressed in SIP primitives such as FFTs and matrix multiplication.

Ignoring the setup of the filter coefficients, the VSIPL++ version of the fast time filter requires a single line, performing two data-parallel operations sequentially. By contrast, the C reference implementation is more verbose and thus more error prone. In addition, because the C code is iterative, it is more difficult to divide among multiple processors or to optimize for different architectures.

The C and VSIPL++ implementations were benchmarked on both a conventional Xeon processor running at 3.6 GHz and a Cell/B.E. processor running at 3.2 GHz. The entire front-end processing chain was run looping over the data 10 times to average out the measurements. On the Xeon platform, the VSIPL++ library used the Intel Performance Primitives (IPP) library v5 and Intel Math Kernel Library (MKL) v7.21 as well as FFTW v3.1.2. On the Cell/B.E. platform, the VSIPL++ library used the Cell Math Library v1.0 and FFTW v3.2-alpha3. The VSIPL++ code needed no changes to run on both architectures; the VSIPL++ library utilized these underlying math libraries without explicit direction from the developer.

The extra mile: Unlocking hardware's potential with strategic optimization.

Finally, opportunities for optimization were investigated in the VSIPL++ implementation for the Xeon and Cell/B.E. processors. For example, in the fast time filter computation, the VSIPL++ code performs an FFT followed by a matrix multiplication. Given large enough data sets, memory accesses in the second step cause cache misses on a Xeon processor, leading to expensive reads from main memory. Rewriting this loop so that each row is processed one at a time (that is, taking the FFT and then performing the vector multiplication) results in a 1.6x speed improvement for this portion of the code. The change is trivial to implement, requiring only a net increase of eight lines of code, yet it yields a 20% improvement in the execution time of the entire front-end stage.

On the Cell/B.E. processor, profiling reveals that interpolation takes almost 40 times longer than matched filtering. A large amount of time is spent in a loop over data in the "range" direction (perpendicular to the flight path), performing a polar-to-rectangular coordinate conversion. A contribution from several inputs for each side-lobe of the sinc

Contd...

Notes

function used in the interpolation is added to calculate the intensity and phase of the corresponding output pixel. This computation cannot be expressed well using VSIPL++ primitives.

To improve performance, CodeSourcery used a VSIPL++ API extension available in Sourcery VSIPL++ called "user-defined kernels." User-defined kernels allow the developer to write a high-performance computational kernel and still leverage the data-handling aspects of the VSIPL++ library. A hand-coded kernel with 208 lines of code speeds up interpolation from 4.23 seconds to 0.18 seconds, an improvement of more than 23 times that of the original implementation.

On Xeon, the final optimized code runs more than 82 times faster than the C reference implementation. On the Cell/B.E., it was 5.7 times faster than on the Xeon and more than 1,400 times faster than the reference C code. Even modest, easy-to-implement changes can significantly improve performance.

Combining performance, productivity, and portability with VSIPL++.

Using a library implementing the open-standard VSIPL++ API made possible the development of a complex application in far fewer lines of code than are necessary in C. Out of the box, this code outperformed the C reference implementation. With limited changes to address performance bottlenecks, performance was further enhanced. And the application remained portable across vastly different architectures.

Questions:

1. Discuss the main hindrances in developing high-performance radar applications using the VSIPL++ API.
2. What are the different measures adopted to overcome such problems?

Source: http://mil-embedded.com/articles/case-applications-using-vsipl-api/

2.5 Summary

- Real-time systems have of late, found applications in wide ranging areas.
- Errors due to A/D and D/A conversion as well as due to limited length word calculations.
- It is not meaningful to talk about guarantying real-time performance.
- Software developing is prone to errors.
- Controllers are usually systematized in a hierarchy.
- Higher level controllers and multiple control loops supervise the behaviour of low-level controllers.
- The term real-time refers to systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.
- Examples of real-time systems include command and control systems, process control systems, flight control systems, and so on.
- Digital signal processing systems analyse, produce, and transform discrete-time signals.
- A discrete-time signal is a function that is defined only at a particular set of values of time.
- Industrial applications constitute a major usage area of real-time systems.
- A cellular system usually maps a city into cells.

Notes

2.6 Keywords

Discrete-time Signal: A discrete-time signal is a function that is defined only at a particular set of values of time.

Radar: Radar is an acronym for Radio Detection and Ranging.

Radar System: The system consists of an Input/Output (I/O) subsystem that samples and digitizes the echo signal from the radar and places the sampled values in a shared memory.

Real-time: The term real-time refers to systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.

Signal Processing: Signal processing is an area of systems engineering, electrical engineering and applied mathematics that deals with operations on or analysis of signals, in either discrete or continuous time.

2.7 Review Questions

1. Define the term digital control.
2. What are the digital controls in real systems?
3. What is meant by term sampled signal?
4. Differentiate between sampled and non-sampled signals.
5. What are the complex control-systems?
6. What are the high level controls?
7. Explain real-time command and controls.
8. What is the signal processing means?
9. What is radar system aimed at?
10. Explain internet and multimedia applications in real-time system.

Answers: Self Assessment

1.	wide	2.	limited length word calculations
3.	performance	4.	prone
5.	systematized	6.	low-level
7.	True	8.	True
9.	True	10.	False
11.	False	12.	True

2.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real Time Systems and Programming Languages*, Addison Wesley.

Notes

C. M. Krishna and K. G. Shin (1997). *Real Time Systems*. McGraw-Hill International Editions..

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

publib.boulder.ibm.com/.../realtime/v1r0/.../realtime/write_introd.html

csperkins.org/teaching/rtes/lecture02.pdf

www.fi.muni.cz/~xpelanek/IA158/slides/intro.pdf

v5.books.elsevier.com/bookscat/samples/.../9780750664714.PDF

Notes

Unit 3: Hard Versus Soft Real-time System

CONTENTS

Objectives

Introduction

3.1 Jobs and Processors

3.2 Release Time

3.3 Deadline and Timing Constraint

3.3.1 Deadline

3.3.2 Time Constraints

3.4 Hard and Soft Timing Constraint

3.4.1 Hard vs. Soft Deadlines

3.5 Hard Real-time Systems and Soft Real-time Systems

3.6 Summary

3.7 Keywords

3.8 Review Questions

3.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain Jobs and Processors
- Define Release Time
- Enumerate Deadline and Timing Constraint
- Explain Hard and Soft Timing Constraint

Introduction

In this unit, we will study Jobs and Processors. We will also define release time. Further, we will enumerate deadline and timing constraint. In the end, we will focus on hard and soft timing constraint.

3.1 Jobs and Processors

A job is a unit of work that is scheduled by the system.

Example: Computation of a control law (digital controller), FFT computation on sample data, transmission of a packet.

A job executes on a processor and may depend on some resources. A processor P is an active object, for example, CPU, Transmission Link, Server.

Notes

Processors have attributes (pre-emptivity, context switch time, speed). Two processors are of the same type, if they are functionally identical and can be used interchangeably.

According to the model the basic component of any real-time application are jobs. The operating system treats jobs as a unit of work and allocates processors and resources.

Did u know? A job J_i is characterized by many parameters:

- execution time
- deadlines
- pre-emptivity
- resource requirements
- soft or hard real-time

3.2 Release Time

If all jobs are released when the system begins execution, then there is said to be no release time. The release time r_i of a job is the instant of time at which the job becomes available for execution. Release time may be jittery so that r_i is in the interval $[r_i-, r_i+]$. Jobs can be scheduled and executed at any time after its release time, whenever its data and control dependency conditions are met.

Did u know? The response time is the length of time from the release time of the job to the time it completes.

Self Assessment

Fill in the blanks:

1. A job is a unit of work that is scheduled by the ……
2. A job executes on a processor and may depend on some ……
3. The basic component of any real-time application are ……
4. The …… system treats jobs as a unit of work and allocates processors and resources.
5. If all jobs are released when the system begins execution, then there is said to be …… release time.
6. The release time r_i of a job is the instant of time at which the job becomes available for ……

3.3 Deadline and Timing Constraint

Deadline is defined as an instant of time a job's execution is required to be completed. If deadline is infinity, then job has no deadline. Absolute deadline is equal to release time plus relative deadline. Timing constraint is a constraint imposed on timing behaviour of a job: hard or soft.

Notes

3.3.1 Deadline

A deadline is a timing milestone. If a deadline is missed by a computer-controller, the controlled system may transit to an undesirable state.

In hard real-time systems, according to the usual definition, a deadline that is not met can lead to a catastrophic failure. This means that the criteria used to establish deadlines are safety based. Control system engineers, on the other hand, use performance criteria to establish the desired response time of a controlling computer.

The deadlines suggested by these scientific communities are not mutually exclusive but only different entities perceived in particular and equally important contexts. Moreover, they show the disassociation of controllers' timing constraints into those related to safety – hard deadlines – and those related to performance – performance deadlines.

Performance deadlines are usually more confining than hard deadlines. Therefore, a computer-controller, designed to meet performance deadlines, does not drive the controlled system to an unsafe state as soon as one of them is missed, but only later, when a hard deadline is disrespected. Performance and hard deadlines are thus separated by a grace-time. This notion can help in the design of low-cost, yet highly reliable control systems.

Lab Exercise Go to URL http://paginas.fe.up.pt/saic/Membros/apmag/Ficheiros/drts.pdf and collect more information on unified view of deadlines.

3.3.2 Time Constraints

Real-time systems are usually classified into soft and hard. Classically, in a soft real-time system, missing a deadline is inconvenient but not damaging to the environment; in hard real-time system, missing a deadline can be catastrophic, and thus unacceptable.

The traditional view of the temporal merit of a hard real-time computation (i.e., the relationship between the computation completion time and the resulting temporal merit of that computation) is usually modelled by a step time-value function: if a controller service is completed before a given deadline it yields a constant positive value while completing it any time later may incur in a catastrophic failure. From this point of view, hard deadlines are established in a safety-based context.

This means that when a computer is part of a hard real-time system, all the software running on it has to be tuned to satisfy all controlled system deadlines.

Notes **Computations**

Computations often present non-binary time constraints, even when a large merit penalty is incurred for completing it after a deadline. Also, there are many cases in real-time applications where some diminished merit is attained for completing a computation within an allowable period after a deadline. Moreover, the acceptability of the completion times of a set of computations must consider their collective merit instead of the individual ones presents and schedules some smooth time-value functions illustrating Jensen's point of view.

Notes

It is commonly accepted that a controlled process can sporadically tolerate a missed deadline, if not by much. This notion presupposes a controller not tuned to meet hard deadlines but some other kind of time limit.

However, the characterisation of a deadline is by itself a relatively unexplored problem in the real-time community. Most of the literature seems to consider that deadlines are somehow provided by others, possibly by control system engineers. Moreover, techniques to calculate systems' deadlines are very seldom presented.

Nevertheless, soft and hard deadlines are universally used, and many suggestions appear in the literature reasoning about the existence of other kinds of deadlines, besides these classical ones. Moreover, there is a growing tendency to classify real-time services according to their associated benefit/cost functions, and to establish on them a set of pertinent points of time concerning application goals. This means that there is a growing feeling that traditional definitions and interpretations of deadlines are poor since they cannot describe reality in a satisfactory way nor can be explicitly employed for a best-effort scheduling.

Task Discuss in groups, different techniques to calculate systems.

3.4 Hard and Soft Timing Constraint

Common definitions are based on:

- ***functional criticality:***
  - *soft:* meeting the constraints (deadlines) is desirable, but a few misses do no serious harm,
  - *hard:* missing a deadline is a fatal fault;
- ***usefulness of late results:***
  - *soft:* usefulness decreases gradually with tardiness,
  - *hard:* usefulness drops to zero or becomes negative when tardiness is larger than zero;
- ***deterministic or probabilistic nature of constraints:***
  - *soft:* deadlines can be missed occasionally, with low probability,
  - *hard:* deadlines must never be missed.

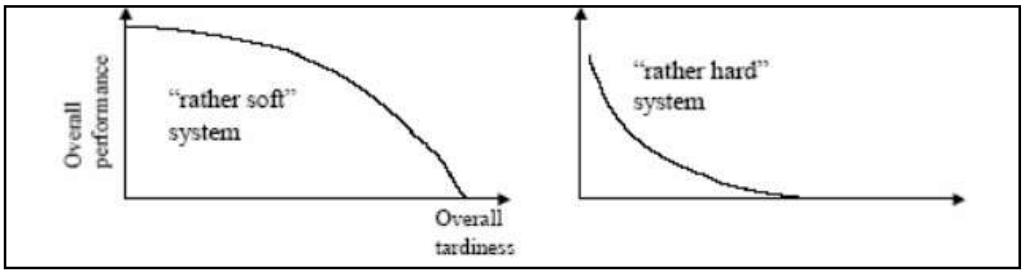
3.4.1 Hard vs. Soft Deadlines

- ***Hard Deadline:*** Late result may be a fatal flaw, useless, or cause disastrous consequences
- ***Soft Deadline:*** Timely completion desirable.

Late results useful to some degree

- ***Quantitative Measure:*** Overall system performance as function of tardiness of jobs.

Notes



Timing constraints can be specified in different terms:

1. deterministic constraints (e.g. response time d" 50 ms)
2. probabilistic constraints (e.g. probability that response time > 50 ms less than 0.1)
3. in terms of some usefulness function (e.g. usefulness function e" 0.8)

In practice hard timing constraints are of the first type: specified as deterministic constraints (easier to validate than the latter two types).

Steps in validating timing constraints are discussed below:

1. Verify that in each component of the system the timing constraints are specified correctly: ensure that the timing constraints of each component are:
   - Mutually consistent,
   - Consistent with the high-level RT requirements.
2. Verify the feasibility of each component with the available HW and SW resources: ensure that each component in the system can meet its timing constraints if:
   - It executes alone
   - It has all the required resources.
3. Verify that the whole system behaves as specified: ensure that, when scheduled together with some scheduling algorithm, the timing constraints of all components (competing for the available resources) are always met.

Now let us discuss how to validate timing constraints.

1. To verify that in each component of the system the timing constraints are specified correctly: use formal methods.
2. To verify the feasibility of each component with the available HW and SW resources: use performance analysis.
3. To verify that a scheduling algorithm can meet the timing constraints of all components, competing for the available resources is done.

3.5 Hard Real-time Systems and Soft Real-time Systems

Real-time systems are classified as hard or soft real-time systems. Hard real-time systems have very strict time constraints, in which missing the specified deadline is unacceptable. The system must be designed to guarantee all time constraints. Every resource management system such as the scheduler, input-output (I/O) manager, and communications, must work in the correct order to meet the specified time constraints.

Military applications and space missions are typical instances of hard real-time systems. Some applications with real-time requirements include telecom switching, car navigation, the medical

Notes

instruments with the critical time constraints, rocket and satellite control, aircraft control and navigation, industrial automation and control, and robotics.

Soft real-time systems also have time constraints; however, missing some deadline may not lead to catastrophic failure of the system. Thus, soft real-time systems are similar to hard real-time systems in their infrastructure requirements, but it is not necessary that every time constraint be met. In other words, some time constraints are not strict, but they are nonetheless important.

Caution A soft real-time system is not equivalent to non-real-time system, because the goal of the system is still to meet as many deadlines as possible.

Some applications with soft real-time requirements include web services such as real-time query, call admittance in voice over internet protocol and cell phone, digital TV transmissions, cable and digital TV set-top-boxes, video conferencing, TV broadcasting, games, and gaming equipment. Multimedia systems in general are examples of soft real-time systems (e.g., dropping frames while displaying video).

Even in some typical hard real-time applications, some functions have soft real-time constraints. For instance, in Apollo 11, the lunar module guidance computer could not keep up with the data stream from the landing radar. However, it was discovered that the missed deadlines were nonfatal, and the scheduler automatically adjusted, to meet soft real-time behaviour for the landing tasks.

Self Assessment

State whether the following statements are True or False:

7. Deadline is defined as an instant of time a job's execution is required to be completed.
8. If deadline is infinity, then job has finite deadline.
9. Absolute deadline is equal to release time minus relative deadline.
10. Timing constraint is a constraint imposed on timing behaviour of a job: hard or soft.
11. Hard real-time systems have very strict time constraints, in which missing the specified deadline is acceptable.
12. Soft real-time systems also have time constraints; however, missing some deadline may not lead to catastrophic failure of the system.

Case Study

Process and Machine Description

"Punching" is a general term describing the process of cutting a hole in a metal sheet. The punching process results from the motion of two sharp, closely adjoined edges on a material placed between them and comprise three stages:

- The *Deformation Phase*, as the cutting edges begin to close;
- The *Penetration Phase*, as the cutting edges penetrate the material causing initial fracture lines on both sides of the material;
- The *Fracture Point*, the point where the upper and lower fracture lines meet.

Contd...

Notes

Punching reactive forces depend on material thickness, type and hardness, condition of the cutting edges, diameter of hole, etc. Yet, they always increase for the initial tool displacement, decreasing later. The fracture occurs when the resistive force significantly decreases. Figure 1 shows a simplified model of a punching force profile.

This case study concerns a prototype flexible hydraulic press devoted to multiple metal sheet working, including punching. It exhibits a maximum force capacity of 1.6 MN (approx. 160 Ton) and may execute up to 2 punches per second. Press configuration is based on a large area main cylinder, C1, a small auxiliary cylinder, C2, and two hydraulic valves, V1 and V2 (Figure 2). The main cylinder is responsible for press force capacity while the auxiliary one drives the press on its fast ascending and descending movement when no resistive force exists. The auxiliary cylinder is driven by a hydraulic circuit that is not relevant

Figure 1: A Punching Force Profile

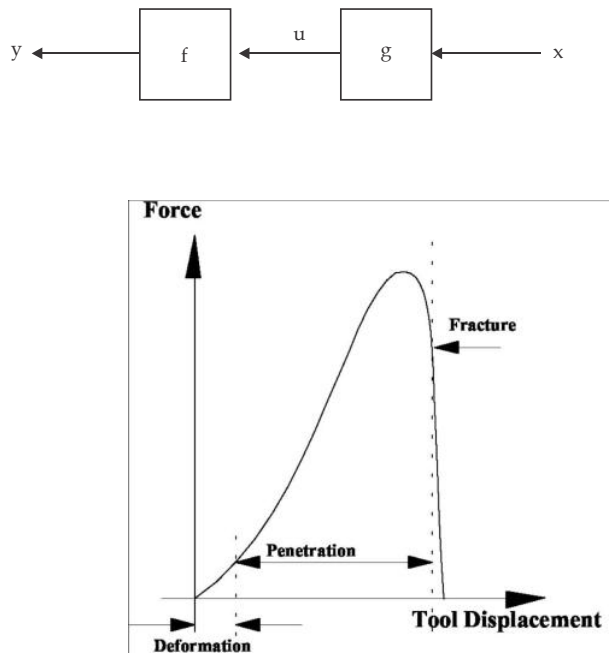
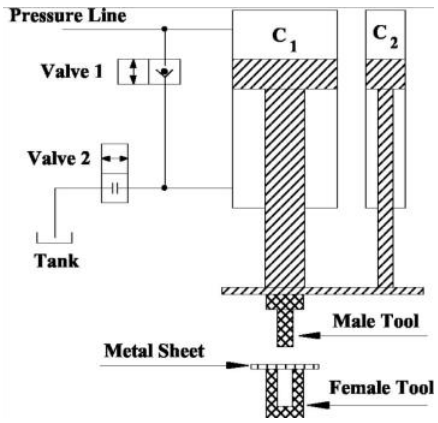


Figure 2: Press Simplified Hydraulic Scheme



Contd...

Notes

An important problem of this algorithm (and of all the algorithms of this kind) is that, to determine if one set of tasks is schedulable, we have to calculate the resulting processor load. To do that we have to sum the worst case execution time of all the tasks in the set. This leads to the processor being idle most of the time since usually the average execution time of a task is significantly shorter than the worst case.

The valve 1 is an electrically operated logic element. When active, it connects the upper and lower main cylinder chambers leading them to a similar pressure and disabling press force capacity. When not, and if the lower chamber pressure exceeds the upper one, it conducts an ascending flow. In electrical terms, this valve may be roughly defined as a switch presenting a diode in parallel, whose cathode is connected to the upper chamber; when the valve 1 is active the switch is closed.

Since the main cylinder exhibits a constant pressure in its upper chamber, decompressing its lower chamber enables press force production. This action is achieved by opening the valve 2 when valve 1 is off.

When a punching cycle is initiated, the male tool is some distance apart from the metal sheet. Valve 1 and valve 2 are off. By this time, the main cylinder exhibits a significant pressure on both chambers. The punching cycle involves the following sequential steps:

- The auxiliary cylinder initiates a fast descending movement and valve 1 conducts a flow from the lower to the upper chamber.
- When the tool gets close to the metal sheet, the valve 2 is opened starting lower chamber decompression. Press yields a growing force capacity and the tool progressively penetrates the material.
- When the resistive force presents a quasi-fracture value, the press controller issues a closing order to valve 2. This action is crucial to press work efficiency, since any flow escaping the lower camber after the fracture is considered an energy waste.
- Soon after the fracture, the valve 1 is activated disabling the press force capacity and the auxiliary cylinder initiates the press ascending movement.

Questions:

1. What do you infer from the case study?
2. Write down the case facts.

Source: http://paginas.fe.up.pt/saic/Membros/apmag/Ficheiros/drts.pdf

3.6 Summary

- A job is a unit of work that is scheduled by the system.
- A job executes on a processor and may depend on some resources.
- The basic component of any real-time application is jobs.
- The operating system treats jobs as a unit of work and allocates processors and resources.
- If all jobs are released when the system begins execution, then there is said to be no release time.
- The release time r_i of a job is the instant of time at which the job becomes available for execution.
- Deadline is defined as an instant of time a job's execution is required to be completed.

Notes

- If deadline is infinity, then job has no deadline.
- Absolute deadline is equal to release time plus relative deadline.
- Timing constraint is a constraint imposed on timing behaviour of a job: hard or soft.
- Hard real-time systems have very strict time constraints, in which missing the specified deadline is unacceptable.
- Soft real-time systems also have time constraints; however, missing some deadline may not lead to catastrophic failure of the system.

3.7 Keywords

Computations: Computations are non-binary time constraints, even when a large merit penalty is incurred for completing it after a deadline.

Hard Real-time System: It is the system which guarantees that critical tasks complete on time and this goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the operating system to finish any request made of it.

Job: This is a unit of work that is scheduled and executed by a system and executed by the operating system on a processor and may depend on some resources.

Processor (P): It is an active object on which job executes. It has a speed attribute which determines the rate of progress a job makes toward completion.

Release Time: It is the time of job, or is the instant of time at which the job becomes available for execution.

Soft Real-time System: It is a system where a critical real-time task gets priority over other tasks and retains that priority until it completes.

Timing Constraints: These are the factors/parameters on which the correctness of real-time tasks depends.

3.8 Review Questions

1. Define the term the job in general.
2. Explain the concept of job and processors in real-time system.
3. Define release time.
4. How many algorithms used to find the release time?
5. Explain ITR algorithm in detail.
6. Algorithm ITR terminates after a finite number of iterations. Comment on it.
7. Define the terms deadline and timing constraints.
8. Discuss different types of timing constraints.
9. Differentiate between soft and hard timing constraints.
10. What is the basic component of any real-time application?

Notes

Answers: Self Assessment

1.	System	2.	Resources
3.	Jobs	4.	Operating
5.	No	6.	Execution
7.	True	8.	False
9.	False	10.	True
11.	False	12.	True

3.9 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the Real World.*

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

en.wikipedia.org/wiki/Real-time_computing

wiki.answers.com/.../Difference_between_Hard_realtime_system_and_soft_real_time_system

csperkins.org/teaching/rtes/lecture02.pdf

www.edaboard.com/thread86102.html

Notes

Unit 4: A Reference Model of Real-time Systems

CONTENTS

Objectives

Introduction

4.1 Processors and Resources

4.2 Temporal Parameters of Real-time Workload

4.2.1 Fixed Jittered and Sporadic Release Times

4.2.2 Execution Time

4.3 Periodical Task Model

4.3.1 Periods, Execution Times and Phases of Periodic Tasks

4.3.2 Aperiodic and Sporadic Jobs

4.4 Precedence Constraint and Data Dependency

4.4.1 Precedence Graph and Task Graph

4.4.2 Data Dependency

4.5 Summary

4.6 Keywords

4.7 Review Questions

4.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Processors and Resources
- Enumerate Temporal Parameters of Real-time Workload
- Define Periodical Task Model
- Explain Precedence Constraint and Data Dependency

Introduction

A reference model of real-time systems is needed to focus on aspects of the system relevant to its real-time timing and resource properties. A system may be characterized by 3 models: A workload model describing the applications (i.e. control laws and their time complexity) supported by the system; a resource model describing the system resources (e.g., compute cycles or time on the processor hardware to be used) available to the applications. The scheduling and resource management algorithms define how the application uses the resources (i.e. how the workload is scheduled on the system).

Notes

4.1 Processors and Resources

All the resources can be divided into two parts.

- Processors
- Resources

Two processors are of the same type of they are functionally identical and can be used interchangeably.

- Symmetric Multi-processors.
- Two transmission links connected with the same transmission rate between a pair of sender and receiver.

A transmission link connecting an on-board flight management system to the ground controller is different from the link connecting two air traffic control centres together as they cannot be used interchangeably. By resources we specifically mean passive resources such as memory, mutexes, database locks. A job needs resources as well as processor to execute.

Resources always mean a re-usable resource. A resource is plentiful if no job is ever prevented from execution by the lack of this resource, i.e. a resource that can be shared by infinite number of jobs for example a file readable by all. These resources need not to be explicitly modelled. Memory is also essential, but many real-times system are designed in such a way that whenever a job is scheduled to execute, it always has a sufficient amount of memory. We even omit memory from our model whenever not required. Consider I/O bus, every computation job must have the I/O bus in addition to the processor to execute. Often it is modelled as plentiful, but what if we want to determine the real-time performance or effect of I/O bus arbitration.

4.2 Temporal Parameters of Real-time Workload

The number of tasks is one parameter. In most embedded systems, the number of tasks is fixed in a particular operation mode. In some systems, however, the number of tasks may change as tasks are added or deleted while the system executes. In an air traffic controller, each surveillance task monitors a single aircraft. The number of tasks may change (added or deleted) when aircrafts enter or leave the coverage area. Each job is characterized by temporal parameters, functional parameters, resource parameters, and interconnecting parameters. Temporal parameters tell us its timing constraints. Interconnection parameters describe how it depends on other jobs and others depend on it.

Functional parameters specify the intrinsic properties of job. Resource parameters give us its resource requirements.

4.2.1 Fixed Jittered and Sporadic Release Times

In many systems, we do not know the actual release time r_i of each job J_i, we are given a range $[r_i-,r_i+]$. It can be as earliest as r_i- or as late as r_i+. This range is called release time jitter. Almost every real-time system is required to respond to external events which occur at random instants of time. On such events the system executes a set of jobs in response. The release time of these jobs is not known until the event triggering them occurs.

These jobs are called sporadic or aperiodic jobs as these are released at random time. The pilot may disengage the autopilot system at any time. When this occurs; the autopilot system changes from cruise mode to standby mode. The jobs which executed for this mode change are sporadic.

Notes

4.2.2 Execution Time

Another temporal parameter of a job, J_i, is its execution time, e_i. It the time required to complete the execution of J_i when it executes alone and has all the resources it requires. The actual amount of time to complete its execution may vary for reasons like execution depending on the input data. The actual amount of time to transmit each frame of MPEG compressed video is different from frame to frame, because of the number of bits used to encode different frames. What can in fact be determined a priori are the maximum and minimum amounts of time required to complete each job (analysis and measurement and testing) We know that the execution time e_i of the job J_i is in the range $[e_i+, e_i+]$.

We usually assume that we know e_i+ and e_i- of every hard real-time job J_i but the actual execution time may be unknown.

Self Assessment

Fill in the blanks:

1. A …… model of real-time systems is needed to focus on aspects of the system relevant to its real-time timing and resource properties.
2. A system may be characterized by …… models.
3. A …… model describes the applications.
4. A …… model describes the system resources.
5. The scheduling and …… algorithms define how the application uses the resources.
6. Resources always means a …… resource.

