I, Raghav Prasad, hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:
TASK SCHEDULE GENERATOR FOR AN RTOS

Student's name: Raghav Prasad

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.
Committee member: George Purdy, Ph.D.
Committee member: Xuefu Zhou, Ph.D.
Task Schedule Generator for an RTOS

A thesis submitted to the

University of Cincinnati

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

Department of Electrical and Computer Engineering

of the

College of Engineering and Applied Science

October 2015

by

Raghav Prasad

B.Tech. Electrical and Electronics Engineering, Anna University,

May 2011

Thesis Advisor and Committee Chair: Dr. Carla Purdy
Abstract

A real time operating system (RTOS) is an important component of many embedded applications which are becoming more common in our day-to-day lives. These systems can be divided into soft or hard real time systems based on whether the timing parameters are flexible or not. Scheduling algorithms play a major role in contributing to the working of an RTOS, they are used to schedule the various tasks present in the system. It is therefore crucial for embedded system engineers to understand the working of various scheduling algorithms and the internal design structure of an RTOS.

In this thesis, we design a task schedule generator using the C language. This schedule generator provides schedulability analysis for periodic, independent, preemptive tasks using Rate Monotonic scheduling, schedulability analysis and a schedule for periodic, independent, non-preemptive tasks using Earliest Deadline First scheduling, and a schedule for independent tasks using Priority scheduling. Further, we develop a variety of user functions that can help the user understand the internal working of our generator. This tool can be included in an RTOS that supports a variety of scheduling algorithms for a variety of functions. Further, the tool developed in this thesis can be used to study the internal workings of an RTOS schedule generator.
Acknowledgements

Firstly, I would like to thank Dr. Carla Purdy for her patience and her time during the course of my thesis. Without her guidance and support, completing my thesis would have been impossible. I am thankful to her for giving me the opportunity to participate in the research and development of Real Time Operating Systems.

I thank Dr. George Purdy and Dr. Zhou for taking the time out of their busy schedules to serve on my thesis committee.

I would also like to thank Eric B Swegert for his time and effort to help me with my thesis.

I would like to thank the University of Cincinnati and the Department of Electrical and Computer Engineering for providing me with the opportunity and an excellent platform to develop my skills as an engineer.

This acknowledgement would be incomplete if I did not express my gratitude to my family and friends without whose constant support and motivation I would not have made it this far.

Thank you all.

- Raghav Prasad
# Table of contents

Abstract........................................................................................................................................i

Acknowledgments..................................................................................................................................iii

List of figures.........................................................................................................................................vi

1 Introduction.....................................................................................................................................1
   1.1 Motivation.................................................................................................................................1
   1.2 Schedulability tests......................................................................................................................2
   1.3 Common scheduling algorithms...............................................................................................3
      1.3.1 Priority scheduling................................................................................................................3
      1.3.2 Rate Monotonic scheduling................................................................................................3
         1.3.2.1 Rate Monotonic scheduling Theorem 1.................................................................3
         1.3.2.2 Rate Monotonic scheduling Theorem 2.................................................................4
      1.3.3 Earliest Deadline First scheduling.....................................................................................5
   1.4 Response time analysis..............................................................................................................6
   1.5 Thesis overview..........................................................................................................................7
   1.6 Thesis outline..............................................................................................................................7

2 Background and related work.......................................................................................................9
   2.1 Real time systems....................................................................................................................9
   2.2 Scheduling protocols...............................................................................................................11
   2.3 Basic components of a real time system................................................................................13
   2.4 Scheduling algorithms in a real time system..........................................................................17
      2.4.1 Rate Monotonic scheduling.............................................................................................19
      2.4.2 Earliest Deadline First scheduling..................................................................................21
      2.4.3 Priority scheduling............................................................................................................22
   2.5 Design of real time operating systems..................................................................................23
      2.5.1 General characteristics of RTOS design.........................................................................23
      2.5.2 RTOS examples................................................................................................................24
         2.5.2.1 General overview of µC/OS-II.................................................................25
         2.5.2.2 General overview of µC/OS-III.................................................................32
         2.5.2.3 General overview of eCos.................................................................35
         2.5.2.4 General overview of Wind River VxWorks.........................................36
3 Design and implementation...........................................................................38
  3.1 Basic structure.........................................................................................38
  3.2 Task Control Block..................................................................................41
  3.3 Function Table.........................................................................................42
  3.4 Task state diagram..................................................................................42
  3.5 Function design.......................................................................................40
    3.5.1 Primary functions............................................................................43
    3.5.2 Ancillary functions...........................................................................44

4 Results........................................................................................................47
  4.1 Test bench 1...........................................................................................48
    4.1.1 Earliest Deadline First scheduling....................................................49
    4.1.2 Priority scheduling............................................................................51
    4.1.3 Rate Monotonic scheduling...............................................................53
  4.2 Summary of test benches.......................................................................55

5 Conclusions and future work.................................................................56
  5.1 Summary of work....................................................................................56
  5.2 Applications............................................................................................56
  5.3 Future work.............................................................................................56

References......................................................................................................58

Appendix A: Test benches............................................................................60
Appendix B: Tutorial.......................................................................................91
List of figures

Figure 2.1 Basic structure of a real time embedded system..................................................10
Figure 2.2 Real time application scheme..............................................................................11
Figure 2.3 Non-preemptive scheduling....................................................................................11
Figure 2.4 Preemptive scheduling............................................................................................12
Figure 2.5 Task state diagram...............................................................................................14
Figure 2.6 Multitasking using Task Control Blocks..............................................................14
Figure 2.7 Task set for Rate Monotonic scheduling.............................................................20
Figure 2.8 Rate Monotonic schedule of tasks.......................................................................20
Figure 2.9 Task set for Earliest Deadline First scheduling....................................................21
Figure 2.10 Earliest Deadline First schedule of tasks............................................................22
Figure 2.11 Task set for Priority scheduling..........................................................................22
Figure 2.12 Priority schedule of tasks....................................................................................23
Figure 2.13 Making tasks run in µC/OS-II...........................................................................26
Figure 2.14 Removing tasks from ready list in µC/OS-II......................................................26
Figure 2.15 Making highest priority task run in µC/OS-II.....................................................26
Figure 2.16 Working with µC/OS-II’s ready list..................................................................27
Figure 2.17 Initializing µC/OS-II.........................................................................................29
Figure 2.18 Initializing µC/OS-II........................................................................................30
Figure 2.19 µC/OS-II state diagram......................................................................................31
Figure 2.20 µC/OS-III state diagram.....................................................................................33
Figure 2.21 µC/OS-III internal state diagram.......................................................................34
Figure 2.22 eCos bitmap and multilevel schedulers.............................................................35
Figure 2.23 Microkernel profile for VxWorks.......................................................................36
Figure 3.1 RTOS schedule generator tool flowchart..............................................................40
Figure 3.2 Task Control Block parameters..........................................................................41
Figure 3.3 Task states............................................................................................................42
Figure 3.4 RTOS schedule generator tool schematic............................................................46
Figure 4.1 User interface for RTOS schedule generator tool..................................................47
Figure 4.2 Selecting the scheduling algorithm for the RTOS schedule generator tool..........47
Figure 4.3 Task parameters for test bench 1.................................................................48
Figure 4.4 Schedule parameters for test bench 1..........................................................48
Figure 4.5 Task parameters for Earliest Deadline First scheduling based on Figure 4.3......49
Figure 4.6 Task input parameters for Earliest Deadline First scheduling based on Figure 4.5...49
Figure 4.7 Simulation output for test bench 1 based on Earliest Deadline First scheduling.....50
Figure 4.8 Task schedule parameters for Priority scheduling...........................................51
Figure 4.9 Task parameters based on Figure 4.8..........................................................51
Figure 4.10 Simulation output for test bench 1 based on Priority scheduling....................52
Figure 4.11 Task parameters for Rate Monotonic scheduling.........................................53
Figure 4.12 Task parameters for Rate Monotonic scheduling based on Figure 4.11...........53
Figure 4.13 Simulation output for test bench 1 based on Rate Monotonic scheduling.......54
Figure 4.14 Summary of results for various test benches.................................................55
Figure A.1 Task parameters for test bench 2.................................................................60
Figure A.2 Schedule parameters..................................................................................61
Figure A.3 Task parameters for Priority scheduling.......................................................61
Figure A.4 Task input parameters for test bench 2........................................................61
Figure A.5 Simulation output for test bench 2 based on Priority scheduling.....................62
Figure A.6 Task parameters for Earliest Deadline First scheduling..............................63
Figure A.7 Task input parameters for Earliest Deadline First scheduling based on Figure A.6...63
Figure A.8 Simulation output for test bench 2 based on Earliest Deadline First scheduling.....64
Figure A.9 Task parameters for Rate Monotonic scheduling.........................................65
Figure A.10 Task parameters for Rate Monotonic scheduling based on Figure A.9...........65
Figure A.11 Simulation output for test bench 2 based on Rate Monotonic scheduling........66
Figure A.12 Task parameters for test bench 3...............................................................66
Figure A.13 Schedule parameters..............................................................................66
Figure A.14 Task parameters for Priority scheduling......................................................67
Figure A.15 Task input parameters for Priority scheduling .................................................. 67
Figure A.16 Simulation output for test bench 3 based on Priority scheduling ..................... 68
Figure A.17 Task parameters for Earliest Deadline First scheduling ................................. 68
Figure A.18 Task parameters for Earliest Deadline First scheduling based on Figure A.17 .... 69
Figure A.19 Simulation output for test bench 3 based on Earliest Deadline First scheduling ... 71
Figure A.20 Task parameters for Rate Monotonic scheduling ........................................... 71
Figure A.21 Task input parameters for Rate Monotonic scheduling based on Figure A.20 ....... 71
Figure A.22 Simulation output for test bench 3 based on Rate Monotonic scheduling .......... 72
Figure A.23 Task parameters for test bench 4 ................................................................. 73
Figure A.24 Schedule parameters ..................................................................................... 73
Figure A.25 Task parameters for Priority scheduling ....................................................... 73
Figure A.26 Task input parameters for Priority scheduling ................................................. 74
Figure A.27 Simulation output for test bench 4 based on Priority scheduling .................... 76
Figure A.28 Task parameters for Earliest Deadline First scheduling ................................. 76
Figure A.29 Task parameters for Earliest Deadline First scheduling .................................. 77
Figure A.30 Simulation output for test bench 4 based on Earliest Deadline First scheduling ... 81
Figure A.31 Task parameters for Rate Monotonic scheduling ........................................... 81
Figure A.32 Task parameters for Rate Monotonic scheduling based on Figure A.31 .......... 82
Figure A.33 Simulation output for test bench 4 based on Rate Monotonic scheduling ......... 83
Figure A.34 Task parameters for test bench 5 ................................................................. 84
Figure A.35 Schedule parameters ..................................................................................... 84
Figure A.36 Task parameters for Priority scheduling ....................................................... 84
Figure A.37 Task input parameters for Priority scheduling ................................................ 85
Figure A.38 Simulation output for test bench 5 based on Priority scheduling .................... 86
Figure A.39 Task parameters for Earliest Deadline First scheduling ................................. 87
Figure A.40 Task parameters for Earliest Deadline First scheduling based on Figure A.39 ... 87
Figure A.41 Simulation output for test bench 5 based on Earliest Deadline First scheduling ... 88
Figure A.42 Task parameters for Rate Monotonic scheduling ........................................... 88
Figure A.43 Task input parameters for Rate Monotonic scheduling based on Figure A.42……89
Figure A.44 Simulation output for test bench 5 based on Rate Monotonic scheduling........90
Figure B.1 User interface and sample input values.........................................................91
Figure B.2 User interface to select the algorithm..............................................................91
Chapter 1: Introduction

