IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727 Volume 13, Issue 2 (Jul. - Aug. 2013), PP 44-51
www.iosrjournals.org

Survey of Real Time Scheduling Algorithms

Swati Pandit [1], Rajashree Shedge [2]
[1](Computer Department, Ramrao Adik Institute of Technology/ Mumbai University,India)
[2](Computer Department, Ramrao Adik Institute of Technology/ Mumbai University,India)

Abstract: Real-Time systems are becoming pervasive. In a Real-Time System the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. A missed deadline in hard real-time systems is catastrophic and in soft real-time systems it can lead to a significant loss .This work talks about static and dynamic scheduling algorithms for real time task. The problem of real-time scheduling spans a broad spectrum of algorithms from simple uniprocessor to highly sophisticated multiprocessor scheduling algorithms which are priority driven and divided into three classes fixed priority, dynamic priority and hybrid priority. Finally conclusion shows that Instantaneous utilization factor scheduling Algorithm gives better result in uniprocessor scheduling algorithms and Modified Instantaneous utilization factor scheduling Algorithm gives better context switching, response time and CPU utilization as compared to previous scheduling algorithms.

Keywords- Deadline, Laxity, Utilization, Precedence, context switching.

I. Introduction

A Real Time Scheduling System is composed of the scheduler, clock and the processing hardware elements. In a Real-Time system, a process or task has schedulability; tasks are accepted by a real-time system and completed as specified by the task deadline depending on the characteristic of the scheduling algorithm.. A deadline is defined as the time required for a task to be processed. In a critical operation the task must be processed in the time specified by the deadline [1]. A system is said to be unschedulable when tasks cannot met the specified deadlines. Goals of the real time Scheduling algorithms are Meeting the timing constraints of the system, preventing simultaneous access to shared resources and devices, attaining a high degree of utilization while satisfying the timing constraints of the systems however this is not a primary driver, Reducing the cost of context switches caused by preemption, reducing the communication cost in real-time distributed systems; we should find the optimal way to decompose the real-time application into smaller portions in order to have the minimum communication cost between mutual portions each portion is assigned to a computer [2].

Several studies have developed optimal scheduling policies for implicit deadline task systems. So far however, studies have failed to develop effective scheduling strategies for more general task systems such as constrained deadline tasks. We argue that a narrow focus on deadline satisfaction (urgency) is the primary reason for this lack of success [3]. Different priority driven algorithms for uniprocessor such as Earliest Deadline First, Least Laxity First, and Maximum Urgency First, Instantaneous utilization factor scheduling Algorithm and their corresponding enhanced or modified version in the multiprocessor that is Earliest Deadline until Zero Laxity, Improved Least Laxity First, Modified Maximum Urgency First, Modified Instantaneous utilization factor scheduling Algorithm are explained in detail with their performance analysis.

The report proceeds with what other related work we have done before in chapter 2. Chapter 3 explains study of uniprocessor priority driven real time scheduling strategies precedes study of multiprocessor priority driven real time scheduling strategies in chapter 4. After that in chapter 5 Analysis of the all the algorithms is done with showing that how Modified Instantaneous utilization factor scheduling Algorithm is efficient, finally conclusion is produced in chapter 6 and the report ends with references that we have taken for whole work.

II. Literature Survey

The literature survey starts with the introduction of RTS and focuses on some basic concepts of real time system and scheduling.

1. Real Time Systems

A real Time System is a computer-controlled mechanism in which there are strict timing constraints on the computer's actions. In another words, the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced. Such systems can be as simple as an alarm clock or can be the most complex systems built (or considered), such as the once-planned strategic defense initiative missile defense system.

RTS can be divided further into two categories:

Hard-Real-Timesystem which must meet its deadline, otherwise it will cause undesirable damage or a fatal error to the system. Soft-Real-Time System, which has an associated deadline that is desirable but not mandatory, it still makes sense to schedule and complete the task even if it has passed its deadline. We will focus on the Hard-Real-Time system.