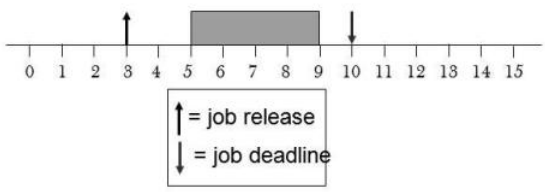
4.3 Periodical Task Model

The concept of periodical task model is discussed below:

4.3.1 Periods, Execution Times and Phases of Periodic Tasks

Each computation or data transmission that is executed repeatedly at regular or semi-regular time intervals in order to provide a function of the system is modelled as a period task. Simply a periodic task occurs at regular intervals. Each task T_i is characterized by the following parameters. The release time (r): the time when the task first released. The period (T): the constant interval between jobs. The relative deadline (D): the maximum acceptable delay for task processing. The computation time (C): the worst case execution time (WCET) of any job. The tasks in the system can be denoted by T_1, T_2,…,T_n. Individual jobs in a task Ti can be denoted as $T_{i,1}$, $T_{i,2}$….$T_{i,n}$ 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. Different tasks may have the same phase

Example:

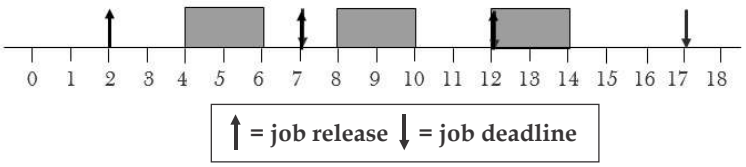


Notes

Job is released at time 3. Its (absolute) deadline is at time 10. Its relative deadline is 7. Its response time is 6. A time interval of length H is called a hyper period of the periodic tasks. It is the least common multiple of p_i for i=1, 2, …n. The length of a hyper period of three periodic tasks with periods 3, 4, 10 is 60

The maximum number N of jobs in each hyper period is equal to $_{i=1}^{n}\Sigma$ H/pi. The total number of job in the hyper period is 41. The ratio $u_i = e_i/p_i$ is the utilization of the task $T_{i..}$ U_i is equal to the fraction of time a truly periodic task with period p_i and execution time e_i keeps a processor busy. The total utilization U of all the tasks in the system is the sum of the utilizations of the individual tasks in it. If the execution times of the three periodic tasks are 1, 1, 3 and periods are 3,4,10 then their utilizations are 0.33, 0.25 and 0.3. Total utilization is 0.88 (These tasks keep processor busy 88% of time).

Example: A periodic task T_i with $r_i = 2$, $p_i = 5$, $e_i = 2$, $D_i = 5$ executes like this:



4.3.2 Aperiodic and Sporadic Jobs

A task is aperiodic if the jobs in it have either soft deadlines or no deadlines. The task to adjust radar's sensitivity is an example. We want the system to be responsive, that is, to complete each adjustment as soon as possible. But a late response is annoying but tolerable. We therefore want to optimize the responsiveness of the system for the aperiodic jobs, but never at the expense of hard real-time tasks has whose deadlines must be met at all timed. In contrast, an autopilot system is required to response to a pilot's command to disengage the autopilot and take over the control manually within a specific time. The jobs that execute in response to these events have hard deadlines. The jobs that execute in response to these events have hard deadlines. Tasks containing jobs that are released at random time instants and have hard deadlines are sporadic tasks.

4.4 Precedence Constraint and Data Dependency

Data and control dependencies among jobs may constrain the order in which they can execute. 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. Consider queries to an information server. Suppose that before each query is processed and the requested information retrieved, its authorization to access the requested information is first checked. The retrieval job cannot begin execution until the authentication job completes. Similarly in a communication system, the jobs that generate acknowledgements of a message and transmit the acknowledgement message cannot begin until the job that receives and processes the message completes.

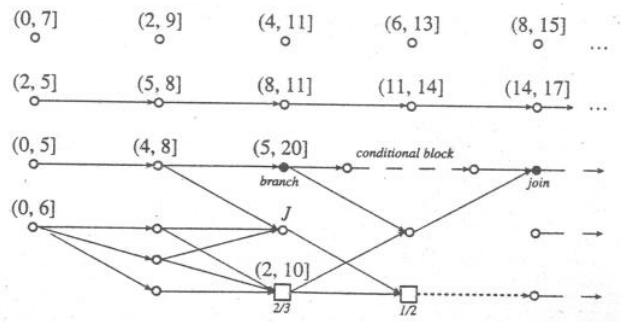
4.4.1 Precedence Graph and Task Graph

We use a partial-order relation <, called a precedence relation, over the set of jobs to specify the precedence constraints among jobs. 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. In short we can write $J_i < J_{k.}$ Ji is the immediate predecessor of Jk if Ji < Jk and there is no other job J_j such that $J_i<J_j<J_{k.}$ Two jobs are independent

Notes

when neither $J_i<J_k$ nor $J_k<J_i$. 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. A task graph, which gives us a general way to describe the application system, is an extended precedence Graph. The vertices in a task graph represent jobs.

A task Graph is shown below:



The task represented in first row has phase 0, period 2 and relative deadline 7 and all jobs are independent. 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. The second row represents a periodic task has phase 2, period 3 and relative deadline 3 and jobs are interdependent. The first job is predecessor of 2nd and 2nd is predecessor of 3rd job. The last two tasks are not periodic.

4.4.2 Data Dependency

Data dependency cannot be captured by a precedence graph. In many real-time systems, jobs communicate via shared data. Each producer can place the data generated by it in a shared address space to be used by the consumer at any time so precedence graphs shows them as independent.

What if consumer and producer need to be synchronized?

In an avionics system, the navigation job updates the location of the air plane periodically. Whenever the flight management job needs navigation data, it read the most current data produced by the navigation data from the shared space. Data dependencies among jobs are represented explicitly by data dependency edges among jobs.

Self Assessment

State whether the following statements are True or False:

7. Each computation or data transmission that is executed repeatedly at regular or semi-regular time intervals in order to provide a function of the system is modelled as a period task.
8. A periodic task occurs at irregular intervals.
9. A task is said to be periodic if the jobs in it have either soft deadlines or no deadlines.
10. The jobs are said to have precedence constraints if they are constrained to execute in some order.
11. If the jobs can execute in any order, they are said to be dependent.
12. Data dependency cannot be captured by a precedence graph.

Notes

Case Study

Virtual University Reference Model

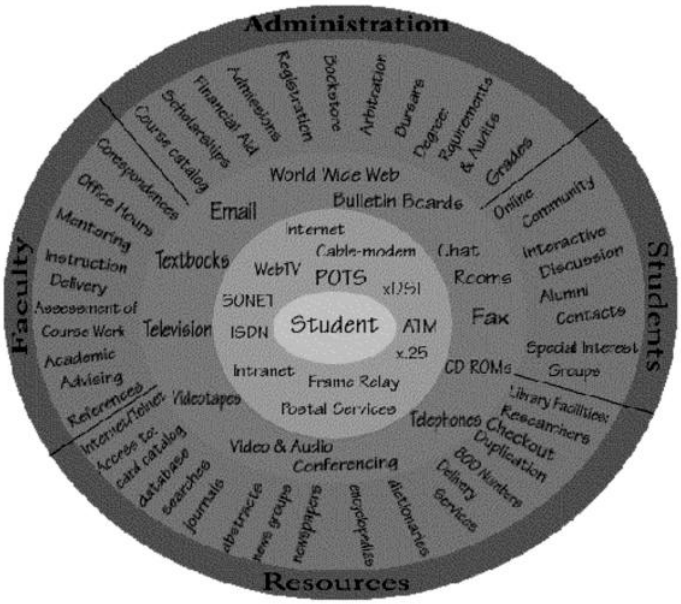
Planning and designing a virtual university or a virtual campus is a complex task involving many different aspects of higher education administration and instructional delivery? In the early days of online courses, just putting course syllabi on the Web is worthy of attracting some attention. Nowadays many online courses are offered using a combination of asynchronous and synchronous computer conferencing, slide presentation on the Web, and file transfer systems. Though course delivery is an important component of virtual university, it is not the only component. In order to create a successful academic environment for a distance learner, various support services to students and faculty members have to be included in the plan as integral part of a virtual university.

To aid planning and designing of a virtual university by existing as well as new higher education institutions, this paper will present the Virtual University Reference Model (VURM). This model is intended to be a guideline or a framework for colleges and universities which plan to deliver instruction and support services to distance learners or to be a checklist to evaluate existing distance learning programs. It should assist in quickly identifying necessary service elements and mechanisms for offering a various degree of online distance learning programs.

General Principles

Figure 1 depicts the Virtual University Reference Model. In the model, a virtual university is broken down to four major components: administrative services, student services, resource services, and faculty services. Each component has a different purpose and provides students with different services.

Figure 1: Virtual University Reference Model



Contd...

Notes

The second outer ring in the model shows the types of services a student receives from each of the four noted component areas. The inner three rings represent (from the innermost): (1) the student and his or her relationship to each of these four areas; (2) transmission systems with which the services can be accessed by students; and (3) applications and tools to be used in offering the service elements in the outer ring. It is important to note that in this model students are the centre of the model and all the service components and elements are depicted in relation to the students.

Transmission Systems

The basic assumption of a virtual university is that all the services and instructions are offered at a distance using some kind of transmission systems or telecommunications technology. In the early days of distance learning, postal mail and telephone were the predominant systems of delivery. However, with the advancement of telecommunications technology and the prevalence of the Internet, a variety of delivery technologies become available though the cost is still the limiting factor for students to have access to the full array of technologies available. Still the most dominant mode of transmission systems used by students is plain old telephone lines with up to 33.6kbps modems. Some higher bandwidth technologies such as Integrated Services Digital Network (ISDN) and cable modems are becoming common among those who use the Internet very heavily for business, entertainment, or education from home. As more and higher bandwidth applications utilizing voice and video capabilities become available and integrated into online educational delivery, the attention must be paid not to discriminate students based on their technological accessibility. There is a large difference in terms of technological accessibility between corporate students who have access to the Internet through a high-speed local area network and home-based students who only have access to the Internet through regular telephone lines with a modem. The students with a limited bandwidth should not be penalized with the limited technological accessibility in comparison to those who have a high-speed access. To ensure that, the technology of least common denominator has to be deployed or the institution has to be prepared to provide those less fortunate students with necessary technologies.

In future, it is foreseeable that the bandwidth will become less of an issue as many alternative telecommunications technologies provided by a variety of telecommunications companies, not limited to existing local telephone companies, will become widely available at an affordable rate.

Tools

In a traditional classroom, information is exchanged between the teacher and students, students and other students, and students to the teacher. Sights and sounds are used for exchanging the information. Communication is either direct between the teacher and students or mediated using various media such as overhead projector transparencies, video and audio tapes.

In distance learning information must be exchanged over a distance. Therefore, telecommunication technology is often employed. Teachers and students at different locations can no longer see and hear each other face-to-face; instead, telecommunications media such as video and audio tapes, bulletin boards, email, facsimile, chat rooms, audio and video conferencing, on-line applications over the Internet, etc., must now be used to facilitate communication. Similarly, information presented by the instructor must be communicated and support services must be provided over various telecommunications media to the distant students.

Contd...

Notes

Questions:

1. Comment upon the statement: "A virtual campus is a complex task involving many different aspects of higher education administration and instructional delivery".
2. What is the transmission systems meant for?
3. How does information are exchanged in a traditional classroom.

Source: http://www.westga.edu/~distance/aoki13.html

4.5 Summary

- A reference model of real-time systems is needed to focus on aspects of the system relevant to its real-time timing and resource properties.
- A system may be characterized by 3 models.
- A workload model describing the applications.
- A resource model describing the system resources.
- The scheduling and resource management algorithms that define how the application uses the resources.
- Resources always mean a re-usable resource.
- Each computation or data transmission that is executed repeatedly at regular or semi-regular time intervals in order to provide a function of the system is modelled as a period task.
- A periodic task occurs at regular intervals.
- A task is aperiodic if the jobs in it have either soft deadlines or no deadlines.
- 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.
- Data dependency cannot be captured by a precedence graph.

4.6 Keywords

Aperiodic Task: A task is aperiodic if the jobs in it have either soft deadlines or no deadlines.

Periodical Task Model: The periodic task model is a well-known deterministic workload model.

Resource Model: A resource model describes the system resources available to the applications.

Sporadic Tasks: Tasks containing jobs that are released at random time instants and have hard deadlines are sporadic tasks.

Transmission Link: A transmission link connecting an on-board flight management system to the ground controller is a different type of processor from the link connecting two air traffic control centres.

Workload Model: Workload model is the model that describes the applications supported by the system.

Notes

4.7 Review Questions

1. What are processors and resources? Give some examples.
2. What are passive resources?
3. Describe the temporal parameters of real-time workload.
4. What do you mean by fixed, jittered, and sporadic release times?
5. Explain release-time jitter.
6. Define execution time.
7. What are periodic task model?
8. What are precedence constraints?
9. Define the term data dependency.
10. List some of the major differences between precedence graph and task graph.
11. Give some of the usual examples of data dependency.

Answers: Self Assessment

1.	Reference	2.	Three
3.	Workload	4.	Resource
5.	Resource management	6.	Re-usable
7.	True	8.	False
9.	False	10.	True
11.	False	12.	True

4.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

csperkins.org/teaching/rtes/lecture02.pdf

en.wikipedia.org/wiki/Reference_model

www.dei.unipd.it/corsi/so2/05-Reference-Model.pdf

www.iust.edu.sy/courses/4.Reference_Model_RTS.pdf

Notes

Unit 5: Real-time System Dependencies

CONTENTS

Objectives

Introduction

5.1 Other Types of Dependencies

5.2 Functional Parameters

5.3 Resource Parameters of Jobs and Parameters of Resources

5.3.1 Pre-emptivity of Resources

5.3.2 Resource Graph

5.4 Scheduling Hierarchy

5.4.1 Feasibility, Optimality and Performance Measures

5.5 Summary

5.6 Keywords

5.7 Review Questions

5.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss the Types of Dependencies
- Explain the Functional Parameters
- Define the Resource Parameters of Job
- Analyse Parameters of Resources
- Enumerate Hierarchy Scheduling

Introduction

Tasks often have explicit dependencies specified among themselves, implicit dependencies are more common. Tasks might become inter-dependent for several reasons. A common form of dependency arises when one task needs the results of another task to proceed with its computations. The positional error computation task can meaningfully run only after the "current position determination" task completes. Further, even when no explicit data exchanges are involved, tasks might be required to run in certain order. Dependency among tasks severely restricts the applicability of the results on task scheduling. The reason is that EDF and RMA schedulers impose no constraints on the order in which various tasks execute. Schedules produced by EDF or RMA might violate the constraints imposed due to task dependencies. In this unit, we will discuss real-time system dependencies.

5.1 Other Types of Dependencies

In this section, we will discuss some other types of dependencies.

Notes

Temporal Dependency

Some jobs may be constrained to complete within a certain amount of time relative to one another. Jobs are said to have temporal distance constraint, if their distance must be no more than some finite value. Consider the display of video frames and the accompanying audio of a person speaking.

Did u know? To have lip synchronization, the time between the display of each frame and the generation of the corresponding audio segment must be no more than 160 msec.

In a task graph, temporal distance constraints among jobs are represented by the temporal dependency edges.

AND/OR Precedence Constraints

In classical model, a job with more than one immediate predecessor must wait until its immediate predecessors have been completed before its execution can begin. We can call such jobs AND jobs and dependencies among them AND precedence constraints. AND jobs are represented by unfilled circles in the task graph. For example, a job transmits message to user, its immediate predecessors are jobs that set up a connection for message, encrypt the message and check the user's account for properties like quality of delivery service. OR job is one which can begin execution at or after its release time provided one or some of its immediate predecessors has been completed. It is represented as square vertices.

Conditional Branches

Similarly in the classical model, all the immediate successors of a job must be executed; an outgoing edge from every vertex expresses an AND constraint.

This convention makes it inconvenient for us to represent conditional execution of jobs.

For every second do the following

```
       Process radar returns
       Generate track records
       Perform track association
       for the target T on each of the established tracks do

If the target T is within distance D from self, do the following

       Analyze the trajectory of T.

       If T is on collision course with self, sound alarms

Else

       Compute the current distance of T from self

       If the current distance is larger than previous distance

                                                        Drop the track of T
```

Notes

```
        End  if

End  if
        End  for
```

Conditional Branches and Pipeline Relationship

This system can be easily moulded by a task graph that has edges expressing OR constraints. Only one of all the immediate successors of a job whose outgoing edges express OR constraints is to be executed. Such job is called a branch job. These jobs are represented by filled circles. The sub-graph that begins from a vertex representing a branch job and ends at the vertex representing the associated join job is called a conditional block.

Caution Only one conditional branch in each conditional block is to be executed.

Pipeline Relationship

A dependency between a pair of producer-consumer jobs that are piped can theoretically be represented by a precedence graph. A pipeline relationship between jobs by a pipeline edge is exemplified by the dotted edge between the jobs.

5.2 Functional Parameters

Following are the functional parameters of real-time system dependencies:

Pre-emptivity of Jobs

Executions of jobs can often be interleaved. The scheduler may suspend the execution of a less urgent job and give the processor to more urgent job. The suspended job can be resumed again. A job is pre-emptible if its execution can be suspended at any time to allow the execution of other jobs and later it can be resumed from the point of suspension. A job is non-pre-emptible if it must be executed from start to completion without interruption. Consider jobs that model the transmission of data frames in a token ring. If transmission of frame is interrupted before its completion then the entire frame must be re-transmitted from the start. So to save bandwidth, we can make this job non-pre-emptible.

Sometimes, a job may be pre-emptible everywhere except of r a small portion which is constrained to be non-pre-emptible. An interrupt handling job usually begins by saving the state of the processor (register, stack pointer, program counter).This small portion of the job has to be mad non-pre-emptible to because suspending it at that state may cause serious errors in the data structures shared by other jobs. During pre-emption, the system must first save the state of the pre-empted job at the time of pre-emption so it can resume the job from that state. The system must prepare the execution environment for the pre-empting job before starting the job. Save the current contents of register, load the new processor status register.

Criticality of Jobs

The criticality of a job is a positive number that indicates how critical the job is with respect to other jobs. It is also called priority or weight. The more important the job the higher its priority is. During an overload when it is not possible to schedule all the jobs to meet their deadlines, it

Notes

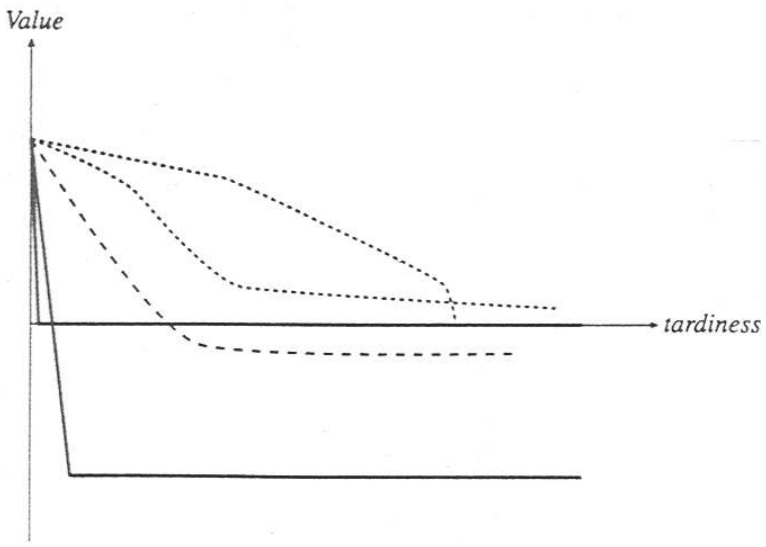
may make sense to sacrifice the less critical jobs so that the more critical jobs can meet their deadline. In a flight control and management system, the job that controls the flight of the aircraft is more critical than the navigation job that determines the current position relative to the chosen course.

Notes **Laxity Type and Function**

The laxity type of a job indicates whether its timing constraints are soft or hard. The laxity type of a job is sometimes supplemented by a usefulness function. In hard real-time systems solid steps are given in figure further. The usefulness of the result becomes zero or negative as soon as the job is tardy. It is then better not to execute the job than to execute it and complete it late. Release a bomb on the target. The dashed and dotted lines show two other usefulness functions dotted, e.g. transaction on Point of sales system, more delay more unsatisfied customer. Eventually usefulness is zero when customer leaves.

Dashed, usefulness is decreasing more rapidly and becomes negative, Stock exchange, late results will be misleading. Usefulness function is shown below:

Figure 5.1: Usefulness Function



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Self Assessment

Fill in the blanks:

1. Tasks often have …… dependencies specified among themselves, …… dependencies are more common.
2. Tasks might become …… for several reasons.
3. A common form of dependency arises when one task needs the results of another task to proceed with its ……

Notes

4. The …… computation task can meaningfully run only after the "current position determination" task completes.
5. In the classical model, all the immediate successors of a job must be executed; an outgoing edge from every vertex expresses an …… constraint.
6. A dependency between a pair of producer-consumer jobs that are piped can theoretically be represented by a …… graph.

5.3 Resource Parameters of Jobs and Parameters of Resources

Every job requires a processor throughout its execution. A job may also execute in parallel on a number of processors, and the number of processors it requires may vary during its execution. The amount of time required by a job to complete may be the function of the number of processors used to execute the job. The resource parameters of a job give us the type and number of processors required by a job to execute as well as how the length of time of computation varies with different number of processors. In this course we will mostly consider jobs that are executed on a single processor.

In addition to processor, a job may also require some other resources. The resource parameters of each job give us the type of processor and the units of each resource type required by the job and the time intervals during its execution when the units are required. These parameters provide the information that is needed to support resource management decisions.

5.3.1 Pre-emptivity of Resources

A resource is non-pre-emptible if each unit of the resource in constrained to be used serially, i.e. once a non-pre-emptible resource is allocated to a job, other jobs needing the unit must wait until the job completes its use. If jobs can use every unit of resource in an interleaving fashion, the resource is pre-emptible. The lock on a data object in a database is non-pre-emptible. The transaction can be preempted on the processor by other transactions that are not waiting for the locks held by it. In the token ring example, this is better to model the token ring as a non-pre-emptible resource (to avoid resubmission) and leave message transmission jobs pre-emptible. We can preempt the transmission of a less urgent message in preference to more urgent massages.

5.3.2 Resource Graph

The configuration of the resources can be described using a resource graph. There is a vertex R_i for every processor or resource R_i in the system. The attributes of the vertex are the parameters of the resource. The resource type of a resource tells us whether the resource is a processor or a passive resource, and its number give us the available number of units. Edges in a resource graph represent the relationship among resources.

There are two types of edges.

- An edge from a vertex R_i to another R_k can mean that R_k is a component of R_i (memory is part of a computer)
- This edge is an is-a-part-of edge.
- The root of each tree represents a major component which contains subcomponents represented by vertices in the tree.
- Computer is root, main memory and so on are children.

Notes

Some edges in resource graphs represent connectivity between components. These edges are called accessibility edges. If there is a connection between two CPUs in the computer, then each CPU is accessible from the other computer and there is an accessibility edge from each computer to the CPU on the other computer.

5.4 Scheduling Hierarchy

Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols. The module which implements these algorithms is called the scheduler. The scheduler assigns processors to jobs, or jobs to processors. We say that a job is scheduled in a time interval on a processor if the processor is assigned to the job, and hence the job executes on the processor, in the interval. The total amount of processor time assigned to a job according to a schedule is the total length of all the time intervals during which the job is scheduled on some processor. By a schedule we mean an assignment of all the jobs in the system on the available processors produced b the scheduler. During this course, we will assume that the scheduler works correctly. By correctness, we mean that the scheduler produces only valid schedulers.

A valid schedule satisfies the following conditions.

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

We are assuming that jobs do not run in parallel on more than one processor to speed up their execution.

5.4.1 Feasibility, Optimality and Performance Measures

A valid schedule is a feasible schedule if every job completes by its deadline. We say that a set of jobs is schedulable according to a scheduling algorithm if when using the algorithms the scheduler always produces a feasible schedule. A hard real-time scheduling algorithm is optimal if the algorithms (scheduler) always produce a feasible schedule if the given set of jobs has feasible schedules. Or, if an optimal algorithm fails to find a feasible schedule, we can conclude that the given set of jobs cannot feasibly be scheduled by any algorithm.

In the case where all the jobs have the same release time and deadline, the problem of scheduling the jobs to meet their deadline is in essence the same as that of scheduling to minimize the completion time of the job which completes last among all jobs. The response time of this job is the response time of the set of jobs as a whole and is often called the makespan of the schedule. If the makespan is less than or equal to the length of their feasible interval, the jobs can meet their deadline.

The most frequently used performance measure of jobs that have soft deadlines is their average response time. The performance of scheduling algorithms is compared on a given set of jobs based on the average response times of jobs when scheduled according to them. The smaller the average time better the job is. In a system that has mixture of hard and soft deadline jobs, the objective is to minimize the average response time of soft deadline jobs and making sure that all hard deadline jobs complete in time. Since there is no advantage in completing hard deadline jobs, we can delay their execution to get a better average response time.

Notes

Task Prepare a chart depicting certain performance measure of jobs that have soft deadlines and compare it with those having hard deadlines.

For many soft real applications, it is acceptable to complete some jobs late or discard late jobs. For such application, suitable performance measure includes the miss rate and loss rate. The former gives the percentage of jobs that are executed but completed too late, and the latter give the percentage of jobs thwart are discarded. When it is impossible to complete all the jobs in time, a scheduler may choose to discard some jobs. Thus scheduler may increases the loss rate but would complete more jobs in time. Similarly, reducing the loss rate may lead to an increase in miss rate. Depends on the application invalid rate is the sum of miss and loss rates and the percentage of all jobs that do not produce a useful result. We want to keep invalid rate as small as possible.

Lab Exercise Search a well-labelled diagram for internal component parameter.

Self Assessment

State whether the following statements are True or False:

7. The configuration of the resources cannot be described using a resource graph.
8. Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols.
9. The most frequently used performance measure of jobs that have soft deadlines is heir average response time.
10. For many soft real applications, it is acceptable to complete some jobs late or discard late jobs.
11. When it is impossible to complete all the jobs in time, a scheduler may choose to discard some jobs.
12. Scheduler may increase the loss rate but would complete more jobs in time.

Case Study

Python as Technology Enabler for TTTech's Development Software

TTTech, founded in 1998, focuses on developing a technology for safety-critical real-time systems. Our central technology is the Time-Triggered Protocol (TTP), a communication protocol used in embedded systems for distributed fault-tolerant applications, such as drive-by-wire and fly-by-wire vehicles. For TTP, we provide chip models for implementing the necessary hardware in silicon, services to customers using the technology, and a broad range of tools.

When starting out in 1998, we urgently needed tools for configuring TTP-based networks. Several people in the team already had experience in writing configuration tools for

Contd...

Notes

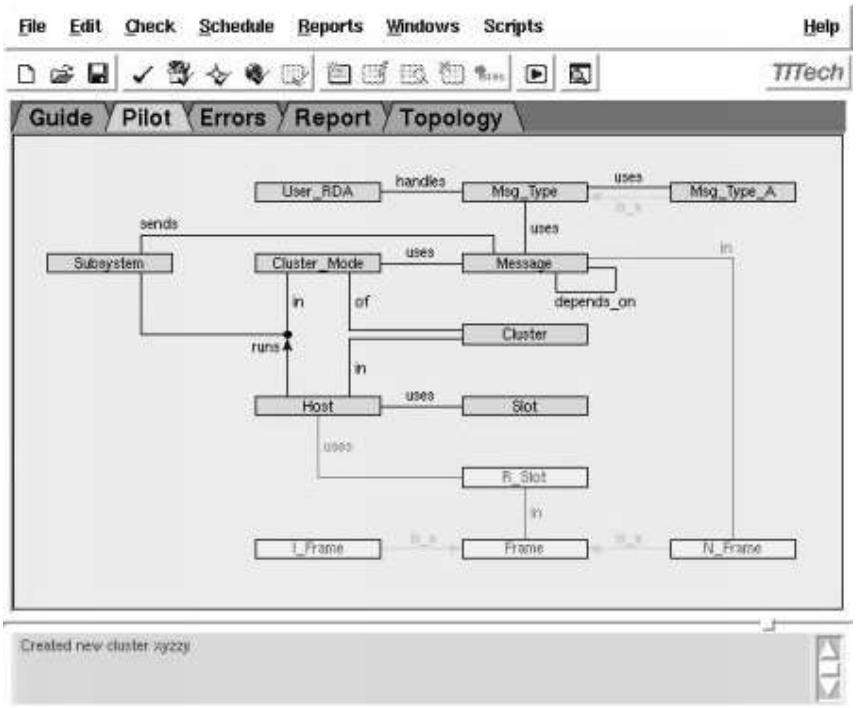
embedded systems, mostly in C and C++. Based on those experiences, it was clear that we needed some way to speed up development. One of the requirements was platform independence, our target platforms being all different Windows platforms and optionally Linux. The configuration tools had to be scriptable, in order to interface easily with other tools used by our customers.

