Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Modified Maximum Urgency First Scheduling Algorithm with EDZL for Multiprocessors in Real Time Applications

H. S. Behera, Naziya Raffat, Minarva Mallik
Department of Computer Science & Engineering,
Veer Surendra Sai University of Technology(VSSUT)
Burla, Sambalpur, Orissa, India,768018
nazraffat09@gmail.com

***Abstract*-- This manuscript proposes the implementation of Modified Maximum Urgency First (MMUF) algorithm in case of multiprocessors and uses EDZL (earliest deadline until zero laxity) algorithm as dynamic priority. Urgency based scheduling is an effective method of scheduling the resources which combines the advantages of both fixed and dynamic scheduling for improved performance of the CPU. In case of uniprocessor scheduling we have various algorithms which exploit various dimensions of urgency based scheduling like maximum urgency first scheduling, modified maximum urgency first scheduling etc. In modified maximum urgency first (MMUF) scheduling algorithm, earliest deadline first (EDF) policy is used as the dynamic priority. Over years EDF has evolved as an efficient and reliable dynamic scheduling policy for uniprocessors in real time systems but the same can't be stated true for scheduling of multiprocessors. On the contrary, it is observed that EDF is not optimal for multiprocessor scheduling. Hence, to resolve this, in the proposed algorithm EDZL has been used as the dynamic priority to schedule tasks for multiprocessors using MMUF.**

***Keywords* -- EDF, EDZL, multiprocessor scheduling, MMUF, real time system, ZL policy**

I. INTRODUCTION

Real time systems are required to provide results within a specific time frame. The correctness of the system behavior depends not only on the logical results of the computations but also on the physical instants at which these results are produced.

Scheduling of tasks in a multiprocessor system is also known as multiprocessor scheduling. Multiprocessor systems are used in parallel execution of tasks for effective scheduling and high performance. The multiprocessing scheduling is done to minimize total time of program execution and also maximize processors utilization at the same time. Multiprocessing strengthens multiprogramming.

Two important types of multiprocessor systems are there identical and uniform systems. In identical multiprocessing system the processing power of all processors is the same, whereas, in case of uniform the processing power varies from processor to processor [2]. In this paper we have considered identical multiprocessing system.

Two important parameters affecting the performance of multiprocessor scheduling algorithms are preemption and migration. A task is said to be preempted if the execution of the task is not completed on the processor it was running, rather it is assigned to some other processor [3]. Preemption occurs if a higher priority process arrives at the ready queue and demands the allotment of processor. If a task whose execution has been preempted, resumes its running on another processor, a migration has occurred [4].

There are two types of scheduling policies in multiprocessor environments named global scheduling and partition scheduling based on partition and migration of tasks. In global scheduling algorithm all the arrived tasks with non-zero remaining execution time are assigned to a queue that is common among the processors. In a system with *m* processors, in every moment, *m* tasks having the highest priorities should be executing considering preemptions and migrations, if necessary [5].

As the name suggests, partition scheduling algorithm divides the task set into partitions (subsets) such that all the tasks within a partition are assigned to a processor. In this policy task migrations are not allowed [5]

Scheduling is the method of allocating system resources to various threads, processes or data flows [6]. Scheduling is mainly divided into 2 types- static scheduling and dynamic scheduling [7]. In static scheduling, scheduling decision is taken at compile time. First come first serve, shortest remaining time next etc are examples of static scheduling. Dynamic scheduling makes scheduling decision at the execution time of the tasks or the processes and uses schedulability test to determine whether a set of tasks can meet their deadlines. Rate monotonic algorithm, earliest deadline first algorithm, least laxity first algorithm are few examples of dynamic scheduling.

Modified Maximum Urgency First scheduling algorithm has been proposed by V.Salmani et.al [1]. It combines the advantages of fixed and dynamic scheduling to provide dynamically changing systems with flexible scheduling.

Here in this paper we are proposing the implementation of Modified Maximum Urgency First (MMUF) algorithm in case of multiprocessors that uses EDZL (earliest deadline until zero laxity) algorithm as dynamic priority.

A. Preliminaries

A program in execution is called a process. CPU utilisation is the process of keeping the CPU busy with the useful work. Burst time of a process is defined as the worst case execution time of that particular process. In simple words it is the execution time of a process. Arrival time is the time at which

the process is invoked and arrives at the ready queue. Turnaround time is the time between submissions of a process to its completion. Waiting time is the amount of time a process spends in the ready queue. Laxity is the remaining time to complete a process. Response time is the amount of time it takes from when a request was submitted until the first response is produced. Throughput is the number of processes that completes their execution per unit time. The number of time a CPU switches from one process to another is the number of context switches [6].