A RTS consists of four parts: Physical process (which is controlled by the computer for some productive end); Sensors (Converts state of physical process into information analog or digital); Computer (Based on information from sensors, deduces state of physical process and issues commands to control the process) and Actuators (In response to commands issued computer, modifies the physical process)sa shown in figure 2.1

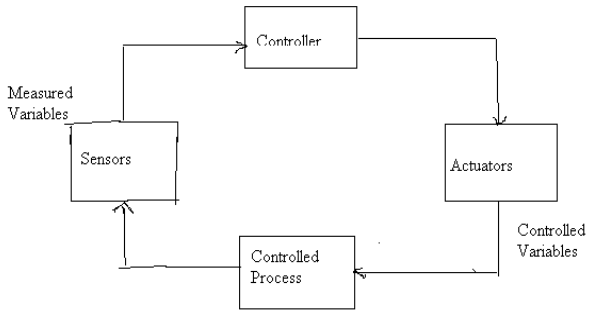

Fig.1. Real Time System Architecture

Characteristics of Real Time Systems:

- Determinism: An operating system is deterministic to the extent that it performs operations at fixed, predetermined times or within predetermined time intervals.
- Responsiveness: Responsiveness is concerned with how long, after acknowledgment, it takes an operating system to service the interrupt.
- User control: A real-time system may also allow the user to specify such characteristics as the use of paging or process swapping, what processes must always be resident in main memory, what disk transfer algorithms are to be used, what rights the processes in various priority bands have and so on.
- Reliability: Reliability in real-time system is responding to and controlling events in real time, loss or degradation of performance may have catastrophic consequences, ranging from financial loss to major equipment damage and even loss of life.
- Fail-soft operation: the ability of a system to fail in such a way as to preserve as much capability and data as possible.

2 Real Time Task Model

2.1 Task Description

A task model has been defined with the main timing parameters. A task is defined by chronological parameters denoting delays and by chronometric parameters denoting times. The model includes primary and dynamic parameters as shown in figure 2.2. Primary parameters are:

• r, task release time, i.e. the triggering time of the task execution request.
• C, task worst-case computation time, when the processor is fully allocated to it.
• D, task relative deadline, i.e. the maximum acceptable delay for its processing.
• T, task period (valid only for periodic tasks).

When the task has hard real-time constraints, the relative deadline allows computation of the absolute deadline d = r +D.

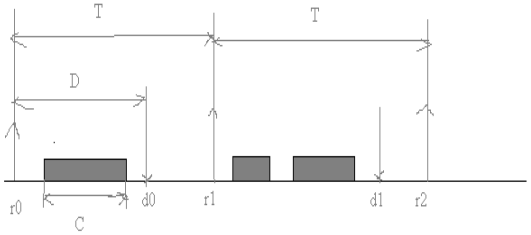

Fig.2. Task Model (Timing Diagram)

The parameter T is absent for an aperiodic task. A periodic task is modeled by the four previous parameters. Each time a task is ready, it releases a periodic request. The successive release times (also called request times, arrival times or ready times) are request release times at rk = r0 + kT, where r0 is the first release and rk the k + 1th release; the successive absolute deadlines are dk = rk + D. If D = T , the periodic task has a relative deadline equal to period. A task is well formed if $0 < C \leq D \leq T$ [4].

2.2 Task Characteristics

- **Arrival Patterns**

Based on their arrival patterns over time, tasks can be divided into two categories: periodic and aperiodic. A task is said to periodic when it is available for execution, regardless of processor availability and inter-task dependencies. Aperiodic tasks have arbitrary arrival patterns.

- **Inter-Task Dependencies**

Depending on the system's definition of the task, there may be dependencies among tasks. A task j is defined to be dependent on a set of tasks S (j) if j cannot start executing until all tasks in S (j) have completed their execution.

- **Deadline Laxity**

All real-time tasks define a deadline by which they have to complete executing. However, the problem of finding an optimal schedule for real-time tasks has been shown to be NP-complete.

III. Uniprocessor Scheduling Algorithms

1 Earliest-Deadline-First Scheduling (EDF)

EDF Scheduling is proposed by Liu and Leyland [6]. This Scheduling Algorithm is based on uniprocessor dynamic-priority preemptive scheme and optimal among all dynamic priority scheduling algorithm. This Algorithm schedules the task with the earliest deadline first.

The algorithm of EDF scheduling is following as;

Step 1: Set up all tasks' start time, end time, remaining time and deadline.

Step 2: If system is idle, then add task to schedule and go to step 4. Otherwise go to step 3.

Step 3: If new task's deadline is earlier than processing task's deadline, then update processing task's remaining time and exchange tasks. Otherwise update new task's start time.

Step 4: If all the tasks have not been scheduled, then go to step 2. Otherwise stop.

