2010 International Conference on Complex, Intelligent and Software Intensive Systems

Least Slack Time Rate first: New Scheduling Algorithm for Multi-Processor Environment

Myunggwon Hwang
Dept. of Computer Engineering
Chosun Univ.,
Gwangju, South Korea
mg.hwang@gmail.com

Pankoo Kim
Dept. of Computer Engineering,
Chosun Univ.,
Gwangju, South Korea
pkkim@chosun.ac.kr

Dongjin Choi
Dept. of Computer Engineering
Chosun Univ.,
Gwangju, South Korea
Dongjin.Choi84@gmail.com

***Abstract*— Real-time systems have to complete the execution of a task within the predetermined time while ensuring that the execution results are logically correct. Such systems require scheduling methods that can adequately distribute the given tasks to a processor. Scheduling methods that all tasks can be executed within a predetermined deadline are called an optimal scheduling. In this paper, we propose a new and simple scheduling algorithm (LSTR: least slack time rate first) as a dynamic-priority algorithm for a multi-processor environment and demonstrate its optimal possibility through various tests.**

Keywords-multi-processor scheduling, scheduling algorithm, optimal scheduling, least slack time rate, LSTR

I. Introduction

Real time systems not only complete tasks within the predetermined time but also obtain logically correct results. In addition to providing these functions, such systems also have to satisfy basic requirements such as reliability, predictability, and adaptability in single-processor and multi-processor environments. [2] According to the extent to which the value of an executed result satisfies the requirements, systems are categorized as a hard or a soft real-time system. A hard real-time system has to complete all given tasks within each predetermined deadline, otherwise some of the results would be useless and a serious disaster could occur. In a soft real-time system, the faster the finish time is, the greater the usefulness of the system is. Even if the execution of the task is completed after its deadline, its result is still useful.

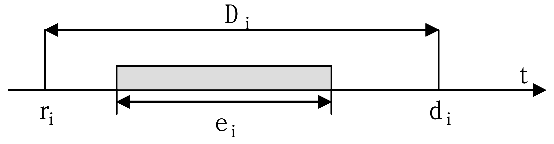

Figure 1. Timing constraints: release time, relative deadline, absolute deadline, and execution time.

All tasks in real-time systems have timing constraints that basically comprises the release time, relative deadline, absolute deadline, and execution time, as shown in figure 1. The release time (r_i) represents the time when a task arrives in a ready queue for execution. The relative deadline (D_i, *or* d_i-r_i) is the maximum amount of time within which a task should be completed. The absolute deadline (d_i, *or* D_i+r_i) is the time within which the execution of a task should be completed. Finally, the execution time (e_i) represents the amount of time (theoretical minimum execution time) required for the task process.

Many tasks occur simultaneously and each has its own timing constraints. For satisfying these constraints, especially task deadline, systems require scheduling methods that suitably distribute tasks to a processor. They are classified into the dynamic and the static (fixed) priority scheduling algorithms according to the scheduling method used. Dynamic priority scheduling algorithms such as EDF (Earliest Deadline First) and LST (Least Slack Time First) assign different priorities to an individual job in each task. On the other hand, static priority scheduling algorithms such as RM (Rate Monotonic) and DM (Deadline Monotonic) assign the same priority to all jobs in each task [1-3].

A scheduling algorithm that can satisfy all the predetermined deadlines of each task is called an optimal scheduling algorithm. In this paper, we propose a simple and new dynamic priority scheduling algorithm for a multi-processor environment and demonstrate its possibility through various tests. We call our algorithm Least Slack Time Rate first (LSTR). The algorithm can feasibly assign all of tasks into processor(s). This significant method is a dynamic priority scheduling algorithm and is useful for hard real time system.

The second chapter describes the background of this research such as EDF, LRT, and LST. In the third chapter, we indicate the limitations of existing algorithms and then propose our scheduling algorithm. Further, we show the

978-0-7695-3967-6/10 $26.00 © 2010 IEEE
DOI 10.1109/CISIS.2010.20

experimental results and performance evaluation in the fourth chapter. Finally, we summarize our research.