B. CPU Load Factor- Multiprocessor Scheduling

Load factor of a task T_i or CPU utilization factor of a task T_i in a uniprocessor system is defined as the ratio of its worst case computation time C_i to its request period T_i [1]. The total utilization factor of a task set is the sum of the ratios of processing time to arrival periods of every task.
CPU utilisation for n periodic task is computed as
$U_n=\sum_{i=0}^{n} C_i/T_i <=1$

In case of multiprocessor scheduling $U_n<=m$, where m is the number of processors. For a platform comprising of m identical unit-capacity processors, utilization factor of the system can at most be equal to m. It has been previously shown that Usys ≤ m is a necessary condition for feasibility of the task system on m processors [9].

In this paper we present a brief overview of multiprocessor scheduling in real time system and simultaneously, we propose urgency based scheduling algorithm for multiprocessors. The rest of the paper is structured as follows. Section 1 describes the basic definitions and fundamental concepts of real time scheduling and multiprocessor scheduling. Section 2 portrays a survey of the related work done in this area and section 3 proposes our algorithm, section 4 concludes the paper and lastly section 5 contains the references used for the preparation of this manuscript.

II. PRIOR WORK

Dynamic scheduling means apart from making scheduling decisions for the tasks already present it also makes scheduling decisions for the new arrivals and for the tasks which arrive in the due course of scheduling. Dynamic algorithms produce schedules during execution time using appropriate and up-to-date information about the tasks and the environment. Since these algorithms perform on-line, they are supposed to be efficient and their sophistication affects the overall system's performance.

A. Earliest Deadline First Scheduling (EDF)

EDF algorithm [8] a task is assigned the highest priority if it is having the shortest deadline. The highest priority belongs to the task with the closest deadline while the task with the longest deadline has the lowest priority. Deadline of a task plays an important role in EDF scheduling. This algorithm has a schedulability of 100% and here $U_n <=1$ for all tasks.

B. Maximum Urgency First (MUF) Scheduling

Maximum Urgency First (MUF) scheduling algorithm resolves the problem of unpredictability of the system during transient overload that is when CPU load factor exceeds 100%. This algorithm is urgency based scheduling algorithm for uniprocessor system. It is a mixed priority scheduling algorithm and employs both fixed as well as dynamic priority for efficient scheduling of tasks. With this algorithm, each task is given an urgency which is defined as a combination of two fixed priorities (criticality and user priority) and a dynamic priority that is inversely proportional to the laxity. The critical priority is set to 1 if tasks are present in the critical set and the CPU load factor for these tasks is less than 1.
Critical priority > dynamic priority > user priority

C. Modified Maximum Urgency First (MMUF) Scheduling

To overcome the drawbacks of MUF[6], MMUF has been proposed. Modified maximum urgency first scheduling algorithm as proposed by V.Salmani et.al [1] is basically a slight modification in maximum urgency first (MUF)[6] scheduling algorithm. User priority is set in the beginning according to the importance of the tasks. Task with the highest importance are given user priority as 1 and task with the second highest priority is assigned user priority 2 and so on. After the user priority has been set first n tasks with CPU utilization less than 100% are allotted to the critical set and assigned critical priority as 1. Remaining tasks are allotted to the non-critical set and critical priority is 0 for these tasks. Unlike, MUF it is not always that the task with the shortest period is the most important one. Here EDF is used as the dynamic priority. Here number of context switches is reduced to a great extent resulting in an enhanced system performance.

User priority>critical priority> dynamic priority
The MMUF scheduling algorithm as proposed by V.Salmani et.al is as follows:
The MMUF algorithm consists of two phases with the following details:

In phase 1, fixed priorities are defined and the tasks are arranged in the decreasing order of their user priorities. First N tasks having CPU utilization<100% are taken in critical tasks and the remaining tasks are considered in non-critical task set.

In phase 2, dynamic priorities are calculated and accordingly the tasks are selected for execution. If there is only 1 critical task the task is executed. If more than 1 critical task is there, the task with the earliest deadline is picked up for execution. If there is more than 1 task with the same deadline then the task with the highest importance is considered and scheduled.