2 Least Laxity First (LLF)

David B. Stewart and Pradeep K. Khosla proposed this algorithm [7]. The minimum-laxity- first algorithm assigns a laxity to each task in a system, and then selects the task with the minimum laxity to execute next. This algorithm is also called as Least Laxity First (LLF).Laxity is defined as follows:

laxity=deadline_time- current_time - CPU_time_still_needed

Laxity is a measure of the flexibility available for scheduling a task. A laxity of tl means that even if the task is delayed by tl time units, it will still meet its deadline. A laxity of zero means that the task must begin to execute now or it will risk failing to meet its deadline. The laxity of a process is defined as the deadline minus remaining computation time. The algorithm gives the highest priority to the active job with the smallest laxity [5]. Then the job with the highest priority is executed. While a process is executing, it can be preempted by another whose laxity has decreased to below that of the running process. A problem arises with this scheme when two processes have similar laxities. One process will run for a short while and then get preempted by the other and vice versa. Thus, many context switches occur in the lifetime of the processes. The least laxity first algorithm is an optimal scheduling algorithm for systems with periodic real-time tasks. If each time a new ready task arrives, it is inserted into a queue of ready tasks, sorted by their laxities.

3 Maximum-Urgency-First scheduling algorithm (MUF)

David B. Stewart and Pradeep K. Khosla proposed this algorithm [7]. The maximum-urgency-first scheduling algorithm which we have developed is a combination of fixed and dynamic priority scheduling, also called **mixed priority** scheduling. With this algorithm, each task is given urgency. The urgency of a task is defined as a combination of two fixed priorities, and a dynamic priority. One of the fixed priorities, called the **criticality,** has higher precedence over the dynamic priority. The other fixed priority, which we call **user priority,** has lower precedence than the dynamic priority. The dynamic priority is inversely proportional to the laxity of a task.

The MUF algorithm consists of two parts. The first part is the assignment of the criticality and user priority, which is done apriori. The second part involves the actions of the MUF scheduler during run-time.

The steps in assigning the criticality and user priority are the following:

- As with RM, order the tasks from shortest period to longest period.

- Define the critical set as the first N tasks such that the total worst-case CPU utilization does not exceed 100%. Assign **high** criticality to all tasks in the critical set, and **low** criticality to all other tasks.
- Optionally assign a unique user priority to every task in the system.

The static priorities are defined once, and do not change during execution. The dynamic priority of each task is assigned at run-time, inversely proportional to the laxity of the task. Before its cycle, each task must specify its desired start time, deadline time, and worst-case execution time. Whenever a task is added to the ready queue, a reschedule operation is performed.

MUF scheduler is used to determine which task is to be selected for execution, using the following algorithm:

1). Select the task with the highest criticalness.

2). If two or more tasks share highest criticalness, then select the task with the highest dynamic priority (i.e. minimum laxity). Only tasks with pending deadlines have a non- zero dynamic priority. Tasks with no deadlines have a dynamic priority of zero.

3). If two or more tasks share highest criticalness, and have equal dynamic priority, then the task among them with the highest user priority is selected.

4). If there are still two or more tasks that share highest criticalness, dynamic priority, and highest user priority, then they are serviced in a first-come-first-serve manner.

4 Instantaneous utilization factor scheduling Algorithm

Radhakrishna Naik proposed this algorithm targets the soft real time systems. It schedules the periodic tasks. This algorithm is based on instantaneous Utilization Factor [IUF]. Instantaneous utilization Factor [IUF] is the processor utilization of a task at any time instant. Here for every task after one time quantum the instantaneous utilization of task is computed. The task having highest instantaneous utilization is given the highest priority and task is given to the processor. The quantum for which task is applied to CPU is Qi. The total sum of quantum of all task (for that quantum iteration only) is $\Sigma Qi=Q$. Here, the Periodic task cycle (PTC) is computed for a given task set which is the LCM of period of invocation of all tasks. After computation of initially the CPU utilization is computed. CPU Utilization is the ratio of initial computation time and its initial period. Depending on this initial utilization the task having highest initial utilization is mapped to the CPU. In a given PTC one task has executed for one time quantum. Now for calculating new instantaneous utilization factor gain the value of computation time and period is computed. For computing new computation time, the total sum of quantum of all tasks is subtracted from the previous computation time. In a similar way the new period is calculated. After this, the new instantaneous utilization is computed. Likewise the process will be repeated [6].

