International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:9, 2007

A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks

Vahid Salmani, Saman Taghavi Zargar, and Mahmoud Naghibzadeh

***Abstract*—**This paper presents a modified version of the maximum urgency first scheduling algorithm. The maximum urgency algorithm combines the advantages of fixed and dynamic scheduling to provide the dynamically changing systems with flexible scheduling. This algorithm, however, has a major shortcoming due to its scheduling mechanism which may cause a critical task to fail. The modified maximum urgency first scheduling algorithm resolves the mentioned problem. In this paper, we propose two possible implementations for this algorithm by using either earliest deadline first or modified least laxity first algorithms for calculating the dynamic priorities. These two approaches are compared together by simulating the two algorithms. The earliest deadline first algorithm as the preferred implementation is then recommended. Afterwards, we make a comparison between our proposed algorithm and maximum urgency first algorithm using simulation and results are presented. It is shown that modified maximum urgency first is superior to maximum urgency first, since it usually has less task preemption and hence, less related overhead. It also leads to less failed non-critical tasks in overloaded situations.

***Keywords*—**Modified maximum urgency first, maximum urgency first, real-time systems, scheduling.

I. Introduction

IN real-time systems, every real-time task has a deadline before or at which it must be completed. Due to the criticality of the tasks, the scheduling algorithms of these systems must be timely and predictable. Tasks could either be periodic or aperiodic. In this paper, we will only consider periodic tasks. Real-time scheduling algorithms may assign priorities statically, dynamically, or in a hybrid manner, which are called fixed, dynamic and mixed scheduling algorithms, respectively. These algorithms may allow preemptions to occur or may impose a non-preemptive method.

A dynamically reconfigurable system can change in time without the need to halt the system and hence needs a dynamic scheduler. The *Earliest Deadline First* (EDF) algorithm proposed by Liu and Layland is an optimal dynamic scheduling algorithm [1]. This algorithm, however, is not predictable in a transient overloaded situation in the sense that it does not guarantee to execute the set of higher priority tasks before their deadlines.

This paper proposes a *Modified Maximum Urgency First* (MMUF) scheduling algorithm as an optimization of *Maximum Urgency First algorithm* (MUF) [2] which is used to predictably schedule dynamically changing systems. The MMUF is a preemptive mixed priority algorithm for predictable scheduling of periodic real-time tasks.

The rest of this paper is organized as follows. Section 2 briefly describes dynamic priority scheduling algorithms including EDF, Least Laxity First (LLF), Modified Least Laxity First (MLLF) and MUF. In section 3 we propose our MMUF scheduling algorithm. Section 4 describes our simulation experiments and results. Finally, Section 5 is a short conclusion.

II. Related Works

The *earliest deadline first* [1] and *least laxity first* [3] algorithms are presented as optimal dynamic priority scheduling algorithms. However, the later is impractical to implement because laxity ties result in poor system performance due to the frequent context switches among the tasks. The *modified least laxity first* algorithm proposed by Oh and Yang solves the problem of the LLF algorithm by reducing the number of context switches [4]. With all these algorithms a transient overload in the system may cause a critical task to fail, which is certainly undesirable. Stewart and Khosla have presented a mixed priority algorithm called *Maximum Urgency First* which defines a critical set of tasks that is guaranteed to meet all its deadlines during a transient overload [2].

A. Earliest Deadline First Algorithm

The earliest deadline first (also known as *nearest deadline first*) is a dynamic priority algorithm which uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority, while the lowest priority belongs to the task with the latest deadline. Since priorities are dynamic, the periods of tasks can change at anytime. In addition, this algorithm has the schedulability bound of 100% for all task sets.

The CPU utilization of task T_i is computed as the ratio of its worst-case execution time E_i to its request period P_i. The total utilization U_n for n periodic tasks is calculated as follows [1]:

V. Salmani is a graduate student with the Computer Engineering Department, Ferdowsi University, Mashad, Iran (e-mail: salmani@um.ac.ir).

