A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks ⋆

Sung-Heun Oh Seung-Min Yang
School of Computing, Soongsil University
1-1 Sangdo-dong, Dongjak-ku, Seoul 156-743, KOREA
honestly@realtime.soongsil.ac.kr yang@computing.soongsil.ac.kr

Abstract

The Least-Laxity-First(LLF) scheduling algorithm assigns higher priority to a task with the least laxity, and has been proved to be optimal for a uniprocessor systems. The algorithm, however, is impractical to implement because laxity tie results in the frequent context switches among the tasks. The Modified Least-Laxity-First (MLLF) scheduling algorithm proposed in this paper solves the problem of the LLF scheduling algorithm by reducing the number of context switches significantly. By reducing the system overhead due to unnecessary context switches, the MLLF scheduling algorithm avoids the degradation of system performance and conserves more system resources for unanticipated aperiodic tasks.

In this paper, we propose the MLLF scheduling algorithm and prove its optimality. We show the performance enhancement of the proposed MLLF scheduling algorithm by using simulation results.

1. Introduction

The laxity of a real-time task T_i at time t, $L_i(t)$, is defined as follows:

$$L_i(t) = D_i(t) - E_i(t)$$

where $D_i(t)$ is the deadline by which the task must be done and $E_i(t)$ is the amount of computation remaining to be complete. In other words, the laxity is the time left by its deadline after the task is completed assuming that the task could be executed immediately without preemption.

A task with zero laxity must be scheduled right away and executed with no preemption to meet the deadline. A task with a negative laxity indicates that the task will miss the deadline. Therefore, the laxity of a task is a measure of its urgency in deadline-driven scheduling[1]. The preemptive Least-Laxity-First (LLF) scheduling algorithm always executes a task whose laxity is the least[1]. This algorithm has been proved to be optimal[1] for a uniprocessor system[2]. With the LLF scheduling algorithm, however, if two or more tasks have same laxities, laxity-tie occurs. Once laxity-tie occurs, context switches takes place every scheduling point until the tie breaks. The laxity-tie in the LLF scheduling algorithm results in the poor system performance due to the frequent context switches[3]. Therefore, the LLF scheduling algorithm, though it is very interesting theoretically, is not practical to implement.

In this paper, we propose a Modified Least-Laxity-First (MLLF) scheduling algorithm to solve the frequent context switches problem of the LLF scheduling algorithm. We prove the optimality of the MLLF scheduling algorithm and show the performance by using simulation results.

The MLLF scheduling algorithm defers the context switch until necessary even if the laxity-tie occurs. That is, the MLLF scheduling algorithm allows the laxity inversion where a task with the least laxity may not be scheduled immediately. The MLLF scheduling algorithm is an optimal algorithm in a sense that if a set of tasks can be scheduled by the LLF scheduling algorithm, those tasks are also schedulable[2] by the MLLF scheduling algorithm. The complexity of the MLLF scheduling algorithm, however, is higher than that of the LLF

⋆ This work is supported partly by KOSEF(Korea Science & Engineering Foundation) under contract # 96-0101-07-01-3(Integrated Object-Oriented Development Environment for Distributed Real-Time Systems)

[1] An algorithm is said to be *optimal* if it may fail to meet a deadline only if no other algorithms of the same class can meet it.

[2] A schedule is said to be *feasible* if all tasks can be completed according to a set of specified constraints. A set of tasks is said to be *schedulable* if there exists at least one algorithm that can produce a feasible schedule[4].

scheduling algorithm. In simulation experiments, we consider both the number of context switches and the cost of the scheduling algorithm. We analyze the global performance between two algorithms and show that the MLLF scheduling algorithm performs better than the LLF scheduling algorithm.

The rest of this paper is organized as follows: Section 2 presents a Modified Least-Laxity-First scheduling algorithm. In section 3, our simulation experiments are described and we conclude in section 4.

2. The Modified Least-Laxity-First (MLLF) Scheduling Algorithm

As long as there exists no laxity-tie, MLLF scheduling algorithm proposed in this paper produces the same schedule as LLF scheduling algorithm. If the laxity-tie occurs, MLLF scheduling algorithm allows the running task to run with no preemption as far as the deadlines of other tasks are not missed. (If there is a laxity-tie and no running task, a task with the earliest deadline is selected.)That is, MLLF scheduling algorithm allows the laxity inversion where a task with the least laxity may not be scheduled. The laxity inversion duration is defined as follows:

Definition 1. Laxity Inversion Duration at time t is the duration that the current running task can continue running with no loss in schedulability even if there exist a task (or tasks) whose laxity is smaller than the current running task.