II. Background

We have studied the definitions and specific features of existing scheduling algorithms known as a dynamic-scheduling method. Based on these studies, we could design LSTR algorithm.

A. *Earliest Deadline First (EDF)*

EDF is a very simple and famous algorithm that the earlier the deadline is, the higher the priority is; the scheduler operates when each processor task is completed or when a task wakes up in the ready queue. Let us assume three tasks with time constraints in a ready queue, as shown in table 1. Each task is scheduled divided into several jobs. According to the timing constraints listed in table 1, the scheduling process is carried out like table 2.

Table 1. Three tasks with timing constraints

Task	r_i	D_i	e_i
T1	0	2	1
T2	0	4	1
T3	0	8	2

r_i : Release time, D_i : relative deadline, e_i: execution time, the tasks are periodic and the deadline coincides with the period.

Table 2. Scheduling process using EDF algorithm

t	0	1	2	3	4	5	6	7
T1	**2**	-	**2**	-	**2**	-	**2**	-
T2	4	**3**	-	-	4	**3**	-	-
T3	8	7	6	**5**	4	3	2	**1**

The three tasks are released at the initial time ($t = 0$). The scheduler attempts to compare the deadlines of each task and then assigns the highest priority to T1 because its deadline is the smallest as value 2. Therefore, a job (J_1) of T1 is executed on the processor. At $t = 1$, the scheduler wakes up since T1 has just completed. The scheduler then determines that T2 has the highest priority. When $t = 2$, T1 is awakened due to its period, and between T1 and T3, the scheduler selects T1 (T2 is already finished when $t = 2$). T3 is then processed by the scheduler for the first time when $t = 4$. The three tasks are processed while each timing constraint is satisfied through iterative operations. In case of that the remaining deadlines are equal, such as T2 and T3 at time 5 or T1 and T3 at time 6, the task having the first index is processed by the tie-break rule that selects the first index or the processed jobs at the last processing time. Therefore, the EDF scheduling algorithm can be considered to be an optimal algorithm only for a single processor.

It is possible to determine whether given tasks are schedulable or not by measuring the processor utilization of timing constraints of given tasks without simulations. The utilization is measured by (1).

$$\Delta u = \sum_{i=1}^{n} \frac{e_i}{D_i} \leq 1 \qquad (1)$$

Equation (1) measures the processor utilization, where Δu denotes the utilization; D_i, the relative deadline; e_i, the execution time of each task; and n, the total count of tasks. If the result of some given tasks as obtained by (1) is 1, the processor utilization is 100%. In other words, the processor should not have an idle state. For example, the utilization of the tasks listed in the table 1 is 1 (1/2 + 1/4 + 2/8 = 1). Therefore, we can assume that these tasks are schedulable.

B. *Latest Release Time First (LRT)*

LRT determines that the later the release time is, the higher the priority is. This is the opposite of EDF. In other words, LRT considers the release time and relative deadlines and schedules jobs in the reverse order, starting from the latest deadline among all jobs. If LRT schedules the same task of table 1 then, the result is same to table 2 scheduled by EDF. This algorithm is also optimal on the condition of satisfying (1).

C. *Least Slack Time First (LST)*

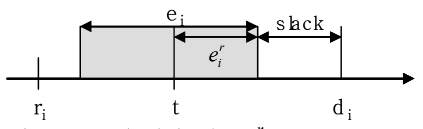

Figure 2. Slack is d_i - e_i^r - t

LST algorithm follows the smaller slack time, the higher priority. The slack time means the remaining spare time (d_i - e_i^r - t) at the absolute deadline. e_i^r represents the time that is required to complete the remaining portion of a job as shown figure 2. LST algorithm can schedule all tasks on condition of satisfying (1). And, it can be considered to be an optimal algorithm for a single processor.