S. Taghavi Zargar is a graduate student with the Computer Engineering Department, Ferdowsi University, Mashad, Iran (e-mail: taghavi@stu-mail.um.ac.ir).

M. Naghibzadeh is a professor with the Computer Engineering Department, Ferdowsi University, Mashad, Iran (e-mail: naghib@um.ac.ir).

$$U_n = \sum_{i=1}^{n} \frac{E_i}{P_i} \quad (1)$$

A significant disadvantage of this algorithm is that there is no way to guarantee which tasks will fail in a *transient overloaded* situation. A transient overload occurs when the worst case utilization is above 100%, in which case it is possible that a critical task may fail at the expense of a lesser important task. This may happen because there is no control of which task fails and the deadline is the only scheduling parameter. However, in a transient overloaded situation we may want to consider task's criticalness as the second parameter.

The EDF algorithm was decided to be considered as a basis for the MMUF algorithm which is proposed in this paper.

B. Least Laxity First Algorithm

The least laxity first (also known as *minimum laxity first*) assigns higher priority to a task with the least laxity. The *laxity* of a real-time task T_i at time t, $L_i(t)$, is defined as follows:

$$L_i(t) = D_i(t) - E_i(t) \quad (2)$$

where $D_i(t)$ is the deadline by which the task must be completed and $E_i(t)$ is the amount of computation remaining to be performed. In other words, laxity is a measure of the flexibility available for scheduling a task. A laxity of $L_i(t)$ means that if the task T_i is delayed at the most by $L_i(t)$ time units, it will still meet its deadline.

A task with zero laxity must be scheduled right away and executed without preemption or it will fail to meet its deadline. The negative laxity indicates that the task will miss the deadline, no matter when it is picked up for execution.

A major problem with LLF algorithm is that it is impractical to implement because *laxity ties* result in the frequent context switches among the corresponding tasks. This will cause the system performance to remarkably degrade. A laxity tie occurs when two or more tasks have the same laxities.

Similar to EDF, LLF has a 100% schedulability bound. Nevertheless, there is no way to control which tasks are guaranteed to execute during a transient overload.

C. Modified Least Laxity First Algorithm

Our purpose in describing the modified least laxity first algorithm in this section is to introduce it as an alternative to the EDF in order to be used as a basis for the MMUF algorithm which is proposed in this paper.

As it is mentioned earlier, the MLLF scheduling algorithm solves the problem of LLF algorithm by significantly reducing the number of context switches. By decreasing the system overhead due to unnecessary context switches, MLLF algorithm avoids the degradation of system performance and conserves more system resources for unanticipated aperiodic tasks.

As long as there is no laxity tie, MLLF schedules the task sets the same as LLF algorithm. If the laxity tie occurs, the running task continues to run with no preemption as far as the deadlines of other tasks are not missed.

The MLLF algorithm defers the context switching until necessary and it is safe even if the laxity tie occurs. That is, it allows the *laxity inversion* where a task with the least laxity may not be scheduled immediately. Laxity inversion applies to the duration that the currently 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 [4].

D. Maximum Urgency First Algorithm

The maximum urgency first algorithm solves the problem of unpredictability during a transient overload for EDF, LLF and MLLF algorithms. This algorithm is a combination of fixed and dynamic priority scheduling, also called *mixed priority* scheduling. 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 criticality has higher precedence over the dynamic priority while user priority has lower precedence than the dynamic priority.

The MUF algorithm assigns priorities in two phases. Phase One concerns the assignment of static priorities to tasks. Static priorities are assigned once and do not change after the system starts. Phase Two deals with the run-time behavior of the MUF scheduler as it is clarified later.

The first phase consists of these steps [2]:

1) It sorts the tasks from the shortest period to the longest period. Then it defines the critical set as the first N tasks such that the total CPU load factor does not exceed 100%. These tasks are guaranteed not to fail even during a transient overload.
2) All tasks in the critical set are assigned high criticality. The remaining tasks are considered to have low criticality.
3) Every task in the system is assigned an optional unique user priority.

