CHAPTER 6

DEVELOPMENT OF NEW SCHEDULING ALGORITHM
FOR REAL TIME APPLICATIONS

6.1 INTRODUCTION

A real-time computing system can be defined as a real-time application which is expected to respond to stimuli within some small upper bound in response to time and any late result is as bad as a wrong one. A real-time system may be one where its application can be considered to be mission critical. Real-time computations can be said to have failed if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load. Thus, correctness of a real-time system could be stated true with logical perfection in the computational result and its timeliness. A soft real-time system is a system that has timing requirements, but occasionally missing the task deadlines has negligible effects. In simple terms, the scheduler defines the state of tasks within the system - running, ready, suspended or killed (deleted). It also includes the dispatching function.

In this chapter the various new scheduling algorithms has been are developed for real-time embedded systems. This section also explains the advantages of proposed algorithms over the existing algorithms.
6.2 SIMPLE ROUND ROBIN ALGORITHM

Round robin algorithm is a preemptive version of first come first serve scheduling algorithm. The tasks are arranged in the ready queue in first come first serve manner and the processor executes the task from the ready queue based on time slice. If the time slice ends and the tasks are still executing on the processor, the scheduler will forcibly preempt the executing process and keeps it at the end of ready queue. The scheduler will allocate the processor to the next task in the ready queue. The preempted task will make its way to the beginning of the ready list and will be executed by the processor from the point of interruption. A scheduler requires a time management function to implement the round robin algorithm and requires a tick timer. The time slice is proportional to the period of clock ticks. The time slice length is critical issue in soft real-time embedded application as missing of deadlines will have negligible effects in the system performance. The time slice must not be too small which results in frequent context switches and should be slightly greater than average task computation time.

6.2.1 Drawbacks of Round Robin in Soft Real-time Systems

Round robin when implemented in soft real-time systems faces two drawbacks. They are high rate of context switch and low throughput. These two problems of round robin algorithm are interrelated.

Larger Waiting Time and Response Time

The waiting time for a task is the time to wait in the ready queue for processor execution and the time the task completes its job and exits from the taskset is called turnaround time. Larger waiting and response times are clearly a drawback in round robin algorithm as it leads to degradation of system performance.
Low Throughput

Throughput is defined as number of tasks completed per time unit. If round robin is implemented in soft real-time systems, throughput will be low which leads to severe degradation of system performance. If the number of context switches is low then the throughput will be high.

Context Switch

When the time slice of the task ends and the task is still executing on the processor the scheduler forcibly preempts the tasks on the processor and stores the task context in stack or registers and allocates the processor to the next task in the ready queue. This action which is performed by the scheduler is called context switch. Context switch leads to the wastage of time, memory and leads to scheduler overhead. Context switch and throughput are inversely proportional to each other.

6.3 NEED FOR MODIFICATION IN ROUND ROBIN ALGORITHM

The aim is to develop the three different ways in which round robin algorithm is modified and made suitable to be implemented in real-time and embedded systems. The scheduling algorithm plays a significant role in the design of real-time embedded systems. Simple round robin algorithm is not efficient to be implemented in embedded systems because of higher context switch rate, larger waiting time and larger response time. Missing of deadlines will degrade the system performance in soft real-time systems. The main objective is to develop the scheduling algorithms which removes the drawbacks in simple round robin algorithm. A comparison with round robin algorithm to the proposed algorithms has been made. It is observed that the proposed algorithms solves the problems encountered in round robin algorithm in soft real-time by decreasing the number of context switches waiting time and response time thereby increasing the system throughput.
The new approach describes the scheduling of tasks in modified round robin algorithm. Simple round robin algorithm is not efficient if the taskset has tasks with variable CPU burst. The tasks arriving in round robin will be allocated time slice in first come first serve manner. These drawbacks of variable task bursts of tasks in a taskset leads to larger waiting time. The larger response time of tasks with less CPU burst leads to increase in waiting time and response time of system, thereby degrading the system performance. The proposed algorithms have the advantage of tasks with less CPU burst will have a better waiting time and response time than compared to simple round robin algorithm and covers the drawbacks of round robin algorithm.

Real-time system has always a time constraint on computation. Each task should be invoked after the ready time and must complete before its deadline. An attempt has been made to satisfy these constraints. There were various methods in which the round robin algorithm was modified such as implements a new priority queue in the round robin algorithm that gives priority to tasks with short CPU burst thereby improving the performance of the tasks with CPU burst.