These requirements suggested the use of a scripting language able to support a cross-platform GUI toolkit. A prototype of the user interface was built with Python and Tkinter over a weekend. The quick result dispelled top-management doubts, and the project was started with Python.

Implementation

Within three months, an initial team of two people (one person working full-time, one part-time) was able to develop and release the first version of our TTP network configuration tool, TTP-Plan. We had anticipated that parts of the program, notably the scheduler, would have to be recoded as Python extensions in C after the first prototype's completion. In fact, however, TTP-Plan is still coded in pure Python, yet our customers appreciate it for its excellent performance.

Figure 1: The *TTT-Plan Pilot Allows* Users to Manipulate Object Data and Links between Objects Zoom in



For the first tool, TTP-Plan, we created a framework in Python that let us apply Python's introspection features in order to reuse, for several different purposes, the information already present in the code.

For example, the help texts present in the tool are reused to produce the reference section in the written documentation. The object model of the configuration tool is visible to the user, displayed in a "Pilot-Window". The Pilot lets the user manipulate object data, and links between objects, by clicking on the respective item (see Figure 1). This click opens an

Contd...

Notes

object editor for modification of attributes. The object editor is parameterized with the type of object in the model. The code for drawing the object model, for the object editor, and for extracting the help information, is all part of the framework, and can be reused by other tools in the TTP-Tools family. This framework was of great help when building the node configuration tool TTP-Build, which uses a network configuration generated by TTP-Plan to produce the configuration of an individual node in the network.

Productivity of Programmers Using the Framework

Meanwhile, new tools were built using the existing framework. A programmer new to both Python and our framework was able to build a new configuration tool for the LIN protocol, a product that is now called LIN-Plan, within three months. The tool was initially built for a first-time customer and is now generally available. It will soon see its second release.

Tools not Written in Python

The good experiences with the Python-based tools are confirmed by other projects originally coded in C++. The development of TTP-Load, the download tool, started in 1999. We decided to write this tool in C++ because the tool had to use a low-level Ethernet driver in order to communicate with an Ethernet-to-TTP gateway, called a monitoring node. The protocol for communicating with the monitoring node at the time used raw Ethernet frames.

The year 2001 saw two major changes. A new TTP controller was introduced, and we faced a redesign of TTP-Load because the download protocol of the chip had changed, and support of different download targets was difficult. Communication over raw Ethernet to the monitoring node used a hard-to-maintain driver that needed constant adaptation to new Windows versions. Instead of porting the driver to new Windows versions at that time Windows NT was supported but we had urgent requests for Windows 2000 and XP was already visible on the horizon we decided to use standard internet protocols (UDP for the old monitoring node and TCP for the new one) for communication. For TTP-Load, which does not have performance requirements for the user interface, it was decided to rewrite the tool from scratch, entirely in Python. This was done by one person not familiar with the framework within six weeks.

In our experience, programming productivity is substantially higher with Python than with C++. This is substantiated in detail in a paper presented at OSCON 1999 by one of our developers.

Today, five people are involved with the daily development of our Python-coded tools. Relying on our Python framework, we are able to quickly implement new features, and to adapt the tools to newly available hardware on short notice.

To summarize, with Python we were able to produce a mission-critical set of software products success of TTP as a communication protocol depends on the availability of high-quality configuration tools within a short time frame. The code is very easy to maintain, and recoding in Python a whole medium-sized project proved an easy task too. Overall, using Python has led to a successful product line with an ever-growing number of satisfied customers.

Questions:

1. Describe the significance of Python in TTTech.
2. What is the scope of Python?

Source: http://www.python.org/about/success/tttech/

Notes

5.5 Summary

- Executions of jobs can often be interleaved.
- The scheduler may suspend the execution of a less urgent job and give the processor to more urgent job. The suspended job can be resumed again.
- A job is pre-emptible if its execution can be suspended at any time to allow the execution of other jobs and later it can be resumed from the point of suspension.
- A job is non-pre-emptible if it must be executed from start to completion without interruption. Consider jobs that model the transmission of data frames in a token ring.
- If transmission of frame is interrupted before its completion then the entire frame must be re-transmitted from the start.
- A valid schedule is a feasible schedule if every job completes by its deadline.
- We say that a set of jobs is schedulable according to a scheduling algorithm if when using the algorithms the scheduler always produces a feasible schedule.
- A hard real-time scheduling algorithm is optimal if the algorithms (scheduler) always produce a feasible schedule if the given set of jobs has feasible schedules.
- If an optimal algorithm fails to find a feasible schedule, we can conclude that the given set of jobs cannot feasibly be scheduled by any algorithm.
- In the case where all the jobs have the same release time and deadline, the problem of scheduling the jobs to meet their deadline is in essence the same as that of scheduling to minimize the completion time of the job which completes last among all jobs.
- The response time of this job is the response time of the set of jobs as a whole and is often called the makespan of the schedule.
- If the makespan is less than or equal to the length of their feasible interval, the jobs can meet their deadline.

5.6 Keywords

Conditional Block: The sub-graph that begins from a vertex representing a branch job and ends at the vertex representing the associated join job is called a conditional block.

Conditional Process Graph (CPG): A graph captures both the flow of data and its control.

Job Resource Parameters: These are indicated processor and resource requirements (what, how many and when).

Job: A unit of work scheduled and executed by system.

Non-pre-emptible Job: A job is non-pre-emptible if it must be executed from start to completion without interruption.

Parameters: These are variables that used for run-time substitution in the properties of Resources.

Pre-emptible Job: A job is pre-emptible if its execution can be suspended at any time to allow the execution of other jobs and later it can be resumed from the point of suspension.

Temporal Distance Constraint: Jobs are said to have temporal distance constraint, if their distance must be no more than some finite value.

Notes

5.7 Review Questions

1. What do you understand by dependencies?
2. Explain the importance of conditional process graph.
3. Define the term parameter.
4. What are resource parameters?
5. Write down major differences between job and task.
6. Explain the term job resource parameter.
7. What is functional parameter?
8. What is feasibility?
9. Define the term optimality.
10. What are various performance measures in real-time systems?

Answers: Self Assessment

1.	Explicit, implicit	2.	inter-dependent
3.	computations	4.	positional error
5.	AND	6.	precedence
7.	False	8.	True
9.	True	10.	True
11.	True	12.	True

5.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the Real World*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall).

Online links

nptel.iitm.ac.in/courses/Webcourse.../**Real**%20**time**%20**system**/.../module3.pdf

csperkins.org/teaching/rtes/lecture02.pdf

https://wiki.engr.illinois.edu/.../realTimeSystems/Dependency+Management+Framework

www.uni-ulm.de/fileadmin/website_uni_ulm/.../KollmannPKS2010.pdf

Notes

Unit 6: Commonly used Approaches to Real-time Scheduling

CONTENTS

Objectives

Introduction

6.1 Clock Driven Approach

6.2 Weight Round–Robin Approach

6.3 Priority Driven Approach

6.4 Dynamic versus Static System

6.5 Summary

6.6 Keywords

6.7 Review Questions

6.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Clock Driven Approach
- Enumerate Weight Round–Robin Approach
- Explain Priority Driven Approach
- Analyse Dynamic versus Static System

Introduction

Commonly used Approaches to Real-time Scheduling are clock driven approach, weight round robin approach, priority-driven approach, etc. Round-robin is commonly used for scheduling time-shared applications. Weighted Round-Robin extends the basic round-robin algorithm with weights. Priority-Driven Scheduling assigns priorities to jobs. Overview of different classes of real-time scheduling algorithms is given in this unit.

6.1 Clock Driven Approach

In clock-driven scheduling, decisions on what jobs execute at what times are made at specific time instants. Instants are chosen a priori before the system begins execution. Usually all the parameters of hard real-time jobs are fixed and known.

Caution Schedule of the jobs is computed off-line and is stored for use at run time.

The scheduler schedules the jobs according to this schedule at each scheduling decision time thus scheduling overhead during run-time is minimized. A hardware timer can be used; the timer is set to expire periodically without the intervention of the scheduler.

Notes

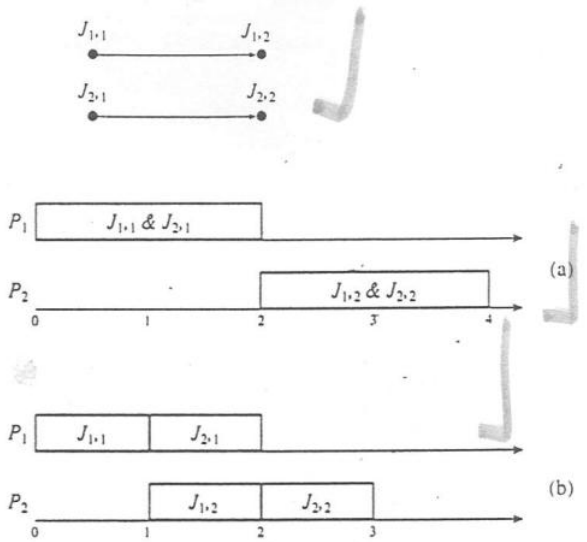
Notes When system is initialized, scheduler selects and schedules the jobs that will executed until the next scheduling decision time and then waits for expiration of the timer. When the timer expires, the scheduler awakes and repeats these actions.

6.2 Weight Round–Robin Approach

Commonly used for scheduling time-shared applications. Every job joins a FIFO queue when it is ready for execution. The job at the head of the queue executes for at most onetime slice. Time slice is the basic granule of time that is allocated to jobs, typically in the order of tens of milliseconds. If the job doesn't complete by the time slice, it is preempted and placed at the end of the queue.

Did u know? If there are n ready jobs in the queue, each job gets one time slice every n time slices, that is a round. That's why WRR approach is also called processor sharing algorithm.

Real-time traffic in high-speed switched networks is an example Different jobs may also be given different weights. A job with weight wt get wt time slices every round, and the length of a round is equal to the sum of the weights of all ready jobs. We can adjust the weights of jobs to speed up or retard the progress of each job towards its completion. If used to schedule precedence constrained jobs, the response time of a chain of jobs can be very large, and that is why not good for such jobs. But a successor job may be able to incrementally consume what is produced by a predecessor. Since a job and its successors can execute concurrently is a pipelined fashion, this is a nice approach for such jobs. In the figure (a), the release time of all jobs are 0, and their execution times are 1, $J_{1,1}$ and $J_{2,1}$ execute on processor P1 and $J_{1,2}$ and $J_{2,1}$ on processor P2. Suppose $J_{1,1}$, is the predecessor of $J_{1,2}$ and J2,1 is the predecessor of $J_{2,2}$. Both sets of jobs complete approximately at time 4.



(b) shows that if the jobs on each processor are executed one after another, one of the chains can complete at time 2, while the other can complete at time 3. 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.

Notes

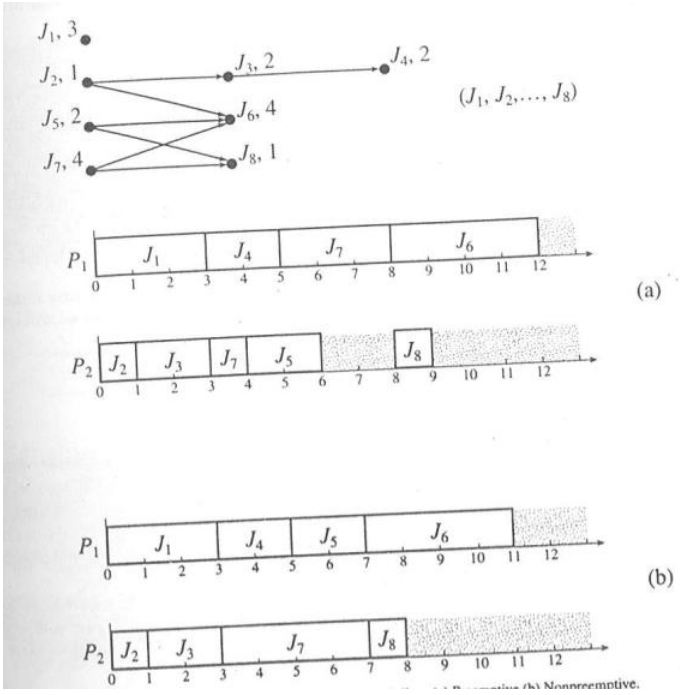
Self Assessment

Fill in the blanks:

1. Round-robin is commonly used for scheduling …… applications.
2. …… Scheduling assigns priorities to jobs.
3. In …… scheduling, decisions on what jobs execute at what times are made at specific time instants.
4. …… are chosen a priori before the system begins execution.
5. Usually all the parameters of hard real-time jobs are …… and known.
6. The …… of a round is equal to the sum of the weights of all ready jobs.

6.3 Priority Driven Approach

It refers to a large class of scheduling algorithms that never leave any resource idle intentionally. A resource is idle only when no job requiring the resource is ready for execution. Scheduling decisions are made when events such as releases and completions of jobs occur. Hence these are also called Event-Driven. It is also called Greedy, list and work-conserving scheduling. When a processor or resource is available and some job can use it to make progress, such an algorithm never makes the job wait. Jobs ready for execution are placed in one or may more queues ordered by the priorities of the jobs. At the scheduling time, the jobs with the highest priorities are scheduled and executed on the available processors. As we can dynamically change the priorities of jobs, even round robin scheduling can be thought of as priority-driven. The Figure below shows a classical precedence graph. Number next to jobs is their execution time. J5 is released at time 4. All other jobs are released at time 0. We want to schedule and execute the jobs on two shared memory processors P1 and P2 (cost of communication is negligible). The schedulers on the processors keep one common priority queue of ready jobs.



Notes

The priority list is J1>J2>J3……>J8 All jobs are preemptible, scheduling decisions are made whenever some job becomes ready for execution or some job complete. (a) shows the schedule of the jobs on the two processors generated by priority-driven algorithms. At time 0, jobs J1, J2, J5 and J7 are ready but J1, J2 have higher priority than J7.

6.4 Dynamic versus Static System

In the discussed methods, jobs that are ready for execution are placed in a priority queue common to all processors. When a processor is available, the job at the head of the queue executes on the processor. Such multiprocessor systems are referred as a dynamic system as jobs are dynamically dispatched to processors. Jobs were migratable and a job migrates if it starts execution on a processor, is preempted, and later resumes on a different processor. Another approach 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. Jobs are moved only when the operation mode of the system changes or some processor fails. Such a system is called a static system. 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. For example considering last figure, put J1, J2, J3 and J4 on P1 and the remaining jobs on P2. The priority list is segmented into (P1,P2,P3,P4) and (P5,P6,P7,P8). The scheduler of processor P1 uses the former and that of P2 uses the later. Jobs on P1 will finish at 8 and on P2 at 11. J2 completes by time 4 and J6 starts at time 6 so precedence constraint is also satisfied. While dynamic systems may be more responsive on the average, their worst case real-time performance may be poorer than static systems. As there are no reliable techniques to validate the timing constraints of dynamic systems (exist for static), Most Hard RTSs are Static.

Self Assessment

State whether the following statements are True or False:

7. Priority Driven Approach refers to a small class of scheduling algorithms that never leave any resource idle intentionally.
8. A resource is non-idle only when no job requiring the resource is ready for execution.
9. Scheduling decisions are made when events such as releases and completions of jobs occur.
10. Jobs that are ready for execution are placed in a priority queue common to all processors.
11. When a processor is available, the job at the head of the queue executes on the processor. Such multiprocessor systems are referred as a dynamic system as jobs are dynamically dispatched to processors.
12. A job does not migrate if it starts execution on a processor, is preempted, and later resumes on a different processor.

Task Prepare a chart to specify differences between dynamic and static system.

Notes

Case Study

AVL, Routing and Scheduling

Intelligent AVL provides a Web-based view of real-time information on delivery arrivals at distribution centres or hubs, enabling dispatchers to adapt routes quickly and efficiently according to exceptions. The AVL Service also features a wireless application that uses Global Positioning Satellite (GPS) locating capabilities to provide real-time data on driver progress against established route plans and enable on-the-spot decision making to meet customer commitments.

Wawa Improves Delivery Operations with Intelligent AVL Service from Descartes

Philadelphia-based convenience store chain Wawa, Inc. has a history that spans more than two centuries. Originally a textile supplier in the 1800s, by the turn of the century it began operations as a provider of home delivery services for milk products. In 1964 the opening of the first Wawa Food Markets marked the beginning of the Wawa chain, a provider of fresh produce, ready-to-go salads and deli products to customers in five states.

Today the Wawa family includes more than 1,500 convenience stores and 13,000 associates and wholesale customers (schools, hospitals and nursing homes) across five states. Wawa offers a large selection of food items that includes its award-winning freshly brewed coffee, and Wawa branded dairy products, juices and iced tea beverages.

Wawa uses a private fleet to manage deliveries for its direct distribution operations (all other delivery functions are managed by third party providers). Its fleet includes 40 twenty-foot trucks, 65 tractors and 80 trailers. Store deliveries range from two to four times a week on a fixed schedule depending on location, with a two-hour time window allowance.

A Limited View to a Growing Network

As a high end convenience store chain, Wawa has unique delivery challenges. For example, deliveries cannot be made between 10 a.m. and 7 p.m.; and drivers must also reach their destinations within two-hour time windows. Over the years, Wawa has relied on various feature-rich elements of the Descartes Routing and Scheduling solution to streamline its delivery operations. Wawa first began using basic mapping functions provided by Descartes, and before long, extended its implementation to include automated master route planning, **followed by integration with its warehouse management system to automatically generate order charges based on routes and sequences.**

Through each implementation phase, Wawa has achieved significant results. For example, by simply optimizing its route planning functions, Wawa was able to reallocate eight trucks to wholesale deliveries, providing significant cost savings on the direct distribution side of the business.

As its delivery network grew however, it became a constant challenge for Wawa to ensure that deliveries were reaching their destinations within designated time windows. Wawa knew that it needed to find more efficient ways to track its drivers, allocate resources and optimize capacity utilization to maintain margins and reduce fuel costs without compromising quality of service.

"As a high-end convenience store and producer with a distribution network that spans five states, having up-to-the-minute information on driver and fleet activities is critical to maintaining high levels of customer service," said Don Kane, Distribution Manager at Wawa.

Contd...

Notes

Wawa realized that not all of its drivers were following the planned routes and in some cases were by-passing stops. "Since drivers were paid hourly, we wanted to have a better sense of where they were and what they were doing," adds Kane. To truly understand whether drivers were following planned routes and making the scheduled stops on time, Wawa realized it needed real-time insight into the delivery activities throughout the day.

This would enable them to save costs relating to added miles and overtime hours, as well as maximize existing resources while minimizing the impact of delays and exceptions en route.

Gaining Real-time Insight with Descartes Intelligent AVL

Wawa decided to leverage its long-established relationship with Descartes and once again extend its implementation by integrating the Descartes Intelligent Automated Vehicle Locator™ (AVL) Service into its direct distribution operations. "We have had so many years of success with Descartes Routing and Scheduling solution, that when we wanted to integrate AVL functions into our operations, Descartes was the natural choice," Kane explains.

Descartes' Intelligent AVL provides a Web-based view of real-time information on delivery arrivals at distribution centres or hubs, enabling dispatchers to adapt routes quickly and efficiently according to exceptions. The AVL Service also features a wireless application that uses Global Positioning Satellite (GPS) locating capabilities to provide real-time data on driver progress against established route plans and enable on-the-spot decision making to meet customer commitments. Both are designed to work with Descartes Routing and Scheduling solutions.

Improved Efficiency, Reduced Costs

By integrating AVL services with its existing Descartes Routing and Scheduling solution, Wawa has improved security; is better able to manage driver salaries and overtime by building highly effective, balanced routes and tracking them; and has introduced the ability to compare planned versus actual truck locations. It has also been able to increase capacity utilization to 90% (the 10% spare capacity is required to accommodate last minute deliveries).

In addition, customer service representatives can now access the system and advice customers on the status of their deliveries in real-time. "Before, customer support was calling the dispatcher at least 10 times a day on average to find out the status of a delivery – and that was on a good day," said Kane. "You can imagine how many more customer calls would come in if there was a truck breakdown or a major accident delay. With the new system, customer support can be proactive and answer the questions to any inbound calls on the spot."

Descartes solution can also be scaled as Wawa continues to expand its delivery operations. "We can easily put new drivers on the road or on new routes," explains Kane. He adds that the stores can also benefit from the implementation. "The beauty of this system is, because we charge the stores for delivery costs, they get to enjoy the savings too." According to Joanne Cochrane, Implementation Consultant at Descartes

"Wawa now has greater insight into whether hourly-paid drivers are following planned routes and making the scheduled stops. Having this level of visibility empowers them to save significant costs relating to added miles and overtime hours, among other areas. It is an outstanding example of how our customers are building on their existing Descartes solution offerings to drive even more efficiencies and savings.

Contd...

Notes

Questions:

1. Describe how Wawa Improves Delivery Operations with Intelligent AVL Service does from Descartes.
2. Explain the Limited View to a Growing Network.
3. How are Improved Efficiency and reduced Costs interrelated?

Source: http://knowledge.descartes.com/resources/mobile-resource-management-performance-management/avl—routing-scheduling-case-study-gaining-real-time-delivery-insight-with-intelligent-avl-Wawa

6.5 Summary

- Round-robin is commonly used for scheduling time-shared applications.
- Priority-Driven Scheduling assigns priorities to jobs.
- In clock-driven scheduling, decisions on what jobs execute at what times are made at specific time instants.
- Instants are chosen a priori before the system begins execution.
- Usually all the parameters of hard real-time jobs are fixed and known.
- The length of a round is equal to the sum of the weights of all ready jobs.
- Priority Driven Approach refers to a large class of scheduling algorithms that never leave any resource idle intentionally.
- A resource is idle only when no job requiring the resource is ready for execution.
- Scheduling decisions are made when events such as releases and completions of jobs occur.
- Jobs that are ready for execution are placed in a priority queue common to all processors.
- When a processor is available, the job at the head of the queue executes on the processor. Such multiprocessor systems are referred as a dynamic system as jobs are dynamically dispatched to processors.
- A job migrates if it starts execution on a processor, is preempted, and later resumes on a different processor.

6.6 Keywords

Priority-driven Algorithms: This type of algorithms refers to a large class of scheduling algorithms that never leave any resource idle intentionally. It is greedy because it tries to make locally optimal decisions.

Round-robin Approach: It is commonly used for scheduling time-shared applications 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.

Scheduler: It repeats a pre-computed schedule the pre-computed schedule needs to be stored only for one major cycle.

Weighted Round-robin Algorithm: It has been used for scheduling real-time traffic in high-speed switched networks.

Notes

6.7 Review Questions

1. What is Priority-driven approach?
2. Describe the priority-driven algorithms.
3. What do you understand by Round-robin approach? Describe it.
4. What is Weight Round-robin approach?
5. Who are schedulers?
6. List any one difference between Round-robin approach and Weight Round-robin approach.
7. What are Clock-driven approach, and its algorithms? Describe in brief.
8. Describe the differences between dynamic versus static system.

Answers: Self Assessment

1.	Time-shared	2.	Priority-Driven
3.	Clock-driven	4.	Instants
5.	Fixed	6.	Length
7.	False	8.	False
9.	True	10.	True
11.	True	12.	False

6.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall).

Online links

xlab.cn.nctu.edu.tw/.../Embedded%20OS%204-1%202007%20handout.pdf

faculty.cs.tamu.edu/bettati/Courses/663/2009C/.../sched_approaches.pdf

en.wikipedia.org/wiki/Scheduling_analysis_real-time_systems

www.ict.kth.se/courses/.../CommonApproachesToScheduling-1x2.pdf

Notes

Unit 7: Commonly used Algorithm to Real-time Scheduling

CONTENTS

Objectives

Introduction

7.1 Real-time Scheduling

7.1.1 Interrupts and Tasks

7.2 Effective Release Time and Deadlines

7.3 Optimality of the EDF and LST Algorithm

7.4 Non-optimality of the EDF and LST Algorithm

7.5 Summary

7.6 Keywords

7.7 Review Questions

7.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Real-time Scheduling
- Enumerate Effective Release Time and Deadlines
- Explain Optimality of the EDF and LST Algorithm
- Analyse Non-optimality of the EDF and LST Algorithm

Introduction

The use of computers to control safety-critical real-time functions has increased rapidly over the past few years. As a consequence, real-time systems — computer systems where the correctness of a computation is dependent on both the logical results of the computation and the time at which these results are produced — have become the focus of much study. Since the concept of "time" is of such importance in real-time application systems, and since these systems typically involve the sharing of one or more resources among various contending processes, the concept of scheduling is integral to real-time system design and analysis. In this unit, we will discuss commonly used Algorithm to Real-time Scheduling.

7.1 Real-time Scheduling

A real-time operating system is a system that schedules execution of tasks in a timely deterministic manner, and is scalable.

The scheduler follows a set of algorithms that determine which task executes at each moment. Preemptive priority-based scheduling is a *mandatory* property of the operating system we

Notes

evaluate for use in our application. A task or 'thread' in this context is a sequential execution stream in a multi-tasking OS.

Key characteristics for a real-time system are

- reliability (stability),
- predictability (it must be deterministic),
- performance (the faster the better),
- compactness to code size (space is money) and
- scalability (one starts with 2 tasks and eventually one ends up with a seven-headed dragon).

The RTOS we are looking at follow the 'scheduler paradigm'.

Did u know? There are two main levels or categories of execution that can code-execute logic:

- The interrupt level, and
- The task level

7.1.1 Interrupts and Tasks

An interrupt is an asynchronous exception of which the source is an internal or external hardware device (note: if the OS supports system calls then software interrupts are possible as well). Implementation dependent Programmable Interrupt Controllers (PICs) prioritize multiple interrupt sources so that at any time the highest priority interrupt is presented to the core CPU for processing.

Did u know? Nesting interrupts refers to the ability of a higher priority interrupt source to preempt the processing of a lower priority interrupt.

Higher priority interrupt will interrupt the lower priority interrupt, the higher priority interrupt will run to completion (unless interrupted by a higher priority interrupt), and then the lower priority interrupt will resume. The above explanation refers to so-called maskable interrupts: lower priority interrupts are masked (prevented from running) by higher priority interrupts. Non-maskable interrupts are also possible; these have the highest priority (sometimes even higher than the RTOS itself) and cannot be prevented from happening.

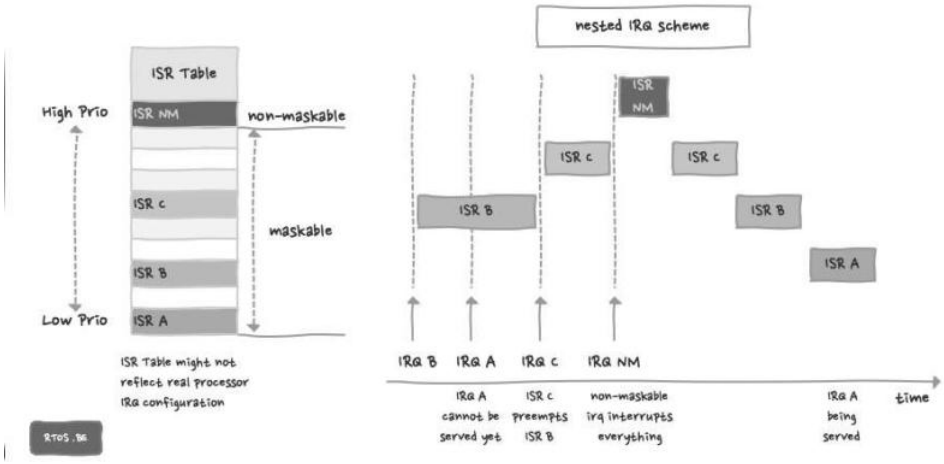
The performance of an embedded (real-time) system is proportional to:

- the latencies involved with the interrupt, and
- the task handling scheme.