In the second phase, the MUF scheduler follows an algorithm to select a task for execution. This algorithm is executed whenever a new task is arrived to the ready queue. The algorithm is as follows:

1) If there is only one highly critical task, pick it up and execute it.
2) If there are more than one highly critical task, select the one with the highest dynamic priority. Here, the task with the least laxity is considered to be the one with the highest priority.
3) If there is more than one task with the same laxity, select the one with the highest user priority.

The MUF scheduler has been implemented as the default scheduler of CHIMERA II, a real-time operating system being used to control sensor-based control systems [5].

III. Modified Maximum Urgency First Algorithm

Although MUF is an efficient algorithm, it has a major disadvantage. Since the rescheduling operation is performed whenever a task is arrived to the ready queue [2], [6], there is

the possibility of failing a critical task in certain situations. In these situations, a task with minimum laxity may be selected whose remaining execution time is greater than remaining time to another task's laxity. This problem is due to performing the rescheduling operation whenever a new task is added to the ready queue. According to [7], the scheduling should be performed at any given instant and the scheduler will choose the highest priority task to run among all available tasks. Therefore, the schedule should be produced in such a way that the task having the highest priority always be running.

Consider two tasks, T_1 and T_2, shown in Table I. Fig. 1 shows the schedule which is produced by the MUF algorithm for the task set in Table I.

TABLE I
AN EXAMPLE OF TASK SET

	Remaining Execution Time	Deadline	Remaining Time to Laxity
T_1	4	6	2
T_2	1	4	3

As it is shown in Fig. 1 the MUF will select the task with minimum laxity (T_1) at time zero. The remaining execution time of task T_1 is greater than remaining time to T_2's laxity. This selection will cause task T_2 to miss its deadline.

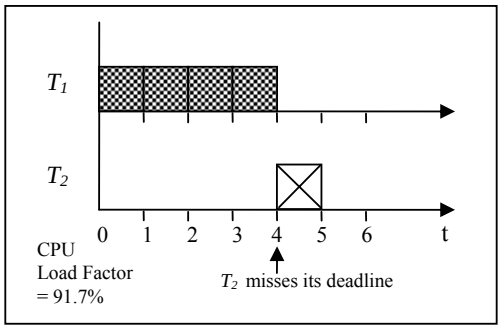

Fig. 1 Schedule generated by the MUF scheduling algorithm

The modified maximum urgency first algorithm we propose in this paper is a modified version of MUF algorithm which resolves the mentioned MUF algorithm's deficiency. In addition it has some extra advantages which will be explained later.

The modifications are as follows: With this algorithm, we use a unique *importance* parameter, instead of using tasks' request intervals, to create the critical set. The importance parameter is a fixed priority which can be defined as user priority or any other optional parameter which expresses the degree of the task's criticalness. It is trivial that the task with the shortest request period is not necessarily the most important one. Furthermore, using the importance parameter, it is not needed to use *period transformation* [8] as it is done in MUF algorithm [2]. With the MMUF algorithm, either EDF or MLLF can be used to define the dynamic priority. Another optimization made in this algorithm is the elimination of unnecessary context switches which in turn reduced the overall system overhead. This is done firstly by using MLLF instead of LLF and secondly, by letting the currently running task to keep running while there are some other tasks with the same priority.

The MMUF algorithm consists of two phases with the following details:

Phase 1: In this phase fixed priorities are defined only once as follows. These fixed priorities will not change during execution time.

1) Order the tasks from the most importance to the least importance
2) Add the first N tasks to the critical set such that the total CPU load factor does not exceed 100%

Phase 2: This phase calculates the dynamic priorities at every scheduling event and selects the task to be executed next.