Once all the tasks present in the critical set are finished, the same set of steps are repeated for the tasks in the non-critical task set.

D. ZL Policy

The ZL (Zero-Laxity) policy [10] is exercised in the proposed algorithm for a multiprocessing environment. There are two important terms which needs to be studied in case of multiprocessing scheduling using ZL policy, these are remaining time to deadline of a job D(t) and remaining execution time of a job E(t). D(t) relates to urgency and E(t) is concerned with parallelism. Parallelism can be defined as the phenomenon of multiple executions of a same task on different

processors in the same time slice. ZL policy , as very much clear from the name itself is related to the scheduling of real time tasks in which tasks with negative or zero laxity are given preference over the remaining tasks. It assigns the highest priority to any zero or negative laxity job and then prioritizes the remaining jobs based on the policy of the original algorithm. The algorithm is priority driven, work-conserving and pre-emptive [10]. ZL policy has been used very effectively in the proposed algorithm to resolve the inefficiency of EDF when multiprocessor scheduling is concerned. Here, it is assumed that minimum or almost nil penalties is incurred if a task is preempted or a task is migrated from one processor to another. A base algorithm has been mooted by J.Lee et.al [10] to throw light on ZL policy. This algorithm is as follows. A employing the ZL policy is described as AZL where A is any random algorithm. That is, AZL assigns the highest priority to any zero or negative laxity job (with arbitrary tie-breaking), and then prioritizes the remaining jobs based on the policy of A. AZL is said to dominate A, if every task set that can be scheduled by A is also schedulable by AZL[14]. One such dominance has been put forward in EDZL and EDF (Park et al., 2005) [11], which clearly show the improved performance of a system while using EDZL over EDF. Thus, EDZL is used in the proposed algorithm since it outperforms EDF. Such influence has been justified by generalizing it into a theorem ie. AZL performs better than A where A is any algorithm [10].

E. Earliest Deadline First until zero laxity (EDZL) Scheduling

EDZL is a hybrid preemptive priority scheduling scheme in which jobs with zero laxity are given highest priority and other jobs are ranked by their respective deadlines. It has been proved in the paper proposed by T.P.Baker et.al [12] that a number of jobs missing their deadline are significantly reduced if scheduled by EDZL on m identical processors [12]. It was previously shown by Cho et al [13] that when EDZL is applied as a global scheduling algorithm for a platform with *m* identical processors its ability to meet deadlines is never worse than pure global EDF scheduling. EDZL is nothing but an extension of EDF. EDZL and EDF schedule the tasks in the same way until a zero laxity task has been encountered.

III. OUR CONTRIBUTION

In this paper MMUF scheduling algorithm is applied to the multiprocessor systems taking EDZL as dynamic priority. In MMUF scheduling policy as proposed by V.Salmani et.al both the kind of priority fixed as well as dynamic priority are considered. First phase in MMUF consists of assigning the fixed priority which is the user priority and the second phase is of assigning dynamic priority. EDF is used as dynamic priority in MMUF in uniprocessor system. In this proposed paper extension of MMUF scheduling in case of multiprocessors has been done. Since EDF is not optimal for multiprocessor scheduling [13] hence EDZL proposed by [12] based on zero laxity policy is used to set the task's dynamic priority in multiprocessing scheduling. Here if a process with zero laxity is encountered then it is scheduled first without preemption because at each time instant its laxity will be zero.

A. Proposed Algorithm

Phase-1:- This phase defines the fixed priorities. These priorities should not be changed any further.

1) Tasks are ordered in the increasing order of user priority that is from highest user priority to least user priority.
2) The first N task are added to the critical set such that the CPU load factor does not exeed 100%. And for the tasks present in the critical set critical priority is assigned 1 else critical priority is 0.
3) The tasks present in the critical set must be scheduled before the tasks in the non-critical set. This concept of scheduling, henceforth reduces the unpredictability of the system

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 tasks are 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 as laxity= deadline - (execution time + current clock cycle)
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.