III. Main Discourse

In this chapter, we analyze the existing scheduling algorithms described in the background chapter and indicate their limitations in a multi-processor environment. We then propose a scheduling algorithm that can show optimal possibility for both uni- and multi-processor environments.

A. *Limitations of existing algorithms*

Algorithms described in the previous chapter are optimal in uni-processor environment, but not in multi-processor ones. Table 4 lists the scheduling results on three processors using EDF according to timing constraints listed in table 3. For the optimal scheduling, the utilization of each processor should be 1, but processor 3 is idle at time (t) 1, 3, and 5. In case of using LST, it also leaves some processors idle. Therefore, missing deadline occurs.

Table 3. Five tasks with timing constraints

Task	r_i	D_i	e_i
T1	0	2	1
T2	0	2	1
T3	0	2	1
T4	0	8	6
T5	0	8	6

r_i : Release time, D_i : relative deadline, e_i: execution time, the tasks are periodic and the deadline coincides with the period.

Table 4. Scheduling process using EDF and LRT

t		0	1	2	3	4	5	6	7
	T1	**2**	-	**2**	-	**2**	-	**2**	-
	T2	**2**	-	**2**	-	**2**	-	2	**1**
TASK	T3	**2**	-	**2**	-	**2**	-	2	1
	T4	8	**7**	6	**5**	4	**3**	**2**	**1**
	T5	8	**7**	6	**5**	4	**3**	**2**	**1**
PRO-	P1	J1	J4	J1	J4	J1	J4	J4	J4
CE-	P2	J2	J5	J2	J5	J2	J5	J5	J5
SSOR	P3	J3	idle	J3	idle	J3	idle	J1	J2

Ji: job of ith task (Ti), Pi: ith processor

Missing deadlines frequently occur because the scheduling algorithms depend on the release time or slack time of given tasks. Further, scheduling based on these factors cannot exactly determine which task should be first executed. Therefore we suggest new scheduling algorithm that can assign all of jobs on processor(s) without idle state.

B. *Least Slack Time Rate first (LSTR)*

1) Prerequisites: LSTR scheduling algorithm has the same prerequisites as those in existing dynamic scheduling algorithms for real-time systems. These prerequisites are listed in table 5. The first condition is preemptive scheduling. It means that the execution of jobs can be interleaved. The scheduler suspends the execution of a less urgent job and gives processor control to a more urgent job. Later, when the urgent job has executed, the scheduler returns processor control to the previous job. The second prerequisite is migration. In a multi-processor environment, jobs that are ready for execution are placed in a priority queue. When a processor is available, the job at the head of the queue executes on the processor. In other words, an available processor can immediately execute any job in ready queue. And, we assume that all the tasks are periodic and the release time is 0 for well-understood behavior of the algorithm. The last prerequisite is that all timing constraints have to satisfy (2).

$$\sum_{i=1}^{n}\frac{e_i}{D_i}\le n_c, \qquad \frac{e_i}{D_i}\le 1 \tag{2}$$

Equation (2) represents the ranges of timing constraints. n_c denotes the number of processors. The total processor

Table 5. Prerequisites of the scheduling algorithm

CONDITION	PREQUISITE
Preemption	Accept
Migration	Accept
Periodic Task	Only
Release time	0
Timing constraints	Tasks should satisfy (2)

utilization of a given set of tasks cannot be higher than the number of processors and the processor utilization of each task is lower than 1. If the utilization of a task equals to 1, it implies that the task should occupy a processor till completion. In addition, if the total utilization equals to the number of processors, optimal scheduling is possible if and only if all processors work without an idle state.

The five prerequisites mentioned above are fundamental for the dynamic scheduling algorithm in a real-time system; the scheduling algorithm described in this paper also depends on these prerequisites.