Lemma 1. In LLF scheduling, consider T_i and T_j satisfying $L_i(t) \le L_j(t)$. After T_i has being executed for $L_j(t) - L_i(t)$, the laxities of T_i and T_j will become the same. If $L_j(t) - L_i(t) \ge E_i(t)$, T_i will be completed before the laxities of T_i and T_j becomes the same. And, if $L_j(t) - L_i(t) < E_i(t)$, the laxities of T_i and T_j will tie before T_i is completed.

Proof:

Let t' $(t \le t')$ be the time the laxities of T_i and T_j become the same, and ω_{other} the time spent for execution of the other tasks except T_i and T_j over $\lfloor t, t' \rfloor$. Since T_i always has higher priority than T_j over $\lfloor t, t' \rfloor$, T_j will never be executed in the interval $\lfloor t, t' \rfloor$. Let ω_i be the execution time of T_i in the interval $\lfloor t, t' \rfloor$. At time t', T_i and T_j satisfy the following:

$$L_i(t') = L_j(t')$$

$$L_i(t) - \omega_{other} = L_j(t) - \omega_{other} - \omega_i .$$

Notice that the laxities of T_i and T_j are the same after T_i has been executed for $\omega_i = L_j(t) - L_i(t)$. If $\omega_i \ge E_i(t)$, $L_i(t) - \omega_{other} \le L_j(t) - \omega_{other} - E_i(t)$ is satisfied at time t', so that T_i will be completed before or at time t'. And if $\omega_i < E_i(t)$, $E_i(t') = E_i(t) - \omega_i > 0$ holds at time t', so that the laxity of T_i and T_j will become the same before T_i is completed. ■

From Lemma 1, the following cases need to be considered in scheduling of T_i and T_j with LLF:

Case 1) If $L_j(t) - L_i(t) \ge E_i(t)$, T_i will be completed before the laxities of T_i and T_j become the same. This means $D_j(t) > D_i(t)$.

Case 2) If $L_j(t) - L_i(t) < E_i(t)$, the laxities of T_i and T_j will be the same before T_i is completed. This case can be divided into two cases:

Case 2.1) If $E_i(t) - (L_j(t) - L_i(t)) > E_j(t)$ (or if $D_i(t) > D_j(t)$), T_j is completed before the completion of T_i. T_i will be executed during $L_j(t) - L_i(t) + E_j(t)$ (or $D_j(t) - L_i(t)$) before T_j is completed.

Case 2.2) If $E_i(t) - (L_j(t) - L_i(t)) \le E_j(t)$ (or if $D_i(t) \le D_j(t)$), T_i will be completed before T_j is completed.

Observe that if $D_i(t) \le D_j(t)$, T_i will be completed before T_j is completed in the LLF scheduling. Therefore, in this case, we can execute T_i with no preemption until its completion and T_j will not be miss its deadline. In case of $D_i(t) > D_j(t)$, we can execute T_i with no preemption for $D_j(t) - L_i(t)$ and by that time T_j must be executed in order not to miss its deadline.

Lemma 2. At time t, suppose there is a schedulable task set with T_i and T_j, satisfying $L_i(t) < L_j(t)$ and $D_i(t) > D_j(t)$. Then, T_i and T_j are still schedulable at time $t + \alpha$ when T_i has been executed with no preemption for α, where $0 < \alpha \le D_j(t) - L_i(t)$.

Proof:

At time t, the following conditions are true for T_i and T_j:

$$L_i(t) \ge E_j(t) \qquad (1)$$

$$L_j(t) \ge 0 \qquad (2)$$

Now, in order to be schedulable at time $t + \alpha$ for T_i and T_j, the following conditions must be satisfied:

$$L_i(t + \alpha) \ge E_j(t + \alpha) \qquad (3)$$

$$L_j(t + \alpha) \ge 0 \qquad (4)$$

Since $L_i(t+\alpha) = L_i(t)$ and $E_j(t+\alpha) = E_j(t)$, condition (3) is true. And since $L_j(t+\alpha) = L_j(t) - \alpha$ and $L_j(t) - \alpha \geq L_i(t) - E_j(t) \geq 0$, condition (4) is also true. ■