1.1 Motivation

An operating system (OS) is a piece of software that manages programs and resources in a computer system. A real time operating system (RTOS) is specifically designed to complete tasks by specified deadlines. RTOS timing constraints occur in a variety of applications including vehicle control, manufacturing and robotics.

An operating system can be viewed as a resource allocator. A computer system has many resources (hardware/software) that may be required to solve a problem: CPU time, memory space, file storage space, I/O devices, and so on. The OS allocates these resources to the programs and users based on their needs. Since there are many, possibly conflicting, requests for the resources, the operating system must decide to allocate resources to operate the computer system efficiently and fairly. The common functions of controlling the system and allocating resources are bought together in the operating system [12].

The main function of an RTOS is to provide a real time computing system in a controlled environment. This means that there are two sets of times to be taken into consideration while working with an RTOS, the chronological time and the chronometric time [1]. While the chronometric time deals with the time in the real world, chronological time deals with the time it takes for machine instructions to execute.

The components of an RTOS include [1]:

- Passive objects such as physical resources or logical resources
- Communication objects including messages and ports
• Synchronization objects

• Active objects such as processes, threads, etc.

• Modules created from structuring, grouping and combining objects

In order to achieve the required real time performance, an RTOS schedules tasks based on the timing constraints of every task. Tasks are scheduled based on scheduling algorithms. An operating system simply provides an environment within which other programs can do useful work.

In this thesis we develop a scheduling tool for a RTOS that supports non-preemptive scheduling of independent periodic tasks using Earliest Deadline First (EDF) scheduling, independent non-periodic tasks using Priority scheduling (PS) and check for the schedulability of independent preemptive periodic tasks for Rate Monotonic Scheduling (RMS). This tool can be incorporated into an RTOS in the future.

1.2 Schedulability tests

Schedulability tests are tests that allow the checking of whether a periodic task set that is submitted to a given scheduling algorithm might result in a feasible schedule [1]. A schedulability test is sufficient if all task sets that are schedulable by the algorithm are actually schedulable, i.e., whenever the test says a task is schedulable, a schedule exists. A schedulability test is necessary if all task sets that the test says are not schedulable, are actually not schedulable. Schedulability tests that are both necessary and sufficient are called exact [6].
A Boolean schedulability test is one which reports whether the tasks are schedulable or not. Response time tests on the other hand provide the exact worst-case execution time of each task and report on the schedulability of the task set.

1.3 Common scheduling algorithms

Scheduling algorithms determine the order in which tasks are scheduled in an RTOS. There are various factors based on which a task is scheduled. Once a task is ready to be scheduled, it is given control of the processor.

1.3.1 Priority scheduling (PS)

Tasks can be scheduled based on the priority they have been assigned. The scheduler determines the highest priority task that is ready to run and executes it. The operating system, μC/OS-II supports Fixed Priority scheduling [3].

1.3.2 Rate Monotonic scheduling (RMS)

The basic principle of Rate Monotonic scheduling (RMS) is as follows:

“The shorter the task’s period, the higher the priority.” [7]

This algorithm assigns task priorities as a monotonic function of their rates (frequencies) and hence is called the Rate Monotonic scheduling algorithm.

1.3.2.1 Rate Monotonic scheduling Theorem 1:

A set of n independent periodic tasks are schedulable using RMS if equation (1) is satisfied assuming the task deadline is equal to the task period.
\[ \sum_{1 \leq i \leq n} \left( \frac{E_i}{T_i} \right) \leq n \left( 2^{1/n} - 1 \right) \]  

(1)

where

- \( E_i \) - Maximum execution time of task \( i \)
- \( T_i \) - Execution period of task \( i \)

\( n \) - Total number of tasks

As the value of \( n \) increases to infinity, the total value of processor utilization reaches 0.693 approximately. This means tasks can be scheduled using RMS if the total processor utilization is less than or equal to 0.693.

However, the above algorithm is not a necessary condition to schedule tasks based on RMS. Theorem 2 defines a necessary and sufficient condition to schedule tasks based on RMS.

### 1.3.2.2 Rate Monotonic scheduling Theorem 2:

A set of \( n \) independent periodic tasks are schedulable using RMS if the inequality in equation (2) is satisfied assuming task deadline is equal to task period [15].

\[
S_i = \sum_{1 \leq j \leq i} \left( \frac{C_i}{IT_k} \right)^* \left[ \frac{IT_k}{T_j} \right] 
\]

(2)

\[ W(i) = \{(k,l), 1 \leq k \leq i, l=1,\ldots,\left\lfloor \frac{T_i}{T_k} \right\rfloor \} \]

(3)

\[ \forall i \min W(i) \quad S_i \leq 1 \]

(4)

where

- \( C_i \) - Maximum execution time of task \( i \)
- \( T_j \) - Execution period of task \( j \)

\( n \) - Total number of tasks

\( \min \) is calculated over \((k,l) \in W_i \)
This means that a task can be scheduled if it can complete its execution before its first deadline. The equation used in Theorem 2 checks if the task set can be executed for various scheduling points. Scheduling points are time instants at which any task period ends [7]. Tasks can be schedulable based on the Thus a task set can be scheduled using RMS Theorem 2 if each task can complete execution before its first deadline.

The following steps need to be followed to check if tasks can be scheduled based on Theorem 2[7].

Step 1: Apply Theorem 1 on all tasks until there is a case where a task does not satisfy it.

Step 2: Determine all the Schedulability points from time 0 to the end of the period of the lowest frequency task.

Step 3: Construct an inequality, where the left hand side contains the execution times of all the tasks that can run before this schedulability point and the right hand side is the value of time corresponding to this schedulability point.

Step 4: Check if the left hand side is lesser than or equal to the right hand side for any of the inequalities. The task set is schedulable if that is possible, else the task set is not schedulable.

1.3.3 Earliest Deadline First scheduling (EDF)

In this algorithm tasks are scheduled based on the earliest absolute deadline. The ready task with the earliest absolute deadline is given the highest priority. Absolute deadline is the instant of time a task needs to complete after it has been triggered. Equation (5) is a necessary condition to implement EDF scheduling:
\[
\sum_{1 \leq i \leq n} \left( \frac{E_i}{T_i} \right) \leq 1
\]  \hspace{1cm} (5)

where
- \( E_i \) - Maximum execution time of task \( i \)
- \( T_i \) - Execution period of task \( i \)
- \( n \) - Total number of tasks

### 1.4 Response time analysis

Response time analysis is used to provide the exact worst-case execution times taken by tasks to execute before their deadlines. It can be applied to any fixed priority scheduling algorithm. The advantage of response time analysis over Theorem 2, is that response time analysis is able to quantify the extent by which a task will miss its deadline [2].