6.3.1 Modified Round Robin Algorithms with Priority

In soft real-time systems missing of task deadline will not lead to catastrophic effects on the system, but will lead to negligible effects in the system which results in decrease in system performance. The proposed algorithm eliminates the drawbacks of round robin algorithms when implemented in soft real-time systems. The proposed algorithms focuses on the drawbacks of simple round robin algorithm which gives equal priority to all the tasks (tasks are scheduled in first come first serve manner), because drawback round robin algorithm is not efficient for tasks with smaller CPU burst.
The proposed algorithms eliminate the defects of implementing simple round robin algorithm in by scheduling of tasks based on the CPU burst. A dedicated small processor is used to reduce the burden of the main processor. The small processor rearranges the process in the ascending order based on the CPU burst of the process (lower to higher) as shown in Figure 6.1. The proposed algorithms have greater waiting time response time and throughput thereby improving the system performance.

![Figure 6.1 Function of dedicated smaller processor](image)

**6.4 DESIGN AND IMPLEMENTATION OF PROPOSED NEW ALGORITHM: ROUND ROBIN PRIORITY**

The proposed algorithm round robin priority eliminates the drawbacks of implementing a simple round robin algorithm in real-time system by introducing a concept called intelligent time slicing which depends on two aspects priority and context switch. The proposed algorithm allows the user to assign priority to the system. A dedicated small processor is used to reduce the burden of the main processor which calculates the time slice for the tasks based on their priority and these tasks are arranged based on priority, executed in the main processor with their individual time slices. The proposed algorithm can be implemented in real-time because of greater response time and throughput and the users can allocate priority to every individual task. It
works efficiently when the range of CPU burst for tasks is large and the scheduler follows static scheduling. The function of the small processor in priority based round robin algorithm is shown in Figure 6.2.

**Figure 6.2 Priority based round robin algorithm**

6.4.1 Time Slice Calculation for Proposed Algorithm

The time slice for the proposed algorithm is shown

\[
\text{Time slice} = \frac{\text{Range (R)} \times \text{Total no. of Task in the system (N)}}{\text{Priority (Pr)} \times \text{total no. of Priority in the system (P)}}
\] (6.1)

\[
\text{Range} = \frac{\text{maximum CPU burst} + \text{minimum CPU burst}}{2}
\] (6.2)

The step by step procedure to calculate time slice is shown as flowchart in Figure 6.3.
Figure 6.3 Steps Calculating Time Slice in the Proposed Algorithm

### 6.4.2 Case Study

Five Tasks have been defined (with their Priority), these five tasks are scheduled in the proposed algorithm. The context switch, waiting time, turn around time have been calculated and the results compared. The Task ID burst time and Priority are defined as shown in Table 6.1.
Table 6.1 Input component for the processor

<table>
<thead>
<tr>
<th>Task ID</th>
<th>CPU burst time (milliseconds)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

6.4.2.1 Execution in Simple Round Robin Algorithm

The time slice of 4 milliseconds has been used in round robin algorithm and no task is allocated to CPU for more than 1 time slice in a row. If the CPU task exceeds one time slice, that task is preempted and put into the ready queue. The task is preempted after the first time quantum and the CPU is given to the next task which is in ready queue. Similarly continues for all the task in the taskset and completes the first cycle In the second cycle Task (p2) does not require 4 milliseconds time slice (requires 1 millisecond time slice). So it quits before its time quantum expires. The CPU is given to next Task (p3). In the same way all other tasks in the system are scheduled. Once each task has received one time slice, the CPU is returned to task 1 for additional time slice. Figure 6.4 shows the task execution in simple round robin algorithm.
6.4.2.2 Execution in Proposed Algorithm 1

The time slice is calculated for all the above five tasks based on the priority. The tasks are scheduled in the ascending order based on priority. Figure 6.5 shows that task (p3) has the time slice of 15 milliseconds and has the highest priority, so it is scheduled at the beginning. The task 3 executes for 15 milliseconds and is preempted and given to task (p1) which has the next highest priority. The task 1 executes in time slice of 8 milliseconds by the Processor. In this way, all the tasks are scheduled and executed. The range (R) and the time slice has been calculated as shown in Table 6.2.

Table 6.2 Time slice calculation in priority round robin algorithm

<table>
<thead>
<tr>
<th>Task No.</th>
<th>CPU Burst</th>
<th>Priority (Pr)</th>
<th>Range (R)</th>
<th>Total No. of Task in the system (N)</th>
<th>Total No. of Priority in the system (P)</th>
<th>Time slice(ms) = $\frac{RxN}{Pr.xP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>2</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>1</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>5</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>4</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
The execution of tasks in priority round robin algorithm is shown in Figure 6.5.