Interrupt latency is the time from when the interrupt request is triggered until the ISR (= the code of the interrupt service routine) is running. Context information handling (of a task or interrupt) is additional overhead and increases the latency but should also be deterministic and known. Task switching latency is the time when a task should be made to 'run', involving the time it takes to store the context information of the running task and restoring the context information of the to-be-run task.

Notes

Figure 7.1: Example for Interrupts and Tasks



Source: http://www.rtos.be/wp-content/uploads/2012/11/interrupts.png

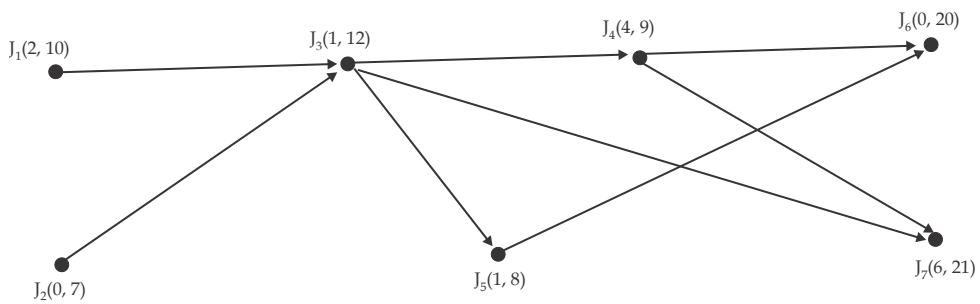
Notes **Interrupt Priorities**

Interrupt priorities are assigned within the PIC; the system designer usually has full or at least some control on them (this is PIC platform specific), but how to determine the priorities of the various tasks? Generally, the more important the task, the higher the priority it should get assigned. This is true for interrupts and tasks, but more criteria are important and should be considered.

7.2 Effective Release Time and Deadlines

Sometimes given release times and deadlines of jobs are inconsistent with the precedence constraints of the jobs. That is the release time of a job may be later than that of its successors, and its deadline may be earlier than that of its predecessors. So we derive a set of effective release times and deadlines from these timing constraints together with precedence constraints.

Figure 7.2: Example of Release Time and Deadlines



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Notes

If there is only one processor then

- ***Effective Release time:*** of a job without predecessors is equal to its given release time. The effective release time of a job with predecessors is equal to the maximum value among its given release time and the effective release times of all of its predecessors.
- ***Effective Deadline:*** without a successor is equal to its given deadline. The effective deadline of a job with successors is equal to the minimum value among its given deadline and the effective deadlines of all of its successors.

Here in figure,

- The numbers are the release times and deadlines.
- As J1 and J2 have no predecessors, their effective release times are 2 and 0.
- Release time of J3 is 1, but that of J2 is 2. Hence, its effective release time is 2.
- Effective release times of the rest of the jobs are 4, 2, 4, 6 respectively.
- J6 and J7 have no successors so their effective deadlines are 20 and 21.
- No issue with deadlines of J4 and J5 as well.

The given deadline of J3 is equal to 12 which is larger than 8 and 9, Hence its effective deadline is 8 and that of J1 and J2 8 and 7.

- The effective deadline of a job should be as early as the deadline of each of its successors minus the execution time of the successor.
- The effective release time of a job is that of its predecessor plus the execution time of the predecessor
- The more accurate calculating is unnecessary if there is only one processor.
- It has been shown that it is feasible to schedule any set of jobs on a processor according to their given release times and deadlines if and only if it is feasible to schedule the set according to their effective release times and deadlines just defined.
- Not true for multiple processors.

Task Draw the effective timing constraints process.

Self Assessment

Fill in the blanks:

1. A real-time operating system is a system that schedules execution of tasks in a timely deterministic manner, and is ……
2. The …… follows a set of algorithms that determine which task executes at each moment.
3. …… priority-based scheduling is a *mandatory* property of the operating system we evaluate for use in our application.
4. An …… is an asynchronous exception of which the source is an internal or external hardware device.

Notes

5. Release times and deadlines of jobs are …… with the precedence constraints of the jobs.

6. Effective Release time of a job without …… is equal to its given release time.

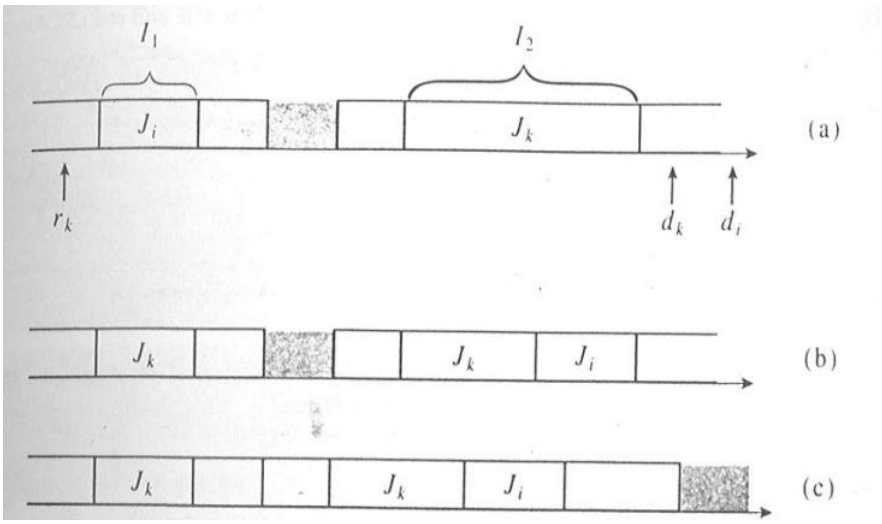
7.3 Optimality of the EDF and LST Algorithm

Priorities can be assigned to jobs according to their deadlines. The earlier the deadline, the higher the priority is. Such scheduling algorithm is called the Earliest-Deadline-First (EDF) algorithms. When Preemption is allowed and jobs do not contend for resources, the EDF algorithm can produces a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if J has feasible schedules. Any feasible schedule of J can be systematically transformed into an EDF schedule.

Example: Suppose that in a schedule, parts of J_i and J_k are scheduled in the intervals I1 and I2. The deadline d_i of J_i is later than the deadline d_k of J_k, but I_1 is earlier than I_2. There are two cases; first the release time of J_k may be later than the end of I_1 so J_k cannot be scheduled in I_1 as the two jobs are already scheduled on the EDF basis in these intervals.

In the second case the release time r_k of J_k is before the end of I_1; we assume that r_k is no later than the beginning of I_1. We swap J_i and J_k, if the interval I_1 is shorter than I_2, we move the portion of J_k that fits in I_1 forward to I_1 and move the entire portion of J_i scheduled in I_1 backward to I_2 and place it after J_k (Figure b). We can do a similar swap if the interval I_1 is longer than I_2; we move the entire portion of J_k scheduled in I_2 to I_1 and place it before J_i and move the portion of J_i that fits in I_2. The result of this swap is that these two jobs are now scheduled on the EDF basis. This transformation is repeated for every pair of jobs that are not scheduled on the EDF basis according the given non-EDF schedule until no such pair exists. The schedule may still not be an EDF schedule if some interval is left idle while there are jobs ready for execution but are scheduled in a later interval. This idle time can be eliminated by moving one or more of these jobs forward into the idle interval and leave the interval where the jobs were scheduled idle. This process is repeated if necessary until the processor never idles when there are jobs ready for execution. Thus every feasible schedule can be transformed into a preemptive EDF schedule.

Figure 7.3: Example 1 for Optimality of the EDF and LST Algorithm



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Notes

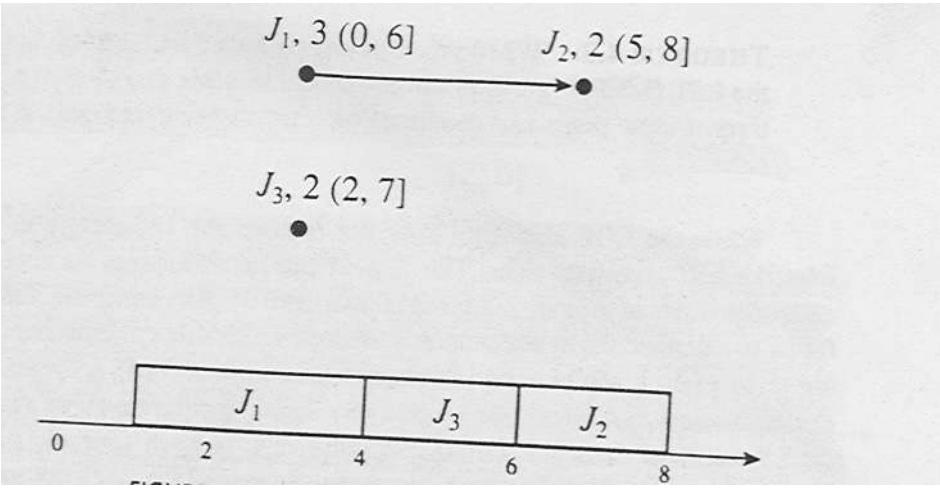
Caution When the goal of scheduling is to meet deadline, there is no advantage to completing any job sooner than necessary.

We may want to postpone the execution of hard real-time jobs for some reason. So we can use another latest release time (LRT) algorithm. This algorithm treats release times as deadlines and deadlines as release times and schedules the jobs backwards, starting from the latest deadline of all jobs, in priority-driven manner, to the current time. The priorities are based on the release times of jobs; the later the release time, the higher the priority. As it may leave the processor idle when there are jobs ready for execution, the LT algorithms is not a priority-driven algorithm.

Consider the figure below:

- IN the precedence graph, the number next of the job name is the executing time of the job.
- Its feasible interval is given by the range of time next to its execution time.

Figure 7.4: Example 2 for Optimality of the EDF and LST Algorithm



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

- The latest deadline among all jobs is 8. Hence time starts at 8 and goes backwards to 0.
- At time 8, J2 is ready and is scheduled. At time 7, J3 is also ready to be scheduled, but because J2 has a later release time, it has a higher priority.
- J2 is scheduled from 7 to 6. When J2 completes at time 6, J1 is ready.
- However as J3 has a higher priority and is, there from, scheduled from 6 to 4.

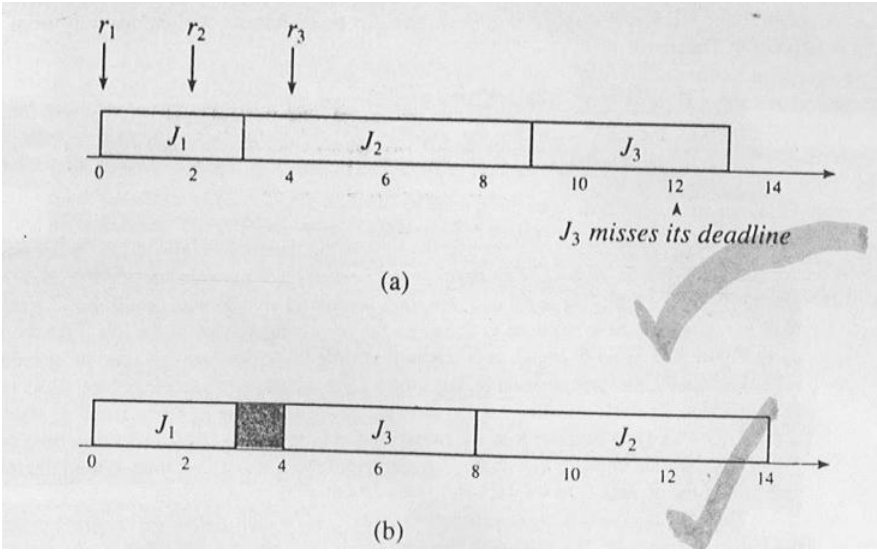
Finally J1 is scheduled from 4 to 1. The result is a feasible schedule.

7.4 Non-optimality of the EDF and LST Algorithm

It is natural to ask whether EDF and the LST algorithms remain optimal if preemption is not allowed or there is more than one processor.

Notes

Figure 7.5: Non-optimality of the EDF and LST Algorithm



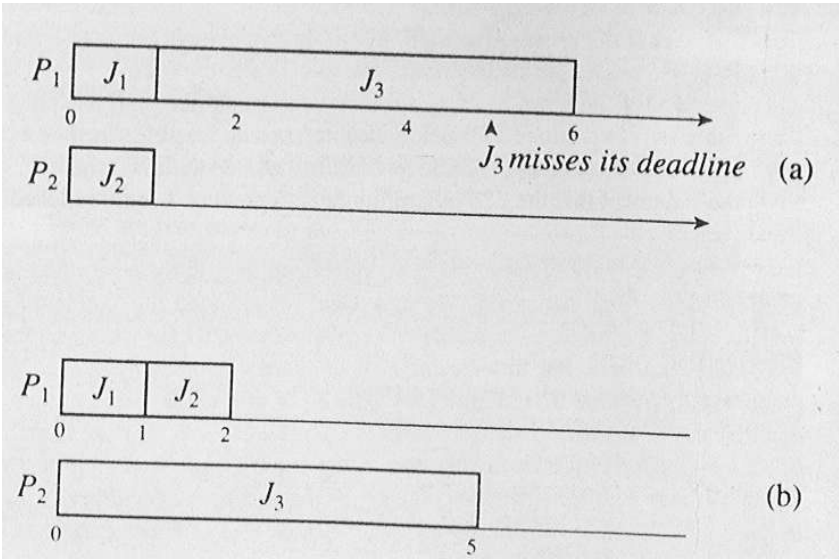
Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

The system shown in the figure has three independent, non-preemptible jobs J1, J2 and J3. The explanation of the figure is as follows:

- The release times are 0, 2 and 4 and execution times are 10, 14 and 12.
- (a) shows the schedule produced by the EDF algorithm
- When J1 completes at time 3, J2 has already been released but not J3. Hence J2 is scheduled.
- When J3 is released at time 4, J2 is executing.
- Even though J3 has an earlier deadline and hence a higher priority, it must wait until J2 completes s there is no preemption
- Thus J3 misses its deadline.
- (b) provides a feasible schedule.
- At time 3 when J1 completes, the processor is left idle, even though J2 is ready for execution.
- When J3 is released at 4, it can be scheduled ahead of J2, allowing both to meet deadline.
- (b) Can never be provided by a priority-driven scheduling algorithm as they never leave a processor idle when jobs are ready.
- This figure shows that the EDF algorithm is not optimal for scheduling preemptible jobs on more than one processor.
- J1, J2 and J3 have execution times 1, 1, 5 and deadlines 1, 2, 5 respectively and release time 0.
- On two processors, J1 and J2 are scheduled on the processors at time 0 because of higher priority so J3 misses deadline.
- An algorithm that could assign higher priority to J3 could do the job. (like LST)

Notes

Figure 7.6: EDF Algorithm



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Caution LST is not optimal for scheduling jobs on more than one processor.

Self Assessment

State whether the following statements are True or False:

7. Effective Deadline with a successor is equal to its given deadline.
8. Priorities cannot be assigned to jobs according to their deadlines.
9. The later the deadline, the higher the priority. Such scheduling algorithm is called the Earliest-Deadline-First (EDF) algorithms.
10. The priorities are based on the release times of jobs; the later the release time, the higher the priority.
11. The EDF algorithm can produces a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if J has feasible schedules.
12. Any feasible schedule of J can be systematically transformed into an EDF schedule.

Case Study

Real-Time Dynamic Voltage Scaling

This case study concerns the performance of real-time embedded systems under dynamic voltage scaling schedulers. Dynamic voltage scaling is a technique used to address the trade-off between the battery life and the performance of battery powered processors.

Contd...

Notes

In the classic model of real-time embedded systems we have a set of tasks T1, Tn that need to be executed periodically. Each task Ti has an associated period (Pi) and worst case execution time (Ci). The task Ti is released every Pi time units and is required to complete its execution before some deadline which is typically defined as the end of its period, i.e. it must finish before its next release.

Schedulers

The real-time DVS schedulers we consider are taken from [PS01], which are based on two of the standard real-time schedulers, namely Rate Monotonic (RM) and Earliest-Deadline First (EDF) schedulers.

- RM schedulers are static and assign the task priority according to the period of the tasks- they always select the task with the shortest period that is ready to run.
- EDF schedulers are dynamic and order tasks by their deadlines, giving highest priority to the released task with the most immediate deadline.

Static Voltage Scaling Schedulers

The first schedulers that consider simply extend RM and EDF by selecting the lowest possible operating frequency that will allow the scheduler to meet all the deadlines of the task set. The frequency is set statically and is only changed when the task set is changed.

When selecting the operating frequency, one must take into account that, if the frequency is scaled by a factor of a("(0,1]), then the worst case execution time is scaled by a factor of 1/a, while the period and deadline remain unchanged. Extending the standard schedulability tests for RM and EDF schedulers in [PS01], the following static voltage scaling algorithms were reached:

```
Static_RM:
rm_test(á):
if (" Ti " { T1, … ,Tn | P1d" … d"Pn }. ceil(Pi/P1) × C1 + … + ceil(Pi/
Pi) × Ci d" á)
return true else return false
select_frequency
use lowest frequency fi such that RM_test(fi/f1) is true
Static_EDF:
edf_test(á):
if (C1/P1+ … +Cn/Pn ≤ α) return true
else return false
select_frequency
use lowest frequency fi such that EDF_test(fi/f1) is true
```

Cycle Conserving Schedulers

In general tasks are completed far sooner than their worst case execution time and the following algorithms of aim to take advantage of this. In particular, when a task uses less than its worst case execution time, the scheduler can use this fact to reduce the operating frequency. Note that, when a task is released for execution we do not know how much computation time it will require, and therefore at this point the algorithms make the conservative assumption that the task will require its worst case execution time. However, when a task completes, one can calculate the number of cycles that were not required by the task, and hence at this time the algorithms recalculate the frequency taking into account these unused cycles (hence the term "cycle conserving").

Contd...

Notes

Below, we give the cycle conserving extensions of the RM and EDF schedulers

```
Cycle_Conserving_RM:
assume fj is the frequency set by the static scaling algorithm
select_frequency:
set sm to the maximum cycles until the next deadline
use lowest frequency fi such that
(d1+ … +dn) / sm d” fi/fm
upon task_release(Ti):
set c_lefti to Ci
set cm to the maximum cycles until the next deadline
set sj to sm × fi/fm
allocate_cycles(sm)
select_frequency
upon task_completion(Ti):
set c_lefti to 0
set di to 0
select_frequency
during_task_execution(Ti):
decrement c_lefti and di
allocate_cycles(k):
for i=1 to n, Ti “ { T1, … Tn | P1 d” … d” Pn } // tasks sorted by period
if c_lefti < k
set di equal to c_lefti
set k equal to k-c_lefti
else
set di equal to k
set k equal to 0
Cycle_Conserving_EDF:
select_frequency:
use lowest frequency fi such that
(U1+ … +Un d” fi/f1)
upon task_release(Ti):
set Ui to Ci/Pi
select_frequency
upon task_completion(Ti):
set Ui to ci/Pi where ci equals the actual cycles used in the invocation
select_frequency
```

Prism Models

For this case study, we consider a processor with three operating frequencies, 1, 0.75 and 0.5, with corresponding voltages 5,4 and 3. The task set is of size three with the following parameters:

Contd...

Notes

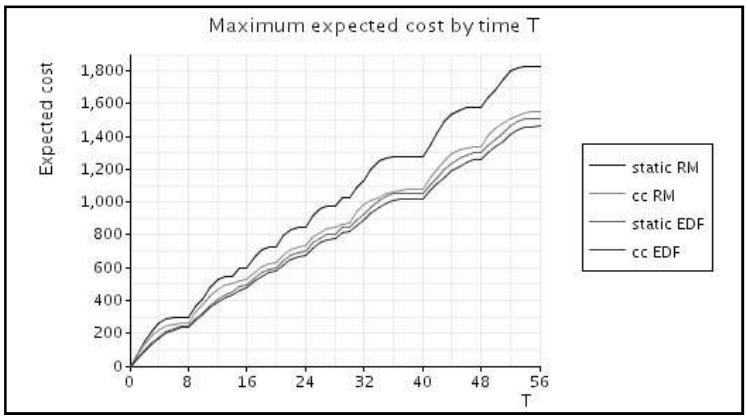
task:	Pi (period)	Ci (WCET)
T1	8	3
T2	10	3
T3	14	1

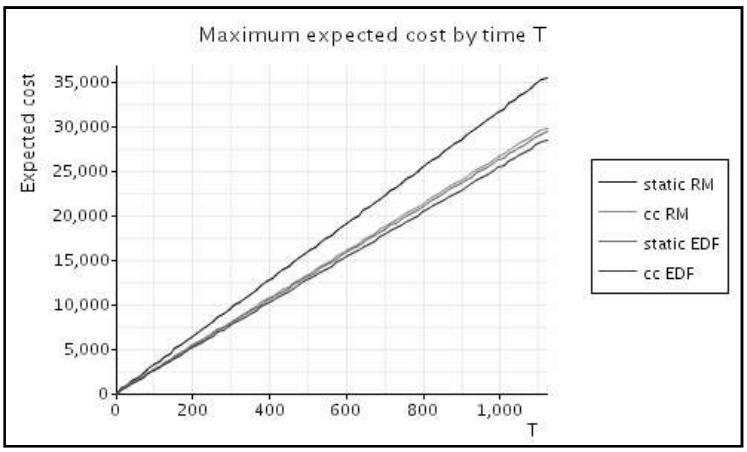
To model the system in Prism, we discredit time and, to simplify the modelling process, the division of one time unit is such that, for each of the possible frequencies, the number of discrete time steps required by a task to finish on its WCET is an integer. For example, if a task is run at the frequency 0.75 and the task takes WCET time units, it will require WCET*4/3 time units to complete. Furthermore, we suppose that the completion time distribution of a task is uniformly distributed between 1 and its WCET.

Results:

We analyses the different schedulers by comparing the (maximum) expected cost by time T as T varies. The cost function we consider is given by the square of the current voltage since this value is proportional to the energy consumption.

The graphs below plot the expected costs for the different schedulers as T varies. The EDF based approach can lead to greater power savings compared to the RM based algorithms.





Questions:

1. Analyse the case and interpret it.
2. Write down the case facts.

Source: http://www.prismmodelchecker.org/casestudies/voltage.php

Notes

7.5 Summary

- A real-time operating system is a system that schedules execution of tasks in a timely deterministic manner, and is scalable.
- The scheduler follows a set of algorithms that determine which task executes at each moment.
- Preemptive priority-based scheduling is a *mandatory* property of the operating system we evaluate for use in our application.
- An interrupt is an asynchronous exception of which the source is an internal or external hardware device.
- Release times and deadlines of jobs are inconsistent with the precedence constraints of the jobs.
- Effective Release time of a job without predecessors is equal to its given release time.
- Effective Deadline without a successor is equal to its given deadline.
- Priorities can be assigned to jobs according to their deadlines.
- The earlier the deadline, the higher the priority. Such scheduling algorithm is called the Earliest-Deadline-First (EDF) algorithms.
- The priorities are based on the release times of jobs; the later the release time, the higher the priority.
- the EDF algorithm can produces a feasible schedule of a set J of jobs with arbitrary release times and deadlines on a processor if and only if J has feasible schedules.
- Any feasible schedule of J can be systematically transformed into an EDF schedule.

7.6 Keywords

Earliest-Deadline-First (EDF) Algorithm: It is an algorithm that is optimal when used to schedule jobs on a processor as long as preemption is allowed and jobs do not contend for resources.

Latest Release Time (LRT) Algorithm: This is an algorithm that treats release times as deadlines and deadlines as release times and schedules the jobs backwards, starting from the latest deadline of all jobs, in "priority-driven" manner, to the current time.

Real-time Operating System: A real-time operating system is a system that schedules execution of tasks in a timely deterministic manner, and is scalable.

Real-time Scheduling: It is the form of scheduling that offers a way of predicting the timing behaviour of complex multi-tasking computer software.

RM Schedulers: These are static and assign the task priority according to the period of the tasks – they always select the task with the shortest period that is ready to run.

7.7 Review Questions

1. Define the term real-time scheduling.
2. What is effective release times and deadline?
3. What is the limitation of effective deadline?

Notes

4. Differentiate between optimality of the EDF and LST algorithms.
5. Define earliest-deadline-first.
6. What is an effective deadline means?
7. Differentiate between non-optimality of the EDF and the LST algorithms.
8. What is latest release time algorithm?
9. Differentiate between preemption and non-preemption.
10. Differentiate between EDF and the LST Algorithms.

Answers: Self Assessment

1.	Scalable	2.	Scheduler
3.	Preemptive	4.	interrupt
5.	inconsistent	6.	predecessors
7.	False	8.	False
9.	False	10.	True
11.	True	12.	True

7.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M.(1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall).

Online links

research.cs.queensu.ca/TechReports/Reports/2005-499.pdf -

www.cs.unc.edu/~anderson/papers/rtss08b.pdf

www.dca.ufrn.br/~affonso/DCA_STR/aulas/stankovic2.pdf

www.ulb.ac.be/di/ssd/goossens/baruahGoossens2003-3.pdf

Notes

Unit 8: Working of Real-time Scheduling

CONTENTS

Objectives

Introduction

8.1 Challenges in Validating Timing Constraint in Priority Driven System

8.2 Off-line Versus On-line Scheduling

8.3 Summary

8.4 Keywords

8.5 Review Questions

8.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Give an overview of multiprocessor and distributed real-time systems
- Describe Challenges in Validating Timing Constraint in Priority Driven System
- Analyse Off-line Versus On-line Scheduling.

Introduction

In multiprocessor and distributed real-time systems, scheduling jobs dynamically on processors can be used to achieve better performance. However, analytical and efficient validation methods for determining whether all the timing constraints are met do not yet exist for systems using modern dynamic scheduling strategies, and exhaustive methods are often infeasible or unreliable since the execution time and release time of each job may vary.

8.1 Challenges in Validating Timing Constraint in Priority Driven System

Priority-driven system or schedulers are online schedulers which comprise of following features:

- Schedule is not pre-computed.
- Scheduler assigns priority to released jobs and places them in ready job queue in priority order.
- At each scheduling decision time, scheduler updates the ready job queue and schedules job with highest priority.

The assumptions of Priority-Driven Scheduling are:

- Every job is ready for execution as soon as it is released, and can be preempted at any time.
- Scheduling decisions are made immediately upon job releases and completions.
- The context switch overhead is negligibly small compared with execution times.
- The number of priority levels is unlimited.

Notes

Did u know? Priority-driven scheduling is easy to implement. It does not require the information on the release times and execution times of the jobs a priori.

Apart from the merit Priority-driven scheduling has demerits too. The timing behavior of a priority driven system is non-deterministic.

Caution It is difficult to validate that all jobs scheduled in a priority-driven manner meet their deadlines when the job parameters vary.

Validating Priority-Driven Systems is Much Harder than Time-Driven Systems

Timing characterization of Priority-Driven systems is difficult since these systems are not as deterministic as the Time-Driven systems.

Validation of a system requires determination that all jobs meet their deadlines as specified by the system requirements:

- Predictability of tasks
- Execution of jobs in a set of independent preempt-able jobs is predictable if priority-driven scheduling is used on a processor
- Validation of a set of independent, preemptive, static, single processor priority-driven systems are relatively easy since the execution of the jobs are predictable
- Maximum execution time is used to validate these systems

Self Assessment

Fill in the blanks:

1. In multiprocessor and distributed real-time systems, scheduling jobs …… on processors can be used to achieve better performance.
2. Analytical and efficient …… methods for determining whether all the timing constraints are met do not yet exist for systems using modern dynamic scheduling strategies.
3. Exhaustive methods are often …… or …… since the execution time and release time of each job may vary.
4. Priority-driven scheduling is …… to implement.
5. Priority-driven system or schedulers are …… schedulers.
6. In event triggered systems, events invoke an online scheduler, which takes a decision based on a set of …… rules.