Equation (6) is used to calculate the response time of a task as follows:

\[
R_i = C_i + I_i
\]  \hspace{1cm} (6)

where
- \( R_i \) is the response time of task \( i \)
- \( C_i \) is the execution time of task \( i \)
- \( I_i \) is the interference of task \( i \)

The interference time is the time task \( i \) can be preempted by higher priority tasks during its execution [10].
\[ R(i)^{m+1} = C_i + \sum_{1 \leq i \leq n} \lceil R(i)^m / T_j \rceil C_j \] 

Equation (7) is iterated until the value for \( R_i^m \) converges. The result is the worst case execution response time for a given task.

1.5 Thesis overview

This thesis aims to develop a scheduling tool for an RTOS. This tool aids in checking for the schedulability of the algorithms discussed and simulates the schedule for Earliest Deadline First and Priority scheduling. The results are stored in a text file. Further, we discuss the internals of the tool developed in this thesis.

1.6 Thesis outline

The thesis is organized as follows:

Chapter 1:

This chapter dealt with the basic scheduling algorithms used in this thesis. It discusses the necessary and sufficient conditions required to schedule tasks based on the respective algorithms.

Chapter 2:

In this chapter, a detailed description of the background required to set the tone of the remainder of the thesis is given. Also, we discuss the general design of an RTOS and the structure of several operating systems including \( \mu \text{C}/\text{OS-II} \) [3], \( \mu \text{C}/\text{OS-III} \) [4] and eCos [5].

Chapter 3:

The design used in the RTOS schedule generator tool developed is discussed in this chapter.
Chapter 4:

The results for various test benches for the designed RTOS schedule generator tool are given in this chapter.

Chapter 5:

Further improvements to the developed RTOS schedule generator tool are proposed in this chapter.
Chapter 2: Background and related work

This chapter is divided into four major sections. Section 2.1 describes the basics of real time embedded systems and their applications. Section 2.2 covers the basic components of a Real Time Operating System (RTOS). Section 2.3 discusses a few scheduling algorithms used in a Real Time Operating System. Section 2.4 explains the generic structure of a Real Time Operating System with examples.

2.1 Real time systems

An embedded system is a special purpose hardware-software system for a specific set of applications. Often it is implemented using a microprocessor. Any information processing activity or system which has to respond to externally-generated input stimuli within a finite and specified period is called a real time system [1]. Real time systems are used for various applications including engine controls, printers, heart pacemakers, etc.

Based on the response time taken by a real time system, it can be classified as either a hard real time system or a soft real time system. Hard real time systems are systems where it is absolutely imperative that responses occur within the specified deadlines. Aircraft missile systems are an example of a hard real time system, where it is imperative that the system should meet its deadline. Soft real time systems are those where the response is important but the system can function correctly if deadlines are missed. Telephone switching systems are examples of soft real time systems, where importance is given to the completion of a task by the system over the deadline. The key point to be noted is that the computers function to process information rather than perform the required operation. Figure 2.1 represents the basic structure of a real time embedded system.
A real time system provides the user with real time facilities which can be used to specify the computation time, the deadline, the period and the jitter of every task. The computation time of a task is the time required for the task to complete its execution. The deadline of a task is the maximum allowable delay for a task. The period of a task is used for periodic tasks and is the occurrence of the task at regular intervals. The jitter of a task is the deviation of a task from its expected execution. These parameters improve the performance of the system by providing a model of the tasks to be executed. Further, the real time system can respond to situations where the timing requirements of a task are changed dynamically or a situation where not all timing requirements are met.

Real time operating systems are typically used to control or process. They have a variety of advantages [8]:

- Shared resources in an RTOS can be accessed using semaphores or sharing mechanisms, thereby preventing data corruption.
- An RTOS provides task management, time management, interrupt management, communication and synchronization methods.

![Real time application scheme](image)

Figure 2.2 Real time application scheme [1]

### 2.2 Scheduling protocols

There are two families of scheduling protocols: non-preemptive and preemptive protocols. The two scheduling protocols differ in the method used to select the next task to run. Non-preemptive or cooperative multitasking occurs when tasks cooperate with each other to share the CPU. When an interrupt occurs, the higher priority task is made ready to run and the lower priority task must wait on the higher priority task. When the higher priority task finishes execution, control is given to the lower priority task as shown in Figure 2.3.
A preemptive scheduling protocol is one where a running task can be interrupted as soon as a task with a higher priority occurs. This means that the running task can be interrupted during its execution and context switched for a higher priority task as shown in Figure 2.4.
One complication with a system where tasks can be interrupted for a higher priority task is priority inversion [3]. Assume there are three tasks, Task 1, Task 2 and Task 3. Task 1 has a higher priority than Task 2, and Task 2 has a higher priority than Task 3. Now, Task 3 is executing and has acquired a semaphore to access a shared resource. Now, an interrupt occurs and Task 1 preempts Task 3. If Task 1 requires the shared resource at this time, it is suspended as Task 3 owns the shared resource. Task 3 is now made to execute and Task 1 has to wait until Task 3 completes execution using the shared resource. This phenomenon where a higher priority task has to wait for a lower priority task to complete execution before it executes is known as priority inversion. While Task 1 is waiting on Task 3, if Task 2 becomes ready to run and does not require the shared resource. At this point, Task 2 preempts Task 3 and gets control of the processor. This adds to the phenomenon of priority inversion. In the case there are more tasks with a priority higher than Task 3 but lower than Task 1, it results in further preemption of the task set and results in an unbounded priority inversion. In order to manage priority inversion there are a variety of protocols including Priority Ceiling Protocol (PCP) and Priority Inheritance Protocol (PIL). These protocols are outside the scope of this thesis and will not be discussed here.

2.3 Basic components of a real time system

A real time system is split into smaller portions of programs called tasks that are responsible for performing certain functions. A task is a program that thinks it has complete control of the CPU. Tasks are created from a program to be executed and they are assigned a priority, a set of registers and a stack area.

Figure 2.5[12] represents a generic task state diagram. Tasks are generally in one of the following five states:
- **New**: The task is being created and resides in memory. However, it has not yet been made ready for multitasking.
- **Ready**: The task can execute when it has the highest priority in the scheduling scheme.
- **Running**: This task has control of the processor at that given instance of time and its instructions are being executed.
- **Waiting**: The task requires a particular event to occur before it can move to another state.
- **Terminated**: The task has finished execution.

![Task state diagram](image)

Figure 2.5 Task state diagram [12]

Tasks are implemented in a real time operating system using Task Control Blocks. A Task Control Block (TCB) is a data structure present in an operating system that helps in managing a particular process. TCBs store all the information pertaining to a particular task. Task management properties such as the priority, the task state, scheduling parameters and stack pointer are stored in a TCB. A TCB also stores properties for memory and file management. Tasks are assigned priorities based on the algorithm used for scheduling. In this thesis, the lower the value assigned to a task, the greater its priority.
During multitasking (Figure 2.6), the kernel performs a context switch when it decides to run a different task. The current task’s context is saved in the current task’s storage area, i.e., either a common stack or the area for each task. When the operation is completed, the context of the new task is restored from its storage area and execution of the new code is resumed. Context switches add timing overhead to the system. The overhead is determined by the number of CPU registers used and the CPU efficiency.

Figure 2.6 Multitasking using Task Control Blocks [3]
Management and communication between tasks is done by the kernel. Context switching is one of the services offered by the kernel. Tasks can be divided and managed by the kernel. A kernel is a piece of software that manages the time and resources of a RTOS and provides services for applications [4]. Kernels consume extra ROM and additional RAM for the kernel data structures apart from consuming CPU time. Other services offered by the kernel include managing semaphores and time delays, etc.

The scheduler is the part of the kernel responsible for determining which task will run next. It makes use of a scheduling algorithm to determine the highest priority task ready to run. The highest priority task that is ready to run is given control of the processor. The scheduling protocol could be either preemptive or non-preemptive depending on the user’s requirements.

Interrupts are hardware mechanisms that are used to inform the processor that an asynchronous event has occurred. When an interrupt occurs, the processor performs a context switch, i.e, it saves the context of the executing task and executes an Interrupt Service Routine (ISR), after which it selects the next task to be executed based on the scheduling protocol. Clock ticks are special interrupts that occur periodically. They allow the kernel to delay tasks for an integral number of clock cycles to provide a timeout.

However, there are certain sections of code that must not be interrupted while they execute. These sections of code are called critical sections. Interrupts are generally disabled before critical sections of code are executed and enabled after they are executed. Disabling interrupts should be kept to a minimum, as it results in interrupt latency and may cause interrupts to be missed.
Tasks make use of entities that are known as resources. These resources include memory, structures, pointers and I/O devices. Shared resources are used by more than one task. Data corruption is prevented by following the principle of mutual exclusion. This results in tasks getting exclusive access to a shared resource for some amount of time.

2.4 Scheduling algorithms in a real time system

As discussed in the previous section, the scheduler is responsible for scheduling tasks based on their priority and state. The highest priority task that is ready to run is selected using a scheduling algorithm. The scheduling algorithm varies based on the application for which the task is to be executed.

In static task scheduling, the system designer assigns a priority to a task based on certain parameters. The priority of the task remains the same throughout the lifetime of the task. In dynamic task scheduling, the priority of each task is assigned dynamically at runtime.

The criteria used in selecting a scheduling algorithm include [12]:

- CPU Utilization: The CPU is kept as busy as possible. CPU Utilization varies from 0% to 100%, in the case of an RTOS it is standard that it varies from 40% (lightly loaded) to 90% (heavily used system)
- Throughput: Throughput is defined as the number of processes completed per time unit.
- Turnaround time: Turnaround time is the interval between from the time of submission to the time of completion of the task. This is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the processor and doing I/O.
• Waiting time: The waiting time is the time the processor spends waiting for the ready queue. It is the sum of the periods spent waiting in the ready queue.