2) Scheduling Time: Every scheduling algorithm has a scheduling time that is used by a scheduler to determine which task should be first. Generally, a scheduler operates when tasks are released, when the execution of a task is completed, or when a task is awakened in a ready queue (when the time for its execution has arrived). In case of the timing constraints of tasks given in tables 1 and 3, a scheduler operates in accordance with these rules. A scheduler differently operates according to the timing constraints of a given set of tasks and scheduling algorithms. Moreover, in the case of the Round-Robin scheduling algorithm[1], either the users or the systems make a decision with regard to the scheduling time [4].

The LSTR scheduling algorithm for efficient scheduling has an additional condition with regard to scheduling time. We defined maximum occupation time (MOT) which limits the maximum time which one job (of a task) can occupy one processor. The MOT is determined by the timing constraints of a given task set. Further, the scheduler for LSTR starts operating when a task on a processor completes its execution time or when a task is awakened in a ready queue (the period of a task arrives). In other words, this algorithm works in the same manner as the existing scheduling algorithm, but the processing time which is occupied by one job cannot exceed the MOT. The MOT can be obtained as follow:

$$MOT = \underset{i\in task\_set}{\arg}\ \min\left(D_i - e_i\right) \tag{3}$$

where, i denotes an index of the given tasks. For example, the MOT is 1 when using the timing constraints shown in table 3; the MOT is resulted by T1 because it has the minimum value. Equation (3) plays a core role of making a decision of scheduling time of LSTR in the coming chapter.

[1] Round-robin Scheduling, http://en.wikipedia.org/wiki/Round-robin_scheduling

3) Scheduler: The LSTR scheduling algorithm measures the rate of the execution time of each task at every scheduling time. The aim of LSTR is not to allow idle state in any processor. All tasks have the deadline and execution time as a timing constraint. The scheduler determines the task at the scheduling time to be executed on a processor. Tasks are executed on the processor(s) and then both the remaining execution time and the remaining deadline of these tasks decrease. However, the remaining execution times of other tasks do not decrease. This explains the limitations of existing scheduling algorithms such as EDF, LRT, and LST which use only the deadline, release time, or slack time. Therefore, these algorithms cannot determine a task which is really urgent and once the scheduler determines the highest priority, others cannot execute for some fixed time which the task is executed. This causes some processors to have an idle state. On the other hand, the LSTR scheduling algorithm determines the priority by computing the rate between the remaining execution time and the remaining deadline in order to satisfy all given timing constraints of tasks. Equation (4) presents this scheduler. A higher rate is a higher priority is.

$$rate(task_i) = \frac{e_i^r}{d_i - t} \qquad (4)$$

Although this is a simple equation, it can schedule tasks without idle states of processors. Here, t denotes the current time; e_i^r, the remaining execution time of *i-th* task at t; and d_i, the remaining absolute deadline of *i-th* task at t. The LSTR algorithm is similar to the LST algorithm which uses the slack time; the difference is that the LSTR algorithm uses the rate of the remaining execution time and the remaining deadline. This is why we named this algorithm least slack time rate first. As described in the previous chapter, the scheduling operates when the tasks are released, when the current running job is completed, and when the period of tasks arrives. Moreover, each job of tasks cannot excess MOT calculated by (3). Under these conditions, LSTR returned the optimal scheduling result in both uni- and multi-processor environments. As an example, table 6 shows the scheduling results under the condition of table 4 (five tasks) for multi-processor environment (three processors). As shown in the tables 6, no processor has an idle state.