Algorithm for IUF:

7. Initially for given task set, calculate CPU utilization of each task using formula

$$U^i_0 = C^i_0 / P^i_0 \quad (1)$$

U^i_0 =Initial utilization of ith task.

C^i_0 =Initial computation time.

P^i_0 =Initial period of invocation.

Based on utilization [U^i_0] the task which is having higher value of Utilization is mapped for the CPU.

2): Now in a given PTC, one task has executed for one quantum of time. Again calculate value of C^i_1 , P^i_1 by using following formula-

$$C^i_1 = C^i_0 - Q_i \quad (2)$$

$$P^i_1 = P^i_0 - Q \quad (3)$$

Where $\Sigma qi=Q$.

Then calculate new Instantaneous Utilization factor for ith task for using formula 1 and 2

$$U^i_1 = C^i_1 / P^i_1. \quad (4)$$

Where,

C^i_1 =Instantaneous computation time for ith task

P^i_1= Instantaneous period of execution of ith task.

U^i_1=Instantaneous Utilization of ith task.

For second iteration of time derive the table (Ti, C^i_1, P^i_1, U^i_1)

Again the task which is having highest instantaneous utilization will be having highest priority of execution for second iteration quantum. Likewise, calculate

$$C^i_j = C^i_{j-1} - Q_i \quad (5)$$

$$P^i_j = P^i_{j-1} - Q \quad (6)$$

Calculate U using equation 4 and 5

$$U^i_j = C^i_j / P^i_j. \quad (7)$$

Where,

U^i_j =Instantaneous utilization of ith task for the jth iteration of quantum

j = PTC end point.

3): Hence task sequence in first PTC is derived. It is observed that at every step we can check whether the instantaneous utilization is less than initial utilization U^i_0. If at any given instant of time, it is observed that it is greater than U^i_0, it means that task is going to miss its deadline.

IV. Multiprocessor Scheduling Algorithms

1. Earliest Deadline first until Zero Laxity (EDZL)

Yi-Hsiung Chao proposed this algorithm [10]. EDF cannot guarantee all periodic tasks to meet their deadlines if the total utilization is greater than (m+1)/2 when the system has m processors. On the other hand, LLF has a good schedulability condition but has a high context switching overhead and causes a large number of preemptions.

Earliest Deadline first until Zero Laxity (EDZL) algorithm which combines the EDF and MLF algorithms. EDZL schedules jobs based on their deadlines and laxities. When all jobs have positive laxities, EDZL schedules jobs according to EDF. Whenever the laxity of a job becomes zero, EDZL schedules the job with the highest priority. Algorithm for EDZL

1). While true do
2). If there is any job with zero laxity
3). If there is an idle processor
4). Execute the zero-laxity job on the idle processor
5). Else if there is any task with positive laxity
6). Preempt the task with largest deadline
7). Else Output "Fail to Schedule"
8). Else Perform EDF
9). End if
10). End While

2. Improved Least Laxity First Scheduling Algorithm (ILLF)

H.S Behera, Satyajit Khuntia & Soumyashree Nayak proposes the Improved Least Laxity First Scheduling Algorithm [11]. This algorithm with intelligence time slice finds the time quantum by taking the greatest common divisor (GCD) of all the execution time of the processes. After every unit of time slice the laxity of each remaining process (present in the ready queue) is calculate. The loop is continued until all the processes are being executed by the CPU. Here as the GCD of execution the execution time is always greater than equal to 1 so loop will be continue for lesser no of time or same no of time Our proposed ILLF algorithm shows less context switching to least laxity First algorithm.

The algorithm is as follows:

1). Initialize all the processes in ready queue.
2). Initialize ready queue.
3). Calculation of the Time Quantum using GCD of the execution time of all the processes.
4). If execution time of process Pi is 0 then remove the process form ready queue and goto Step 5 otherwise calculate laxity time of the Pi and goto step 6.
5). If n==0 then stop and Exit otherwise goto step 4.
6). If i<n then increment i and goto 4 otherwise goto 7.
7). Sort P1 to Pn according to laxity in ascending order goto 8.
8). If (Lj==Lj+1) then goto 9; otherwise Execute Pj for
TQ time;
for (k=0;k<n; k++)
{Dk=Dk – TQ ;}
Ej=Ej-TQ; goto 4;
9). If (Ej>=Ej+1) then goto 9;
Otherwise {Execute Pj+1 till completion
for (k=0;k<n; k++)
{Dk=Dk-Ej+1; Ej+1=0;
goto 4 ;}
10). Execute Pj till completion
for (k=0; k<n; k++)
{Dk=Dk-Ej; Ej=0; goto 4 ;}