8.2 Off-line Versus On-line Scheduling

The heated debate on whether offline or online scheduling is preferable for real-time systems has taken attention away from the question how the two are actually defined and what their essential differences are. Results and arguments in the debate are typically based on a conglomerate of assumptions, system design issues, and particular views of scheduling rather

Notes

than the actual cores of the ways to schedule themselves. We do not wish to get involved into the "TT' vs. ET" debate; rather have a look the related scheduling paradigms to determine what their differences are. We conclude they are much smaller than perpetuated clichees.

One of the central choices for scheduler design is that of the activation paradigm, i.e., when are events recognized, who initiates activities, when are these decisions taken? In conventional systems, the event triggered approach is prevalent, in which occurrences of events initiate activities in the system immediately. In time triggered systems, activities are initiated at predefined points in time.

Offline Scheduling: We start with the time triggered approach, contrasting some properties already here to those of event triggered. Initiating activities in the system with the progression of time requires thorough, complete understanding of the system and the environment it will operate in. Scheduling for TT is usually carried out via a scheduling table, which lists tasks and their activation times. An offline algorithm takes complete information about the system activities, which reflect the knowledge about anticipated environmental situations and requirements, and creates a single table, representing a feasible solution to the given requirements. 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. Should a feasible solution not be found, retries are possible, for example, by changing the parameterization of the algorithm or the properties of the task set. At runtime, a very simple runtime dispatcher executes the decisions represented in the table, i.e., which (portion of a) task to execute next. Typically, a minimum granularity of time is assumed for the invocations of the runtime scheduler, so called slots.

Online Scheduling: In event triggered systems, events invoke an online scheduler, which takes a decision based on a set of pre-defined rules, e.g., represented as priorities. An offline 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. Major representative lines of such algorithms are based on fixed priorities, e.g., rate monotonic or dynamic priorities.

Task Write the procedural steps to use the online scheduling.

Fundamentally Different

Given the different assumptions, approaches, and properties, offline and online scheduling could be perceived as very different or even opposing paradigms. We will take a closer look. The ET real-time scheduling process takes sets of tasks with timing constraints and performs a test if these constraints can be met if a given algorithm is used at runtime. The algorithm may take properties of tasks, notably priority or deadline, as input, or determine them as directives, artifacts for the online scheduling algorithm then, the task properties, "priority" are separated from the importance of a task, "deadline" from the timing constraint. Rather, they both serve only to direct the online scheduling algorithm to execute the proper rules for schedulability.

Notes **Scheduling Table**

The scheduling table of offline scheduling provides a "proof by construction" that all timing constraints will be met. In contrast to a proof that no timing constraints could be

Contd...

Notes

violated in any situation, an offline approach only needs to show that one situation exists, i.e., the scheduling table, in which the timing constraints are met: instead of a "for all" proof, considering even situations which may never occur during the runtime of a system, a "there exists one" suffices.

Thus, the process follows the steps: task set with timing constraints-schedulability test and determination of rules (for example, via directives priority or deadline) – execution of rules by runtime scheduler-timing constraints met.

Looking closer, we can see that TT real-time scheduling work in the same way. Instead of a definition of rules, e.g., earliest deadline first, "the decisions on which task to execute are represented in the scheduling table, schedule next task as given table".

While TT scheduling has to assume a periodic world and ET provides flexibility for tasks with not fully known parameters, example, aperiodic, the difference concerns mostly runtime execution without guarantees. When offline guarantees are required, task parameters have to be known offline: without worst case execution, period or maximum arrival frequency, offline guarantees cannot be given, independent of the scheduling paradigm used.

Hence, we can conclude that the terms "offline" and "online" scheduling cannot be seen as disjoint in general. Real-time scheduling requires offline guarantees, which require assumptions about online behavior at design time. At runtime, both offline and online execute according to some (explicitly or implicitly) defined rules, which guarantee feasibility. The question "offline" vs. "online" is thus less black and white, but more about how much of the decision process is Thus, both offline and online are based on a substantial offline part. The question is then where to set the tradeoff between determinism – all decisions offline - and flexibility – some decisions online.

Lab Exercise Search through internet and make points of differences between the terms "offline" and "online" scheduling.

Self Assessment

State whether the following statements are True or False:

7. Defined rules are represented as priorities.
8. An offline 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.
9. Major representative lines of such algorithms are based on fixed priorities, e.g., rate monotonic or dynamic priorities.
10. The scheduling table of offline scheduling provides a "proof by construction" that all timing constraints will be met.
11. When offline guarantees are required, task parameters have to be known online.
12. The terms "offline" and "online" scheduling cannot be seen as disjoint in general.

Notes

Case Study

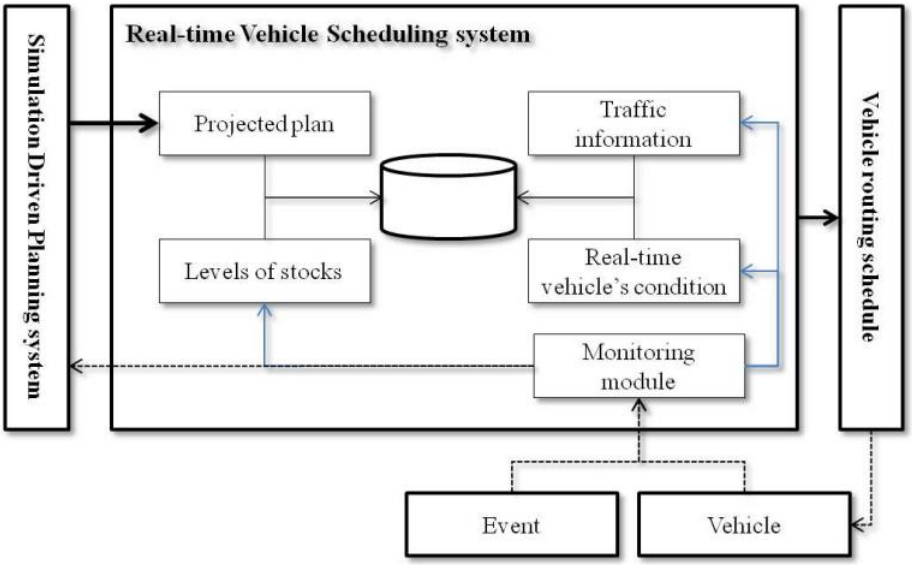
Real-time Vehicle Scheduling System

The real-time vehicle scheduling system (see Figure below) generates the vehicle routing schedule by combining the projected plan provided by the simulation-driven planning system with information which provided by the monitoring modules; this system include:

Traffic information includes traffic reports and estimated travel time of each travel course. This module has a strong effect on travel time.

Level of stocks directly affects production flow, so it is very important. Shortage of stock means loss of business opportunity. Vehicles, in the overage, cannot perform the next schedule before unloading process so the efficiency of vehicles is decreased.

Real-time vehicle's condition provides the vehicle's location, whether it is loading or unloading, and the items being loaded. It is needed to assign work to the appropriate vehicle at the re-planning.



Monitoring Module tracks various data to estimate the throughput of the generated vehicle routing schedule. It senses the changes of production plan and other unexpected occurrences to adjust the generated schedule and determine whether re-planning process is necessary. It is supported by other technology like RFID, which can measure travel time and waiting time automatically

Questions:

1. Explain the concept of real-time vehicle scheduling system.
2. How do we schedule the traffic problem?
3. What is monitoring module means?

Source: orsc.edu.cn/apiems/public/Abstracts/315.doc

Notes

8.3 Summary

- In multiprocessor and distributed real-time systems, scheduling jobs dynamically on processors can be used to achieve better performance.
- Analytical and efficient validation methods for determining whether all the timing constraints are met do not yet exist for systems using modern dynamic scheduling strategies.
- Exhaustive methods are often infeasible or unreliable since the execution time and release time of each job may vary.
- Priority-driven scheduling is easy to implement. It does not require the information on the release times and execution times of the jobs a priori.
- Priority-driven system or schedulers are online schedulers.
- Priority-driven scheduling is easy to implement.
- In event triggered systems, events invoke an online scheduler, which takes a decision based on a set of pre-defined rules, e.g., represented as priorities.
- An offline 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.
- Major representative lines of such algorithms are based on fixed priorities, e.g., rate monotonic or dynamic priorities.
- The scheduling table of offline scheduling provides a "proof by construction" that all timing constraints will be met.
- When offline guarantees are required, task parameters have to be known offline.
- The terms offline" and online" scheduling cannot be seen as disjoint in general.

8.4 Keywords

Efficiency: It is the capacity to do any work**.**

On-line Scheduling Algorithm: In this form of scheduling, scheduler makes each scheduling decision without knowledge about the jobs that will be released in the future the parameters of each job become known to the on-line scheduler only after the job is released.

Validation Algorithm: It is the form of algorithm that **allows** us to determine whether all jobs in a system indeed meet their timing constraints despite scheduling anomalies.

8.5 Review Questions

1. Explain the challenges in validating timing constraints in priority driven system.
2. What are priority-driven schedulers?
3. What is online scheduling?
4. What is offline scheduling?
5. Distinguish between off-line versus on-line scheduling.
6. Write a short note on *validation algorithm.*

Notes

Answers: Self Assessment

1.	dynamically	2.	validation
3.	infeasible, unreliable	4.	easy
5.	online	6.	pre-defined
7.	True	8.	True
9.	True	10.	True
11.	False	12.	True

8.6 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall).

Online links

beru.univ-brest.fr/~singhoff/cheddar/publications/audsley95.pdf

cs.uga.edu/~shelby/pubs/goossensBF2002-8.pdf

www.seas.upenn.edu/~lee/10cis541/lecs/lec-RT-sched-part1-v2-1x2.pdf

csperkins.org/teaching/rtes/lecture03.pdf

Notes

Unit 9: Concept of Clock-driven Scheduling

CONTENTS

Objectives

Introduction

9.1 Notations and Assumptions

9.2 Static, Timer-Driven Schedules

9.3 General Structure of Cyclic Schedules

9.4 Cyclic Executives

9.5 Summary

9.6 Keywords

9.7 Review Questions

9.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Notations and Assumptions
- Enumerate Static, Timer-Driven Schedules
- Explain General Structure of Cyclic Schedules
- Analyse Cyclic Executives

Introduction

The clock-driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock. In the event-driven ones, the scheduling points are defined by certain events which precludes clock interrupts. Clock driven scheduling has an ability to consider complex dependencies, communication delays, and resource contention among jobs when constructing the static schedule, guaranteeing absence of deadlocks and unpredictable delays. Entire schedule is captured in a static table. Different operating modes can be represented by different tables. No concurrency control or synchronization required. If completion time jitter requirements exist, it can be captured in the schedule. In this unit, we will discuss the concept of clock driven system.

9.1 Notations and Assumptions

Clock-driven approach is appropriate only when the system is deterministic, excluding a few aperiodic and sporadic jobs. The restricted assumptions are:

- There are n periodic tasks in the system; n is fixed in an operation mode.
- The parameters of all periodic tasks are known a priori. Variations in the inter release times of job in any periodic task are negligibly small i.e. each job in T_i is released p_i units of time after the previous job in T_i.

Notes

- Each job $J_{i,k}$ is ready for execution at its release time $r_{i,k}$.
- We refer to a periodic task T_i with phase $Ö_i$, period p_i, execution time e_i and relative deadline D_i by the 4-tuple ($Ö_i$, p_i, e_i, D_i)
- (1,10,3,6) means the first job in the task is released and ready at time 1 and must be completed by time 7; second job is ready at 11 and must be completed by 17 and so on.
- Each job executes for 3 units of time.
- Default deadline of each job is its period and phase is 0.

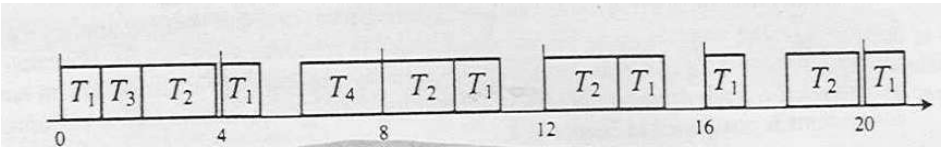
There are aperiodic jobs released at unexpected time instants and there are no sporadic jobs.

9.2 Static, Timer-Driven Schedules

Whenever the parameters of jobs with hard deadlines are known before the system begins t_i execute, a straightforward way to ensure that they meet their deadlines is to construct a static schedule of the jobs off-line. This schedule specifies exactly when each job executes. In this schedule, the amount of processor time allocated to every job is equal to its maximum execution time, and every job completes by its deadline. Among all the feasible schedules, we may want to choose one that is good according to some criteria (processor idles periodically to accommodate aperiodic jobs).

Consider a system that contains four independent periodic tasks. They are T1 = (4,1), T2 = (5,1.8), T3=(20,1) and T4 = (20,2). The utilizations are .25, .36, .05 and .01 , totalling .76 We need to create a schedule for only first hyper period which is 20. A schedule is shown in the figure given below.

Figure 9.1: Scheduler



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Some intervals, such as (3.8, 4), (5, 6) and (10.8, 12) are not used by periodic tasks. We can use these for executing aperiodic jobs.

A straight forward way to implement the scheduler is to store the precomputed schedule as a table. Each entry (t_k, $T(t_k)$) in this table gives a decision time t_k, which is an instant when a scheduling decision has been made and $T(t_k)$ is the name of the task whose job starts at t_k or I. The scheduler timer is used. Immediately after all the tasks have been created and initialized and then at every scheduling decision time, the scheduler, sets the timer so the timer will expire and request an interrupt at the next decision time. Upon receiving a timer interrupt at t_k, the scheduler sets the timer to expire at t_{k+1} and prepares the task $T(t_k)$ for execution. The scheduler then suspends itself, letting the task have processor and execute, when the timer expires again the scheduler repeats its operation.

Input: Stored schedule (t_k, $T(t_k)$) for k = 0,1,…N-1

```
Task SCHEDULER:
See the next decision point i and table entry k to 0;
        set the timer to expire at t_k
        do forever
```

Notes

```
accept timer interrupt;

If an aperiodic job is executing, preempt the job;

Current task T = T(t_k);

increment i by 1;

compute the next table entry k = i mod(N);

set the timer to expire at (i/N )*H + t_k;

if the current task T is i,

       let the job at the head of the aperiodic job queue execute;

else, let the task T execute;

sleep;
End Scheduler
```

Self Assessment

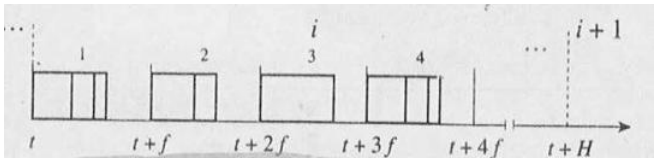
Fill in the blanks:

1. The clock-driven schedulers are those in which the scheduling points are determined by the …… received from a clock.
2. In the event-driven ones, the scheduling points are defined by certain events which …… clock interrupts.
3. …… scheduling has an ability to consider complex dependencies, communication delays, and resource contention among jobs when constructing the static schedule, guaranteeing absence of deadlocks and unpredictable delays .
4. Entire schedule is captured in a …… table .
5. Different operating modes can be represented by different ……
6. Clock-driven approach is appropriate only when the system is ……, excluding a few aperiodic and sporadic jobs.

9.3 General Structure of Cyclic Schedules

Figure below shows a good structure of cyclic schedules.

Figure 9.2: Cyclic Frames



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

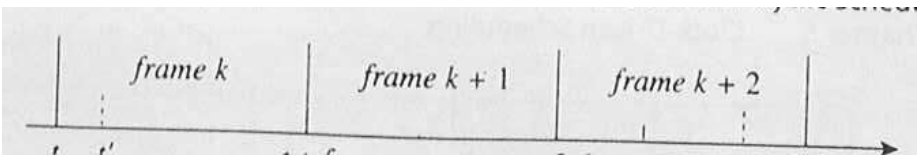
Notes

A constraint enforced by this structure is that scheduling decisions are made periodically, rather than at arbitrary times. The scheduling decision times partition the timeline into intervals called frames. Each frame has length f which is the frame size. Scheduling decisions are made only at the beginning of every frame so there is no preemption in a frame. Phase is always a multiple of frame size. Scheduler also carries out monitoring and enforcement actions at the beginning of each frame to check whether every job scheduled in the frame has indeed been released and ready for execution.

Frame Size constraints:

- The frames should be sufficiently longs so that every job can start and complete its execution within a frame (no preemption).
- Possible If we make the frame size f larger than the execution time of every task Ti (**Constraint 1**).
- The frame size would be chosen so that it divides H. This condition is met if f divides the period pi of at least one task T_i.
- $(p_i/f\ ) - p_i/f = 0$ for at least one i (**Constraint 2**). Thus there are an integer number of frames in each hyperperiod.
- F is the number of frames in a hyperperiod.
- A hyperperiod that begins at the beginning of (kF+1)st frame, for any k =0,1… is called a major cycle.
- To make it possible for the scheduler to determine whether every job completes by its deadline, we want the frame size to be sufficiently small so that between the release time and deadline of every job, there is at least one frame (**Constraint 3**).
- The figure below illustrates the suitable range of f for a task.
- When f is in this range, there is at least one frame between the release time and deadline of every job in the task.
- Now, the figure given below constraints the frame size to be no less than 2. As hyper period is 20; 2, 4, 5, 10 are possible frame sized.
- However only 2 satisfies $(p_i/f\ ) - p_i/f = 0$.
- If we have (15, 1, 14), (20, 2, 26), and (22, 3) we must have only 3, 4, 5.

Figure 9.3: General Structure Cyclic Schedules



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

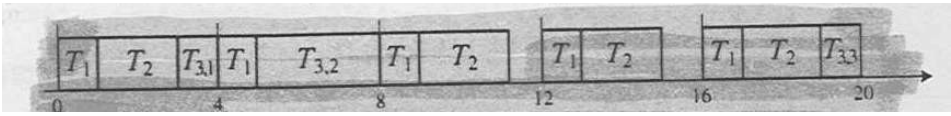
Job Slices

At times the constraints cannot be met, for example, (4,1), (5,2,7), (20,5). For constraint 1 we must have f>= 5 but for constraint 3 we must have f<=4. Now we are forced to partition each job in a task that has a large execution time into slices (sub jobs) with smaller execution times (IN message transmission, divide message into several segments, when job is computational divide

Notes

into non-preemptive procedures). Now the lower bound by constraint 1 has been reduced. Now we can divide each job in (20,5) into a chain of three slices with execution time 1,3,1. So (20,5) now in fact is (20,1), (20,3), (20,1). Now we have 5 tasks and we can choose frame size 4.

Figure 9.4: Example of Job Slices



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

We have not chosen to decompose (20,5) into (20,3) and (20,2) as it would not be possible to fit these tasks together with T1 and T2 in five frames of size 4. T1 must be scheduled in each frame. T2 must be scheduled in 4 out of 5 frames. This leaves one frame with 3 units of time for T3. The other frames only have one unit of time left for T3.

We can schedule two subtasks each with 1 unit of execution time in these frames, but there is not time in any frame for a subtask with execution time 2.

Task Draw a diagram for a cyclic schedule with frame size 4.

9.4 Cyclic Executives

The clock driven scheduler must be modified to accommodate the restriction that scheduling decisions are made only at frame boundaries. The term cyclic executives refer to a scheduler that deterministically interleaves and sequentializes the execution of periodic tasks on a CUP according to a given cyclic schedule. It also makes scheduling decision only at the beginning of each frame and deterministically interleaves the execution of periodic tasks.

However it allows aperiodic and sporadic jobs to use the time not used by periodic tasks. The cyclic executives take over the processor and executes at each of the clock interrupts, which occur at the beginning of frames. When it executes, the cyclic executive copies the table entry for the current frame into the current block. It then wakes up a job, called period task server (Not needed where periodic tasks never overrun, executive just executes the job slices) , and lets the server execute the job slices in the current block. Upon the completion of the periodic task server, the cyclic executive wakes ups the aperiodic jobs in the aperiodic job queue in turn and allows them to use the remaining time in the frame.

The supposition here is that whenever the server or a job completes, the cyclic executives wake up and executes. The system may also have an aperiodic task server, which when awaked executes aperiodic jobs in the aperiodic job queue. In addition to scheduling, the cyclic executive also checks for overruns at the beginning of each frame. If the last job executed in the previous frame is not complete at that time, the cyclic executive preempts the execution of the job if the last job is an aperiodic job. The job remains in the aperiodic job queue and will be resumed whenever there is time again for aperiodic jobs.

If the cyclic executive finds the periodic task server still executing at the time of a clock interrupt, a frame overrun occurs. After checking for overruns, the cyclic executive makes sure that all the job slices scheduled in the current block are ready for execution and then wakes up the periodic task server to execute them. If there is still time after all the slices in the current block are

completed and the aperiodic job queues is nonempty, it lets the job at the head of the aperiodic job queue execute.

Lab Exercise Consider a system that contains four independent periodic tasks. They are $T1 = (3, 2)$, $T2 = (4, 1.8)$, $T3 = (10, 2)$, and $T4 = (15, 2)$, construct a static schedule for the first hyper period of the task.

Self Assessment

State whether the following statements are True or False:

7. A straight forward way to implement the scheduler is to store the precomputed schedule as a table.
8. A constraint enforced by this structure is that scheduling decisions are made periodically, rather than at arbitrary times.
9. The scheduling decision times partition the timeline into intervals called frames.
10. Each frame has length f which is the frame size.
11. Scheduling decisions are made only at the end of every frame so there is no preemption in a frame.
12. Phase is always a multiple of frame size.

Case Study

The MARS Kernel

Time driven real-time systems are of increasing importance in the field of critical computer control applications. Because of their predictable behaviour they are well suited for systems whose correct operation in the time domain must be guaranteed already in the design phase of an application. Time driven systems allow the proof of the correct timing behaviour of an application by construction of a feasible schedule. In the MARS system the time driven approach is realized. The structure of the MARS operating system kernel differs significantly from that of others because of the specific demands which a distributed time driven system imposes on its underlying operating system. Based on the experiences with the first prototype of the MARS operating system (MARS-1), a new operating system kernel, MARS-2, has been developed from scratch. There have been some motivations for the development of MARS-2:

New processor boards ('MARS components') have been developed to fully support the MARS concepts. These boards provide mechanisms to achieve a high self-checking coverage and a highly predictable timing behaviour.

- The introduction of new concepts and mechanisms into the MARS system (e.g. membership protocol, time redundant process execution, shadow component) requires support by the runtime system.
- A predictable timing behaviour should be achieved by the new kernel. Although the system overhead caused by the old implementation was bondable in principle,

Contd...

Notes

the calculated bounds were too high to guarantee the correct timing behaviour of an application already at design time.

- The self-checking coverage of the MARS components has to be high because the fault tolerance mechanisms of MARS are based on it. Whereas the old kernel was not specifically designed in order to meet this requirement, MARS-2 uses both hardware and software mechanisms to increase the self-checking coverage to a sufficiently high degree.MARS-2 is based on a microkernel operating system architecture in contrast to the monolithic kernel of MARS-1.

Questions:

1. What are the functions of time driven real-time systems?
2. Explain in what ways, the MARS system was different from other systems.
3. Describe the self-checking coverage concept.

Source: dl.acm.org/citation.cfm?id=506416

9.5 Summary

- Clock-driven schedulers schedule periodic tasks according to a cyclic schedule.
- Aperiodic and sporadic jobs can be scheduled, if they do not influence other scheduled jobs.
- Applicable to static systems, with a small number of aperiodic jobs.
- The scheduling decision times partition the time line into intervals called frames.
- A straight forward way to implement the scheduler is to store the precomputed schedule as a table.
- A constraint enforced by this structure is that scheduling decisions are made periodically, rather than at arbitrary times.
- The scheduling decision times partition the timeline into intervals called frames.
- Each frame has length f which is the frame size.
- Scheduling decisions are made only at the end of every frame so there is no preemption in a frame.
- Phase is always a multiple of frame size.

9.6 Keywords

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

Cyclic Executive: It refers to a scheduler that deterministically interleaves and sequentializes the execution of periodic-tasks on a CPU according to a given cyclic schedule.

Event-driven Schedulers: In the event-driven ones, the scheduling points are defined by certain events which precludes clock interrupts.

Frame Size: Each frame has length f which is the frame size.

Frames: The scheduling decision times partition the timeline into intervals called frames.

Notes

Parameters of Jobs: These are, with hard deadlines are known before the system begins to execute, a straightforward way to ensure that they meet their deadlines is to construct a static schedule of the jobs off-line.

9.7 Review Questions

1. Describe the notations and assumptions for clock driven scheduling.
2. What is the static scheduler?
3. What are the roles of frames and major cycles?
4. What are the frame size constraints?
5. What is the job slices in scheduling?
6. Why we differentiate the jobs in slices?
7. What are cyclic executives?
8. Describe the various uses of cyclic executive in scheduling.
9. What is the scheduling block?
10. What are scheduling decisions?

Answers: Self Assessment

1.	Interrupts	2.	Precludes
3.	Clock driven	4.	Static
5.	tables	6.	deterministic
7.	True	8.	True
9.	True	10.	True
11.	False	12.	True

9.8 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages,* Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions..

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

www.people.vcu.edu/~wzhang4/egre691/slide2.pdf

www.cse.buffalo.edu/~bina/cse321/fall2012/cyclicExecutiveOct3.pptx

www.ict.kth.se/courses/IL2212/1011/.../Clock-DrivenScheduling-1x2.pdf

Notes

Unit 10: Working of Clock-driven Scheduling

CONTENTS

Objectives

Introduction

10.1 Improving the Average Response Time of Aperiodic Jobs

10.2 Scheduling Sporadic Jobs

10.3 Summary

10.4 Keywords

10.5 Review Questions

10.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Introduce clock driven scheduling
- Describe Improving the Average Response Time of Aperiodic Jobs
- Explain Scheduling Sporadic Jobs
- Analyse maximum execution time of each sporadic job.

Introduction

In clock driven scheduling, a schedule of jobs is computed off-line. All scheduling decisions are made a priori. In clock driven scheduling, schedule is static, finite (but generally is applied cyclically). The run-time scheduler is driven by a hardware timer and simply follows this schedule. There is minimal runtime overhead. Behaviour is completely predictable and some variation can be accommodated using multiple tables which apply to different operational modes of the system. In this unit, we will discuss the working of clock driven scheduling.

10.1 Improving the Average Response Time of Aperiodic Jobs

There is no advantage to complete a job before its hard deadline. So far the jobs are scheduled in the background after all the jobs slices with hard deadlines scheduled in each frame are completed. Strategy of delaying aperiodic jobs is not a good one if hard deadlines are competing early. The sooner the aperiodic jobs complete, the more responsive the system is.

One way is to execute aperiodic jobs ahead of the periodic jobs whenever possible. This approach is called slack stealing. For this scheme to work, every periodic job slice must be scheduled in a frame that ends no later than its deadline.

Let the total amount of time allocated to all the slices scheduled in the frame k be x_k. The slack time available in the frame is equal to $f - x_k$ at the beginning of the frame. If the aperiodic job queue is nonempty at this time, the cycle executive can let aperiodic jobs execute for this amount of time without causing any job to miss its deadline. When an aperiodic job executes ahead of slices of periodic tasks, it consumes the slack in the frame. After y units of slack time are used by aperiodic jobs, the available slack is reduced to $f-x_k - y$. The cyclic executive can let aperiodic jobs

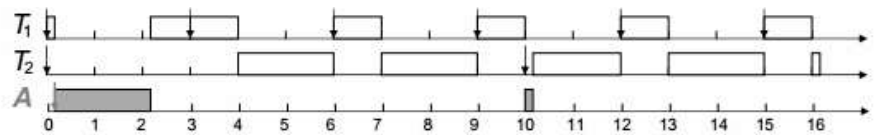
Notes

execute in frame k as long as there is slack. When the cyclic executive finds the aperiodic job queue empty, it lest the periodic task server execute the next slice in the current block. The amount of slack remains the same during this execution.