Now let T_a be a task with the smallest remaining execution time among a set of tasks with the least laxities. This means T_a's deadline is the earliest among those tasks. And let $T_{\min}$ be a task which has the earliest deadline among all tasks, but has larger laxity than T_a. With no loss in schedulability (i.e. in order not to miss the deadlines of other tasks), T_a may be executed with no preemption for $D_{\min}(t) - L_a(t)$, where $D_{\min}(t)$ is the deadline of task $T_{\min}$, and $L_a(t)$ is the laxity of task T_a. If $D_a(t) \leq D_{\min}(t)$ is satisfied, T_a will be completed with no preemption.

Therefore, MLLF scheduling algorithm allows laxity inversion for $E_{\min}(t) - 1$, where $E_{\min}(t)$ is the remaining execution time of task $T_{\min}$ at time t. MLLF scheduling algorithm reevaluates the priorities of tasks when any of the following conditions is met:

1) when the current running task T_a is terminated,
2) when the currently running task T_a has executed for $D_{\min}(t) - L_a(t)$,
3) when a new task arrives.

Figure 1 presents the MLLF scheduling algorithm. If $T_{\min}$ does not exists, T_a will be executed until its completion.

Consider two tasks, T_1 and T_2, shown in Table 1, whose laxities are the same. Figure 2 and Figure 3 show the schedules produced by LLF and MLLF scheduling algorithms, respectively, for the task set in table 1. The context switches point is indicated by a down-arrow. Five context switches occur with LLF, whereas only one context switches with MLLF. In this case Laxity Inversion Duration is 2, from time 1 to from 3.

Table 1 . An example of task set

	Remaining Execution Time	Deadline	Laxity
T_1	3	6	3
T_2	4	7	3

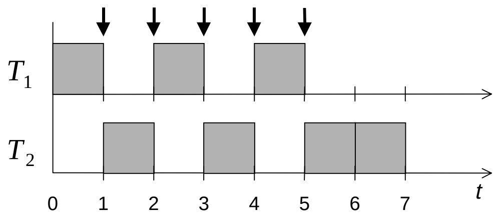

Figure 2. Schedule generated by the LLF scheduling algorithm

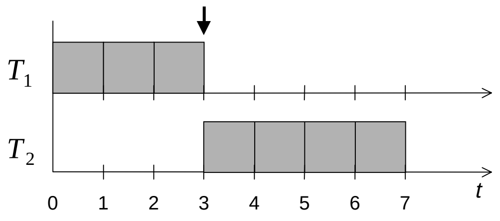

Figure 3. Schedule generated by the MLLF scheduling algorithm

Here, we prove the optimality of the proposed MLLF scheduling algorithm.

```
/*  Rescheduling point conditions                                  */
/*   1. $T_a$ Terminates                                            */
/*   2. $T_a$ uses up the time quantum $E_a(t)$ or $D_{\min}(t) - L_a(t)$  */
/*   3. new tasks are requested                                     */

Algorithm MLLF
begin
    finds $T_a$ that satisfies $V_1 = \{T_i \mid L_i(t) \leq L_j(t),\ T_i, T_j \in T\}$ and $T_a = \{T_i \mid E_i(t) \leq E_j(t),\ T_i, T_j \in V_1\}$ ;
    finds $T_{\min}$ that satisfies $T_{\min} = \{T_i \mid D_i(t) \leq D_j(t) \text{ and } L_i(t), L_j(t) > L_a(t),\ T_i, T_j \in T\}$ ;
    executes $T_a$ until satisfying any rescheduling point conditions ;
end
```

Figure 1. MLLF scheduling algorithm

Theorem 1. MLLF is optimal.

Proof:

The proof is by the induction. Let S^n be a task set at the n th rescheduling point(or time t^n) and S^{n+1} be a task set at the ($n+1$)th rescheduling point(or time t^{n+1}). Suppose S^n is schedulable and if MLLF always produces a schedulable set S^{n+1}, MLLF is optimal.

In MLLF, rescheduling occurs ① when the current running task T_a is terminated, ② when the currently running task T_a has executed for $D_{\min}(t) - L_a(t)$ and ③ when a new task arrives.

