Turk J Elec Eng & Comp Sci, Vol.18, No.6, 2010, © TÜBİTAK
doi:10.3906/elk-0906-77

Behaviors of real-time schedulers under resource modification and a steady scheme with bounded utilization

Refik SAMET, Orhan Fikret DUMAN
Department of Computer Engineering, Faculty of Engineering, Ankara University
06100, Besevler, Ankara-TURKEY
e-mail: samet@eng.ankara.edu.tr, o_duman701@hotmail.com

Abstract

In this article we present an analysis for task models having random resource needs and different arrival patterns. In hard real-time environments like avionic systems or nuclear reactors, the inputs to the system are obtained from real world by using sensors. And it is highly possible for a task to have different resource needs for each period according to these changing conditions of real world. We made an analysis of schedulers for task models having random resources in each period. Since feasibility tests for usual task models are just limited to some specific schedulers and arrival patterns, we made our analysis using different preemptive schedulers with different arrival patterns. Another issue that this paper presents is that for task sets having processor utilization factors of below 100%, a new scheduler assigns more resource values to the tasks than they originally have by extending and shrinking the resource values. This provides a better utilization of the resources as well as providing all the tasks to finish in their timing constraints and never assigning lower resource values than the initial resources for all tasks. We exposed a dynamic priority driven scheduling scheme that implements the newly developed Steady Scheme with Bounded Utilization for changing resources and simulated to reach the results depicted.

Key Words: *Scheduling, real-time scheduler, task arrival patterns, simulation, dynamic priority, random resource, resource modification*

1. Introduction

Scheduling concept is being analyzed thoroughly in the last decades since it applies to hard real-time environments and expects more efficiency and stability from the algorithms used. As the algorithms designed for scheduling are used basically on real world sensor-based controller systems, we need to know what behaviors we will face when that kind of a real world situations happen [1]. There are many analysis and comparison works done on the behaviors of the algorithms in different situations with some assumptions [2–4]. Lots of techniques are used for real-time scheduling of tasks which can have either periodic, aperiodic or some other arrival patterns. These techniques are distinguished according to their way of thinking in order to solve some specific problems. So this makes different algorithms superior to each other from different aspects and under different conditions.

In real-time systems, every cycle in the scheduling process includes the processes of data gathering by sensor systems, processing those data, having and transmitting a control signal according to the results obtained from these data, test process and fault tolerance if included [5–8]. The most optimal scheduling obtained either statically or dynamically and sequencing of these processes realization time have vital importance for efficient and proper working of hard real-time systems.

Schedulers for hard real-time systems struggle with the constraint of expecting the tasks to accomplish at the right time while assigning resources to tasks under some policies. So, a real-time system needs an efficient algorithm to process all the time-critical tasks on time and to achieve a lower bound for the miss or loss rate for tasks when proceeding through time not to lose much information about the given issue. All these things clarify the timing constraint of the aforementioned systems.

As well as satisfying the timing constraints, the scheduling methods also arrange the sharing of resources along the procedures or tasks to be done. They determine which task to run and when to run a new task according to the schedulers used. In a non-idling system, algorithms should be efficient enough for minimizing processor idling providing a better utilization. The admission control process which decides the tasks to be processed or not should be planned so well not to have a rejection or late completion for mission-critic tasks.

In this work, the behaviors of the scheduling algorithms under some certain conditions, like tasks having random resources in each period and having different arrival patterns, are analyzed and the superiorities of the existing methods under different conditions are examined. A resource modification scheme, which increments or decrements the resource values of the task set according to the momentary loads and tasks needs to accomplish the simulation period, is put forward.

The existing scheme and the motivation of this work will be explained in Section 2. The scheme of changing resources and task set generation are investigated in Section 3, the results of changing resource conditions for dynamic schedulers and suggested methods are given with the simulation results later in Section 4 followed by the conclusion in Section 5.

2. Motivation and related works

The studies on scheduling algorithms consider some certain conditions, analyze the behaviors of the algorithms under these conditions, expose the superiorities of each one and suggest a better scheduler under those certain conditions [3, 4, 9].

When comparing the scheduling algorithms, the works generally take the resource parameter as an input that never changes during execution and make their research on a specific arrival pattern.

To have a better idea about the behaviors of the algorithms under different conditions, we focused on dynamic, preemptive algorithms and tasks with changing resources in each activation with periodic arrival patterns having some different effective delays between two wake-up times.

2.1. Parameters

The basic methods to schedule real-time tasks are founded on the task priority which is to determine the behavior that the scheduler will use when tasks come into execution with the need for processor sources which is named *preemption.* Preemption is the basic issue of dynamic priority driven scheduling. To have a better idea about the runtime overhead of the algorithms, *context switch* (exchange of the shared resources between tasks) parameter is another important issue that should be taken into consideration.

2.2. Arrival patterns

Tasks have *arrival patterns* which describe the way the tasks are activated. An *aperiodic* task is only activated once in the system that ends up with a unique execution while a *periodic* task is activated many times and the delay between two activations is a fixed one. A *Poisson* process task is activated many times like periodic tasks and the only difference is that the delay between two activations is not fixed but it is random. The randomization rule used to generate these delays is an exponential one (Poisson process). A *sporadic* task is a task which is activated many times with the minimal delay between two successive activations.