Example of slack stealing is discussed below:

- T1= (3, 1); T2= (10, 4): RM scheduled; + aperiodic job A:r = 0.1 , e = 2.1

Figure 10.1: Example of Slack Stealing



Source: http://www.dei.unipd.it/corsi/so2/12-aperiodic-sporadic-tasks-1.pdf

- The initial amount of sack in each frame can be pre-computed along with the cyclic schedule and stored in the table defining the schedule.
- It is necessary for the cyclic executive to keep track of the amount of available slack and update this amount as it consumes the slack.
- This can be done using an interval timer.
- At the beginning of each frame, the cyclic executive sets the timer to the value of the initial slack in the frame.
- The timer counts down whenever an aperiodic job executes ahead of any slice in the current block.
- When the timer expires, indicating that there is no more slack, the cyclic executive preempts the executing aperiodic job and lets the execution of the next job slice in the current block begin.
- Most operating systems do not offer interval timers of sub-millisecond granularity and accuracy.
- This scheme is practical only when the temporal parameters of periodic tasks are in orders of hundreds of milliseconds or seconds.

Self Assessment

Fill in the blanks:

1. In clock driven scheduling, a schedule of jobs is computed ……
2. All scheduling decisions are made a ……
3. In clock driven scheduling, schedule is static, finite but generally is applied ……
4. The run-time scheduler is driven by a …… timer and simply follows this schedule.
5. There is …… runtime overhead. Behaviour is completely predictable and some variation can be accommodated using multiple tables which apply to different operational modes of the system.
6. The sooner the aperiodic jobs complete, the …… responsive the system is.

Notes

10.2 Scheduling Sporadic Jobs

Like jobs in periodic tasks, sporadic jobs have hard deadlines. Their minimum release times and maximum execution times are unknown a priori. It is impossible to guarantee a priori that all sporadic jobs can complete in time. A common way to deal with this situation is to have the scheduler perform an acceptance test when each sporadic job is released. During and acceptance test, the scheduler checks whether the newly released sporadic job can be feasibly scheduled with all the jobs in the system at the time. Here by a job we mean either a periodic job, for which time has already been allocated or sporadic job, which has been scheduled but not yet completed. If according to schedule, there is a sufficient amount of time in that frame s before its deadline to complete the newly released sporadic job without causing any job in the systems to complete too late, the scheduler accepts and schedules the job. Otherwise the sporadic job is rejected and thus scheduler gives the application systems as much time as there is to take any necessary recovery action. Consider a quality control system.

A sporadic job that activates robotic arm is released when a defective part is detected. The arm, when activated, removes the part from the conveyor belt. Thus job must complete before the part moves beyond the reach of the arm. When the job cannot be completed in time, it is better for the system to have this information as soon as possible. The system can slow down the belt, stop the belt, or alert an operator to manually remove the part. If it was late, its lateness could only be detected at its deadline, if the recovery action is taken there, the defective item may already have been packed. We assume that the maximum execution time of each sporadic job becomes known upon its release. It is impossible for the scheduler to determine which sporadic jobs to admit and which to reject unless this information is available.

So the scheduler must maintain information on the maximum execution times of all types of sporadic jobs that the system may execute in response to the events it is required to handle. Suppose at the beginning of frame t, an acceptance test is done on a sporadic job S(d,e). Suppose that the deadline d of S is in frame l+1 (l ends before d and l+1 after d) and l>=t.

The job must be scheduled in the lth or earlier frames. The job can complete in time only if the current amount of slack time in frames t,t+1,….,l is equal to or greater than its execution time e. Therefore the scheduler should reject S if e > amount of slack.

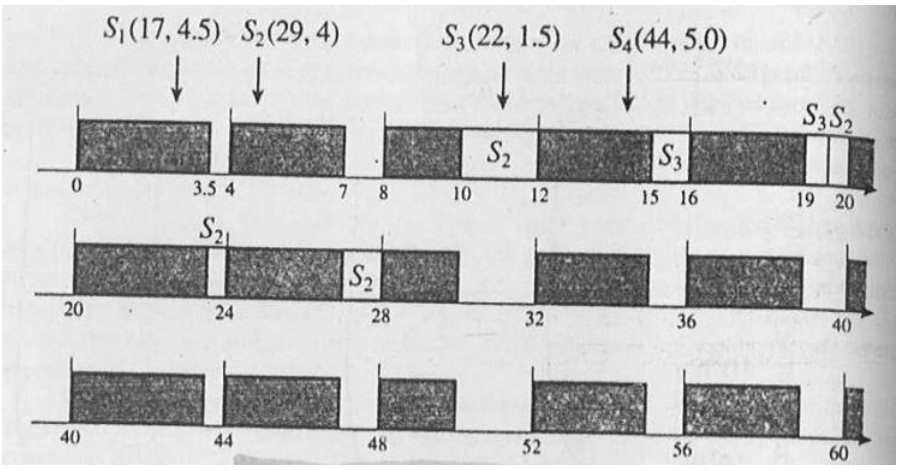
The scheduler also has to check whether accepting a new job may cause some sporadic jobs in the system to complete late.

The scheduler accepts the new job only if no sporadic jobs in system are adversely affected. As there can be more than one sporadic job, a good way to order them is on the earliest Deadline First (EDF) basis. The scheduler maintains a queue of accepted sporadic jobs in non-decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order. Whenever all slices of periodic tasks scheduled in each frame are completed, the cyclic executive lets the jobs in the sporadic job queue execute in the order they appear in the queue. The cyclic executive assumes that each newly released sporadic job is placed in a waiting queue without the intervention of the scheduler. The scheduler allows aperiodic jobs to execute only when the accepted sporadic job queue is empty. Whenever a server or job completes the cyclic executive wakes up to execute. In the figure given on p. 99, the frame size used is 4.

The shaded boxes show where periodic tasks are scheduled. Suppose at time 3, a sporadic job S1 (17,4.5) is released. Acceptance test is done at time 4, when frame 2 begins. S1 must be scheduled in frames 2, 3, 4 where total slack time is 4 thus S1 is rejected. At time 5, S2 (29,4) is released and frame 3 to 7 end before its deadline. Acceptance test is done at time 8, as the total slack time is 5.5 so S2 is accepted. At time 11, S3 (22, 1.5) is released. The scheduler finds 2 units of slack time in frames 4, 5. There is still enough slack to complete S2 event though S3 executes ahead of S2. So scheduler accepts and executes this job in frame 4. At time 14, S4 (44, 5) is released.

Notes

Figure 10.2: Example of Scheduling Sporadic Jobs



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

At time 16 acceptance test is done, the scheduler finds only 4.5 units of time available in frames before the deadline of S4, after accounting the slack time already committed to the remaining portions of S2 and S3, therefore it is rejected. When the remaining portion of S3 completes in the current frame, S2 executes until the beginning of the next frame. The last portion of S2 executes in frames 6 and 7.

Self Assessment

State whether the following statements are True or False:

7. Like jobs in periodic tasks, sporadic jobs have soft deadlines.
8. A sporadic job that activates robotic arm is released when a defective part is detected.
9. The arm, when activated, removes the part from the conveyor belt.
10. The scheduler accepts the new job only if no sporadic jobs in system are adversely affected.
11. The scheduler maintains a queue of accepted sporadic jobs in non-decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order.
12. Whenever a server or job completes the cyclic executive wakes up to execute.

Case Study

6-STAGE FPU

Clock scheduling is proposed on the premise that typical workloads exhibit a high degree of cycle-level variations in activity due to intermittent bubbles that persist in the instruction streams. In order to verify this premise, this section presents a brief case study of the characteristics of workloads found in the SPECfp benchmark suite running on a FPU like that found in IBM's POWER4.

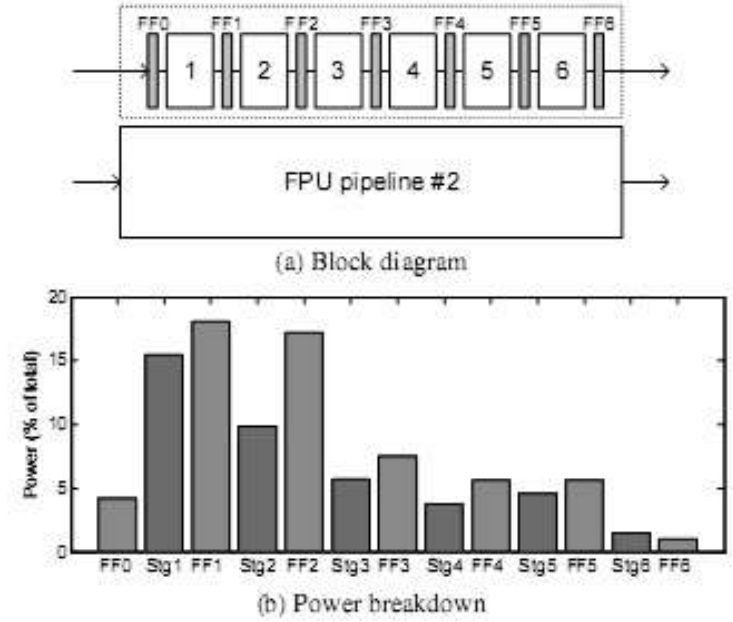
We assume a FPU that consists of two parallel pipelines with each consisting of 6 pipeline stages divided by FFs, as shown in Figure. 1(*a*). Data enters the pipelines through FF0.

Contd...

Notes

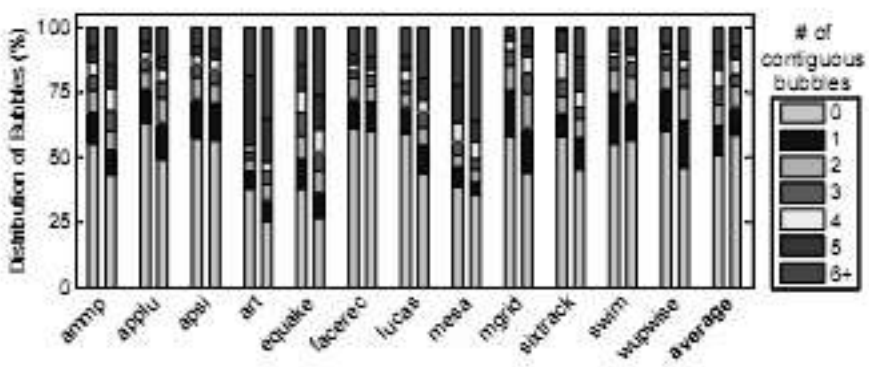
Detailed power analysis of the RTL shows that ~60% of the total power consumed in the FPU can be attributed to the clocks with logic switching factor of ~16% for random data. A fully-active power consumption breakdown of the FPU per FF and logic stage is shown in Figure. 1(*b*). The top heavy distribution of power consumption can be attributed to highly parallel structures (e.g., Wallace Tree) towards the front of the block. We will see later that such attributes are desirable for clock scheduling and glitch mitigation.

Figure 1: FPU Pipeline Characteristics Assumed in the Case Study



To augment the FPU power dissipation breakdowns, we collect architectural utilization to determine potential benefits from clock scheduling. We use the Turandot processor simulator to model a POWER4-like processor with two parallel 6-stage FPU pipelines. We simulate 100M-instruction traces of the SPECfp benchmark suite. Figure 2 presents a stacked bar graph showing the distribution of contiguous bubbles observed. The figure shows 50% of consecutive fp instructions have one or more bubbles between them. Clock scheduling can utilize these bubbles to reduce clock power.

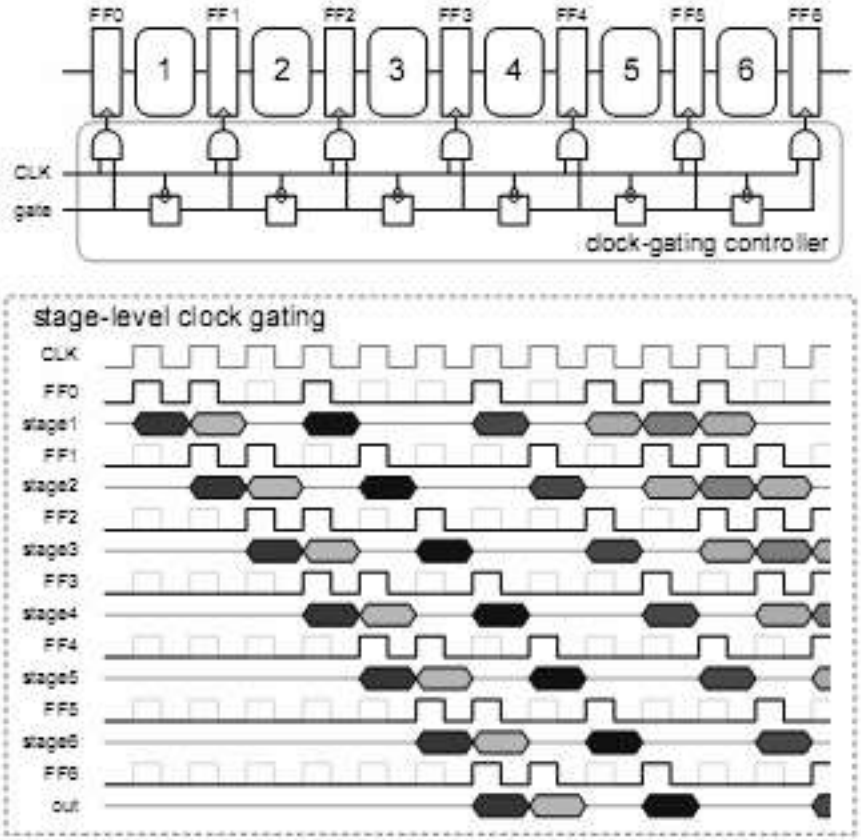
Figure 2: Distribution of Contiguous Bubbles Found in two Parallel 6-Stage FPU Pipelines Found in a POWER4-Like Processor



Contd...

Notes

Figure 3: Block and Timing Diagrams of Conventional Clock Gating



Questions:

1. How many parallel and non-parallel pipelines are used in FPU?
2. Define the term stacked bar graph.
3. What is meant by FPU power dissipation breakdowns?

Source: http://www.eecs.harvard.edu/~dbrooks/wei_islped08.pdf 10.3

10.3 Summary

- In clock driven scheduling, a schedule of jobs is computed off-line.
- All scheduling decisions are made a priori.
- In clock driven scheduling, schedule is static, finite but generally is applied cyclically.
- The run-time scheduler is driven by a hardware timer and simply follows this schedule.
- There is minimal runtime overhead. Behaviour is completely predictable and some variation can be accommodated using multiple tables which apply to different operational modes of the system.
- The sooner the aperiodic jobs complete, the more responsive the system is.
- Like jobs in periodic tasks, sporadic jobs have hard deadlines.

Notes

- A sporadic job that activates robotic arm is released when a defective part is detected.
- The arm, when activated, removes the part from the conveyor belt.
- The scheduler accepts the new job only if no sporadic jobs in system are adversely affected.
- The scheduler maintains a queue of accepted sporadic jobs in non-decreasing order of their deadlines and inserts each newly accepted sporadic job into this queue in this order.
- Whenever a server or job completes the cyclic executive wakes up to execute.

10.4 Keywords

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

Aperiodic Tasks: It is the average fraction of processor time required by all the aperiodic tasks in the system.

Cyclic EDF Algorithm: It is the only class of algorithms that perform acceptance tests at the beginnings of frames.

Deadline First Algorithm: It is universal for sets of sporadic and periodic tasks and for sets of concrete sporadic tasks.

EDF Algorithm: The EDF algorithm is a good way to schedule accepted sporadic jobs.

Sporadic Job: It is in the system is one that was accepted earlier but is not yet complete at the time.

10.5 Review Questions

1. Define the term slack stealing.
2. How does slack stealing work with aperiodic jobs?
3. What is the average response time? Give some example.
4. Differentiate between average queuing time and total average utilization.
5. What is the acceptance test and explain their significance?
6. What is cyclic EDF algorithm?

Answers: Self Assessment

1.	off-line	2.	priori
3.	Cyclically	4.	Hardware
5.	Minimal	6.	More
7.	False	8.	True
9.	True	10.	True
11.	True	12.	True

10.6 Further Readings

Notes

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990. *Principles of Concurrent and Distributed Programming*, Prentice Hall).

Online links

https://support.dce.felk.cvut.cz/psr/prednasky/liu/Chapter_5.pdf

www.ict.kth.se/courses/IL2212/1011/.../Clock-DrivenScheduling-1x2.pdf -

https://support.dce.felk.cvut.cz/psr/prednasky/liu/Chapter_5.pdf

askguru.net/d/real-world-example-of-clock-driven-scheduling

Notes

Unit 11: Application of Clock-driven Scheduling

CONTENTS

Objectives

Introduction

11.1 Practical Considerations and Generalizations

11.1.1 Frame overruns

11.1.2 Mode Changes

11.1.3 Multi-Processors

11.2 Algorithm for Constructing Static Schedules

11.3 Pros and Cons of Clock-Driven Scheduling

11.4 Summary

11.5 Keywords

11.6 Review Questions

11.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Practical Considerations and Generalizations
- Enumerate Algorithm for Constructing Static Schedules
- Analyse Pros and Cons of Clock-Driven Scheduling

Introduction

We can schedule aperiodic jobs in the background after completing the scheduling of periodic jobs in each frame. Otherwise, the average response time of aperiodic jobs can be improved. This can be done by scheduling them in the slack in every frame. Likewise, we schedule sporadic jobs during the slack times not utilized by periodic job slices in the frames. When a sporadic job is accepted, it is scheduled between all sporadic jobs in the EDF order during the time intervals not utilized by periodic jobs slices. In this unit, we will discuss some practical considerations of clock driven scheduling. Also we will discuss the pros and cons of clock driven scheduling.

11.1 Practical Considerations and Generalizations

In this section, we will discuss various practical considerations.

11.1.1 Frame Overruns

A frame overrun can occur for many reasons as follows:

- When the execution time of a job is input data dependent, it can become unexpectedly large for some rare combination of input value which is not taken into account in the precomputed schedule.

Notes

- A hardware fault may cause some job to execute longer than expected.
- A software flaw that was undetected during debugging and testing can also cause this problem.

There are many ways to handle overrun as follows:

- A way to handle overruns is to simply abort the overrun job at the beginning of the next frame and log the premature termination of the job.
  - Such a fault can then be handled by some recovery mechanism later when necessary.
  - This way seems attractive for applications where late results are not useful, like control-law computation of a robust digital controller.
  - Premature termination of overrun jobs may put the system in some inconsistent state and the actions required recovering from the state and maintain system integrity may be costly.
  - That is why in most real time systems, a job that overruns its allocated time and is found executing at the end of the frame is preempted immediately, if it is not in a critical section at the time, or as soon as it exits the critical section, if it is.
  - The unfinished portion executes as an aperiodic job during the slack time in the subsequent frames or in the background whenever there is spare time.
- Another way to handle overrun is to continue to execute the offending job.
  - The start of the next frame and the execution of jobs scheduled in the next frame are then delayed.
  - Letting a job postpone the execution and completion of jobs scheduled after it can in turn causer these jobs to be late.
  - Appropriate only if the late result is still useful and occasional late completion of periodic job is acceptable.

11.1.2 Mode Changes

- We have assumed that the number of periodic tasks in the system of their parameters remain constant as long as the system stays in the same operation mode.
- During a mode change a system is reconfigured.
- Some periodic tasks are deleted from the systems as not required in the new mode.
- The periodic tasks which execute in both modes continue to execute in timely fashion.
- We assume that the parameters of periodic tasks to be executed in the new mode are also known.
- The schedule of the tasks executed in the new mode is also precomputed.
- However the new schedule table may not be in the memory, so it must be brought into memory.
- Code of the new tasks must also be brought into memory and memory space for data accessed by them must be allocated before their execution begins.
- The mode change job is released at a mode-change request.
- The mode change job can either have soft or hard deadline.

Notes

- IN both cases we assume that periodic jobs are independent and hence can be added and deleted independently.

Aperiodic Mode Change

- A reasonable way to schedule a mode-change job that has a soft deadline is to treat it just like an ordinary aperiodic job, except that it may be given the highest priority and executed ahead of other aperiodic jobs.
- A periodic task that will not execute in the new mode can be deleted and its memory space and processor time freed.
- During mode change, the scheduler continues to use the old schedule table.
- Before the periodic tasks server begins to execute a periodic job, however, it checks whether the corresponding task is marked and return immediately if the task is marked.
- Thus the schedule of the periodic tasks that execute in both modes remains unchanged.
- The time allocated to the deleted tasks can be used to execute the mod-change job.
- Once the new schedule table and code of the new tasks are in memory, the scheduler can switch to use the new table.
- How aperiodic and sporadic jobs in the system should be dealt with during mode change? (already running)
- Since the deadlines of the remaining periodic jobs are soft, their execution can be delayed until after the mode change.
- The sporadic jobs should not be affected by the mode change.
- One way to ensure their on-time completion is to defer the switchover from the old schedule table to the new schedule table until all the sporadic jobs in the system complete.
- Clearly this option can lengthen the response time of the mode change.
- Another option is to have the mode-change job check whether the sporadic jobs in the system can complete in time according to the new schedule.
- The schedule switchover is deferred only when some sporadic job cannot complete in time according to the new schedule.
- When the delay thus introduces is unacceptably long, the scheduler may be forced to switchover to the new schedule, let some sporadic jobs complete late and leaved the affected applications to handle the timing faults.
- One way to ensure their on-time completion is to defer the switchover from the old schedule table to the new schedule table until all the sporadic jobs in the system complete.
- Clearly this option can lengthen the response time of the mode change.
- Another option is to have the mode-change job check whether the sporadic jobs in the system can complete in time according to the new schedule.
- The schedule switchover is deferred only when some sporadic job cannot complete in time according to the new schedule.
- When the delay thus introduces is unacceptably long, the scheduler may be forced to switchover to the new schedule, let some sporadic jobs complete late and leaved the affected applications to handle the timing faults.
- Pseudo code on next pate describes a mode changer.

Notes

Caution Intuitively, one should give hard real-time jobs higher priority than aperiodic jobs. However, this may lengthen the response time of an aperiodic job.

11.1.3 Multi-Processors

- Searching for a feasible multiprocessor schedule is more complex than searching for a uniprocessor schedule.
- We can construct a global clock schedule which specifies on what processor each job executes and when the job executes.
- As long as the clock drifts on the processors are sufficiently small, we can use the uniprocessor scheduler on each processor to enforce the execution of the jobs according to the global schedule.
- Since the feasible multiprocessor schedule search is done offline, we can use exhaustive and complex heuristic algorithms for this purpose.

Self Assessment

Fill in the blanks:

1. A …… that was undetected during debugging and testing can also cause this problem.
2. Another way to handle …… is to continue to execute the offending job.
3. The schedule switchover is deferred only when some …… cannot complete in time according to the new schedule.
4. A …… that will not execute in the new mode can be deleted and its memory space and processor time freed.

11.2 Algorithm for Constructing Static Schedules

DSP algorithms are usually required to operate under real-time constraints and with limited resources. We are concerned with constructing efficient static (compile-time) schedules for multirate DSP algorithms.

Dataflow models are widely used to represent DSP applications. The one often used for multirate DSP algorithms are synchronous dataflow graphs (SDFGs). Each node (also called actor) in an SDFG represents a computation or function and each edge models a FIFO channel; the sample rates of actors may differ. Practical MultiMate DSP applications modelled with SDFGs include a spectrum analyser, a satellite receiver, etc.

DSP algorithms are non-terminating and repetitive. Static schedules are usually used for them. A static schedule arranges computations of an algorithm to be executed repeatedly. Execution of all the computations for the required number of times is referred to as iteration. An iteration of an SDFG may include more than one execution, often called firing in dataflow, of an actor, and a different number of firings for different actors. The average computation time per iteration is called the iteration period of a schedule. DSP algorithms with recursions (or feedbacks) have an inherent lower bound on the iteration period, referred to as the iteration bound. It is impossible to achieve an iteration period lower than the iteration bound, even when unlimited processors are available. A schedule whose iteration period equals the iteration bound is called a rate-optimal schedule.

Notes

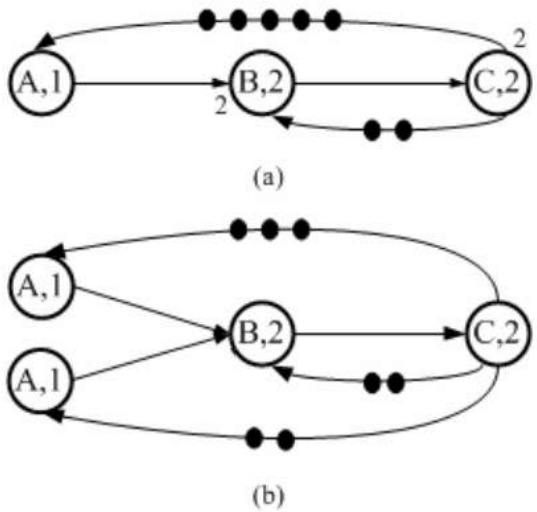
The construction of rate-optimal schedules involves explicit or implicit retiming and unfolding. Unfolding turns f consecutive iterations into a cycle in the unfolded graph. f is called the unfolding factor. Unfolding can lead to a rate optimal schedule, at the cost of multiplying the problem space by the unfolding factor. An unfolding factor is called an optimal unfolding factor if a rate-optimal schedule exists with the factor. Combined with retiming, a smaller optimal unfolding factor may be obtained.

Much work has been done on static rate-optimal scheduling of homogenous synchronous dataflow graphs (HSDFGs), which is a special type of SDFGs. All sample rates of actors of an HSDFG are one. In an iteration of an HSDFG, each actor acts once.

Little work, however, is concerned with general SDFGs. Theoretically, it is always possible to convert an SDFG to its equivalent HSDFG and then use the available methods for HSDFGs. However, converting an SDFG to an HSDFG is very time-consuming when SDFGs scale up. The size of the HSDFG can be exponentially larger than the original SDFG in extreme cases. To the best of our knowledge, only discusses the rate-optimal scheduling of SDFGs. It converts an SDFG to a precedence graph, which is a reduced form of its equivalent HSDFG, then computes the unfolding factor and schedules the precedence graph with the same method as. A precedence graph of an SDFG has less edges than and the same number of actors as its equivalent HSDFG. The conversion procedure is still time and space-consuming.

Did u know? Many digital signal processing algorithms, like the digital IIR-filter are naturally described by data flow graphs.

Figure 11.1: Figure (a) The SDFG G1, where the sample rates are omitted when they are 1 and the computation time of each actor is attached inside the node; (b) the equivalent HSDFG of G1.



Source: http://www.es.ele.tue.nl/~mgeilen/publications/rtas2012final.pdf

A synchronous dataflow graph (SDFG) is a finite directed graph G = ,V,E,t,d, prd, cns>, in which V is the set of actors, modelling the functional elements of the system; E is the set of directed edges, modelling interconnections between functional elements. Each actor v is weighted with its computation time t(v), a nonnegative integer. Each edge e is weighted with three properties: d(e), a nonnegative integer that represents the number of initial tokens associated with e, prd(e), a positive integer that represents the number of tokens produced onto e by each firing of the

Notes

source actor of e, cns(e), a positive integer that represents the number of tokens consumed from e by each firing of the sink actor of e. These numbers are also called the delay, production rate and consumption rate, respectively. The source actor and sink actor of e are denoted as src(e) and snk(e), respectively. If prd(e) = cns(e) = 1 for each e 2 E, then we say that G is a homogeneous SDFG (HSDFG).

Notes A special class of algorithms can be modelled by a synchronous data flow (SDF) graph for which efficient implementation methods exist.

The edge e with source actor u and sink actor v is denoted by e = hu; vi. The set of incoming edges to actor v is denoted by InE (v), and the set of outgoing edges from v by OutE(v). An initial delay distribution of the SDFG G is a vector containing delays on all edges of G, denoted as d(G). An SDFG G is sample rate consistent if and only if there exists a positive integer vector q(V) such that for each edge e in G,

$$q(src(e)) \times prd(e) = q(snk(e)) \times cns(e) .$$

where (1) is called a balance equation. The smallest q is called the repetition vector. We use q to represent the repetition vector directly.

Example: A balance equation can be constructed for each edge of G1 in Fig. 11.1 (a). By solving these balance equations, we have G1's repetition vector q = [2, 1, 1].

One iteration of an SDFG G is a firing sequence in which each actor v occurs exactly q(v) times.