![Figure 6.5 Execution of priority round robin algorithm](image)

6.4.2.3 Comparison of Proposed Algorithm with Simple Round Robin Algorithm

Round robin algorithm cannot incorporate priority in the system. It schedules the tasks in first come first serve manner based on the time slice which is equal for all the tasks in the system. The proposed algorithm incorporates priority the in simple round robin algorithm. The tasks are scheduled based on priority and the time slice has been calculated for each and every individual task. The tasks executes on the processor based on the calculated time slice. As shown in Table 6.2, task 3 has the highest priority, so it has less waiting time and less turn around time when executed in normal round robin algorithm. The proposed algorithm proves that the proposed algorithm is superior to normal round robin algorithm. Tasks with highest priority, less waiting time and less turn around time increase the overall system performance of higher priority tasks.
6.4.3 **Drawbacks in Round Robin Priority Algorithm**

The proposed algorithm has various drawbacks. Although tasks with higher priority get less waiting time and response time, tasks with lower priority are severely affected by this algorithm, which results in the overall decrease in the system performance. Another drawback of this algorithm is that two tasks cannot be given equal priority. These drawbacks lead to the development of intelligent time slice for round robin algorithm.

6.5 **INTELLIGENT TIME SLICE FOR ROUND ROBIN**

The proposed algorithm focuses on the drawbacks of round robin priority algorithm. In soft real-time systems, the missing of task deadlines are not catastrophic but have negligible effects on the system which results in degradation of system performance. The proposed algorithm eliminates the defects of implementing a simple round robin in soft real-time system by introducing a concept called intelligent time slicing which depends on three aspects priority, average CPU burst, context switch avoidance time. The proposed algorithm allows the user to assign priority to the system. An assumption is made on average CPU burst which is reasonable to the system. A dedicated small processor is used to reduce the burden of the main processor. A small processor calculates the time slice. The calculated time slice is different and independent for each task and the tasks are fed into the ready queue and these tasks are executed in the main processor with their individual time slices. The proposed algorithm can be implemented in soft real-time because of greater response time and throughput and the users can allocate priority to every individual task. The intelligent time slice is calculated by the dedicated small processor as shown in Figure 6.6.
Figure 6.6 Intelligent time slice generation

The priority, average CPU burst and context switch avoidance time along with original time slice are given to the small dedicated processor which calculates the new time slice dedicated to the corresponding tasks. The output of small dedicated processor will be the task id no, CPU burst and the time slice. These criteria are given to the main processor. These dedicated time slices are different and are exclusively allocated for each task and the tasks are executed on the processor based on these time slicings.

6.5.1 Intelligent Time Slice Calculation

A new way of intelligent time slice calculation has been proposed which allocates the time frame exclusively for each task based on priority, shortest CPU burst time and context switch avoidance time.

Let the original time slice (OTS) is the time slice to be given to any process if it deserves no special consideration. The intelligent time slice has been calculated.

\[
\text{Intelligent time slice} = \text{Original Time Slice (OTS) + Priority Component (PC) + Shortness Component for CPU burst time (SC) + Context Switch Component (CSC)}
\]

(6.3)
Priority component (PC) is assigned by the user depending upon the priority which is inversely proportional to the priority number (higher the priority, greater the PC).

Shortness component (SC) is assigned inversely proportional to the length of the next CPU burst for the process. Shortest component should be lesser than the assumed CPU burst (ATS).

Context Switch Component (CSC) is calculated as follows:

- Calculate the Computed Component (CC): Priority Component (PC) and Shortness Component (SC) are added to the Original Time Slice (OTS).

\[
\text{Computed component (CC)} = \text{Priority Component (PC)} + \text{Shortness Component (SC)} + \text{Original Time Slice (OTS)}
\]

- CC is deducted from the Assumed CPU burst (ATS). Let the result be called balance CPU burst.

\[
\text{Balanced CPU burst} = \text{Assumed Time Slice (ATS)} - \text{Computed Component (CC)}
\]

If this balance CPU burst is less than OTS, it is considered as Context Switch Component (CSC).