2.3. Task model

A task is a sequential job that is invoked with some frequency and result in a single execution of the instance at a time, handled by a given scheduler. So, we can say a task is an infinite number of instances.

Each periodic task T_i, considered as an infinite sequence of jobs $T_{i,k}$, $(k = 1, 2, ...)$, includes the parameters

$$T_{i,k} = (R_i, D_i, P_i), \tag{1}$$

where R_i, D_i, P_i respectively represent Resource, relative Deadline (note that absolute deadline is equal to the release time of the instance plus the relative deadline) and Period. The execution of the k^{th} request of task T_i, which occurs at time $(k-1) * P_i$ (assuming that no offset delay exists), must finish before or at time $(k-1) * P_i + D_i$, knowing that the deadlines are considered to be hard which means deadline failure is fatal for the system.

Each task also has a start time (release time), the time when the task arrives in the system, parameter which is exactly for the first activation in the schedule.

2.4. Scheduling algorithms

As the scheduling concept emerged, the main scheme was founded on timeline scheduling. At that time most critical control applications were developed using offline table-driven approach (timeline scheduling), according to which the timeline is divided into slots of fixed length (minor cycle) and tasks are statically allocated in each slot based on their rates and execution requirements [10]. The schedule is constructed up to the least common multiple of all periods (called hyper-period, base period or major cycle) and stored in a table. At runtime, tasks are dispatched according to the table and synchronized by a timer at the beginning of each minor cycle. This scheme is improved for dynamic situations by using a priority-based approach, according to which each task is assigned a priority (which can be fixed or dynamic) and the schedule is generated online based on the current priority value.

Priority based algorithms come out with the static priority driven scheduling scheme of Rate Monotonic (RM) [11–14] and the dynamic priority driven scheduling schemes of Earliest Deadline First (EDF) [1, 11] and

Least Laxity First (LLF) [1, 15, 16] which are the base methods for task scheduling. A general comparison and the preemption rule for the algorithms are as follows.

Static priority driven scheduling scheme of RM algorithm results with a scheduling parameter that does not support the instantaneous and dynamically changing situations. The most significant property of the RM algorithm is that you can specify the critical set of tasks that should not fail anytime. But it has the disadvantage of a narrow schedulable bound of 69% and 88% for the average case [12].

The algorithm works according to the index of tasks periods. The task with the minimum period gets the higher priority.

Static Priority: Return (Task with the Minimum Period)

Dynamic priority driven scheduling scheme of EDF and LLF algorithms overcome the difficulty for responding dynamic changes in the system parameters. And the schedulable bound for tasks is 100%. An important deficiency of the algorithms is the inability for specifying the critical tasks that should never fail. When a transient overload appears in the system, you cannot be sure whether a critical task will be omitted or not. And the runtime overheads for these algorithms are more than that of the static priority driven scheduling schemes [1].

EDF algorithm assigns priorities according to the deadline queue which works as the task with the earliest deadline always gets the higher priority.

```
Dynamic Priority:
    Start (Release) Time of Task
    + [(Task Activation Number - 1) * Task Period]
    + Deadline of Task
Return (Task with Minimum Dynamic Priority)
```

LLF algorithm assigns priorities considering the time parameter that the task can be delayed which is named laxity. It assigns higher priority to the task with the smallest laxity value.

```
Dynamic Deadline:
    Start (Release) Time of Task
    + [(Task Activation Number - 1)* Task Period]
    + Deadline of Task
Laxity: Dynamic Deadline - Rest of Resource Need for Task
Return (Task with Minimum Laxity)
```

2.5. Assumptions

We shall consider the problem of scheduling a set $T = \{T_1, T_2, ...\}$ of n periodic tasks on a single processor in presence of the hard real-time constraints. Tasks are assumed independent, meaning that tasks do not have precedence constraints and each task has the parameters defined by equation (1).

We have conducted a deeper analysis on dynamic priority driven scheduling algorithms of EDF and LLF and tried to reveal the behaviors of these two algorithms under following conditions:

1. Given a task set, each task in the task set has varying resource needs for each activation. And the behaviors of the algorithms under this condition are analyzed.

2. Behaviors of the algorithms under different arrival patterns of periodic, sporadic and Poisson delays are analyzed.

3. We make use of schedulability and WCET (Worst Case Execution Time) parameters, and we mention about the feasibility of the algorithms.
4. We consider both algorithms as preemptive. We make use of the preemption and context switch number as parameters.
5. To fulfill the requirement of random resources in each period, we made a random array generation scheme between determined lower and upper bounds.

3. Resource modification and task set generation

A very important feature in scheduling theory is being able to satisfy changing dynamic situations and needs of the system. Because the resource parameter is defined statically for each task in a task set, there's no chance to assign more or less time for tasks having the need for resource modification. We have designed a scheduler which is able to change the resource at each period, assigning random resources which are generated considering the workload of the system.