1) If there is at least one critical task in the ready queue
   a. Select the *critical* task with the earliest deadline (EDF algorithm) if there is no tie
   b. If there are two or more *critical* tasks with the same earliest deadline
      i. If any of these *critical* tasks is already running select it to continue running
      ii. Otherwise, select the *critical* task with the highest importance
2) If there is no critical task in the ready queue
   a. Select the task with earliest deadline (EDF algorithm) if there is no tie
   b. If there are two or more tasks with the same earliest deadline
      i. If any of these tasks is already running select it to continue running
      ii. Otherwise, select the task with the highest importance

As it is mentioned earlier, the MLLF algorithm can be used instead of the EDF in the above algorithm.

The MMUF algorithm needs the start time, deadline parameter and worst-case execution time of each task to be specified before the system starts.

When using EDF algorithm, a scheduling event occurs whenever a task is arrived to the ready queue or a task finishes its execution time. For the MLLF algorithm, there is a third scheduling event which is explained in [4].

A. RM, EDF, LLF, MLLF and MUF as Special Cases of MMUF

It was shown in [5] that the RM (Rate Monotonic [9]), EDF and LLF algorithms are special cases of the MUF algorithm. If the period is used as the importance parameter and the optimization which is made for the reduction of context switches is omitted, the MMUF scheduler with either of EDF or MLLF behaves like the MUF scheduler. If all tasks specify *zero* as the worst-case execution time, the EDF (used in MMUF) behaves like the LLF (used in MUF). In this case the laxity would be a function of deadline [5]. As long as there exists no laxity tie, the MLLF (used in MMUF) behaves like

LLF (used in MUF) [4]. Therefore, the RM, EDF, LLF and MUF algorithms are special cases of the MMUF algorithm. When the processor utilization is less than or equal to one (i.e. there is no non-critical task in the system), if a task which was already running is not selected in the case there are two or more tasks with the same dynamic priority, the MMUF scheduler behaves like the MLLF scheduler. Therefore, MLLF is also a special case of MMUF.

B. MMUF as an Efficient Hybrid Algorithm

The efficiency of MMUF can be inferred from optimality of EDF [10] and MLLF [4] algorithms. As it is explained earlier, EDF and MLLF are special cases of MMUF algorithm. If there is no non-critical task in the system, that is the CPU utilization is less than or equal 100%, the MMUF algorithm behaves exactly like EDF or MLLF algorithms depending on which one is chosen within the MMUF, and hence it is optimal too. If the CPU load factor exceeds 100%, i.e., there are some non-critical tasks, the same things hold for EDF and MLLF. That is, there is no guarantee for these tasks to meet their deadlines.

IV. Performance Evaluation

In this section we first compare our proposed MMUF algorithm in case of using either EDF or MLLF and will show that the MMUF is more efficient when the EDF is used. Therefore, we propose using EDF instead of MLLF in the MMUF algorithm. Then the MMUF algorithm is compared with the MUF algorithm.

We analyze the number of context switches, global performance ratio and the number of failed non-critical tasks by performing the simulation results. In our simulation experiments, we assume that:

1) All tasks are periodic and the deadline parameter of each task is equal to its request period.
2) The period of tasks is chosen randomly between 10 and 200 time units.
3) The worst-case execution time is chosen randomly. It is at least one and at most 30% of the corresponding task's request period.
4) All tasks start simultaneously at time zero.

It is worth mentioning that when a task is being started for execution we have first check to see whether it can be completed in time or not, assuming that it will not be preempted. If the task cannot meet its deadline, it will not be started and it is considered as a failed task [11].

A. Comparison of MMUF using EDF and MLLF

The notations used to evaluate performance are as follows:

1) *MMUF-EDF*, *MMUF-MLLF*: the MMUF algorithm using either EDF or MLLF to calculate the dynamic priorities, respectively.
2) $N_{MMUF\text{-}EDF}$, $N_{MMUF\text{-}MLLF}$: the number of context switches that is produced by MMUF-EDF and MMUF-MLLF scheduling algorithms, respectively.
3) $T_{MMUF\text{-}EDF}$, $T_{MMUF\text{-}MLLF}$: the total scheduling overhead.
4) $F_{MMUF\text{-}EDF}$, $F_{MMUF\text{-}MLLF}$: the number of failed non-critical tasks.