Example: An iteration of G1 in Fig. 11.1 (a), for example, includes two firings of actor A and one firing of B and C, respectively.

Self Assessment

Fill in the blanks:

5. …… are usually required to operate under real-time constraints and with limited resources.
6. The construction of rate-optimal schedules involves …… retiming and unfolding.
7. It is always possible to convert an SDFG to its equivalent …… and then use the available methods for HSDFGs.
8. The average computation time per iteration is called the …… of a schedule.
9. A …… arranges computations of an algorithm to be executed repeatedly.

11.3 Pros and Cons of Clock-Driven Scheduling

The advantages of clock-driven scheduling are discussed below:

- Conceptual simplicity
  - complex dependencies, communication delays, resource contentions can be considered when developing the schedule

Notes

- ❖ no need for any synchronization mechanisms
- ❖ schedule can be represented as tables that is used by the scheduler at run time

- Relatively easy to validate
  - ❖ Because times when jobs execute are deterministic, the system will not exhibit anomalies
  - ❖ Can prove scheduling feasibility by exhaustive simulation and testing

Disadvantages of clock-driven scheduling are discussed below:

- Inflexible, since schedule is computed off-line, small changes mean that new tables have to be generated.
- Release times of jobs must be fixed
  - ❖ Priority-driven does not require this
- A lot information about jobs has to be known beforehand, so that the schedule can be precomputed.
- Difficult to get acceptable response times for aperiodic (soft real-time) jobs.

Task Prepare a presentation on Algorithm for Constructing Static Schedules.

Self Assessment

State whether the following statements are True or False:

10. Synchronization mechanism is very important for Clock –Driven Scheduling.
11. The sporadic jobs should be affected by the mode change.
12. DSP algorithms are non-terminating and repetitive.

Case Study

Innovative Concepts and LynxOX Team Up to Support U.S. Army Communications

CI's mobile, network, and communications systems are developed for global enterprises seeking sophisticated solutions for complex voice, imagery, video, and data. The company's newly launched product line focuses on tactical and wireless embedded communications systems, reinforcing its dynamic vision to 'engineer tomorrow's communications.

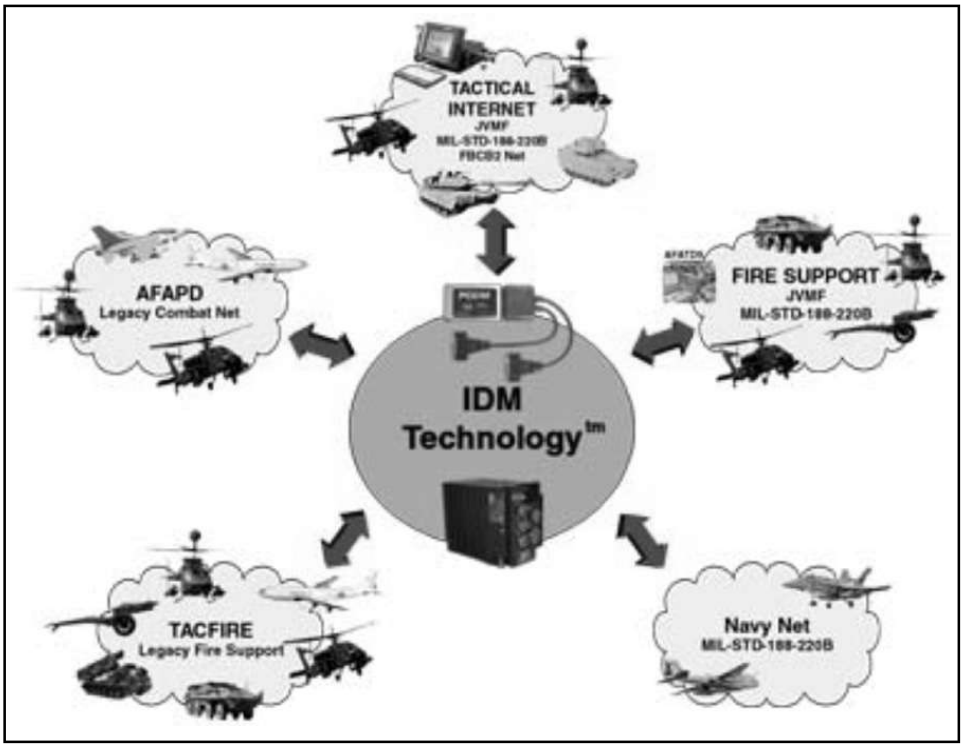
Making Tactical Connections

ICI's military communications equipment is designed for the United States Army and is used primarily by Army aviation helicopters, such as Apache/Longbow, Kiowa Warrior, cargo helicopters, and others. While the Army and Air Force employ many different protocols to communicate, ICI's IDM (improved data modem), a communications and

Contd...

Notes

targeting system, is particularly unique because it can interface between different communications formats.



"The IDM is a complete communications unit and a critical piece of equipment," said Bob Woodward, director, tactical communication systems at ICI. "The IDM essentially provides full networking capability to tactical users and RF-based data communication connectivity to the commercial mobile communication market." It interconnects the U.S. military's major networks for both manoeuvre and fire support, and also provides critical linkage to the military's legacy systems.

Plans are underway for ICI's IDM to incorporate an embedded subset of the Army's innovative Force XXI Battle Command, Brigade and Below (FBCB2) System software called EBC (embedded battlefield command), which satisfies the Army's growing trend toward smaller, handheld computer devices. FBCB2 allows different vehicles on a battlefield to share a joint situational awareness picture in real time. Hypothetically, a Crusader mobile artillery piece could 'see' on its computer battlefield map the same enemy that a Comanche helicopter flying miles away from it has just spotted. The Crusader could then target its weapons using radar data provided by the Comanche helicopter.

While FBCB2 software will ultimately run in every ground vehicle for the Army, EBC will be integrated into virtually all Army aviation equipment, thereby enabling Army personnel to communicate with one another over the Tactical Internet (which uses the same protocols as the Internet but whose IP addresses are tied to Army vehicles constantly on the go).

A Smarter Way to Communicate

Historically, FBCB2 software ran on the Solaris® operating system, which is a large, process-oriented piece of software. However, the addition of EBC and the requirement for compact computer devices prompted the need for a better, more efficient solution. Redesigning the IDM to accommodate a Solaris card, a disk, and another display for the

Contd...

Notes

helicopters proved costly and inefficient. Because VxWorks® was already being used in the IDM, porting FBCB2 to VxWorks seemed a logical approach. However, after 3 years attempting without success to port to VxWorks, the original project team cut its losses and turned to LynuxWorks' LynxOS® real-time operating system (RTOS).

LynuxWorks' LynxOS blends deterministic performance, reliability, openness, and scalability with patented technology for real-time event handling, and also provides complete UNIX® compatibility. The complete RTOS package proved particularly attractive to ICI. "LynxOS® provided us the full UNIX capability we needed in order to meet the Army's requirements," explained Woodward. "VxWorks does not feature all of the UNIX application programming interfaces; it's UNIX-like, but does not provide the true UNIX qualifications necessary to accommodate Army-standard applications. We wanted something that would fit with what we already have yet allowed us to proceed to integrate our new technology."

LynxOS meets strict POSIX® conformance tests, plus the UNIX compatibility making its application interfaces compatible with Linux. To be conformant to the POSIX standard, a hardware platform and operating system must be certified as such. Many operating systems like VxWorks, however, only implement portions of POSIX while still claiming POSIX compliance. This represented a significant reason why efforts to port to VxWorks ultimately failed—it is not a true POSIX-conformant operating system.

Innovative Concepts and the Army became convinced that moving forward with LynxOS signified the best solution. "We knew it would not be difficult to port the FBCB2 code to LynxOS, and it was not. In fact, in just six short months, the port succeeded. We received our first version of EBC and the initial testing was favourable. LynuxWorks went beyond the call of duty to ensure that we achieved our goals, and provided professional, top-level expertise all along the way," said Woodward.

A new standard

As a result of its collaboration with LynuxWorks and the overwhelming success of LynxOS, ICI has shifted its software baseline from VxWorks to LynxOS to support the development and deployment of the Army's standard EBC software for its aviation unit. Moreover, LynxOS has become the de facto standard for virtually all airborne Army communications equipment interfacing with the Tactical Internet.

Questions:

1. What is the smarter and effective way to communicate? Explain on the basis of given case study.
2. Describe the new standard that is used in the communications.

Source: http://www.lynuxworks.com/corporate/news/success/ici.php3

11.4 Summary

- There are many ways to handle overrun.
- A way to handle overruns is to simply abort the overrun job at the beginning of the next frame and log the premature termination of the job.
- Another way to handle overrun is to continue to execute the offending job.
- Periodic tasks that execute in the new mode but not in the old mode are created and added to the system.

Notes

- A reasonable way to schedule a mode-change job that has a soft deadline is to treat it just like an ordinary aperiodic job, except that it may be given the highest priority and executed ahead of other aperiodic jobs.
- Searching for a feasible multiprocessor schedule is more complex than searching for a uniprocessor schedule.
- Dataflow models are widely used to represent DSP applications.
- A synchronous dataflow graph (SDFG) is a finite directed graph G = ,V,E,t,d, prd, cns>, in which V is the set of actors, modelling the functional elements of the system; E is the set of directed edges, modelling interconnections between functional elements.

11.5 Keywords

DSP Algorithms: Digital signal processing (DSP) is the mathematical manipulation of an information signal to modify or improve it in some way.

Iteration Period: The average computation time per iteration is called the iteration period of a schedule.

Iteration: Execution of all the computations for the required number of times is referred to as iteration.

Rate-optimal Schedule: A schedule whose iteration period equals the iteration bound is called a rate-optimal schedule.

Synchronous Dataflow Graph: Data flow programs for signal processing are directed graphs where each node represents a function and each arc represents a signal path.

11.6 Review Questions

1. Discuss the ways or methods to handle overrun.
2. What do you mean by frame overrun?
3. Explain the concept of Aperiodic Mode Change.
4. Elucidate the importance of mode changes in frame overruns.
5. Give an overview on Pros and Cons of Clock-Driven Scheduling.
6. Describe the concept of Algorithm for Constructing Static Schedules.
7. Throw light on synchronous dataflow graph.

Answers: Self Assessment

1.	Software flaw	2.	Overrun
3.	Sporadic job	4.	Periodic task
5.	DSP algorithms	6.	Explicit or implicit
7.	HSDFG	8.	Iteration period
9.	Static schedule	10.	False
11.	False	12.	True

Notes

11.7 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990. *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

www.nptel.iitm.ac.in/courses/Webcourse-contents/.../Pdf/Lesson-29.pdf

https://support.dce.felk.cvut.cz/psr/prednasky/liu/Chapter_5.pdf

www.ict.kth.se/courses/IL2212/1011/.../Clock-DrivenScheduling-1x2.pdf

www.dei.unipd.it/corsi/so2/07-Clock-driven-scheduling.pdf

Unit 12: Priority-driven Scheduling of Periodic Tasks

Notes

CONTENTS

Objectives

Introduction

12.1 Static Assumptions

12.2 Fixed-Priority versus Dynamic-Priority Algorithms

12.2.1 Rate Monotonic and Deadline Monotonic Algorithms

12.2.2 Some Well Known Dynamic Algorithms

12.3 Maximum Schedulable Utilization

12.4 Summary

12.5 Keywords

12.6 Review Questions

12.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Static Assumptions
- Enumerate fixed-Priority Versus Dynamic-Priority Algorithms
- Explain Maximum Schedulable Utilization

Introduction

Priority-driven scheduling is easy to implement. It does not require the information on the release times and execution times of the jobs a priori. The run-time overhead due to maintaining a priority queue of ready jobs can be made small. Priority-driven algorithms differ from each other in how priorities are assigned to jobs. We classify algorithms for scheduling periodic tasks into two types, that is, Fixed-priority and Dynamic-priority. A *fixed-priority* algorithm assigns the same priority to all the jobs in each task. A *dynamic-priority* algorithm assigns different priorities to the individual jobs in each task. Most real-time scheduling algorithms of practical interest assign fixed priority to individual jobs.

12.1 Static Assumptions

Consider the assumptions given below:

- The tasks are independent
- There are no aperiodic and sporadic tasks
- Job is ready for execution as soon as it is released
- Jobs are preemptible
- Scheduling decisions are made immediately upon job release and completion

Notes

- Context switch overhead is negligibly small compared with the execution times of the tasks
- Unlimited priority levels.
- Fixed number of periodic tasks

On each new task creation, application provides its period, execution time, relative deadline and scheduler does an acceptance test. When tasks are independent, the scheduler can delete any tasks and add an acceptable task at any time without causing any missed deadline. A multiprocessor priority-driven system is either dynamic or static. In a static system, all the tasks are partitioned into subsystems. Each subsystem is assigned to a processor, and tasks on each processor are scheduled by themselves.

In a dynamic system, job ready for execution are placed in one common priority queue and dispatched to processors for execution as the processors become available.

The dynamic approach should allow the processors to be more fully utilized on average as the workload fluctuates. The troublesome problem with dynamic systems is the fact that we often do not know how to determine their worst-case and best-case performance. That is why most hard real-time systems built and in use are static when tasks in a static system are independent, we can consider the tasks on each processor independently of the tasks on the other processors. The problem of scheduling in multiprocessor and distributed systems is reduced to that of uniprocessor scheduling.

Self Assessment

Fill in the blanks:

1. When tasks are independent, the …… can delete any tasks and add an acceptable task at any time without causing any missed deadline.
2. When tasks in a …… are independent, we can consider the tasks on each processor independently of the tasks on the other processors.
3. In a …… system, job ready for execution are placed in one common priority queue and dispatched to processors for execution as the processors become available.
4. A …… priority-driven system is either dynamic or static.
5. The problem of scheduling in multiprocessor and …… is reduced to that of uniprocessor scheduling.

12.2 Fixed-Priority versus Dynamic-Priority Algorithms

A priority driven scheduler is an on-line scheduler. There is no pre-computation of schedules of tasks. It assigns priorities to jobs after they are released and places the jobs in a ready job queue in priority order. Preemption is allowed at any time, as scheduling decision is made whenever a job is released or completed. At each scheduling decision time, the scheduler updates the ready job queue and then schedules and executes the job at the head of the queue. Algorithms for scheduling periodic tasks are classified into two types:

- Fixed Priority
- Dynamic Priority

Fixed Priority algorithm assigns the same priority to all the jobs in each task. Dynamic Priority algorithm assigns different priorities to the individual jobs in each task.

Notes

12.2.1 Rate Monotonic and Deadline Monotonic Algorithms

The rate monotonic algorithm assigns priorities to tasks based on their periods. The shorter the period the higher the priority is. The rate of (job releases) of a task is the inverse of its period so the higher the rate, the higher its priority. Each job is placed at the head of the priority queue and is executed as soon as the job is released. When the relative deadline of every task is proportional to its period, the RM and DM algorithms are identical. When the relative deadlines are arbitrary, the DM algorithm performs better in the sense that it can sometimes produce a feasible schedule when the RM fails. The RM algorithm always fails when the DM algorithm fails. As the priorities of the tasks with short relative deadlines are too low, these tasks cannot meet their deadlines.

Did u know? RMA has been proved to be the optimal static priority real-time task scheduling algorithms.

12.2.2 Some Well Known Dynamic Algorithms

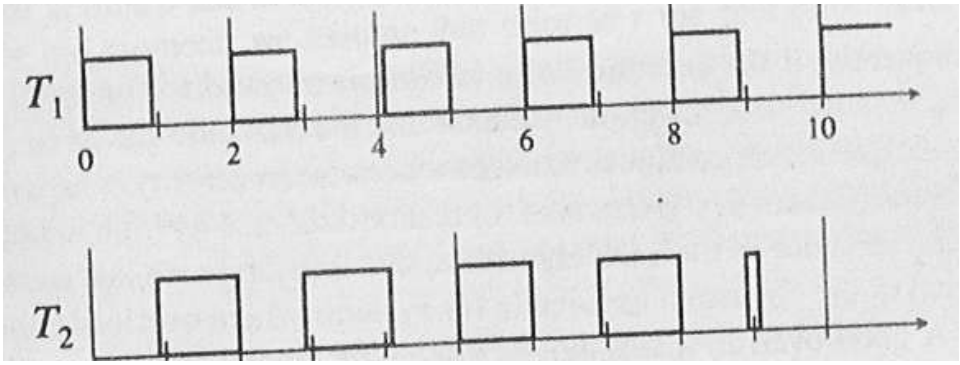
The EDF assigns priorities to individual jobs in the tasks according to their absolute deadlines; it is a dynamic-priority algorithm. Figure given below shows the EDF schedule of the two tasks T_1 and T_2.

Notes The basic principle of EDF is algorithm is very intuitive and easy to understand. The scheduling test for EDF is also simple.

Here,

J_1 is taken as First Job and J_2 is taken as Second Job.

Figure 12.1: EDF Schedule of the Two Tasks T_1 and T_2



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

- At time 0, the first jobs J_1, 1 and J2, 1 of both tasks are ready.
- The absolute deadline of J1, 1 is 2 and that of J2,1 is 5 so J1, 1 is higher priority.
- At time 2, J1, 2 is released and its deadline is 4, earlier than J2, 1.
- Hence J2, 1 is placed ahead of J2, 1 it preempts J2,1 and executes.

Notes

- At time 2.9, J1, 2 completes.
- Processor executes J2, 1.
- At time 4, J1,3 is released with deadline 6, later than the deadline of J2,1, hence processor keeps on executing J2,1
- At time 4.1, J2,1 completes, the processor starts to execute J1,3 and so on.
- The priority of T1 is higher than priority of T2 from time 0 until 4.
- T2 starts to have a higher priority at time 4.
- When the job J2, 2 is released, T2 again has a lower priority.
- Hence the EDF algorithm is a task-level dynamic priority algorithm.
- Once a job is placed in the ready job queue according to the priority assigned to it, its order with respect to other jobs in the queue remains fixed.
- At time 0, the first jobs J1, 1 and J2, 1 of both tasks are ready. The absolute deadline of J1,1 is 2 and that of J2,1 is 5 so J1,1 is higher priority.
- At time 2, J1, 2 is released and its deadline is 4, earlier than J2, 1. Hence J2, 1 is placed ahead of J2, 1 it preempts J2, 1 and executes.
- At time 2.9, J1, 2 completes. Processor executes J2, 1.
- At time 4, J1,3 is released with deadline 6, later than the deadline of J2,1, hence processor keeps on executing J2,1.
- At time 4.1, J2,1 completes, the processor starts to execute J1,3 and so on.
- The priority of T1 is higher than priority of T2 from time 0 until 4.
- T2 starts to have a higher priority at time 4.
- When the jobJ2, 2 is released, T2 again has a lower priority.
- Hence the EDF algorithm is a task-level dynamic priority algorithm.
- Once a job is placed in the ready job queue according to the priority assigned to it, its order with respect to other jobs in the queue remains fixed.
- Thus the EDF algorithm is a job-level fixed-priority algorithm.
- Another dynamic priory algorithm is Least-Slack-Time-First (LST).
- At time t, the slack of a job whose remaining execution time is x and whose deadline is d is equal to d-t-x.
- The scheduler checks the slacks of all the ready jobs each time a new job is released and orders the new job and the existing jobs on the basis of their slack.
- The smaller the slack, the higher the priority.
- Co-incidentally, the schedule of T1 and T2 in the discussed example happens to be identical.
- However the LST schedule of a system may differ in a fundamental way from the EDF schedule.
- Consider and more complicated system that consists of three tasks; T1 = (2, 0.8) T2 = (5, 1.5) and T3 = (5.1, 1.5).

Notes

- When the first jobs J1, 1, J2, 1 and J3, 1 are rele3ased at time 0, their slacks are 1.2, 3.5 and 3.6 respectively.
- J1, 1 has the highest priority, and J3, 1 has the lowest.
- At time 0.8, J1, 1 completes and J2, 1 executes.
- When J1, 2 is released at time 2, its slack is 1.2, while the slacks of J2, 1 and J3, 1 become 2.7 and 1.6.
- Hence J1, 2 has the highest priority, but now J3, 1 has a higher priority than J2, 1.
- We can see that the EDF algorithm is a job-level fixed priority algorithms and LST is a job-level-dynamic priority algorithms.
- In this version of LST, the scheduling decisions are made only at the times when jobs are released or completed, thus we can call it non-strict LST.
- If the scheduler were to follow the LST rule strictly, it would have to monitor the slacks of all ready jobs and compare them with the slack of the execution job.
- It would re-assign the priorities to jobs whenever their slacks change relative to each other.
- J1, 1 has the highest priority, and J3, 1 has the lowest.
- At time 0.8, J1, 1 completes and J2, 1 executes.
- When J1, 2 is released at time 2, its slack is 1.2, while the slacks of J2, 1 and J3, 1 become 2.7 and 1.6.
- Hence J1, 2 has the highest priority, but now J3, 1 has a higher priority than J2, 1.
- We can see that the EDF algorithm is a job-level fixed priority algorithms and LST is a job-level-dynamic priority algorithms.
- In this version of LST, the scheduling decisions are made only at the times when jobs are released or completed, thus we can call it non-strict LST.
- If the scheduler were to follow the LST rule strictly, it would have to monitor the slacks of all ready jobs and compare them with the slack of the execution job.
- It would re-assign the priorities to jobs whenever their slacks change relative to each other.
- Consider the diagram below, the scheduler would find that at time 2.7, the slack of J2, 1 becomes (5-2.7-1.2) = 1.1, the same as that of J1, 2.
- It would schedule the two ready jobs in a round-robin manner until J1, 2 completes.
- The run-time overhead of the strict LST algorithms includes the time required to monitor and compare the slacks of all ready jobs as time progresses.
- According to our classification, FIFO and LIFO algorithms are also dynamic priority algorithms.
- Suppose we have three tasks T1 = (0, 3, 1, 3), T2 = (0.5, 4, 1, 1), T3 = (0.75, 7.5, 2, 7.5).
- If scheduled on FIFO basis J1, 1 has higher priority than J2, 1 which has higher priority than J3, 1.
- Later J1, 4, J2, 3 and J3, 2 are released at times 9, 8.5 and 8.25, and T3 has the highest priority and T1 lowest.

Notes

Example: Consider 3 periodic processes scheduled using **EDF**, the following acceptance test shows that all deadlines will be met.

Process	Execution Time = C	Period = T
P1	1	8
P2	2	5
P3	4	10

The utilization will be:

$$\frac{1}{8}+\frac{2}{5}+\frac{4}{10}=0.925=92.5\%$$

The theoretical limit for any number of processes is 100% and so the system is schedulable.

Self Assessment

State whether the following statements are True or False:

6. Fixed Priority algorithm assigns the different priority to all the jobs in each task.
7. A priority driven scheduler is an off-line scheduler.
8. FIFO and LIFO algorithms are also dynamic priority algorithms.
9. The rate monotonic algorithm assigns priorities to tasks based on their periods.
10. The rate of (job releases) of a task is the inverse of its period so the higher the rate, the higher its priority.

12.3 Maximum Schedulable Utilization

The schedulable utilization UALG of a scheduling algorithm is defined as follows:

A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if the total utilization of the tasks is equal to or less than the schedulable utilization of that algorithm.

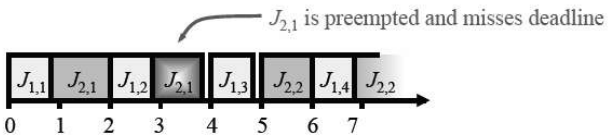
A periodic task is defined as a tuple (ϕ_i, p_i, e_i, D_i) where $u_i = e_i / p_i$

- Total utilization of the system $U = \Sigma_{i=1}^{n} u_i$ where $0 \le U \le 1$
- The ratio of the execution time e_k of a task T_k to the minimum of its relative deadline D_k and period p_k is called the density of the task
- Density of $T_k = e_k/\min(D_i, p_i)$
- The sum of the densities of all tasks in a system is the density of the system and is denoted by Δ
- If the density of a system is larger than 1, the system may not be feasible

Notes

Example: (2, 0.9) and (5, 2.3, 3)

Figure 12.2: Example of Maximum Schedulable Utilization



Source: ansari.szabist-isb.edu.pk/RTS/RTS0508.ppt

Theorem: A system T of independent, preemptible tasks can be feasibly scheduled on one processor if its density is equal to or less than 1. The condition in this theorem is not necessary for a system to be feasible, a system can be feasible when its density is greater than 1, e.g. (2, 0.6, 1) and (5, 2.3).

Self Assessment

Fill in the blanks:

11. A scheduling algorithm can feasibly schedule any set of …… on a processor if the total utilization of the tasks is equal to or less than the schedulable utilization of that algorithm.
12. If the scheduler were to follow the …… strictly, it would have to monitor the slacks of all ready jobs and compare them with the slack of the execution job.

Case Study

Rugby Union Scheduling

In this study we examine the problem of producing sports leagues that are round-robin in structure. Such structures occur when we have n teams (or competitors) in a league, and each team is required to play all other teams exactly m times within a fixed number of rounds. Examples include the Six Nations Rugby Championship (where n = 6 and m = 1), the England and Wales County Cricket Championship (n = 9, m = 2), and most European and South American national soccer leagues. In practice the most common league structures are single and *double* round-robins, where m = 1 and 2 respectively.

It has been known for many years that valid round-robin schedules using the minimal m (n¡1) rounds can be constructed for any positive integer n. It is also known that the number of *distinct* round-robin schedules grows exponentially with n, since this gore is monotonically related to the number of non-isomorphic one-factorizations of a complete graph with n vertices (which is also known to increase exponentially with n). Such a growth rate implies that, for non-trivial problems, examining all possible schedules in order to identify the one that best is the needs of the user will not always be possible in reasonable time (in practice, non-trivial values for n would appear to be anything above 10 when m = 1.) Recent work on round-robin scheduling problems has tended to focus on the use of metaheuristic-based approaches, which have shown to be effective for exploring the large search spaces associated with such problems [1, 3, 10, and 12]. The strategy used in such methods is to apply a two-stage scheme, whereby a valid round-robin schedule is

Contd...

Notes

first constructed, with changes then being made to the schedule via the application of neighbourhood operators that preserve this validity. Until now, such approaches have tended to focus on a specific formulation of round-robin scheduling problem, known as the Travelling Tournament Problem, originally proposed by Easton.

In this work we have chosen to look at the problem of round-robin scheduling in more general terms. Specifically, we choose to view it as a type of graph colouring problem. Graph colouring is a classical combinatorial optimisation problem for which much research has been conducted, both from the perspective of *solving* graph colouring problems [2, 6, 7, 9] and in *exploring* the spaces of feasible and/or optimally coloured graphs [9,11,13]. By using such principles, in this work we analyse the ability of a number of well-known graph colouring algorithms for solving round-robin scheduling problems of different sizes. We also show how the graph colouring model can be extended in order to cope with additional hard (i.e. mandatory) constraints that can often be imposed in practical situations. Finally, we also examine a number of neighbourhood operators, originating from the graph colouring literature that can then be utilised in order to explore the space of feasible round-robin schedules.

In the final part of this work we consider a complex real world scheduling problem that was given to us by the Welsh Rugby Union, based at the Millennium Stadium in Cardiff, Wales. This problem involves producing a double round-robin tournament using $2(n - 1)$ rounds and features a number of additional hard constraints to do with stadium sharing, stadium availability, and pre-assignment issues. In addition, soft constraints involving the spread of related matches throughout the tournament are also imposed. We propose two separate algorithms for tackling this problem, both that make use of our proposed algorithmic operators, and we demonstrate their ability to produce solutions that are superior to those currently used by the league, in short periods of time.

Questions:

1. Describe the main problem of rugby scheduling.
2. How does the Welsh Rugby Union resolve the existing problem?

Source: http://www.mistaconference.org/2009/abstracts/718-719-131-A.pdf

12.4 Summary

- Most real-time scheduling algorithms of practical interest assign fixed priority to individual jobs.
- In a dynamic system, job ready for execution are placed in one common priority queue and dispatched to processors for execution as the processors become available.
- A priority driven scheduler is an on-line scheduler.
- There is no pre-computation of schedules of tasks.
- The rate monotonic algorithm assigns priorities to tasks based on their periods.
- A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if the total utilization of the tasks is equal to or less than the schedulable utilization of that algorithm.