3. Modified Maximum Urgency First Scheduling Algorithm (MMUF)

V. Salmani proposes this algorithm [8]. The MMUF algorithm consists of two phases with the following details:

The algorithm is as follows:

Phase-1:- This phase defines the fixed priorities. These priorities should not be changed any further.
The steps are same as we discussed in MUF algorithm.
Phase-2:-This phase defines the dynamic priority. Dynamic priority is calculated at each clock cycle. Dynamic priorities are set according to EDZL algorithm which is explained in the following steps.
1) If there is only 1 critical task, schedule it at any free processor without pre-emption.
2) If more than 1 critical task is present in the ready queue, schedule the tasks with earliest deadline first (EDF) scheduling algorithm until there is a task with Zero laxity.
3) Laxity at each clock cycle is computed for all the remaining processes in the ready queue
4) If processes with zero laxity are available then these processes are assigned with higher priority over other process having non-zero laxity.
5) The process with zero laxity is scheduled at any available free processor without preemption.
6) If no free processor is available, and zero laxity process is present in the ready queue then preemption occurs. The process with the longest deadline is preempted and the zero laxity process is assigned to that processor.
7) After the execution of all zero laxity processes the remaining processes in the ready queue are assigned to the processor as per EDF scheduling policy.
The above steps are performed again for the non-critical task set.

4. Modified Instantaneous utilization factor scheduling Algorithm (MIUF)
Algorithm for MIUF is as follows [9]:
1)**:** Take input of tasks containing period, mandatory execution time, and optional execution time.
2)**:** Calculate the mandatory utilization and optional utilization for each task.
3): Check the schedulability for tasks according to their utilization.
4)**:** Generate CPU mapping for tasks.
5)**:** Execute the mandatory portion of tasks according to the highest instantaneous utilization. Meanwhile if there is interrupt due to corruption of task while executing the mandatory portion of task then a backup image is maintained so that the task can be restored from the image.
6): After executing mandatory portion of all tasks execute optional portion according the Shortest job first policy.
7)**:** If interrupt occurs due to corruption of task in the optional portion of task then optional portion does not run to its completion and gets aborted.

V. Comparison of Real Time scheduling algorithms

Before doing comparison let us see the definitions of all Performance Metrics:

1. Performance parameters of the Scheduling algorithms.

- **CPU Utilization**
  In the Multiprocessor Real time Operating System, CPU utilization is the parameter which is very important. All the processors in the system must be effectively utilized.
- **Deadline Miss Chances**
  Number of times the task may fail to execute to given time or to reach the deadline.
- **Number of context switching**
  It measures the number of preemptions of a task by a Higher Priority task.
- **Response Time**
  Response time of a task or thread is defined as the time elapsed between the dispatch (time when task is ready to execute) to the time when it finishes its job (One dispatch).
- **Effectiveness**
  Effectiveness of a Scheduler can be measured on any factor for example, Energy.
- **Predictability**
  Predictability for these systems should mean that we are able to satisfy the timing requirements of critical tasks with 100% guarantee over the life of the system, be able to assess overall system performance over various time frames (a stochastic evaluation), and be able to assess individual task and task group performance at different times and as a function f the current system state [14]. If all these assessments meet the timing requirements, then the system is predictable with respect to its timing requirements.
- **Optimality**

Scheduling algorithm is referred to as optimal if it can schedule all of the task sets that can be scheduled by any other algorithm, that is, all of the feasible task sets.

Table 1. Comparative Analysis of scheduling algorithms

Algorithms / Performance Metric	Uniprocessor Algorithms				Multiprocessor Algorithms			
Priority	EDF D	LLF D	MUF Hybrid	IUF D	EDZL D	ILLF D	MMUF Hybrid	MIUF D
CPU Utilization	High	High	High	High	High	High	High	High
No of Context Switching	Less	High	High	High	Very less	Less	Less	Less
Optimal	Yes	Yes	for critical tasks	Yes	No	Yes	Yes	Yes
Deadline miss chances	Average	Average	Less	Less	Less	Less	Less	Very Less
Response Time	High	Average	Low	High	Low	High	Average	Low
Predictability	Not Predictable	Not Predictable	Predictable under transient load	Dynamic predictability	More Predictable than EDF	More Predictable	Predictable under transient load	Dynamic predictability
Effectiveness	Optimal, Easy to implement	Takes execution time into cosideration	Work in transient overload	maximize utilization bound of schedule	Context switching overhead is low	Less context switching	Optimal for noncritical tasks	Improves context switching, response time and CPU utilization
Limitations	Not Work in overload, not optimal for pro.>1	In laxity tie, more context switches occurs	Non critical task may miss deadline	Context switching is very high	chances of deadline miss of the critical tasks	Excut-ion time is more	Only consider static utilization of task set	