Then the intelligent time slice is calculated as follows:

\[
\text{Intelligent Time Slice} = \text{OTS} + \text{PC} + \text{SC} + \text{CSC}
\]
6.5.2 Case Study with Five Tasks

Five tasks have been defined (with their priority). These five tasks are scheduled in round robin and also in the proposed algorithm. The context switch, waiting time and turn around time have been calculated and the results compared. The process ID, burst time and priority are defined as shown in Table 6.3.

<table>
<thead>
<tr>
<th>Process ID</th>
<th>CPU burst time (milliseconds)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

6.5.2.1 Simple Round Robin Algorithm

The above five tasks have been scheduled using simple round robin algorithm. The time slice of four milliseconds has been used. In round robin algorithm no process is allocated CPU for more than one time slice in a row. If the CPU process exceeds one time slice, the concerned process will be preempted and put into the ready queue. The process is preempted after the first time quantum and the CPU is given to the next process which is in the ready queue (process P2), similarly schedules all the process and completes the first cycle. In the second cycle, process P2 requires one millisecond time slice and doesn’t require four milliseconds time slice. So it quits before its time quantum expires. The CPU is given to next process P3. In the same way, all other tasks in the system are scheduled. Once each process has received one time slice, the CPU is returned to process P1 for additional time slice. The process time slicing in simple round robin algorithm is shown in Figure 6.7.
6.5.2.2 Intelligent Time Slice for Round Robin

The above five tasks have been scheduled using proposed algorithm. Intelligent time slice has been calculated and the calculated intelligent time slice is shown in Table 6.4.

Table 6.4 Calculation of intelligent time slice for round robin

<table>
<thead>
<tr>
<th>Process ID</th>
<th>CPU burst</th>
<th>OTS</th>
<th>PC</th>
<th>SC</th>
<th>CSC</th>
<th>Intelligent time slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

The intelligent time slice of process P1 is the same as the original time slice of four milliseconds and time slice of four milliseconds is assigned to process P1. After the execution of four milliseconds time slice, the CPU is allocated to process P2. Since the CPU burst of process P2 is lesser than the assumed CPU burst (ATS), one milliseconds of SC has been included. The
process P3 has the highest priority. So priority component is added and the total of five milliseconds is allocated to process P3. The balanced CPU burst for process P4 is lesser than OTS, context switch component is added and a total of eight millisecond time slices is given to process P4. Process P5 is given a total of five milliseconds with one millisecond of priority component is added to original time slice. After executing a cycle, the processor will again be allocated to process P1 for the next cycle and continuously schedules in the same manner. The process time slicing in the proposed algorithm is shown in Figure 6.8.

![Figure 6.8 Time slicing in the proposed algorithm](image)

The context switch, waiting time and turn around time have been calculated and the comparison has been made as shown in Figure 6.9.
Waiting time is calculated by

\[ \sum_{i=1}^{n} \frac{wt_i}{n} \]

\( wt \) - waiting time of process.
\( n \) - No. of process.

Turn around time is calculated by

\[ \sum_{i=1}^{n} \frac{tt_i}{n} \]

\( tt \) - turned around time of process.
\( n \) - No. of process.

**Figure 6.9 Round robin vs Intelligent time slice for round robin**

The algorithm for implementing intelligent time slice for round robin is shown in Figure 6.10.
1. I/P Process: Cpu burst, priority, time slice
   O/P Process: Average waiting time (awt), Average turn around time (att)

2. Initialize ready queue = 0 awt=0 att=0
3. While (ready queue = null)
   Processor Idle
   Else
   Sort the tasks in the ready queue based on priority
4. Find and assign intelligent time slice (it) for each process
   For each process i = 1 to n
   P[i] → it
5. If a new process arrives
   Update the counter and go to step 2
6. awt, att are calculated
7. Stop and exit

Figure 6.10 Algorithm for intelligent time slice for round robin

Table 6.5 Round robin and intelligent time slice for round robin comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Waiting time in milliseconds</th>
<th>Turn around time in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple round robin 5 tasks 100 tasks</td>
<td>31</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>105.80</td>
<td>141.22</td>
</tr>
<tr>
<td>Intelligent time slice for round robin 5 tasks 100 tasks</td>
<td>25</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>80.408</td>
<td>111.56</td>
</tr>
</tbody>
</table>
A comparative study of round robin algorithm and intelligent time slice for round robin algorithm is made. It is observed that the proposed algorithm is superior as it involves less waiting time, less turn around time and lower no of context switches. Therefore, it is concluded that proposed intelligent time slice for round robin algorithm has the potential to be considered in future research.