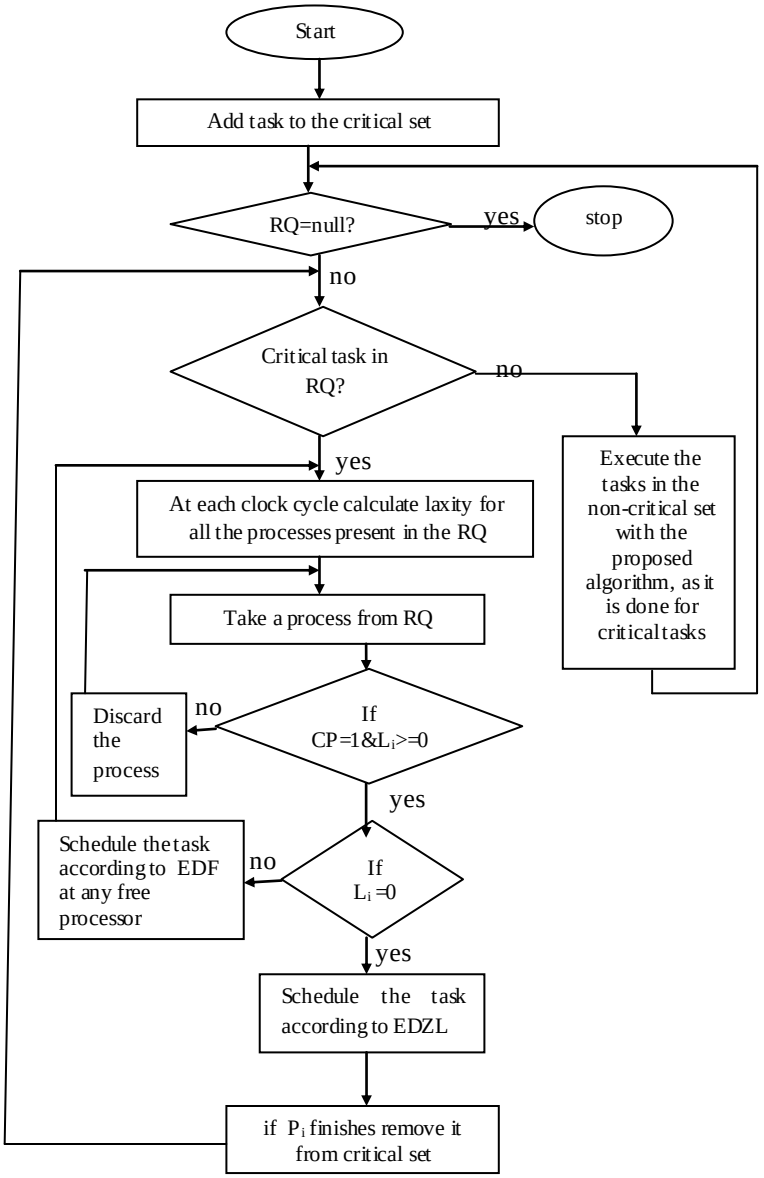
B. Pseudo Code of the Proposed Algorithm

1) n= no.of process
   P_i= process i
   E_i=execution time of the process i
   D_i= deadline of the process i
   AT_i= arrival time of the process i
   CP_i= critical priority of the process i
   UP_i= user priority of the process i
   L_i=laxity of the process i
   m= no. of processors
   Set avg TAT=0, avg WT =0, i=0
2) for(i=0; i<=n; i++)
   {
   If ((E_i/D_i-AT_i)<=m)
   {


```
            Assign Pi to the critical set
          }
        Else
            Assign Pi to the non-critical set
      }
3)  set CPi= 1 for the processes in the critical set
4)  set user priority according to the importance of the
    task
5)  at each clock cycle, calculate laxity is calculated for all
    the remaining process present in the ready queue(y)
6)  for(i=0;i<=y;i++)
    {
       Calculate Li
       If(Li<0)
          {
            Terminate the process from the ready queue
          }
       Else if(Li=0)
            x++;
    }
7)  if(x>0)
    {
      For(i=0;i<=n;i++)
        {
          If( Li =0)
           {
             Allot highest priority to Pi and execute it at
             any free processor
              if(all processors are busy)
                  {
                    Preempt the process with the longest
                    deadline from the processor in which it
                    was running and assign process with
                    zero laxity to that processor
                  }
            }
        }

    Remaining processes will be scheduled according to
    EDF
   }
8)     Else if(x==0)
       {
         Execute the process with EDF
       }
9)  If( x>m) deadline miss occurs for the process with
    least urgency or least priority
10) If(ready queue=! null)   goto step 5
```

C. Flow Chart of the Proposed algorithm



IV. EXPERIMENTAL ANALYSIS

A. Assumptions

All the experiments are performed in a multiprocessor environment and all the processes are independent. Attributes like execution time, user priority, absolute deadline, number of processes, and number of processors are known before submitting the processes to the processor.