Table 6. Scheduling process for multi-processor environment

t		0	1	2	3	4	5	6	7
TASK	T1	**1/2**	-	**1/2**	-	**1/2**	**-**	**1/2**	-
	T2	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**	**1/2**	**-**
	T3	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**	1/2	**1/1**
	T4	**6/8**	**5/7**	**4/6**	3/5	**3/4**	**2/3**	1/2	**1/1**
	T5	**6/8**	5/7	**5/6**	**4/5**	**3/4**	2/3	**2/2**	**1/1**
PRO-CE-SSOR	P1	J4	J2	J5	J2	J4	J2	J5	J3
	P2	J5	J3	J4	J3	J5	J3	J1	J4
	P3	J1	J4	J1	J5	J1	J4	J2	J5

Table 7. Pairs of processors and tasks

Number of Processors	Number of Tasks			
	3	5	7	9
2	480	480	480	480
3	-	480	480	480
4	-	480	480	480
5	-	-	480	480

IV. Experiment and Performance Evaluation

In the previous chapter, we presented the successful scheduling results of the given tasks using the LSTR algorithm. In order to verify the optimal possibility, we implemented a simulation system and tested using total 7,680 task sets. Further, in order to measure the amount of time loss in using the LSTR algorithm, we evaluated the performance by comparing it with the EDF algorithm.

A. Experiment

We developed a random number creator that creates timing constraints for tasks. The creator randomly selects a relative deadline in the range from 2 to 16 because an integer overflow often occurs in the simulation program at the time in which all tasks are concurrently completed (LCM (Least

Table 8. Scheduling process for four processors and nine tasks

Task count	Timing Constraints (MOT = 1)				Utilization
	Task	Release Time	Deadline	Execution Time	
9	T1	0	15	8	4.0
	T2	0	6	5	
	T3	0	10	4	
	T4	0	4	3	
	T5	0	4	2	
	T6	0	3	1	
	T7	0	3	1	
	T8	0	5	1	
	T9	0	60	7	

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…
P1	J2	J2	J7	J5	J4	J6	J2	J2	J2	J8	J2	J7	J2	J2	J7	J2	J4	…
P2	J4	J4	J2	J2	J2	J7	J3	J1	J4	J2	J4	J1	J4	J4	J2	J1	J1	…
P3	J1	J1	J3	J1	J3	J4	J4	J6	J3	J4	J5	J3	J5	J1	J4	J3	J2	…
P4	J5	J6	J4	J8	J5	J1	J5	J7	J5	J1	J6	J8	J3	J6	J5	J6	J5	…

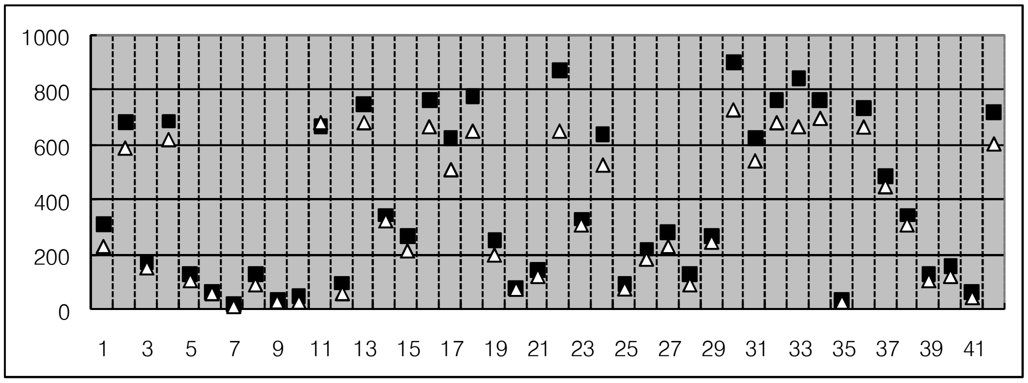

Figure 3. Performance evaluation of scheduling time (42 sets of five tasks).
Black rectangles mean the results by LSTR while white triangles are EDF. X axis is an index of task set while Y axis is elapsed time

Common Multiple) of the task deadlines) if the upper limit is more than 16. Further, we set the range of utilization from *0.96•n* to *n*, where *n* denotes the number of processors. If *n* is 1, it means uni-processor environment, otherwise multi-processor environment.