6.5.3 Case Study with Randomly Generated Task Set

100 tasksets have been randomly created and each taskset has 100 tasks the input parameters for each taskset has been defined. The created 100 task set has been computed with round robin and proposed algorithm. The comparative analysis is shown in Table 6.5.

6.6 A NEW ALGORITHM: PRIORITY IN RATE MONOTONIC AND DEADLINE MONOTONIC

Real-time scheduling algorithms such as rate monotonic and deadline monotonic plays an important role in scheduling real-time tasks in a real-time environment. There are some cases where inconsistencies may arise such as tasks having less task period but their execution is not very important. In this case, when scheduled under rate monotonic algorithm, the CPU unnecessarily spends time in scheduling the tasks that are not uttermost importance. The proposed algorithm eliminates this drawback and combines the advantages of both rate monotonic and deadline monotonic algorithms. It also incorporates a priority component which is specified by the user which denotes the importance of tasks in the system.

Drawbacks of Rate Monotonic

This policy considers a single task timing criterion and task period. Ready tasks are arranged in order dependent on their period. The tasks with
the shortest period is placed in the front of the ready queue, being allocated highest priority. The next task is the one which has the second shortest period, and so on. In this strategy, the most frequent task executes first. A running task may be preempted if a task with a shorter period is readied. In this algorithm all the tasks are considered of equal priority. This may lead to the starvation of tasks having large task periods.

**Drawbacks of Deadline Monotonic**

This scheduling policy is the extension of rate monotonic scheduling. Here the deadlines of a process is less er than the period and the priority will be assigned based on their deadlines. It allows the completion of process execution to be defined more precisely. It allows sporadic process to be easily incorporated. In this algorithm all the tasks are considered of equal priority. This may lead to the starvation of tasks having large task Deadlines. From the literature, it is evident that the rate monotonic and deadline monotonic algorithms considers all tasks to be of equal importance. The proposed algorithm combines the advantages of both rate monotonic and deadline monotonic, by taking both task period and deadline and also priority component have been added to eliminate the drawback of both the algorithm.

The proposed algorithm covers the drawbacks of both rate monotonic (RM) and deadline monotonic algorithms (DM). In rate monotonic all the tasks are assumed to be of equal importance and the tasks are scheduled in the task scheduler based on task period. Because of this drawback, there may be some unimportant tasks which may have less task period that could be scheduled before an important task. The same criteria also implies to deadline monotonic algorithm where all the tasks are assumed to be of equal importance and are scheduled based on deadlines. The proposed algorithm eliminates this drawbacks by incorporating a priority component which is defined by the user in the same way as task period and
deadline. Percentage of weightage will be allocated to all these criteria and based on these weightages, a scheduling component (SC) will be computed. Based on the scheduling value, the tasks will be scheduled in the system. This proposed algorithm can be implemented in real-time system where neither the task period nor deadline vary with time (the same as rate monotonic and deadline monotonic). Table 6.6 shows the comparison between existing rate monotonic and deadline monotonic algorithms to the proposed algorithm.

Table 6.6 Comparison between existing algorithms and proposed algorithm

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Rate monotonic (RM)</th>
<th>Deadline monotonic (DM)</th>
<th>New algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task importance</td>
<td>Equal importance to all tasks</td>
<td>Equal importance to all tasks</td>
<td>Priority component is added</td>
</tr>
<tr>
<td>Scheduling criteria</td>
<td>Task period</td>
<td>Task deadline</td>
<td>Task period</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Task deadline</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Task priority</td>
</tr>
<tr>
<td>Task scheduling</td>
<td>Tasks are arranged in ascending order based on task period</td>
<td>Tasks are arranged in ascending order based on deadline</td>
<td>Percentage of weightage is allocated based on the computations a new component scheduling component will be computed (SC) based on the scheduling component value tasks will be placed in ascending order</td>
</tr>
<tr>
<td>Time difference</td>
<td>Task period does not vary with time</td>
<td>Task deadline does not vary with time</td>
<td>Task period, Task deadline and Task priority does not vary with time</td>
</tr>
</tbody>
</table>
6.6.1 Scheduling Component Calculation

The proposed algorithm deviates from the existing rate monotonic and deadline monotonic as it incorporates priority component. The proposed algorithm contains three components task period, task deadline and task priority components. Scheduling component are calculated by adding these components and the tasks are scheduled based on the scheduling component.