• Response time: This is the time it takes for a task to start responding, but not the time it takes to output that response.

Tasks can be checked for schedulability using a variety of tests. The basic Utilization Test[14] was proposed by Liu and Layland in 1973. This test has four main assumptions.

• The tasks are periodic
• The tasks are independent
• This test is applicable for a uniprocessor system
• The system overhead is assumed to be zero

The processor utilization of a particular task is defined to be the ratio of its computation time to its period as shown in equation (8).

\[
U = \frac{E}{T} \quad (8)
\]

where

\[U\] is the processor utilization of task
\[E\] is the computation time of task
\[T\] is the period of task

Therefore, the total utilization of a given task set is shown in equation (9)

\[
U_{\text{total}} = \sum_{1 \leq i \leq n} \left( \frac{E_i}{T_i} \right) \quad (9)
\]

where

\[U_{\text{total}}\] is the total processor utilization of task the task set
\[E_i\] is the computation time of task \(i\)
\( T_i \) is the period of task \( i \)

\( n \) is the total number of tasks in the task set

In practice, each task time can be incremented slightly to allow for system overhead.

### 2.4.1 Rate Monotonic scheduling

Rate Monotonic scheduling is a method to assign priority based on the highest rate of execution. Priorities are assigned based on the period of the task, the shorter the period of the task, the greater is the priority of the task. The inequality shown in equation (10) needs to be satisfied to assign priority based on RMS [14].

\[
\sum_{1 \leq i \leq n} \frac{E_i}{T_i} \leq n \left(2^{1/n}-1\right)
\]  

(10)

where \( E_i \) is the maximum execution time of task \( i \)

\( T_i \) is the execution period of task \( i \)

\( n \) is the total number of tasks in the task set

As the total number of tasks in the task set tends to infinity, the total processor utilization of the task set is 0.693.

The important characteristic of the utilization is that it defines a limit under which the tasks will always meet their deadlines, not a limit above which the tasks will not meet their deadlines [7].
Consider a task set as shown in Figure 2.7 with three independent and periodic tasks, scheduled using a preemptive protocol:

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Start Time (ms)</th>
<th>Task Computation Time (ms)</th>
<th>Task Deadline (ms)</th>
<th>Task Period (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>0</td>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 2.7 Task set for Rate Monotonic scheduling

The tasks can be scheduled as shown in Figure 2.8. Task \(\tau_2\) takes control of the processor as it has a task period of 5ms. Once it is done executing, task \(\tau_3\) takes control of the processor as it has a task period of 10ms. Task \(\tau_1\) has a task period of 20ms and gets control of the processor next. The cycle continues based on the tasks period.

Figure 2.8 Rate Monotonic schedule of tasks [8]
2.4.2 Earliest Deadline First scheduling

Earliest Deadline First Scheduling algorithm assigns task priority based on the absolute deadline of the task. The earlier the deadline of the task, the higher its priority to be executed.

The schedulability test for EDF is as follows [14] as shown in equation (11):

\[ \sum_{1 \leq i \leq n} \left( \frac{E_i}{T_i} \right) \leq 1 \]  

(11)

where

- \( E_i \) is the maximum execution time of task \( i \)
- \( T_i \) is the execution period of task \( i \)
- \( n \) is the total number of tasks in the task set

The schedulability test does not take into account the periodicity of a task and hence can be used for scheduling both periodic and aperiodic tasks.

Consider a task set as shown in Figure 2.9, with three independent and periodic tasks, scheduled using a non-preemptive protocol:

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Start Time (ms)</th>
<th>Task Computation Time (ms)</th>
<th>Task Deadline (ms)</th>
<th>Task Period (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 2.9 Task set for Earliest Deadline First scheduling

Using the schedulability test, the tasks can be scheduled as shown in Figure 2.10. At time \( t=0 \)ms, task \( \tau_2 \) takes control of the processor as it has the smallest absolute deadline. Once it is done executing, task \( \tau_1 \) takes control of the processor at \( t=2 \)ms as it has the smallest absolute
deadline. Based on the next smallest absolute deadline, task $\tau_3$ gets control of the processor next. The cycle continues based on the next smallest absolute deadline.

![Earliest Deadline First schedule of tasks](image)

**Figure 2.10 Earliest Deadline First schedule of tasks [8]**

### 2.4.3 Priority scheduling

In this type of scheduling, tasks are assigned a constant priority. The highest priority task is always selected to run. A scheduler that works based on Priority Scheduling does not take into account system integrity, instead it works based on the temporal requirements.

Consider the task set shown in Figure 2.11, with three independent tasks scheduled using a non-preemptive protocol:

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Task Start Time (ms)</th>
<th>Task Computation Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

![Task set for Priority scheduling](image)

**Figure 2.11 Task set for Priority scheduling**
The task schedule is based on the priorities of the tasks as shown in Figure 2.12. Task $\tau_2$ has a priority of 1 and executes at time $t=0$ms. Once task $\tau_2$ executes, task $\tau_3$ executes at time $t=2$ms. Finally task $\tau_1$ executes at time $t=4$ms.

![Priority schedule of tasks](image)

**Figure 2.12 Priority schedule of tasks**

### 2.5 Design of real time operating systems

#### 2.5.1 General characteristics of RTOS design

There are two main functions of a scheduler [8]

- Maximizing the processor usage, i.e. the ratio between the active time to the idle time
- Minimizing the response time, i.e. the time between task submission time and the end of execution

Some of the criteria used to evaluate these two functions are [1]:

- Evaluating the task waiting time
- Evaluating the processor throughput
- Computing the total execution time of a given set of tasks
- Computing the average response time of a given set of tasks
The scheduler is implemented using conventional data structures [1]:

- **Election table**: An election table is used to store the schedule of the task set to be executed by the scheduler. The next task to be executed is determined by the scheduler.
- **Priority Queuing List**: A priority queuing list is an ordered list consisting of keys, generally in the form of a heap sorted list or a priority ordered list. This list is used to create an ordered list of tasks to be executed.
- **Constant or Varying Priority**: The priority of a task is a parameter that is assigned based on various parameters in a task model. Based on the static or dynamic nature of the task parameters, task priorities are either constant or varying.
- **Two-level scheduling**: As the complexity of the scheduler increases, it becomes easier to split the scheduler into two parts, a high level part and a low level part. The high level part is used for long-term decisions. The low level part consists of the election table and other short term decisions.

### 2.5.2 RTOS examples

In this section, we discuss the general overview of µC/OS-II, µC/OS-III and eCos. µC/OS-II was developed in 1999 and µC/OS-III was developed in 2009 by Micrium. eCos was developed by Cygnus Solutions in 1998. Windriver developed and released in 1987 by VxWorks. The source code for µC/OS-II and µC/OS-III can be downloaded from Micrium’s website after registration. eCos is an open source runtime system, there is no proprietary involved with it. VxWorks has a proprietary associated with it. Tasks are scheduled using Fixed Priority Scheduling in µC/OS-II. Tasks with equal priority are scheduled using round-robin scheduling in µC/OS-III. eCos has three different schedulers: the bitmap scheduler and the
multilevel queue scheduler, which are actively supported and the lottery scheduler which is not actively supported by eCos. Vxworks has a priority based preemption scheduler.

2.5.2.1 General overview of µC/OS-II

µC/OS-II is a preemptive, portable, multitasking real time system [3]. It was developed in 1998 and is mainly used for embedded systems. Also, µC/OS-II is free to use for educational purposes and the source code can be downloaded after registering on the Micrium website. µC/OS-II is written using C and some assembly language. µC/OS-II supports 255 tasks. These tasks are scheduled using Fixed Priority Scheduling. µC/OS-II can be ported onto a variety of platforms including Altera’s NIOS II processor, Intel 80x86 and Xilinx MicroBlaze.

The three major functions of the scheduler are: making the task ready to run, searching for the highest priority task ready to run and removing a task from the ready list[]. The scheduler makes use of OSRdyTbl[] and OSRdyGrp to perform the three functions.

OSSched() is used to perform the task level scheduling. If OSSched() is called from an ISR or if the application calls OSSchedLock() atleast once, then it exits. The scheduler determines the priority of a task using OSRdyTbl[] and OSRsyGrp. It checks if the highest priority task is not the current task, else a context switch needs to be done. The static counter OSCtxSwCtr is incremented everytime a context switch is performed. OS_TASK_SW() is used to perform a context switch. The code inside OSSched() is considered critical. Interrupts are disabled once control is given to OSSched(). This is done to prevent the ISRs from setting the ready bit of one or more tasks while finding the highest task to run.