At time t^n, a task set V and $\overline{V}$ are defined as followings:

$$V = \{T_i \mid L_i(t^n) \le L_j(t^n), T_i, T_j \in S^n\}$$
$$\overline{V} = S^n - V$$

Since T_a has the earliest deadline among the task set V, execution of T_a with no preemption cannot incur the missing of any other task's deadline in the task set V. However, there may be a task(s) in $\overline{V}$ with earlier deadline than T_a. A task, T_x, in $\overline{V}$ is either in one of the two cases.

A) $D_a(t^n) \le D_x(t^n)$

B) $D_a(t^n) > D_x(t^n)$

where $D_x(t^n)$ is the deadline of T_x at time t^n. In case A, execution of T_a with no preemption cannot incur the missing of T_x's deadlines. In case B, from Lemma 2, execution of T_a with no preemption for $D_x(t) - L_a(t)$ cannot incur the missing of T_x's deadlines. Therefore, execution of T_a with no preemption for $D_{\min}(t) - L_a(t)$ where $T_{\min}$ has the earliest deadline among task set $\overline{V}$ cannot incur the missing of any task's deadlines in the task set $\overline{V}$. ■

3. Performance Evaluation

In this section, LLF and MLLF scheduling algorithms are tested by simulation in order to compare the global performance. When there are n tasks at time t, LLF scheduling algorithm performs the maximum of $n-1$ operations to find a task with the least laxity. MLLF scheduling algorithm, however, performs the maximum of $3n-3$ operations to find T_a and $T_{\min}$.

Definition 2. The notations used to evaluate performance are as follows:

- C_{CS} : the cost of a context switches
- C_{LLF}, C_{MLLF} : the cost of LLF and MLLF scheduling algorithms, respectively
- N_{LLF}, N_{MLLF} : the number of context switches that is produced by LLF and MLLF scheduling algorithms, respectively
- T_{LLF}, T_{MLLF} : the total scheduling cost

$$T_{LLF} = N_{LLF} \times (C_{CS} + C_{LLF})$$
$$T_{MLLF} = N_{MLLF} \times (C_{CS} + C_{MLLF})$$

- *global performance ratio* of MLLF and LLF scheduling algorithm :

$$global\ \ performance\ \ ratio = \frac{T_{LLF}}{T_{MLLF}}$$

We analyze the schedulability, the number of context switches, and global performance ratio by using the simulation results. In simulation experiments, we assume that

- All tasks are periodic and the relative deadline of each task is equal to its period.
- The period of tasks is chosen as a random variable with uniform distribution between 10 and 100 time units.
- The worst-case execution time is computed as a random variable with uniform distribution between 1 and 70 time units.
- C_{LLF} and C_{MLLF} are $n-1$ and $3n-3$, respectively.
- C_{CS} is 500. We assume that the cost of the scheduling algorithm depends on the cost of memory accesses rather than processor operation. Since the time overhead of one context switches between processes is 500 times larger than one memory access on general computer system, C_{CS} is assumed to be 500.

Both the LLF and the MLLF scheduling algorithms have the same schedulability for a task set with processor utilization below 1.0.

In Figure 4, N_{MLLF}/N_{LLF} (i.e., the number of context switches ratio) is shown as a function of processor utilization on the assumption that the number of tasks is random. With the fixed number of tasks = 10 and 20, Figure 5 shows N_{MLLF}/N_{LLF}. As shown in Figures 4 and 5, as the processor utilization increases, MLLF scheduling algorithm performs much better than LLF scheduling algorithm. This is due to the high possibility of laxity-ties as the processor utilization increases.

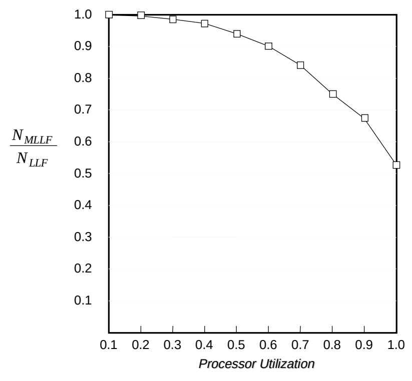

Figure 4. Comparison of the number of context switches

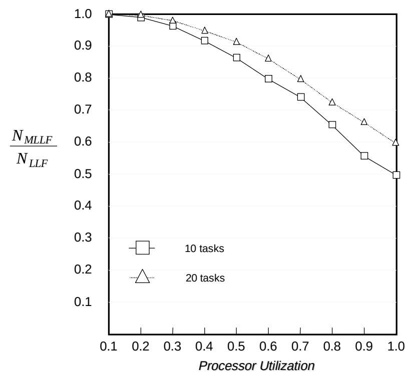

Figure 5. Comparison of the number of context switches with task = 10 and 20

In Figure 6, N_{MLLF}/N_{LLF} is shown when the processor utilization is 0.9. Although as the number of tasks increases the ratio also increases slightly, the number of context switches with MLLF is half on the average.