Since it is a static scheduling algorithm and tasks with two different criteria can be brought together, the third criteria called priority. Number of priority in the system must always be equal to the weightage assigned to the priority component of the system.

**Task Period Component (TPC) is calculated by**

\[
\text{Weightage percentage} \times \text{Task period}
\]

(eg let us assume the task period for task as 40
This implies \(20\% \times \text{task period} = 20\% \times 40 = 8\))

**Task Deadline Component (TDC) is calculated by**

\[
\text{Weightage percentage} \times \text{Task Deadline}
\]

(eg let us assume the task period for task as 40
This implies \(30\% \times \text{task Deadline} = 30\% \times 25 = 10\))

Task Priority component (PC) is assigned by the user

In the above example, weightage of task priority is 50% since the weightage is equal to number of priority in the system. The priority in the system ranges from 1 to 50 eg let us assign the priority component as 25
The priority range does not reflect the number of tasks in the system.

Scheduling component (SC) is calculated by

\[
\text{Scheduling Component (sc)} = \text{task priority component (TPC)} + \text{Task Deadline Component (TDC)} + \text{Priority component (pc)}
\]

(eg \( \text{SC} = 8 + 0 + 25 = 43 \) )

The scheduler rearranges the scheduling component based on ascending order and schedules the tasks.

If the scheduling component of two or more tasks are the same, then the component with next highest percentage of weightage is taken into account here the priority of the task has to be considered. The calculation of SC of proposed algorithm has been shown in Figure 6.11.

![Figure 6.11 Calculation of scheduling component for the proposed algorithm](image)

### 6.6.2 Case Study

The period weightage percentage is assumed as 20% and deadline weightage percentage is assumed as 30% and priority weightage is assumed as 50%.
The input parameters for the tasks are given in Table 6.7.

**Table 6.7 Input component for the processor**

<table>
<thead>
<tr>
<th>Task</th>
<th>Period (milli sec)</th>
<th>Deadline (milli sec)</th>
<th>Cpu Burst (milli sec)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>40</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>60</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>30</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Five tasks have been defined with their priority, period, deadline, CPU burst, These five tasks are scheduled in the proposed algorithm. Table 6.8 shows the scheduling component calculation. The context switch, waiting time and turnaround time have been calculated and the results are compared accordingly.

**Table 6.8 Scheduling component calculation**

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Period Component (TPC)</th>
<th>Task Deadline Component (TDC)</th>
<th>Priority Component (PC)</th>
<th>Scheduling Component (SC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>12</td>
<td>44</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>18</td>
<td>49</td>
<td>77</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>9</td>
<td>40</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>5</td>
<td>48</td>
<td>61</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>42</td>
<td>49</td>
</tr>
</tbody>
</table>
Figure 6.12 Task executions in proposed algorithm

Figure 6.12 shows the task execution in proposed algorithm. Here for task 1 has the period of 25 milliseconds and the deadline is of 40 milliseconds and it has 7 milliseconds CPU burst. The priority of 6 has been assigned to task 1. In rate monotonic algorithm, this task will be scheduled second, since rate monotonic algorithm deals with less task period scheduled first from all the tasks in the task set. In deadline monotonic algorithm, this task will be scheduled fourth since Deadline monotonic algorithm deals with tasks with less deadline are scheduled first. Task period component (TPC), Task deadline component (TDC), and Priority component (PC) are calculated and Scheduling component (SC) is calculated by the addition of all the three criteria, the task with highest scheduling criteria in the task set is allocated with the processor. Task 1 will be the third task which will be allocated to the processor. Here Task 1 and Task 4 have the same scheduling component. But task 4 has the higher priority and will be the first to be allocated by the processor for execution. In this way, all the tasks will be executed in the increasing order of scheduling component. The step by step procedure to calculate scheduling component (SC) is shown in Figure 6.13.
Figure 6.13 Steps to calculate scheduling component for new proposed algorithm

6.7 CONCLUSION

This chapter gives the details about the design and development of two new scheduling algorithms for real-time systems. A comparative study of round robin algorithm, shortest round robin, priority round robin and intelligent time slice for round robin algorithm is made. It is concluded that the proposed algorithms are superior, as it has less waiting, response times,
usually less preemption and context switching thereby reducing the overhead and saving of memory space.

The developed algorithm eliminates the drawback and combines the advantages of both Rate monotonic and Deadline monotonic algorithms. It also incorporates a priority component which is specified by the user which denotes the importance of tasks in the system. A case study with the example was also presented on these proposed algorithms.