In Fig. 2, $N_{MMUF-EDF} / N_{MMUF-MLLF}$ (i.e. the number of context switches ratio) is shown as a function of processor utilization with the fixed number of tasks = 10 and 20. As the processor utilization increases, the number of context switches ratio goes down.

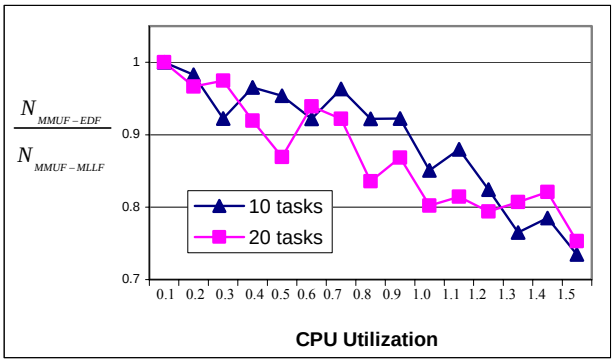

Fig. 2 Comparison of the number of context switches

Fig. 3 shows the $T_{MMUF-EDF} / T_{MMUF-MLLF}$ (i.e. the global performance ratio) with the fixed number of tasks = 10 and 20. As it is shown the global performance ratio, in most cases, is less than one. Furthermore, it is less than one in all cases in which the processor utilization is greater than one.

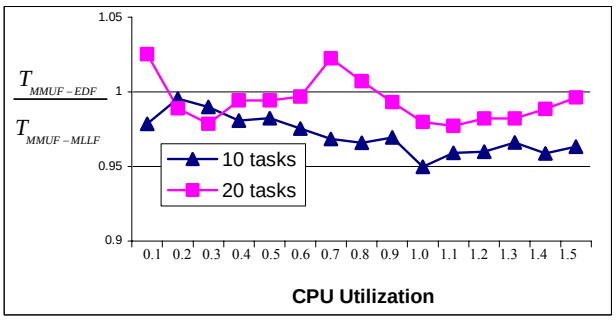

Fig. 3 Global performance ratio

In Fig. 4, the $F_{MMUF-EDF} / F_{MMUF-MLLF}$ (i.e. the number of failed non-critical tasks ratio) is shown as a function of processor utilization with the fixed number of tasks = 10 and 20. The number of failed non-critical tasks ratio is less than one and as the number of tasks increases, the number of failed non-critical tasks ratio reduces too. It is trivial that the number of failed non-critical tasks is zero when the processor utilization is less than or equal to one, since there in no non-critical task in the system.

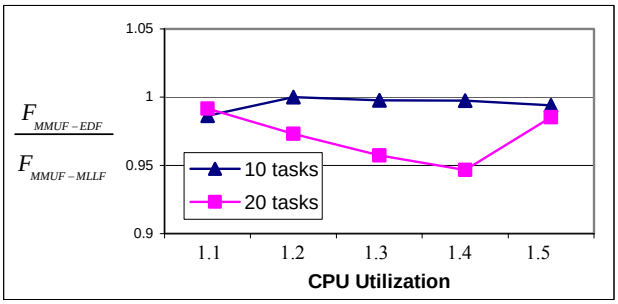

Fig. 4 Comparison of the number of failed non-critical tasks

B. Comparison of MMUF and MUF

The notations used to evaluate performance are as follows:

1) *MMUF*: the MMUF algorithm using EDF to calculate the dynamic priorities.
2) N_{MMUF}, N_{MUF}: the number of context switches that is produced by MMUF and MUF scheduling algorithms, respectively.
3) F_{MMUF}, F_{MUF}: the number of failed non-critical tasks.

In Fig. 5, N_{MMUF} / N_{MUF} (i.e. the number of context switches ratio) is shown as a function of processor utilization with the fixed number of tasks = 10 and 20. In all cases, the number of context switches ratio is less than one and it shows an improvement in MMUF algorithm.