VI. Conclusion

As we have seen, all uniprocessor and their extended versions for multiprocessor scheduling algorithms are priority based. The comparative analysis shows that In EDF context switching is high in overload condition whereas EDZL removes that drawback, in LLF When laxity occurs between two tasks context switching taken place is more that disadvantage is overcome in ILLF, whereas the way of assigning static priorities to the tasks is improved in MMUF, the Instantaneous Utilization Factor scheduling algorithm gives better result in uniprocessor scheduling and Modified Instantaneous Utilization Factor scheduling algorithm gives better context switching, response time and CPU utilization as compared to previous multiprocessor scheduling algorithms.

Acknowledgements

We wish to give our heartiest gratitude to Dr. Leena Ragha, Head of the Department of Computer Engineering, Ramrao Adik Institute of Technology and Prof. Vanita Mane, PG coordinator Department of Computer Engineering for her constant motivation, knowledge sharing and support behind this paper.

References

[1] Jashweeni Nandanwar, Urmila Shrawankar, "An Adaptive Real Time Task Scheduler", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, November 2012.

[2] H. S. Behera, Naziya Raffat, Minarva Mallik, **"** A Modified Maximum Urgency First Scheduling Algorithm with EDZL for Multiprocessors in Real Time Applications", Volume 2, Issue 4, April 2012.

[3] Jinkyu Leea, Arvind Easwaranb, Insik Shina,∗ , Insup Lee, "Zero-laxity based real- time multiprocessor scheduling", The Journal of Systems and Software, 2011.

[4] Francis Cottet,"Scheduling in Real Time Syatem", John W. Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2002.

[5] Arezou Mohammadi and Selim G. Akl, "Scheduling Algorithms for Real-Time Systems", Technical Report No. 2005-499, July 15, 2005.

[6] Yoo, Myungryun and M. Gen, "Study on Scheduling for Real- time Task by Hybrid Multiobjective Genetic Algorithm", Thesis, 2006.

[7] David B. Stewart, Pradeep Khosla, "Real-Time Scheduling of Sensor-Based Control Systems", 1991.

[8] Komal S. Bhalotiya, "Customised Multiprocesssor Scheduling Algorithms For Real time Systems", Proceedings published by International Journal of Computer Applications® (IJCA) ISSN: 0975 – 8887, April, 2012.

[9] Radhakrishna Naik, "Instantaneous Utilization Based Scheduling Algorithms for Real Time Systems", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (2), 654-662, 2011.

[10] Yi-Hsiung Chao, Shun-Shii Lin, Kwei-Jay Lin, "Schedulability issues for EDZL scheduling on real-Time multiprocessor systems", 2008.

[11] H.S Behera, Satyajit Khuntia & Soumyashree Nayak, "An Improved Least-Laxity-First Scheduling Algorithm For Real-Time Tasks", International Journal of Engineering Science and Technology (IJEST), Vol. 4 No.04 April 2012.

[12] G. Umarani Srikanth, A. P. Shanthi, V. Uma Maheswari, "A Survey on Real Time Task Scheduling", Europeans Journal of Scientific Research ISSN 1450-216X Vol.69 No.1 (2012), pp.33-41 © EuroJournals Publishing, Inc. 2012.
[13] Farooq Muhammad, "Ordonnancement de T^aches E_cace _aComplexit_e Ma^_tris_ee pouSyst_emesTempsR_eel' Pd.D. thesis, 2010.
[14] Robert I. Davis and Alan Burns, "A Survey of Hard Real- Time Scheduling for Multiprocessor Systems", ACM Computing Surveys, Vol. 43, No. 4, Article 35, Publication date October 2011.
[15] S.Behera, Naziya Raffat, Minarva Mallik, "Enhanced Maximum Urgency First Algorihm with Intelligent Laxity Real Time Systems", International Journal of Computer Applications (0975 – 8887), Volume 44– No.8, April 2012.