B. Experimental Framework

The experiment consists of several input and output parameters. The input parameters consist of arrival time execution time, absolute deadline, critical task priority, user priority and the number of processes, the number of processors. The output parameters consist of average waiting time, average turnaround time and throughput.

C. Performance Analysis

The significance of our performance analysis for the proposed algorithm is as follows: With the implementation of MMUF [1] in multiprocessors, taking EDZL as the dynamic priority the chances of deadline miss of the critical tasks and the tasks which are most important for the users is very minimal and hence overall performance of the multiprocessing systems increases resulting in an all-round effective and successful scheduling of tasks. The average waiting time, average turnaround time and throughput are calculated for both the examples.

D. Example Implementation

EXAMPLE 1:

We assume ten processes to be executed in 3 identical processors. The processes having arrival time (P_1=2, P_2=4, P_3=4 ,P_4=4, P_5=6, P_6=6, P_7=6, P_8=8, P_9=8, P_{10}=9), & their execution time are (P_1=5, P_2=6, P_3=8, P_4=1, P_5=3,P_6=2, P_7=4, P_8=1, P_9=1, P_{10}=3) and critical task set={P_1, P_2, P_3, P_4} and deadlines (P_1=8, P_2=19, P_3=16, P_4=10, P_5=20 , P_6=21, P_7=16, P_8=15, P_9=15, P_{10}=16). Table.1 contains data to be used in our proposed algorithm. Table 3 shows the performance metrics for the proposed algorithm.

Giant charts are drawn for the proposed algorithm for the 2 identical processors and the gantt charts show how processors are allotted to different processes as per the proposed algorithm.

TABLE 1

Processes	Arrival time	Execution time	Absolute deadline	UP	CP
P_1	2	5	8	1	1
P_2	4	6	19	2	1
P_3	4	8	16	3	1
P_4	4	1	10	4	1
P_5	6	3	20	5	0
P_6	6	2	21	6	0
P_7	6	4	16	7	0
P_8	8	1	15	8	0
P_9	8	1	15	9	0
P_{10}	9	3	16	10	0

GANTT CHARTS OF SCHEDULING FOR THE PROPOSED ALGO IN MULTIPROCESSORS

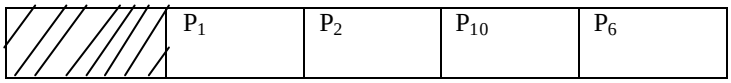


Fig 1: gantt chart for processor 1

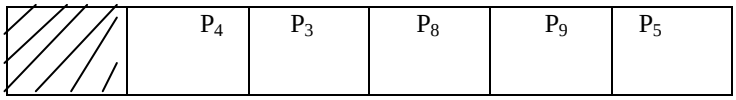


Fig 2: gantt chart for processor 2

EXAMPLE 2:

We assume ten processes to be executed in 3 identical processors. The processes having arrival time (P_1=2, P_2=4, P_3=5 ,P_4=6, P_5=6, P_6=7, P_7=7, P_8=7, P_9=8, P_{10}=8), & their execution time are (P_1=5, P_2=3, P_3=6, P_4=5, P_5=4,P_6=2, P_7=7, P_8=1, P_9=3, P_{10}=3) and critical task set={P_1, P_2, P_3, P_4} and deadlines (P_1=8, P_2=7, P_3=18, P_4=11, P_5=10 , P_6=18, P_7=16, P_8=8, P_9=12, P_{10}=14).

Table.2 contains data to be used in our proposed algorithm.

Giant charts are drawn for the proposed algorithm for the 3 identical processors and the gantt charts show how processors are allotted to different policies as per the proposed algorithm.

TABLE 2

Processes	Arrival time	Execution time	Absolute deadline	UP	CP
P_1	2	5	8	1	1
P_2	4	3	7	2	1
P_3	5	6	18	3	1
P_4	6	5	11	4	1
P_5	6	4	10	5	0
P_6	7	2	18	6	0
P_7	7	7	16	7	0
P_8	7	1	8	8	0
P_9	8	3	12	9	0
P_{10}	8	3	14	10	0