A task can be made ready to run using the following snippet of code taken from the µC/OS-II source code as shown in Figure 2.13.
The last three digits of `prio` are used to determine the bit position into `OSRdyTbl[]` and the next three most significant digits are used to index into `OSRdyTbl[]`. A task is removed from the ready list using the following code snippet. Based on whether all there are any ready tasks in `OSRdyTbl[]`, the snippet of code shown in figure 2.14 clears `OSRdyGrp`.

```c
if ((OSRdyTbl[prio>>3] & ~OSMapTbl[prio & 0x07]) == 0)
    OSRdyGrp &= ~OSMapTbl[prio>>3];
```

Figure 2.14 Removing tasks from ready list in µC/OS-II

The highest priority task ready to run is set using the following snippet of code as shown in figure 2.15.

```c
y=OSUnMapTbl[OSRdyGrp];
x=OSUnMapTbl[OSRdyTbl[y]];
prio=(y<<3)+x;
```

Figure 2.15 Making highest priority task run in µC/OS-II
All tasks are assigned a unique priority level between 0 and OS_LOWEST_PRIO. Idle tasks are assigned OS_LOWEST_PRIO. A ready list consisting of two variables: OSRdyGrp and
OSRdyTbl[], is used to place the tasks ready to run in a ready list. There are eight tasks per group in OSRdyGrp. The bits in OSRdyGrp indicate when a task is ready to run. These tasks set a corresponding bit in the ready table, OSRdyTbl[] when they are ready to run. The lower three bits are used to determine the task position in OSRdyTbl[], the next three are used to represent the index into OSRdyTbl[].

The TCBs store various parameters including the task priority, the task ID, etc. Time management is performed using a periodic time ticker. There are various services including delaying a task, resuming a task, etc. performed using time management.

OSInit() is called before calling any of µC/OS-II’s other services. This is done to initialize all the required data structures in µC/OS-II. The idle task is set to OS_LOWEST_PRIO and is always ready to run. Figure 2.17 and Figure 2.18 show the initialization of the data structures in µC/OS-II.
Figure 2.17 Initializing µC/OS-II[3] (a) Represents the Ready list in µC/OS-II (b) Represents the Priority table used in µC/OS-II and the TCBs that can be accessed.
Figure 2.18 Initializing μC/OS-II[3] (c) Represents the free lists for various structures in μC/OS-II
µC/OS-II state diagram

The state diagram for µC/OS-II is as shown in Figure 2.19.
Tasks in μC/OS-II have five states:

DORMANT: The task is present in memory but is yet to have access to the multitasking kernel.

READY: The task is ready to be executed; however its priority is lesser than the current task being run.

RUNNING: The task has control of the CPU.

WAITING: The task is waiting for an event to occur. These events include waiting for a shared resource to become available, time to expire, etc.

ISR or Interrupted: The task has been interrupted and the CPU is servicing the interrupt.

2.5.2.2 General overview of μC/OS-III

μC/OS-III is a scalable, preemptive, multitasking real time system [4]. μC/OS-III is written using C and some assembly language. μC/OS-III supports an unlimited amount of tasks and can be ported onto a variety of platforms including Altera’s NIOS processor, ARM Cortex-M3 and Xilinx MicroBlaze. Like μC/OS-II, μC/OS-III is free to use for educational purposes and the source code can be downloaded after registration on the Micrium website. Tasks with equal priority are scheduled using round-robin scheduling. The scheduler used in μC/OS-III is similar to μC/OS-II, except when the time quantum of a task expires and there are multiple tasks with the same priority, μC/OS-III will select the next task to run using OS_SchedRoundRobin().
μC/OS-III state diagram

The state diagram for μC/OS-III is as shown in Figure 2.20.
As shown in Figure 2.20, tasks in μC/OS-III can have five states: dormant, ready, running, interrupted and pending. Tasks in μC/OS-III are kept in track using an internal state machine. In the internal state machine, dormant tasks are not tracked as they are unknown to μC/OS-III. Figure 2.21 represents the internal state diagram used in μC/OS-III. There are eight internal states as shown.

Figure 2.21 μC/OS-III internal state diagram [4]
2.5.2.3 General overview of eCos

Embedded Configurable Operating System or eCos is an open source operating system that was created using C/C++ and some assembly language by Cygnus [5]. There are two versions of eCos, the free version and the non-free version. eCos can be ported onto a variety of platforms including Altera’s NIOS II processor, ARM architectures and Intel 80x86. The scheduler used in eCos supports bitmap, lottery and multilevel queue scheduling. Figure 2.22 represents the bitmap and multilevel schedulers used in eCos. There exists a third type of experimental scheduler in eCos called the lottery scheduler. It is not shown in any configuration options as it is not actively supported by eCos.

Figure 2.22 eCos bitmap and multilevel schedulers [5]
2.5.2.4 General overview of Wind River VxWorks

VxWorks is a real time operating system developed by Wind River in 1987 [17]. This real time operating system provides deterministic, real time operation with security and safety certification in some cases. VxWorks is used in a variety of operations in the manufacturing, aerospace and automobile industry. It is supported on various platforms including ARM and Intel 80x86. A preemptive priority scheduler is used in VxWorks. It is easy to fix bugs and test new features in VxWorks as the OS kernel is separate from the middleware, applications and the other packages. Figure 2.23 represents the microkernel profile of VxWorks.

![Figure 2.23 Microkernel profile for VxWorks][17]

This scheduling tool is intended to be developed into an RTOS in the future. The developed RTOS should be able to support various scheduling algorithms (both preemptive and non-preemptive) and provide security and safety to the user over various platforms. Altera’s DE1
board is inexpensive and comes with the source code for Micrium μC/OS-II. This can be used to understand the inner workings of an RTOS. The Tasks schedule generator developed in this thesis is a part of the scheduler of the developed RTOS.
Chapter 3: Design and implementation

In this thesis, we develop a schedule generator tool for an RTOS. The scheduler works for Rate Monotonic scheduling (RMS), Earliest Deadline First scheduling (EDF) and Priority scheduling (PS), depending on the user’s choice. It can be integrated into a complete RTOS or used alone for a cyclic executive system. A cyclic executive is a program or a control structure that explicitly interleaves several periodic tasks in a single CPU, such that the task execution times are deterministic [16]. In this chapter, we discuss the design and implementation of the proposed scheduling tool for an RTOS. There are 6 major header files used in the design of applications using this RTOS. This chapter discusses the functions supported by each header file and their applications.

3.1 Basic structure

The proposed RTOS schedule generator aims to schedule independent tasks using Priority scheduling (PS), check the schedulability of independent preemptive periodic tasks for Rate Monotonic scheduling (RMS) or simulate independent non-periodic preemptive tasks using Earliest Deadline First Scheduling (EDF) based on the user’s choice. Each scheduling algorithm has a header file that performs operations on the tasks based on the respective algorithm.

An ID Table (OSIDTbl[]) is used to keep track of the Task Control Blocks (TCB). Any change made to a TCB is performed only after it passes through the OSIDTbl[]. The OSIDTbl[] makes use of the task ID as the index to the respective TCB. While the priority of a task changes during execution, the task ID remains the same. In this thesis each task is an integer counter that needs to be incremented when a running task gets control of the scheduling framework. The function that contains the code to increment the counter associated with each task is pointed to
by the f uncarr[] table. The task ID is used to index to the respective function code. This is present in the global_variables.h header file.

There are separate doubly linked lists for the TCBs based on the task’s state. These lists have separate pointers to their respective first and last TCB nodes. The task that is running is pointed to by the pointer TCBRunning. The first task in the dormant list, is pointed to by the TCBFirst pointer and the last task in the dormant list is pointed to by the TCBLast pointer. The ready list task set starts at the TCBReadyListFirst pointer and ends at the TCBReadyListLast pointer.

The user initially inputs the total number of tasks and the total time the task set needs to execute. Next, a menu pops up and the user is asked to choose the scheduling algorithm. Based on the scheduling algorithm chosen, the parameters required to create the TCB is input. The results are then computed and output. A text file is created based on the scheduling algorithm and the schedulability analysis and the simulated schedule is stored in the same. Figure 3.1 represents the flowchart for the RTOS schedule generator tool.
Figure 3.1 RTOS schedule generator tool flowchart
3.2 Task Control Block

The TCB contains the following information for each task:

- Task ID
- Task Priority
- Task State
- Task Deadline
- Task Period
- Task Computation time
- Pointer to the next TCB in the list
- Pointer to the previous TCB in the list

The header file os_tcb.h contains the structure for the Task Control Blocks. Figure 3.1 depicts the basic structure of the TCB. The deadline, period and computation time of the task is a float variable, while the priority, id and state are integer variables.

```c
int prio;
int id;
int state;
float period;
float computation_time;
float deadline;
struct os_tcb_orig *next;
struct os_tcb_orig *previous;
```

Figure 3.2 Task Control Block parameters
3.3 Function table

OSIDTbl[] points to these TCBs based on their Task IDs. While the priority of a task changes during execution, the task ID remains the same. The function code associated with each task is pointed to by the funcarr[] table. The task ID is used to index to the respective function code. This is present in the global_variables.h header file.

3.4 Task state diagram

Figure 3.3 represents the transition between the various states of a task in the proposed RTOS schedule generator. The states are altered using the TaskStateChange() function present in the Task_management.h header file.

When a task is initially created using the TaskCreate() function present in the Task_management.h file, it is assigned a default DORMANT state. The application program changes the state of the task to the allowed state transition using TaskStateChange(). The parameters for TaskCreate() include the computation time, deadline, period, id and priority of the task. TaskStateChange() takes the id of the task and the new state of the task as inputs. The state change is applied and the task is placed into the doubly linked list associated with the state.
3.5 Function design

3.5.1 Primary functions

These are the functions used to schedule the TCB structure and transition between states.