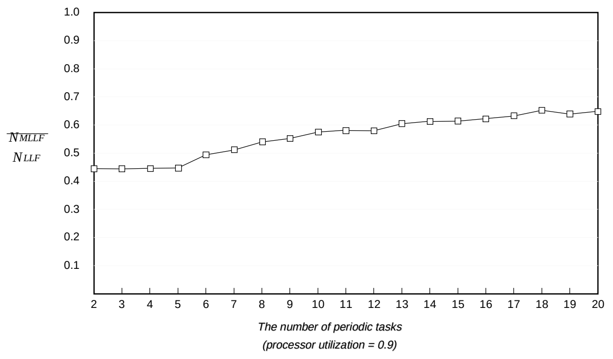

Figure 6. Comparison of the number of context switches

Figure 7 shows the global performance ratio with the fixed number of tasks = 10 and 20. As expected, as the processor utilization increases MLLF performs much better than LLF algorithm.

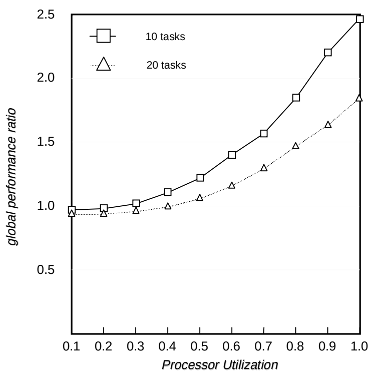

Figure 7. Global performance ratio with task = 10 and 20

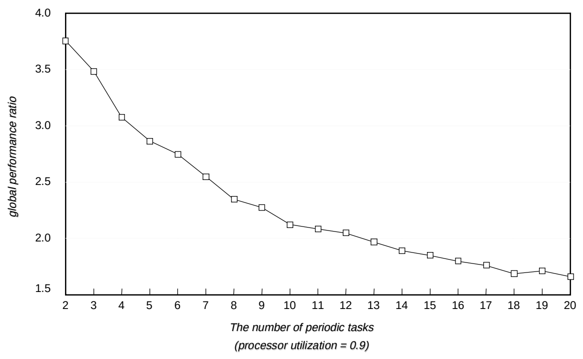

Figure 8. Global performance ratio

In Figure 8, the global performance ratio is shown when the processor utilization is 0.9. As the number of tasks increases, the performance of MLLF goes down due to the cost of the algorithm itself. Therefore, in order to maximize the performance of MLLF algorithm, an efficient implementation approach and data structure are needed.

4. Conclusion

In this paper, we proposed the Modified Least-Laxity-First (MLLF) scheduling algorithm that solves the disadvantage of the LLF scheduling algorithm. MLLF scheduling algorithm defers the preemption by allowing laxity inversion as far as deadlines of tasks are not missed. Hence, MLLF scheduling algorithm avoids the degradation of systems performance. We proved the optimality of MLLF scheduling algorithm. The simulation results showed that MLLF scheduling algorithm performs better than LLF scheduling algorithm especially when the number of processes is small and the processor utilization is high.

We now investigate an efficient implementation approach and data structures to incorporate MLLF scheduling algorithm in the real-time kernel we are developing.

References

[1] M.L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes," *Information Processing 74*, North-Holland Publishing Company, 1974

[2] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," *Ph.D. Thesis*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983

[3] Michael B. Jones, Joseph S. Barrera III, Alessandro Forin, Paul J. Leach, Daniela Rosu and Marcel-Catalin Rosu, "An Overview of the Rialto Real-Time Architecture," *In Proceedings of the Seventh ACM SIGOPS European Works*hop, Connemara, Ireland, pages 249-256, September, 1996

[4] G.C. Buttazzo, "Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications," *Kluwer Academic Publishers*, 1997