Under these conditions, we have performed many experiments for demonstrating the optimal possibility of LSTR. For proving the possibility in a uni-processor environment, we used task counts of 3, 5, 7, and 9 and performed each experiment 480 times. Total 1,920 experiments were performed for a single processor. Through this experiment, we obtained the successful results while satisfying all deadlines of each task set. Further, for a multi-processor environment, we simulated several cases under the environment of different processor counts and task counts, as listed in table 7. The processor count ranges from 2 to 5. We also used task counts of 3, 5, 7, and 9 in the case of two processors; 5, 7, and 9 in the case of three and four processors; and 7 and 9 in the case of five processors. And, we have experimented 480 times for each processor-task set. The experiment was performed total 5,760 times. All experiments have shown that LSTR satisfied each deadline for a task set. Table 8 shows that for the pair (4, 9). In this scheduling process, the MOT value is 1, as determined by T4. The completion time within which all tasks are finished is 60 which is the LCM of relative deadlines. In this paper, we presented the scheduling results until t = 16 because of the size limit. LSTR which does not allow a processor to have an idle state shows optimal possibility for both uni- and multi-processor environments, as verified by performing the experiments 7,680 times.

B. *Performance Evaluation*

In the previous chapter, we demonstrated the optimal possibility of LSTR algorithm. However, this algorithm would have a scheduling time greater than that of other algorithms such as EDF, LRT, and LST. Schedulers of other scheduling algorithm wake up when jobs are released, a job on the processor is completed or a task of higher priority wakes up; in case of LSTR, however, scheduler wakes up at a time determined both by existing way and by (3). Therefore, we need to know how much time is wasted by the LSTR. We have attempted to evaluate the performance by comparing LSTR with EDF because this is one of the simplest and the fastest. However, it is not optimal algorithm for multi-processor environment. So, we evaluated the difference of time loss under a single-processor environment using three and five tasks.

The performance was evaluated on a computer with a 2 GHz CPU running the Windows XP operating system and using JDK 1.6 for Java programming. In Java, a measurable time unit is 1 ms (0.001 s); however, the scheduling time for one task set is lesser than 1 ms. Therefore, we need to iterate each task set 1,000 times to obtain appropriate values. We prepared 300 task sets that comprised 150 sets of three tasks and 150 sets of five tasks. Using these sets, both simulators of the EDF and LSTR operated under the same conditions.

The results of the performance evaluation are shown in figure 3, where the x-axis represents the index of a task set and the y-axis represents the time required (*ms*) for scheduling. The figure shows the results of 42 task sets comprised of five tasks. The scheduling time of LSTR is slightly higher than that of EDF, as indicated in table 9, which presents an analysis of this performance evaluation. In the case of three tasks, (A) has an excess pair of (220 *ms*, 10), (B) has (47, 3), and (C) has (109, –1). In the case of (C), the excess count of LSTR is less than that of EDF, although the total excess time is more. Further, in the case of five tasks, (A) has an excess pair of (2,150, 35), (B) has (2,210, 37), and (C) has (1,685, 31). From the result, the excess count and the total excess time are found to increase depending on the task count. However, these results have been obtained from 1,000 iterations of 50 task sets. It means, for computing the excess time of one task set, it should be divided by 50,000. In the case of five tasks and group (B), which wastes the most excess time, the average excess time is measured to be 0.0442 *ms* which is very small. In figure 3, each actual scheduling time should be divided by 1,000. For example, consider the result having the highest scheduling time (30th

Table 9. Performance evaluation of scheduling time

Total Excess = LSTR - EDF		Group		
		(A) 50 sets	(B) 50 sets	(C) 50 sets
3 Tasks	Total Excess (ms)	220	47	109
	Excess Count	10	3	-1
5 Tasks	Total Excess (ms)	2150	2210	1685
	Excess Count	35	37	31

Table 10. Conditions of performance evaluation on time loss

Number of Processors	Task Count (iteration x tasks set) 3	4	5	6	7
2	1000x50	1000x50	1000x50	-	-
3	-	1000x50	1000x50	1000x50	-
4	-	-	1000x50	1000x50	1000x50

in the figure); the scheduling time of LSTR is 906 *ms* and that of EDF is 734 *ms*. The difference is merely 0.172 *ms* ((906-734)/1,000).