OSTaskCreate()

This function takes in the task parameters for a task and stores it into the TCB associated with the task. OSIDTbl points to the task’s TCB. The default value of the task state is DORMANT. Further, TCBFirst points to the first TCB present in the doubly linked list while TCBLast points to the last TCB present in the doubly linked list.

OSTaskDelete()

The task ID is passed to this function. This function is used to remove the task from the dormant TCB doubly linked list. TCBFirst and TCBLast are updated accordingly.

OSTaskStateChange()

This function allows a task to transition from one state to another. The task ID and its new state are passed to OSTaskStateChange. This function makes use of various functions to remove the TCB from the original task list, place it into the allowed state based on the state diagram and update the state of the task.

RMS_RTA()

This function is used to schedule tasks based on Rate Monotonic scheduling. It accesses the Ready list of the task set and checks if the tasks can be scheduled using Theorem 1 [], Theorem 2[] or response time analysis for Rate Monotonic scheduling. The results are stored in “RateMonotonicScheduling.txt”
PS()

This function schedules tasks based on Priority scheduling. It makes use of the task ready list to schedule the tasks. RMS_RTA() defaults to PS(), if the task set is not schedulable using Rate Monotonic Scheduling and the user decides to default to PS().

output()

This function is used to output the simulated task schedule in real time. It follows PS() in the application code. Further, it creates a text file “PriorityScheduling.txt” that stores the simulated schedule.

EDF_Init()

This function is used to initialize the parameters used for scheduling using EDF().

EDF()

This function is used to schedule tasks based on the Earliest Deadline First algorithm. It works on the ready list to schedule tasks. Also, it creates a text file “EarliestDeadlineFirstScheduling.txt” to store the schedulability analysis and the simulated schedule.

3.5.2 Ancillary functions

These functions can be used in the further development of the RTOS schedule generator into an RTOS.

TaskAccess()

This function takes the task ID as its parameter and allows the user to view the TCB contents.
printready()

This function prints the ready list of the task set.

printwaiting()

This function prints the waiting list of the task set.

printdormant()

This function prints the dormant list of the task set.

delaytimer()

This function is used to delay a task for a certain amount of time. It is used in the application program’s code.

Figure 3.4 represents the structure of the RTOS schedule generator tool.
Figure 3.4 RTOS schedule generator tool schematic
Chapter 4: Results

In this chapter, a variety of test benches are simulated against the algorithms in the RTOS schedule generator tool. In these examples it is assumed that tasks with a lower priority value have higher priority importance.

Tasks scheduled using Priority Scheduling are assigned random priorities in all examples. Tasks are arranged in non-decreasing order of priorities in the case of Rate Monotonic Scheduling and tasks are dynamically assigned priorities based on their absolute deadlines for Earliest Deadline First scheduling.

A variety of test benches are chosen. These test benches are used to prove the working of the developed schedule generator. The user interface is as shown in Figure 4.1. The total number of tasks and the total time the task schedule generator needs to execute for is input in the user interface.

![User interface for RTOS schedule generator tool](image1)

The user then selects a scheduling algorithm as shown with Figure 4.2.

![Selecting the scheduling algorithm for RTOS schedule generator tool](image2)
In this chapter, we discuss two test benches. Each test bench is executed for the three scheduling algorithms. There are a few more test benches simulated in this thesis, the results of which are shown in the Appendix.

4.1 Test bench 1

This test bench was selected from reference [7]. Consider two periodic, independent tasks with parameters as shown in figure 4.3.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 4.3 Task parameters for test bench 1 [7]

Assume these tasks need to run for 100 processor ticks. The task parameters are input into the Task Schedule Generator as shown in Figure 4.4.

Figure 4.4 Schedule parameters for test bench 1
4.1.1 Earliest Deadline First scheduling

The task parameters are input to the Task Schedule generator as shown in figure 4.5.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 4.5 Task parameters for Earliest Deadline First scheduling based on Figure 4.3

The task input parameters for Earliest Deadline First scheduling is shown in figure 4.6.

**TASK INFORMATION:**

Enter Task ID: 1
Enter Task Execution time: 10
Enter Task Deadline: 20
Enter Task Period: 20
Enter Task ID: 2
Enter Task Execution time: 15
Enter Task Deadline: 30
Enter Task Period: 30

Figure 4.6 Task input parameters for Earliest Deadline First scheduling based on Figure 4.5

The task set is simulated using Earliest Deadline First scheduling as shown in Figure 4.7.
As seen in Figure 4.7, the task set is schedulable based on the total utilization of the tasks. Based on the absolute deadlines of each task at any instant, they are given a dynamic priority and are scheduled.
4.1.2 Priority scheduling

The task parameters are as shown in figure 4.8. The task priorities are selected at random. In this case, we assume the task with ID 1 has a priority 1 and the task with ID 2 has a priority 2.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure 4.8 Task parameters for Priority scheduling

These parameters are input into the RTOS schedule generator as shown in figure 4.9.

Figure 4.9 Task parameters based on Figure 4.8

The task set is simulated based on priority scheduling as shown in figure 4.10. Tasks start executing at time 28381 and end at time 28480. Task 1 with priority 1 is executed first followed by Task 2 with priority 2. These tasks continue executing based on their priorities until the total time elapses.
PRIORIT SCHEDULING

SIMULATION:

Schedule Start:

Task 1
Start 28381  End 28391
Task 1 took 10 ticks

Task 2
Start 28391  End 28405
Task 2 took 14 ticks

Task 1
Start 28406  End 28416
Task 1 took 10 ticks

Task 2
Start 28416  End 28431
Task 2 took 15 ticks

Task 1
Start 28431  End 28440
Task 1 took 9 ticks

Task 2
Start 28441  End 28456
Task 2 took 15 ticks

Task 1
Start 28456  End 28465
Task 1 took 9 ticks

Task 2
Start 28466  End 28480
Task 2 took 14 ticks
Schedule End

Figure 4.10 Simulation output for test bench 1 based on Priority scheduling
4.1.3 Rate Monotonic scheduling

The task parameters are input to the Task Schedule generator as shown in figure 4.11.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 4.11 Task parameters for Rate Monotonic scheduling

The parameters used to schedule the tasks based on Rate Monotonic scheduling are input to the RTOS schedule generator tool as shown in Figure 4.12.

```
TASK INFORMATION:

Enter Task ID: 1
Enter Task Execution time: 10
Enter Task Deadline: 20
Enter Task Period: 20
Enter Task ID: 2
Enter Task Execution time: 15
Enter Task Deadline: 30
Enter Task Period: 30
```

Figure 4.12 Task parameters for Rate Monotonic scheduling based on Figure 4.11
The schedulability for the task set based on Rate Monotonic scheduling is checked as shown in figure 4.13.

RATE MONOTONIC SCHEDULING

Theorem 1 (C.L.Liu and J.V.Layland):
Utilization based on n = 0.799763
Utilization based on task set = 1.000000
Task set not schedulable based on Utilization

Theorem 2 (J.Lehoczky, L.Sha and Y.Ding):
Task set not schedulable based on Theorem 2

Response Time Analysis:
Task set not schedulable at Task ID 2
Task set can be rescheduled to Priority 1

Figure 4.13 Simulation output for test bench 1 based on Rate Monotonic scheduling

As shown in Figure 4.13, the task set is not schedulable based on Rate Monotonic Scheduling. The task set cannot be scheduled based on Theorem 1, Theorem 2 or Response Time Analysis.
4.2 Summary of Test benches

There are a few more task sets that have been simulated using the Task schedule generator tool. These results are shown in the Appendix. Figure 4.14 summarizes the results for the simulated test benches.

<table>
<thead>
<tr>
<th>Test bench</th>
<th># of tasks</th>
<th>EDF</th>
<th>PS</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Theorem 1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Schedulable</td>
<td>Schedulable</td>
<td>Not schedulable</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Schedulable</td>
<td>Schedulable</td>
<td>Not schedulable</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Schedulable</td>
<td>Schedulable</td>
<td>Not schedulable</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Not schedulable</td>
<td>Schedulable</td>
<td>Not schedulable</td>
</tr>
</tbody>
</table>

Figure 4.14 Summary of results for various test benches
Chapter 5: Conclusions and future work

In this chapter, we briefly summarize the work that has been accomplished along with applications and future additions to the same.

5.1 Summary of work

The RTOS task schedule generator supports schedulability analysis for periodic, independent, preemptive tasks using Rate Monotonic scheduling, provides the schedulability analysis and a simulated schedule for periodic, independent, non-preemptive tasks using Earliest Deadline First scheduling and simulates a schedule for non-preemptive, independent tasks using Priority scheduling. The results for each algorithm are stored in their respective text files.

5.2 Applications

The schedule generator tool can be used by students to understand the working of the algorithms. A simple user interface is provided to aid in this. Further, the source code was written in C and has been well documented, this means that the student can read the source code files to understand the inner working of the RTOS schedule generator.

5.3 Future work

- The proposed task schedule generator can be developed into an RTOS that supports a variety of preemptive and non-preemptive algorithms over a variety of platforms.
- In this thesis, the tasks are counters that are executed based on the scheduling algorithm. Future work could involve various other applications that require scheduling based on the algorithms. References [9] and [11] discuss the implementation of Rate Monotonic scheduling in industrial applications.
• The design of the task schedule generator is intended to be a prototype for a RTOS. The source code can be optimized, if it is used included in a complete operating system, thereby decreasing the memory used for the same.