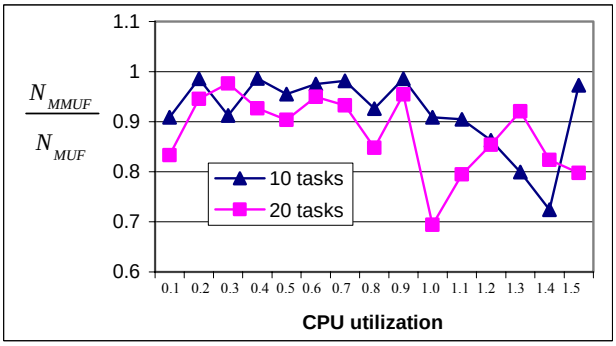

Fig. 5 Comparison of the number of context switches

Fig. 6 shows the F_{MMUF} / F_{MUF} (i.e. the number of failed non-critical tasks ratio) with the fixed number of tasks = 10 and 20. The number of failed non-critical tasks ratio is less than one and as the number of tasks increases, the number of failed non-critical tasks ratio reduces.

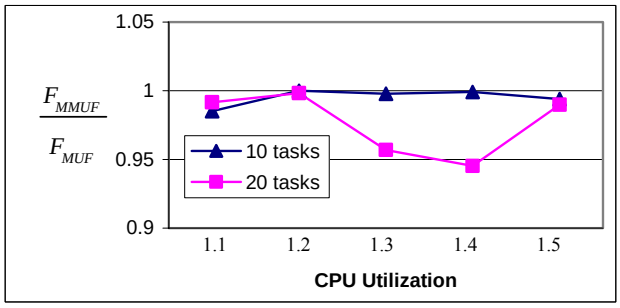

Fig. 6 Comparison of the number of failed non-critical tasks

V. Conclusion

In this paper, we presented a modified version of MUF scheduling algorithm called MMUF which resolves the deficiency of the MUF algorithm in which a critical task may miss its deadline in certain situations. Moreover, some additional optimizations are applied in the MMUF algorithm. The performance of the MMUF was compared to MUF algorithm and showed to be superior. It usually has less task preemption and hence, less related overhead. It also leads to less failed non-critical tasks in overloaded situations in which the CPU load factor is greater than 100%.

References

[1] 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, January 1973.

[2] D. B. Stewart, and P. k. Khosla, "Real-Time Scheduling of Dynamically Reconfigurable Systems," in *Proc. IEEE International Conference on Systems Engineering*, Dayton Ohio, August 1991, pp. 139-142.

[3] A. Mok. "Fundamental Design Problems of Distributed Systems for Hard Real-time Environments". PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1983.

[4] S. H. Oh, and S. M. Yang, "A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks", in *Proc. Fifth International Conference on Real-Time Computing Systems and Applications*, October 1998, pp. 31-36

[5] D. B. Stewart, D. E. Schmitz, and P. K. Khosla, "Implementing Real-Time Robotic Systems using CHIMERA II," in *Proc. IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 1990, pp. 598-603.

[6] 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.

[7] J. Goossens, and P. Richard, "Overview of real-time scheduling problems", in *Proc. the ninth international workshop on project management and scheduling*, Nancy, France, April 2004.

[8] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemtive Scheduling," in *Proc. 10th IEEE Real-Time Systems Symposium,* Santa Monica, CA, December 1989.

[9] M. Naghibzadeh, M. Fathi, "Intelligent Rate-Monotonic Scheduling Algorithm for Real-Time Systems", *Kuwait Journal of Science and Engineering*, vol. 30, no. 2, pp. 197-210, 2003.

[10] M. L. Dertouzos. "Control robotics: The procedural control of physical processes", in *Proc. the IFIP Congress*, 1974, pp. 807–813.

[11] M. Naghibzadeh, "A modified Version of Rate-Monotonic Scheduling Algorithm and its efficiency Assessment", in *Proc. Seventh IEEE International Workshop on Object-Oriented Real-time Dependable Systems*, San Diego, January 2002.