These results indicate that LSTR waste slightly higher scheduling time than that of EDF but, the LSTR is valuable because it shows the possibility of optimal scheduling for multi-processor environment.

In addition to the evaluation, in order to check the time loss by the LSTR algorithm in multi-processor environment, we measure the total amount of time loss based on the conditions given in table 10. Each item in the table is an iterated count and a count of task sets. Table 11 lists the average time of each item in Table 10.

Table 11 shows the maximum time loss of one task set iterated 1,000 times, the total amount of time required for completing 50 task sets iterated 1,000 times, and the average time loss of one task set iterated once. From this result, we could observe that the processor count does not have a significant effect on the time loss. For example, the task counts for LSTR(2, 5), LSTR(3, 5), and LSTR(4, 5) are the same, but the processor counts are not. The result obtained using these conditions shows that LSTR(4, 5) wasted the most time as 953(ms) in side of maximum time loss, but in cases of both sum and average surprisingly LSTR(2, 5) is the biggest as 12.828 and 0.25656 respectively. Further, we could observe that the time loss is mainly affected by task count.

The result of this chapter shows that the LSTR scheduling algorithm can have optimal possibility for multi-processor environment. Further, from the performance evaluation checking the time loss, we could confirm that the LSTR takes a slightly longer time than that of the EDF algorithm which is one of the simplest and fastest scheduling algorithms. We also observed that the amount of time loss depends more on the number of tasks to be scheduled. Hence, the time loss in a multi-processor environment can be similar to that in a single-processor environment. Therefore, the LSTR algorithm is an extremely valuable and novel scheduling algorithm.

V. Conclusion

In this paper, we proposed the LSTR scheduling algorithm; it can complete all tasks within the deadlines for real-time system. Through several experiments with various task-processor sets, we have demonstrated that LSTR could show the optimal possibility for both uni- and multi-processor environments. From performance evaluation, we could observe that its time loss is slightly longer than that of EDF. We strongly believe that the LSTR will be wildly useful, but, for the optimal multi-processor scheduling, a complexity, an acceptance test, or a theoretic verification should be additionally studied. These are remained as future works.

Acknowledgment

Following are results of a study on the "Human Resource Development Center for Economic Region Leading Industry" Project, supported by the Ministry of Education, Science & Technology(MEST) and the National Research Foundation of Korea(NRF).

References

[1] Labrosse, J.J, MicroC OS-II: The Real-Time Kernel (Second Edition), CMPBOOKS, Manhasset NY, 2002.

[2] Liu, J.W.S., Real-Time System, Printice Hall, New Jersey, 2000.

[3] Stallings, W., Operating Systems: Internals and Design Principles (fifth international edition), Printice Hall, New Jersey, 2004.

[4] J. Liu and E.A. Lee, “Timed Multitasking for Real-Time Embedded Software,” IEEE Control System Magazine, Feb., 2003, pp 65 ~ 75, doi: 10.1109/MCS.2003.1172830

Table 11. Results of performance evaluation on time loss

	LSTR (2,3)	LSTR (2,4)	LSTR (2,5)	LSTR (3,4)	LSTR (3,5)	LSTR (3,6)	LSTR (4,5)	LSTR (4,6)	LSTR (4,7)
Maximum Time Loose (ms)	63	390	703	516	860	1188	953	1546	1766
Sum (ms)	562	3343	12828	4251	11376	25344	11922	30797	35704
Average (ms) (Sum / (Set Count x Iteration Count))	0.01124	0.06686	0.25656	0.08502	0.22752	0.50688	0.23844	0.61594	0.71408