Notes

12.5 Keywords

Dynamic-priority Algorithm: A *dynamic-priority* algorithm assigns different priorities to the individual jobs in each task.

EDF: Earliest-deadline-first scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling.

Fixed-priority Algorithm: A *fixed-priority* algorithm assigns the same priority to all the jobs in each task.

Rate Monotonic Algorithm: It is a static priority algorithm and is extensively used in practical applications.

Scheduling Points: These are the points on time line at which the scheduler makes decisions regarding which task is to be run next.

12.6 Review Questions

1. What is Priority-driven scheduling?
2. Explain the static assumptions of Priority-driven scheduling.
3. Define and Differentiate between Fixed-priority and Dynamic-priority.
4. Discuss the concept of schedulable utilization UALG of a scheduling algorithm with example.
5. What do you mean by Rate Monotonic and Deadline Monotonic Algorithms?
6. Discuss types of EDF assigned tasks with the help of figures and illustration.

Answers: Self Assessment

1.	Scheduler	2.	static system
3.	dynamic	4.	multiprocessor
5.	distributed systems	6.	False
7.	False	8.	True
9.	True	10.	True
11.	periodic tasks	12.	LST rule

12.7 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). "*Principles of Concurrent and Distributed Programming*", Prentice Hall).

Notes

Online links

www.cs.uga.edu/~shelby/pubs/goossensFB2001-12.pdf

csperkins.org/teaching/rtes/lecture05.pdf

www.cse.nd.edu/~cpoellab/teaching/cse40463/slides5.pdf

www.iust.edu.sy/courses/4.PriorityDrvnSchdl.pdf

Notes

Unit 13: Working of Priority-driven Scheduling of Periodic Tasks

CONTENTS

Objectives

Introduction

13.1 Optimality of RM and DM Algorithm

13.2 Schedulability Test

13.2 .1 A Schedulability Test for Fixed-Priority Tasks with Short Response Times

13.2 .2 A Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times

13.3 Summary

13.4 Keywords

13.5 Review Questions

13.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe Optimality of RM and DM Algorithm
- Enumerate A Schedulability Test for Fixed-Priority Tasks with Short Response Times
- Explain A Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times

Introduction

Utilization-based conditions are deduced for finding out if a periodic task system fulfils each deadline when scheduled by means of the EDF scheduling algorithm upon a given multiprocessor platform. In the context of the uniprocessor, when exactly one shared processor is obtainable upon which to perform every job generated by each task in the system, it is known that the EDF (earliest deadline first) scheduling algorithm, which executes at every instant in time the presently active job is a best scheduling algorithm. That is, if a system can be arranged such that every deadline can be fulfilled, this system is fulfilled by earliest deadline first. Algorithm will schedule in order to fulfil every deadline. In this unit, we will discuss the Working of Priority Driven Scheduling of Periodic Tasks.

13.1 Optimality of RM and DM Algorithm

Fixed priority algorithms can be optimal in restricted systems. A system of periodic tasks is simply periodic if the period of each task is an integer multiple of the period of the other tasks:

$$p_k = n\, p_i$$

where $p_i < p_k$ (p_i has higher priority than p_k) and n is a positive integer; for all T_i and T_k.

We discussed a flight control system, the shortest period was 1/180 seconds and the other two periods were 2*1/180 and 6*1/180.

Notes

Did u know? RM and DM are optimal in simply periodic systems.

Theorem: A system of simply periodic, independent, preemptible tasks with D_i e" p_i is schedulable on one processor using the RM algorithm if and only if U d" 1

Corollary: The same is true for the DM algorithm

- To prove that this theorem is true, suppose that the tasks are in phase (they have identical phases) and the processor never idles before the task T_i misses a deadline for the first time at t.
- T is an integer multiple of p_i.
- Because the tasks are simply periodic, t is also an integer multiple of the period p_k of every higher-priority task T_k, for k = 1,2,…, i-1
- Hence the total time required to complete all the jobs with deadlines before and at t is equal to Ó $_{k=1}{}^{i}$ $(e_k t_/ p_k)$ which is equal to t times the total utilization Ó $_{k=1}{}^{n}$ u_k of the I highest priority tasks.
- That Ti misses a deadline at t means that this demand for time exceeds t or in other words, Ui > 1

Self Assessment

Fill in the blanks:

1. …… algorithms can be optimal in restricted systems.
2. If a system can be arranged such that every …… can be fulfilled, this system is fulfilled by earliest deadline first.
3. …… will schedule in order to fulfil every deadline.

13.2 Schedulability Test

A test for the purpose of validating that the given application system can indeed meet all its hard deadlines when scheduled according to the chosen scheduling algorithm is called a Schedulability test.

- Checking whether a set of periodic tasks meet all their deadlines is a special case of the following validation problem

  We are given

  1. The period p_i, execution time e_i, and the relative deadline D_i of every task T_i in a system T = {T_1, T_2, …T_n} of independent periodic tasks and
  2. A priority-driven algorithm used to schedule the tasks in T preemptively on one processor

  We are asked to determine whether all the deadlines of every tasks Ti are always met.
- To determine whether the given system of independent periodic tasks surely meets all the deadlines when scheduling according to preemptive EDF algorithm on one processor, we check whether Δ <= 1, if it is, the system is schedulable, if not exhaustive simulation required.

Notes

13.2.1 A Schedulability Test for Fixed-Priority Tasks with Short Response Times

Fixed priority algorithms are predictable and do not suffer from scheduling anomalies. The worst case execution time of the system occurs with the worst case execution time of the jobs, unlike dynamic priority algorithms which can exhibit anomalous behaviour.

- Use this as the basis for a general Schedulability test:
- Find the critical instant when the system is most loaded, and has its worst response time
- Use time demand analysis to determine if the system is schedulable at that instant
- Prove that, if a fixed-priority system is schedulable at the critical instant, it is always schedulable

Critical Instants

A critical instant for a job is the worst-case release time for that job, taking into account all jobs that have higher priority i.e. a job released at the same instant as all jobs with higher priority are released, and must wait for all those jobs to complete before it executes. The response time of a job in *Ti released at a critical instant is called the maximum (possible) response time, and is denoted by Wi*. The schedulability test involves checking each task in turn, to verify that it can be scheduled when started at a critical instant. If schedulable at all critical instants will work at other times. More work than the test for maximum schedulable utilization, but less than an exhaustive simulation.

Caution An important problem that is addressed during the design of a uniprocessor based real time system id to check whether a set of periodic real time tasks can feasibly be scheduled under RMA.

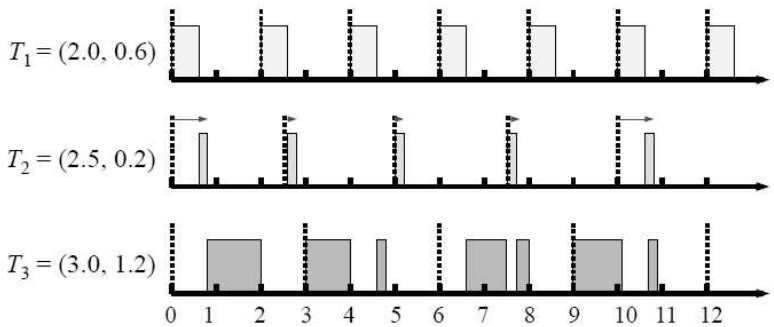
Theorem: In a fixed priority system where every job completes before the next job in the same tasks is released, a critical instant of any tasks Ti occurs when one of its job $J_{i,c}$ is released at the same time with a job in every higher priority task, that is, $r_{i,c} = r_{k,lk}$ for some l_k for every k = 1,2,…i-1

Example: 3 tasks scheduled using rate-monotonic

Response times of jobs in *T2 are:*

r2,1 = 0.8, r2,3 = 0.3, r2,3 = 0.2, r2,4 = 0.3, r2,5 = 0.8, …

Therefore critical instants of *T2 are t = 0 and t = 10*



Notes

Using Critical Instance

Having determined the critical instants, show that for each job *Ji, c* released at a critical instant, that job and all higher priority tasks complete executing before their relative deadlines

- If so, the entire system is schedulable.
- That is don't simulate the entire system, simply show that it has correct characteristics following a critical instant.

– This process is called *time demand analysis*.

Time Demand Analysis

- Schedulability test is more complex than the schedulable utilization test, but more general.
- Works for any fixed-priority scheduling algorithm, provided the tasks have short response time (i.e. pi < Di).
- Can be extended to tasks with arbitrary deadlines.
- Only a sufficient test: guarantees that schedulable results are correct, but requires further testing to validate a result of not schedulable.
- Alternative approach: simulate the behaviour of tasks released at the critical instants, up to the largest period of the tasks.
- Still involves simulation, but less complex than an exhaustive simulation of the system behaviour.

Self Assessment

State whether the following statement are True or False:

4. A critical instant for a job is the worst-case release time for that job, taking into account all jobs that have higher priority.
5. The worst case execution time of the system does not occur with the worst case execution time of the jobs, unlike dynamic priority algorithms which can exhibit anomalous behaviour.

13.2.2 A Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times

When a group of tasks share a common resource (such as a processor, a communication medium,. . .), a scheduling policy is necessary to arbitrate access to the shared resource. One of the most intuitive policies consists of assigning Fixed Priorities (FP) to the tasks, so that at each instant in time the resource is gr anted to the highest priority task requiring it at that instant. De pending on the assigned priority, a task can have longer or shorter response time, which is the time elapsed from request of the resource to the completion of the task.

Critical applications often require that the worst-case response times (i.e. the maximum possible response time) do not exceed a given deadline. Hence it is necessary to perform the Response Time Analysis (RTA) to compute exactly the worst-case response time of each task in the task system.

Notes

Example: Consider a system of n tasks τ1, τ2, . . . , τn, with the ith task ôi characterized by a worst-case execution requirement Ci, a relative deadline parameter Di , and a minimum inter-arrival separation parameter Ti . Without loss of generality, assume that the tasks are indexed in decreasing order of priority. RTA allows for the exact computation of the worst-case response time of each task, by representing the response-time of each job of each task as the solution to a recurrence equation. For example, if Di d" Ti ("i), it has been shown that the worst-case response time of ôi is equal to the smallest t satisfying the following equality:

$$t = C_i + \sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

This equation is easily solved using standard techniques for the solution of recurrence equations, in time pseudo-polynomial in the representation of the task system. Despite this pseudo-polynomial time complexity, RTA has very efficient implementations in practice that renders it quite suitable for feasibility analysis of Fixed Priority (FP) systems. However, there are certain real-time design scenarios during which there are drawbacks to using RTA: The computational complexity of these algorithms may render them unsuitable for use in interactive real-time system design environments. During a process of interactive system design and rapid system prototyping, the system designer typically makes a large number of calls to a feasibility-analysis algorithm, since proposed designs are modified according to the feedback offered by the feasibility-analysis algorithm (and other analysis techniques). In such scenarios, a pseudo-polynomial algorithm for computing the task set feasibility may be unacceptably slow; instead, it may be acceptable to use a faster algorithm that provides an approximate, rather than exact, analysis. (The computation of exact response times is especially expensive when the deadline Di is allowed to be larger than the period Ti, and/or when resource-sharing may lead to limited priority inversion and blocking; for such systems, several recurrences of the form Equation (1), one for each job of ôi within the "first busy period," must be solved in order to compute the exact response time.

Notes If jobs of a task self suspend during their execution, the task no longer behaves like a periodic task: in some intervals it may demand more time than a periodic task.

One of the features of the worst-case response time that is easily seen from Equation (1) above is that it is not a continuous function of system parameters. Informally speaking, this is a consequence of the ceiling function: for certain values of t, an infinitesimally small increase in some Tj will cause t/Tj to increase by one and the right-hand side of Equation (1) to consequently increase by an amount not smaller than Cj. Discontinuities like these are a major hurdle to a process of incremental, interactive system design. Ideally, such a de sign process would allow for the interactive exploration of the state space of possible designs; this would be greatly facilitate d if making minor changes to a design (equivalently, moving small distances in the state space of possible system designs) results in minor changes to system properties.

Self Assessment

Fill in the blanks:

6. …… test is more complex than the schedulable utilization test, but more general.
7. When a group of tasks share a common resource (such as a processor, a communication medium,. . .), a scheduling policy is necessary to …… access to the shared resource.

Notes

8. One of the features of the …… time is that it is not a continuous function of system parameters.

9. …… often require that the worst-case response times do not exceed a given deadline.

10. One of the most intuitive policies consists of assigning …… to the tasks, so that at each instant in time the resource is gr anted to the highest priority task requiring it at that instant.

Case Study

Simulation-based Multi-pass Scheduling

Multi-pass approach is one methodology for rule selection which is to combine a rule-based expert system to select several scheduling rules in a real-time environment through the test which scheduling rule is the best. Wu and Wysk developed a simulator control mechanism that evaluates the candidate dispatching rules and uses various criteria to select the best control strategy for a given time period. Wu and Wysk presented an MPS algorithm that includes a mechanism controller and a flexible simulator. MPS algorithms are defined as scheduling algorithms that solve the scheduling problem of selecting the best dispatching rule among a set of rules. Combinations of dispatching rules over a system's production cycle can produce higher throughput than a single rule alone. Cho and Wysk used neural networks to recommend candidate rules for MPS, used inductive learning and genetic algorithms to build the relationship between shop conditions and rules; some modified techniques using neural networks and production rules have also been used for identifying this relationship.

MPS frameworks consist of five components: (1) recommendation of rules for each problem type, (2) generation of all rule combinations, (3) simulation, (4) evaluation and ranking of rule combinations and (5) scheduling. Whenever the decision-making rules need to be changed, a set of promising rules for resolving each problem type is recommended. A few different types of scheduling problem may occur during the next scheduling period, so all rule combinations must be generated, and each must be evaluated using simulation. The best rule combination is chosen and conveyed to the scheduling component. One method of MPS uses a nested partitioning (NP) method and an optimal computing budget allocation (OCBA) method to reduce the computational load without the loss of throughput.

Questions:

1. Describe the various approaches of multi-pass scheduling.
2. Explain the different components of multi-pass scheduling frameworks.

Source: orsc.edu.cn/apiems/public/Abstracts/315.doc

13.3 Summary

- Algorithm will schedule in order to fulfil every deadline.
- A system of periodic tasks is simply periodic if the period of each task is an integer multiple of the period of the other tasks.
- Depending on the assigned priority, a task can have longer or shorter response time, which is the time, elapsed from request of the resource to the completion of the task.

Notes

- When a group of tasks share a common resource a scheduling policy is necessary to arbitrate access to the shared resource.
- During a process of interactive system design and rapid system prototyping, the system designer typically makes a large number of calls to a feasibility-analysis algorithm, since proposed designs are modified according to the feedback offered by the feasibility-analysis algorithm

13.4 Keywords

Schedulability Test: A test for the purpose of validating that the given application system can indeed meet all its hard deadlines when scheduled according to the chosen scheduling algorithm is called a Schedulability test.

Scheduling Points: The scheduling points of a scheduler are the points on the time line at which the scheduler makes decisions regarding which task is to be run next.

Time Demand Analysis: A schedulability test for fixed-priority systems is the time demand analysis.

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

Valid Schedule: It is one where at most one task is assigned to a processor at a time, no task is scheduled before its arrival time, and the precedence and resource constraints of all tasks are satisfied.

13.5 Review Questions

1. Explain the term Schedulability test.
2. Explain the concept of Schedulability Test for Fixed-Priority Tasks with Short Response Times.
3. What is time demand analysis?
4. Elucidate the Role of critical instants in fixed priority tasks.
5. Discuss the concept of Optimality of RM and DM Algorithm.

Answers: Self Assessment

1.	Fixed priority	2.	deadline
3.	Algorithm	4.	True
5.	False	6.	Schedulability
7.	arbitrate	8.	worst-case response
9.	Critical applications	10.	Fixed Priorities(FP)

13.6 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

Notes

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

www.fi.muni.cz/~xpelanek/IA158/slides/periodic.pdf

csperkins.org/teaching/rtes/lecture05.pdf

faculty.cs.tamu.edu/bettati/Courses/663/2009C/Slides/priority_driven.pdf

www.iust.edu.sy/courses/4.PriorityDrvnSchdl.pdf

Notes

Unit 14: Advance Priority-driven Scheduling of Periodic Tasks

CONTENTS

Objectives

Introduction

14.1 Sufficient Schedulability Conditions for the RM and DM Algorithms

14.1.1 Schedulable Utilization of the RM (Di= pi)

14.1.2 Schedulable Utilization of the RM

14.1.3 Enhanced Schedulable Utilization

14.1.4 Schedulable Utilization of the RM Algorithm for Multi-frame Tasks

14.2 Practical Factors

14.2.1 Blocking and Priority Inversion

14.2.2 Self-Suspension and Context Switches

14.2.3 Tick Scheduling

14.3 Summary

14.4 Keywords

14.5 Review Questions

14.6 Further Readings

Objectives

After studying this unit, you will be able to:

- Define Sufficient Schedulability conditions for the RM Algorithms
- Describe Sufficient Schedulability conditions for the DM Algorithms
- Analyse Practical Factors

Introduction

In case of global scheduling, we can divide schedulers in different groups. This division depends on the priority consumed by task during its execution. If a task priority cannot change during the entire lifetime of task, the algorithm of scheduling consists of "fixed task priority". In case the priority can change only at job boundaries, for example with EDF, and then the algorithm is said to have "fixed job priority". In case the priority can change throughout the job execution also, and then the algorithm is said to have "dynamic priority". In this unit, we will discuss the concept of Sufficient Schedulability conditions for the RM and DM Algorithms.

Notes

14.1 Sufficient Schedulability Conditions for the RM and DM Algorithms

Time-demand analysis method requires the periods and execution times of all the tasks in an application system to determine whether the system is schedulable.

Before we have completed the design of the application system, some of these parameters may not be known.

Sufficient Schedulability Conditions for the RM and DM Algorithms are:

1. Schedulable Utilization of the RM Algorithm for Tasks with Di= pi.
2. Schedulable Utilization of RM Algorithms as Functions of Task Parameters
3. Schedulable Utilization of Fixed Priority Tasks with Arbitrary Relative Deadlines
4. Schedulable Utilization of the RM Algorithm for Multi-frame Tasks

14.1.1 Schedulable Utilization of the RM (Di= pi)

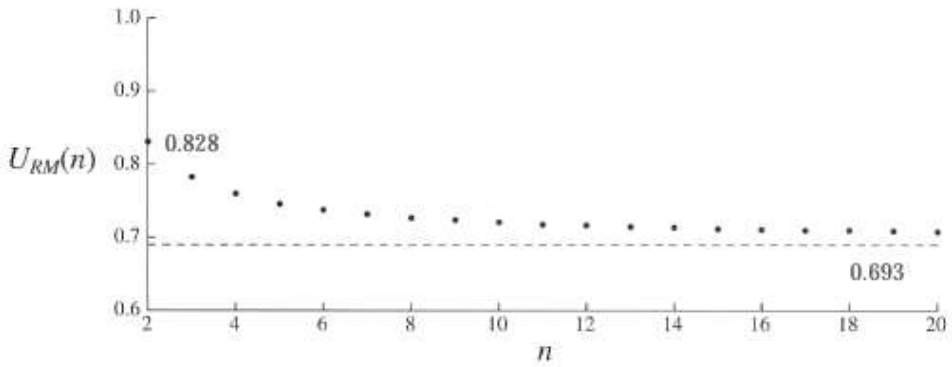
Theorem: A system of n independent, preemptible periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U is less than or equal to $U_{RM}(n) = n(2^{1/n} - 1)$.

Did u know? $U_{RM}(n)$ is the schedulable utilization of the RM algorithm when Di= pi for all.

URM(n) as a Function of n

As long as the total utilization of a system satisfies, it will never miss any deadline. We can reach this conclusion without considering the individual values of the phases, periods, and execution times.

Figure 14.1: URM(n) as a Function of n



Source: http://faculty.csie.ntust.edu.tw/~jenwei/courses/2008RTST/02%20Priority-Driven%20Scheduling%20of%20Periodic%20Tasks%20Part%20I.pdf

14.1.2 Schedulable Utilization of the RM

- Since is not a necessary condition, a system of tasks may nevertheless be schedulable even when its total utilization exceeds the schedulable bound.

Notes

Example: Total utilization of T= {(3, 1), (5, 1.5), (7, 1.25), (9, 0.5)} is 0.85, which is larger than URM(4) = 0.756, but this system is schedulable according to the RM algorithm!

14.1.3 Enhanced Schedulable Utilization

When some of the task parameters are known, this information allows us to improve the schedulable utilization of the RM algorithm:

- The Utilization of Individual Tasks
- The Number n_h of Disjoint Subsets Each Containing Simply Periodic Tasks
- Some Functions of the Periods of the Tasks

14.1.4 Schedulable Utilization of the RM Algorithm for Multi-frame Tasks

Consider a task that models the transmission of an MPEG compressed video over a network link.

- Jobs in this task, modelling the transmissions of individual video frames, are released periodically.
- The size of I-frames can be very large compared with that of Band P- frames, the execution times of the jobs can vary widely.
- When modelled as a periodic task, the execution time of the task is equal to the transmission time of an I-frame.

If we were to determine whether a system of such tasks is schedulable based on the schedulability tests we have learned, we would surely underutilized the processor.

Notes The multi-frame task model is a more accurate model and leads to more accurate schedulability tests.

Self Assessment

Fill in the blanks:

1. …… analysis method requires the periods and execution times of all the tasks.
2. As long as the total utilization of a system satisfies, it will never miss any ……
3. The multi-frame task model is a more …… model.
4. …… is the schedulable utilization of the RM algorithm when Di= pi for all.
5. A system of n independent, preemptible periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a ……

Notes

14.2 Practical Factors

We have assumed that:

- Jobs are preemptible at any time
- Jobs never suspend themselves
- Each job has distinct priority
- The scheduler is event driven and acts immediately
- These assumptions are often not valid and how does this affect the system

14.2.1 Blocking and Priority Inversion

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

These occur because some jobs cannot be pre-empted:

- Many reasons why a job may have non-preemptible sections
  - Critical section over a resource
  - Some system calls are non-preemptible
  - Disk scheduling
- If a job becomes non-preemptible, priority inversions may occur, these may cause a higher priority task to miss its deadline
- When attempting to determine if a task meets all of its deadlines, must consider not only all the tasks that have higher priorities, but also non-preemptible regions of lower-priority tasks.

Add the blocking time when calculating if a task is schedulable

14.2.2 Self-Suspension and Context Switches

Self-Suspension

- A job may invoke an external operation (e.g. request an I/O operation), during which time it is suspended.
- This means the task is no longer strictly periodic and again need to take into account self-suspension time when calculating a schedule.

Context Switches

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

Notes

14.2.3 Tick Scheduling

All of our previous discussion of priority-driven scheduling was driven by job release and job completion events. Alternatively, can perform priority-driven scheduling at periodic events (timer interrupts) generated by a hardware clock i.e. tick (or time-based) scheduling.

Additional factors to account for in schedulability analysis:

- The fact that a job is ready to execute will not be noticed and acted upon until the next clock interrupt; this will delay the completion of the job
- A ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue, the pending job queue
- When the scheduler executes, it moves jobs in the pending queue to the ready queue according to their priorities; once in ready queue, the jobs execute in priority order
- Clear that non-ideal behaviour can affect the schedulability of a system

Self Assessment

Fill in the blanks:

6. A ready job is blocked when it is prevented from executing by a …….. job.
7. A priority inversion is when a lower-priority job executes while a …… job is blocked.
8. The fact that a job is ready to execute will not be noticed and acted upon until the next ……. interrupts.
9. When the scheduler executes, it moves jobs in the …… queue to the ready queue according to their priorities.
10. A ready job that is yet to be noticed by the …….. must be held somewhere other than the ready job queue.

Case Study

Priority Manipulations

It is a POSIX-compliant real-time operating system; we can implement every task and the Meta scheduler as separate POSIX threads. Furthermore, observe that we only require preemptive priority scheduling with FIFO policy. On the other hand, the UA scheduling algorithm embedded in the Meta scheduler framework can be user-defined. Therefore, the Meta scheduler essentially needs to map the sequencing decision of a UA scheduling algorithm to priorities. This requirement manifests itself in the implementation of two complementary operations, namely Meta preempts and Meta resume.

Our approach is a four-priority scheme: the task selected by a UA algorithm is assigned a high priority P2, and all other tasks in the ready queue are assigned a low priorityP1 (P1 < P2) 5. As such, the underlying priority scheduler always schedules the high priority thread, i.e., the task selected by Meta scheduler.

In practice, threads that represent application tasks may reside in different processes and hence in different address spaces other than that of the Meta scheduler. Therefore, the Meta

Contd...

Notes

scheduler thread may not be able to directly adjust priority of an application thread. In such a case an "agent" thread is necessary to reside in each application process. The "agent" thread accepts commands from the Meta scheduler thread and adjusts priority of the target thread accordingly.

Furthermore, application threads are usually created by a "factory" thread. For example, the main thread in each process can use a timer to periodically create a sequence of application threads, which are instances of a periodic task. Conceptually, activations of the "agent" thread and the "factory" thread represent scheduling events, and hence should be "atomic" in the sense that one should not interleave with another. To do that, we assign priority P3 to both the "agent" thread and the "factory" thread. The atomicity property is guaranteed due to the FIFO policy for threads with the same priority. Note that P3 is higher than both P2 and P1, such that a scheduling event can always be issued even during the execution of an application thread. Finally, the Meta scheduler thread is assigned the highest priority, denoted as P4.

Thus, our implementation of Meta preempt and Meta resume only requires four distinct priorities, P1, P2, P3 and P4, where P1 < P2 < P3 < P4. This requirement is satisfied by virtually all POSIX-compliant real-time operating systems.

Questions:

1. Define Meta scheduler.
2. What is the function of Meta scheduler?
3. What is the significance of POSIX-compliant in real-time operating systems?

Source: http://www.computer.org/csdl/trans/ts/2004/09/e0613-abs.html

14.3 Summary

- In case the priority can change only at job boundaries, for example with EDF, and then the algorithm is said to have "fixed job priority".
- In case the priority can change throughout the job execution also, and then the algorithm is said to have "dynamic priority".
- As long as the total utilization of a system satisfies, it will never miss any deadline.
- If we were to determine whether a system of such tasks is schedulable based on the schedulability tests we have learned, we would surely underutilized the processor.
- A ready job is blocked when it is prevented from executing by a lower-priority job; a priority inversion is when a lower-priority job executes while a higher-priority job is blocked.
- A ready job that is yet to be noticed by the scheduler must be held somewhere other than the ready job queue, the pending job queue.

14.4 Keywords

Algorithm: It refers to a step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.

Context Switch: A context switch is the process of storing and restoring the state (context) of a process so that execution can be resumed from the same point at a later time.

Notes

Deadline: The latest time or date by which something should be completed.

Function: It is a named section of a program that performs a specific task.

Processor: It is a program that translates another program into a form acceptable by the computer being used.

Scheduling: It defines a policy of how to order tasks such that a metric is maximized/minimized.

14.5 Review Questions

1. What is meant by tick scheduling?
2. Briefly explain Schedulable Utilization of the RM.
3. What is Enhanced Schedulable Utilization?
4. Discuss Blocking and Priority Inversion briefly.
5. Write short notes on self-suspension and context switches.

Answers: Self Assessment

1.	Time-demand	2.	Deadline
3.	Accurate	4.	$U_{RM}(n)$
5.	Processor	6.	Lower-priority
7.	Higher-priority	8.	Clock
9.	Pending	10.	Scheduler

14.6 Further Readings

Books

Alan Burns and Andy Wellings (2001). *Real-Time Systems and Programming Languages*, Addison Wesley.

C. M. Krishna and K. G. Shin (1997). *Real-Time Systems*. McGraw-Hill International Editions.

O'Reilly Editor (1995). *Programming for the real world*.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming*, Prentice Hall.

Online links

parts.ulb.ac.be/index.../46-priority-driven_scheduling_of_periodic

link.springer.com/article/10.1023%2FA%3A1025120124771

www.ece.mtu.edu/faculty/shiyan/EE5900Spring13/PriorityScheduling.ppt

www.cs.unc.edu/techreports/02-025.pdf