• Given the design of the RTOS schedule generator, it is possible to add more scheduling algorithms to the same. These might include the implementation of Rate Monotonic Scheduling and preemptive versions of Priority scheduling. Reference [13] discusses the implementation of EDF in eCos. Reference [9] discusses the implementation of Rate Monotonic Scheduling on µC/OS-II.
References


Appendix A: Test benches

We discuss 4 test benches in Appendix A. These test benches are taken from [7] and are used to discuss the working of the task schedule generator tool. The task ID is assumed to be equal to the task priority for Priority scheduling. For each algorithm, schedulability analysis is checked based on the algorithm. The schedule is simulated for Earliest Deadline First scheduling and Priority scheduling, while the schedulability is checked for Rate Monotonic scheduling. Test benches 2, 3 and 4 are used to prove that task sets with different number of tasks can be scheduled using Priority scheduling and Earliest Deadline First scheduling. These test benches are also used to show that task sets can be scheduled using Theorem 2 and response time analysis, however they cannot be scheduled using Theorem 1. Test bench 5 is used to show that the task set cannot be scheduled using Earliest Deadline First and Rate Monotonic scheduling. All the test benches are schedulable based on Priority scheduling.

A.1 Test bench 2

In this example, there are three periodic, independent tasks as shown in Figure A.1.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>360</td>
<td>360</td>
</tr>
</tbody>
</table>

Figure A.1 Task parameters for test bench 2 [7]

The user inputs the number of tasks and the total time the simulation runs for as shown in Figure A.2.
A.1.1 Priority scheduling

The input parameters used to simulate priority scheduling are as shown in Figure A.3.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>80</td>
</tr>
</tbody>
</table>

These parameters are input into the RTOS schedule generator as shown in Figure A.4.

The task set is simulated based on priority scheduling as shown in Figure A.5. The tasks start executing at time 11615 and end at time 12009. Task 1 with priority 1 is executed first followed by Task 2 with priority 2 and finally Task 3 with priority 3 is executed. These tasks continue executing based on their priorities until the total time elapses.
Figure A.5 Simulation output for test bench 2 based on Priority scheduling

PRIORITY SCHEDULING

SIMULATION:

Schedule Start:

Task 1
Start 11615   End 11660
Task 1 took 45 ticks

Task 2
Start 11660   End 11710
Task 2 took 50 ticks

Task 3
Start 11710   End 11789
Task 3 took 79 ticks

Task 1
Start 11790   End 11834
Task 1 took 44 ticks

Task 2
Start 11835   End 11884
Task 2 took 49 ticks

Task 3
Start 11885   End 11964
Task 3 took 79 ticks

Task 1
Start 11965   End 12009
Task 1 took 44 ticks

Schedule End
A.1.2 Earliest Deadline First scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>360</td>
<td>360</td>
</tr>
</tbody>
</table>

Figure A.6 Task parameters for Earliest Deadline First scheduling

The parameters used to schedule the tasks based on Earliest Deadline First scheduling are as shown in A.6.

The task input parameters for Earliest Deadline First scheduling is shown in figure A.7.

Figure A.7 Task input parameters for Earliest Deadline First scheduling based on Figure A.6
The task set is simulated using Earliest Deadline First scheduling as shown in Figure 4.22.

**EARLIEST DEADLINE FIRST SCHEDULING**

Task utilization = 0.888889
Task set schedulable based on Utilization Factor

**SIMULATION:**

Schedule Start:

Task 1
start 56366 end 56411
Task 1 took 45 ticks

Task 2
start 56411 end 56461
Task 2 took 50 ticks

IDLE

Task 1
start 56501 end 56546
Task 1 took 45 ticks

Task 2
start 56546 end 56596
Task 2 took 50 ticks

Task 3
start 56596 end 56676
Task 3 took 80 ticks

Task 1
start 56676 end 56721
Task 1 took 45 ticks

Schedule End

Figure A.8 Simulation output for test bench 2 based on Earliest Deadline First scheduling
A.1.3 Rate Monotonic scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>360</td>
<td>360</td>
</tr>
</tbody>
</table>

Figure A.9 Task parameters for Rate Monotonic scheduling

The parameters used to schedule the tasks based on Rate Monotonic scheduling are as shown in Figure A.9. Figure A.10 is the input to the RTOS schedule generator tool.

![Task Information]

Figure A.10 Task parameters for Rate Monotonic scheduling based on Figure A.9

The schedulability for the task set based on Rate Monotonic scheduling is checked as shown in Figure A.11.
A.2 Test Bench 3

In this example, there are three periodic, independent tasks as shown in Figure A.12.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

The user inputs the number of tasks and the total time the simulation runs for as shown in Figure A.13.
A.2.1 Priority scheduling

The input parameters used to simulate Priority scheduling are as shown in Figure A.14.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

These parameters are input into the RTOS schedule generator as shown in Figure A.15.

The task set is simulated based on Priority scheduling as shown in Figure A.16. Tasks start executing at time 11938 and ends at time 11958. Task 1 with priority 1 is executed first followed by Task 2 with priority 2 and finally Task 3 with priority 3. These tasks continue executing based on their priorities until the total time elapses.
A.2.2 Earliest Deadline First scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

The parameters used to schedule the tasks based on Earliest Deadline First scheduling are as shown in Figure A.17.
The task input parameters for Earliest Deadline First scheduling is shown in Figure A.18.

<table>
<thead>
<tr>
<th>TASK INFORMATION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Task ID: 1</td>
</tr>
<tr>
<td>Enter Task Execution time: 1</td>
</tr>
<tr>
<td>Enter Task Deadline: 4</td>
</tr>
<tr>
<td>Enter Task Period: 4</td>
</tr>
<tr>
<td>Enter Task ID: 2</td>
</tr>
<tr>
<td>Enter Task Execution time: 2</td>
</tr>
<tr>
<td>Enter Task Deadline: 5</td>
</tr>
<tr>
<td>Enter Task Period: 5</td>
</tr>
<tr>
<td>Enter Task ID: 3</td>
</tr>
<tr>
<td>Enter Task Execution time: 7</td>
</tr>
<tr>
<td>Enter Task Deadline: 20</td>
</tr>
<tr>
<td>Enter Task Period: 20</td>
</tr>
</tbody>
</table>

Figure A.18 Task parameters for Earliest Deadline First scheduling based on Figure A.17

The task set is simulated using Earliest Deadline First scheduling as shown in Figure A.19.
EARLIEST DEADLINE FIRST SCHEDULING

Task utilization = 1.000000
Task set schedulable based on Utilization Factor

SIMULATION:

Schedule Start:

Task 1
start 45234 end 45235
Task 1 took 1 ticks
Task 2
start 45235 end 45237
Task 2 took 2 ticks
IDLE

Task 1
start 45238 end 45239
Task 1 took 1 ticks
Task 2
start 45239 end 45241
Task 2 took 2 ticks
IDLE

Task 1
start 45242 end 45243
Task 1 took 1 ticks
IDLE

Task 2
start 45244 end 45246
Task 2 took 2 ticks
Task 1
start 45246 end 45247
Task 1 took 1 ticks
A.2.3 Rate Monotonic scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

The parameters used to schedule the tasks based on Rate Monotonic scheduling are as shown in Figure A.20. Figure A.21 is the input to the RTOS schedule generator tool.

**TASK INFORMATION:**

Enter Task ID: 1
Enter Task Execution time: 1
Enter Task Deadline: 4
Enter Task Period: 4
Enter Task ID: 2
Enter Task Execution time: 2
Enter Task Deadline: 5
Enter Task Period: 5
Enter Task ID: 3
Enter Task Execution time: 7
Enter Task Deadline: 20
Enter Task Period: 20
The schedulability for the task set based on Rate Monotonic scheduling is checked as shown in Figure A.22.

RATE MONOTONIC SCHEDULING

Theorem 1 (C.L.Liu and J.W.Layland):

Utilization based on n 0.756828
Utilization based on task set 1.000000
Task set not schedulable based on Utilization

Theorem 2 (J.Lehoczky, L.Sha and Y.Ding):

k = 1 l = 5 sum = 1.000000
k = 2 l = 4 sum = 1.000000
k = 3 l = 1 sum = 1.000000
Task set schedulable based on Theorem 2

Response Time Analysis:
Task set schedulable based on RTA

Figure A.22 Simulation output for test bench 3 based on Rate Monotonic scheduling
A.3 Test bench 4

In this example, there are five periodic, independent tasks as shown in Figure A.23.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution Time</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>1500</td>
<td>1500</td>
</tr>
</tbody>
</table>

Figure A.23 Task parameters for test bench 4 [7]

The user inputs the number of tasks and the total time the simulation runs for as shown in Figure A.24.