GANTT CHARTS OF SCHEDULING FOR THE PROPOSED ALGO IN MULTIPROCESSORS

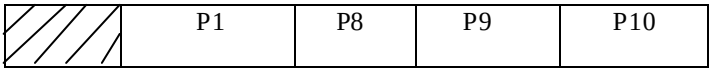


Fig 3: gantt chart for processor 1

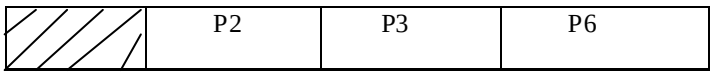


Fig 4: gantt chart for processor 2

For processor 3:-

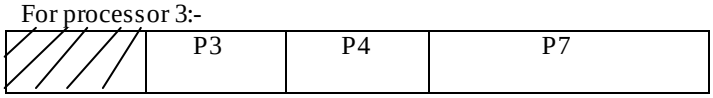


Fig 5: gantt chart for processor 3

TABLE 3: PERFORMANCE METRICS FOR THE PROPOSED ALGORITHM

Example	throughput	Avg. waiting time	Avg. turnaround time
1	0.5	3.8	6.8
2	0.667	1.3	4.8

V. CONCLUSION

In the paper we have proposed urgency based multiprocessor scheduling algorithm for real time systems which extends the MMUF scheduling algorithm proposed by V. Salmani et.al and uses EDZL algorithm as dynamic priority in MMUF scheduling algorithm. From the experimental analysis we conclude that the proposed algorithm performs effectively for multiprocessor system in real time sets. In due course of preparing this manuscript we have realized that this algorithm can be implemented in real life practical examples to enhance the overall performance of a multiprocessor system. Further extensions of this algorithm can be developed in future and can be used to efficiently schedule multiprocessors. Moreover, in coming years this proposed algorithm can also be compared with other multiprocessing scheduling algorithms.

ACKNOWLEDGEMENT

The authors thank the department of computer science and engineering, VSSUT burla, sambalpur for their kind support and the faculties of the department for their guidance which led us successfully towards the preparation of this manuscript.

REFERENCES

[1] V.Salmani, S.zargar and M. Naghibzadeh- a modified maximum urgency first scheduling algorithm for real time tasks- world academy of science and technology 9 2005

[2] J.Goossens, Universit´e Libre de Bruxelles and P. Richard, Laboratoire d'Informatique Scientifiq et Industrielle ENSMA (France), "Overview of real-time scheduling problems"

[3] B. Andersson, and J. Jonsson, Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition, 7th International Conference on Real-Time Computing Systems and Applications (RTCSA'2000), Cheju Island, South Korea, December 12-14, 2000, pp. 337-346.

[4] A. Mohammadi, and S. G. Akl, Scheduling Algorithms for Real-Time Systems. Technical Report, Queen's University, 2005.

[5] S. Funk, J. Goossens, and S. Baruah, On-line scheduling on uniform multiprocessors, the 22nd IEEE Real-Time Systems Symposium(RTSS 2001), London, UK, December 2-6,2001, pp. 183-192.

[6] Baker, T.P., Cirinei, M., 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 178–190

[7] A. Mok. "Fundamental Design Problems of Distributed Systems for Hard Real-time Environments". PhD thesis, Massachusetts Institute ofTechnology, Cambridge, MA, 1983Silberschatz, A., P.B. Galvin and G. Gagne, Operating Systems Concepts. 7th Edn. John Wiley and Sons, USA

[8] C. L. Liu, and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a hard real Time Environment, "Journal of the Association for Computing Machinery, vol.20, no.1, pp. 44-61, January1973.

[9] D. B. Stewart, and P. k. Khosla , "Real-Time Scheduling of Sensor- Based Control Systems", in Proc. Eighth IEEE Workshop on Real-Time Operating Systems and Software, in conjunction with 17th IFAC/IFIP Workshop on Real-Time Programming, Atlanta, GA, May 1991, pp.144-150.

[10] Jinkyu Lee, Arvind Easwaran, Insik Shin, Insup Lee, 2011. "Zero-Laxity based Real-Time Multiprocessor Scheduling", Journal of Parallel and distributed computing, 84, pp. 2324-2333

[11] Park, M., Han, S., Kim, H., Cho, S., Cho, Y., 2005. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. IEICE Transaction on Information and Systems E88-D, 658–661.

[12] T.P.Baker, M. Cirinei, M. Bertogna, "EDZL scheduling analysis". Real-Time Systems. 40:3, 264-289, 2008

[13] Cho S, Lee S-K, Han A, Lin K-J (2002) Efficient real-time scheduling algorithms for multiprocessor systems. IEICE Trans Commun E 85-B(12):2859–2867