![Figure A.24 Schedule parameters [7]](image)

A.3.1 Priority scheduling

The input parameters used to simulate Priority scheduling are as shown in Figure A.25.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>120</td>
</tr>
</tbody>
</table>

Figure A.25 Task parameters for Priority scheduling

These parameters are input into the RTOS schedule generator as shown in Figure A.26.
The task set is simulated based on Priority scheduling as shown in Figure A.27. The tasks start executing at time 91278 and ends at time 93046. Tasks continue executing based on their priorities until the total time elapses.
PRIORITY SCHEDULING

SIMULATION:

Schedule Start:

Task 1
Start 91278   End 91283
Task 1 took 5 ticks
Task 2
Start 91284   End 91320
Task 2 took 36 ticks
Task 3
Start 91320   End 91719
Task 3 took 399 ticks
Task 4
Start 91720   End 91819
Task 4 took 99 ticks
Task 5
Start 91820   End 91940
Task 5 took 120 ticks
Task 1
Start 91940   End 91945
Task 1 took 5 ticks
Task 2
Start 91946   End 91981
Task 2 took 35 ticks
Task 3
Start 91982   End 92382
Task 3 took 400 ticks
Task 4
Start 92382   End 92482
Task 4 took 100 ticks
A.3.2 Earliest Deadline First scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>120</td>
</tr>
</tbody>
</table>

The parameters used to schedule the tasks based on Earliest Deadline First scheduling are as shown in Figure A.28.

The task input parameters for Earliest Deadline First scheduling is shown in Figure A.29.
Figure A.29 Task parameters for Earliest Deadline First scheduling

The task set is simulated using Earliest Deadline First scheduling as shown in Figure A.30.
EARLIEST DEADLINE FIRST SCHEDULING

Task utilization = 0.827333
Task set schedulable based on Utilization Factor

SIMULATION:

Schedule Start:

Task 1
start 87346 end 87352
Task 1 took 6 ticks
IDLE

Task 1
start 87396 end 87402
Task 1 took 6 ticks
IDLE

Task 1
start 87446 end 87452
Task 1 took 6 ticks
IDLE

Task 1
start 87496 end 87502
Task 1 took 6 ticks

Task 2
start 87502 end 87538
Task 2 took 36 ticks
IDLE

Task 1
start 87546 end 87552
Task 1 took 6 ticks
IDLE
Task 1
start 87596 end 87602
Task 1 took 6 ticks
IDLE

Task 1
start 87646 end 87652
Task 1 took 6 ticks
IDLE

Task 1
start 87696 end 87702
Task 1 took 6 ticks
IDLE

Task 1
start 87746 end 87752
Task 1 took 6 ticks

Task 2
start 87752 end 87788
Task 2 took 36 ticks
IDLE

Task 1
start 87796 end 87802
Task 1 took 6 ticks
IDLE

Task 1
start 87846 end 87852
Task 1 took 6 ticks
IDLE

Task 1
start 87896 end 87902
Task 1 took 6 ticks
IDLE
Task 1
start 87946 end 87952
Task 1 took 6 ticks
IDLE
    Task 1
start 87996 end 88002
Task 1 took 6 ticks
    Task 2
start 88002 end 88038
Task 2 took 36 ticks
IDLE
    Task 1
start 88046 end 88052
Task 1 took 6 ticks
IDLE
    Task 1
start 88096 end 88102
Task 1 took 6 ticks
IDLE
    Task 1
start 88146 end 88152
Task 1 took 6 ticks
IDLE
    Task 1
start 88196 end 88202
Task 1 took 6 ticks
IDLE
    Task 1
start 88246 end 88252
Task 1 took 6 ticks
    Task 3
start 88252 end 88652
Task 3 took 400 ticks
A.3.3 Rate Monotonic scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>120</td>
</tr>
</tbody>
</table>

The parameters used to schedule the tasks based on Rate Monotonic scheduling are as shown in Figure A.31. Figure A.32 is the input to the RTOS schedule generator tool.
The schedulability for the task set based on Rate Monotonic scheduling is checked as shown in Figure A.33.
RATE MONOTONIC SCHEDULING

Theorem 1 (<C.L.Liu and J.W.Layland>):

Utilization based on n 0.734772
Utilization based on task set 0.827333
Task set not schedulable based on Utilization

Theorem 2 (<J.Lehoczky, L.Sha and Y.Ding>):

k= 1 l= 18 sum= 0.968889
k= 1 l= 19 sum= 0.924211
k= 1 l= 20 sum= 0.884000
k= 2 l= 4 sum= 0.884000
k= 3 l= 1 sum= 0.884000
Task set schedulable based on Theorem 2

Response Time Analysis:
Task set schedulable based on RTA
A.4 Test bench 5

In this example, there are five periodic, independent tasks as shown in Figure A.34.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution Time</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>800</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>1500</td>
<td>1500</td>
</tr>
</tbody>
</table>

Figure A.34 Task parameters for test bench 5 [7]

The user inputs the number of tasks and the total time the simulation runs for as shown in Figure A.35.

![Task Schedule Generator](image)

**Task Schedule Generator**

Enter total number of tasks
5
Enter total time
2000

Figure A.35 Schedule parameters

A.4.1 Priority scheduling

The input parameters used to simulate Priority scheduling are as shown in Figure A.36.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Task Priority</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>120</td>
</tr>
</tbody>
</table>

Table A.36 Task parameters for Priority scheduling

These parameters are input into the RTOS schedule generator as shown in Figure A.37.
Figure A.37 Task input parameters for Priority scheduling

The task set is simulated based on Priority scheduling as shown in Figure A.38. Tasks start executing at time 52688 and end at time 54692. The tasks continue executing based on their priorities until the total time elapses.
PRIORITY SCHEDULING

SIMULATION:

Schedule Start:

Task 1
Start 52688  End 52693
Task 1 took 5 ticks

Task 2
Start 52694  End 52729
Task 2 took 35 ticks

Task 3
Start 52730  End 53530
Task 3 took 800 ticks

Task 4
Start 53530  End 53630
Task 4 took 100 ticks

Task 5
Start 53630  End 53750
Task 5 took 120 ticks

Task 1
Start 53750  End 53755
Task 1 took 5 ticks

Task 2
Start 53756  End 53791
Task 2 took 35 ticks

Task 3
Start 53792  End 54592
Task 3 took 800 ticks

Task 4
Start 54592  End 54692
Task 4 took 100 ticks

Schedule End

Figure A.38 Simulation output for test bench 5 based on Priority scheduling
A.4.2 Earliest Deadline First scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution Time</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>800</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>1500</td>
<td>1500</td>
</tr>
</tbody>
</table>

Figure A.39 Task parameters for Earliest Deadline First scheduling

The parameters used to schedule the tasks based on Earliest Deadline First scheduling are as shown in Figure A.39. The task input parameters for Earliest Deadline First scheduling is shown in Figure A.40.

![Task Information](image)

Figure A.40 Task parameters for Earliest Deadline First scheduling based on Figure A.39
The task set is simulated using Earliest Deadline First scheduling as shown in Figure A.41.

**EARLIEST DEADLINE FIRST SCHEDULING**

*Task utilization = 1.227333*

*Task set not schedulable based on Utilization Factor*

*Schedule End*

Figure A.41 Simulation output for test bench 5 based on Earliest Deadline First scheduling

### A.4.3 Rate Monotonic scheduling

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution Time</th>
<th>Period</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>800</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>1500</td>
<td>1500</td>
</tr>
</tbody>
</table>

Figure A.42 Task parameters for Rate Monotonic scheduling

The parameters used to schedule the tasks based on Rate Monotonic scheduling are as shown in Figure A.42. Figure A.43 is the input to the RTOS schedule generator tool.
Figure A.43 Task input parameters for Rate Monotonic scheduling based on Figure A.42

The schedulability for the task set based on Rate Monotonic scheduling is checked as shown in Figure A.44.
RATE MONOTONIC SCHEDULING

Theorem 1 (C.L.Liu and J.W.Layland):
Utilization based on $n \geq 0.734772$
Utilization based on task set $1.227333$
Task set not schedulable based on Utilization

Theorem 2 (J.Lehoczky, L.Sha and Y.Ding):
Task set not schedulable based on Theorem 2

Response Time Analysis:
Task set not schedulable at Task ID 3
Task set can be rescheduled to Priority 2

Figure A.44 Simulation output for test bench 5 based on Rate Monotonic scheduling
Appendix B: TUTORIAL

1. Run the code found in ui.c using the GNU compiler. The directory “Sourcecode” contains the header files required to use the task schedule generator tool.

2. A user interface will appear as shown in Figure B.1. Input the total number of tasks and the total time the Task Schedule Generator need to execute for. Press the Enter key.

3. The next screen will ask you to select the scheduling algorithm to execute. Enter 1 to select Rate Monotonic scheduling, 2 to select Earliest Deadline First scheduling and 3 to select Priority scheduling. This is shown in Figure B.2.

4. Based on the algorithm selected, the parameters of the task set to be executed are input in the next screen. If the Priority scheduling algorithm is selected, the task parameters to be input are the task ID, the task priority and the task execution time. However, if either Rate Monotonic scheduling or Earliest Deadline First scheduling algorithm is chosen, the task parameters to be input are task ID, task execution time, task deadline and task period. The priorities of the tasks are calculated based on the algorithm.

5. Once the parameters are input, press the Enter key.
6. The task set is checked for schedulability and is simulated based on the scheduling algorithm. If the scheduling algorithm chosen is either Priority scheduling or Earliest Deadline First scheduling, then the schedulability analysis and the simulation of the task set is output. If the scheduling algorithm chosen is Rate Monotonic scheduling, the output is the schedulability analysis.

7. The results of scheduling are stored in text files. Priority scheduling results are stored in “PriorityScheduling.txt”. Earliest Deadline First scheduling results are stored in “EarliestDeadlineFirstScheduling.txt” and Rate Monotonic scheduling results are stored in “RateMonotonicScheduling.txt”.