The aims in developing such a system are:

- To have a flexible and dynamic scheduler which is able to satisfy the changing resource needs of the system and
- To be able to analyze the behaviors of the algorithms.

We made our analysis according to the successive topics:

1) Comparing the basic dynamic schedulers of EDF and LLF under:
    a) Changing resource needs considering the parameters of;
        - Schedulability (SCH),
        - Preemption (PR),
        - Context Switch (CS),
        - Worst Case Execution Time (WCET).
    b) Comparing these results with the task sets having static resources.
2) Making all these tests with periodic task sets as well as tasks with other arrival patterns of sporadic and Poisson delays.
3) Task sets used in the experiments are generated considering the total utilization in the system.

The definitions and the computation of the parameters that we used in our system are as follows.

The Hyper Period (Base Period) (HP) parameter is defined as a cycle such as the arrival pattern of a periodic tasks set recurs similarly [17]. It is;

$$HP = LCM\{P_i\}, \tag{2}$$

where $(1 \leq i \leq n)$.

The Processor Utilization (U) is the fraction of processor time spent in the execution of the task set [11].

$$U = \sum_{1}^{n} (R_i/P_i) \tag{3}$$

The necessary condition for the feasibility of any task set is that $U \leq 1$.

The Workload $(W(t))$ of a system is; given a critical instant (s_i) at time 0 $(i \in [1, n], s_i = 0)$ the amount of processing time requested by all instances whose release times are in the interval $[0, t]$ [18].

$$W(t) = \sum_{1}^{n} (HP/P_i) * R_i \tag{4}$$

Processor Availability $(B(t))$ is the available time in the base period for a specific task in a task set considering the other tasks resource needs in the base period. Formula is;

$$B(t) = HP - CT - W(t) \tag{5}$$

where CT is Current Time and means a critical instant.

Resource Need $(W_{rk}(t))$ in the base period is the specified task's need for the processor, considering the tasks activation number and assuming that the task has completed the running activation.

$$W_{rk}(t) = (HP/P_i) * R_i - P_i * Task\_Activation\_Number(i), \tag{6}$$

3.1. Task set generation

We have generated a suitable task set for testing the behaviors of the algorithms. To be able to make these tests sensible, we generated our task set by considering some works done on the choice of task sets, beginning with the selection of the test interval, which we mentioned as a hyper period. For some special cases, a feasibility check is conducted. The feasibility check of a system means to simulate the system in some feasibility interval, i.e. a finite interval, to ensure that no deadline is missed; and if there is only these requests in this interval, all deadlines in this interval will be met [19]. The method gives the idea that, for better scheduling simulation, and not to widen the work area, it is better to keep the hyper period and the simulation of the task set small. So, in order to reduce the simulation duration and to be able to make a judicious choice of the periods for the tasks we have chosen our task periods having harmonic relations which can make full utilization even for a static scheduler and provide a smaller hyper period [20].

Our task set $T = \{T_1, T_2, T_3, T_4\}$ is as follows with the parameters of resource, deadline and period.

As seen from Table 1, deadlines are relative [21] and equal to the periods of the tasks that have harmonic relations with the periods of 4, 8, 16, 32. This makes the hyper period as $HP = LCM\{4, 8, 16, 32\} = 32$ and provides a flexible simulation environment, as in 32 units of time we can make one pattern of arrival for all tasks and have all tasks request for access to the system at the hyper period. At time 32, T_1 should be activated for $HP/P_1 = 32/4 = 8$ times while T_2 for $32/8 = 4$, T_3 for $32/16 = 2$ and T_4 for $32/32 = 1$ time.

We also considered the processor utilization (U) parameter having the initial load of $U = 1/4 + 3/8 + 3/16 + 5/32 = 0.96875$, which is near maximum utilization (100%).

Table 1. Task set with periods having harmonic relations.

Name	Resource	Deadline	Period
T_1	1	4	4
T_2	3	8	8
T_3	3	16	16
T_4	5	32	32

When assigning random resource values to tasks after the execution of one period, we keep the following and reach the results shown in Table 2:

1. Compute the minimum processor utilization with all tasks having resource value of 1 and compute the maximum resource value that can be assigned to the tasks, each having 25% processor utilization.
2. Obtain a random value for the first task (between 1 and a quarter of the task period, to have a maximum processor load of 25% for the first task).
3. Obtain a random value for the second task (between 1 and a quarter of the period of the second task plus the unused resource, if exist, from the first task).
4. Keep applying rule 3 to the successive tasks.

Table 2. Minimum and maximum values of task resources.

Task	Period	Min	U (Min)	Max	U (Max)
T_1	4	1	0.25	1	0.25
T_2	8	1	0.125	2	0.25
T_3	16	1	0.0625	4	0.25
T_4	32	1	0.03125	8	0.25
Total Utilization			0.46875		1.00

Assigned random resources (RR) to tasks are computed according to the following formulas:

RR1 = Rand (Min1, Max1);
RR2 = Rand (Min2, Max2 + abs (Max1 - RR1));
RR3 = Rand (Min3, Max3 + abs (Max1 - RR1) +abs (Max2 - RR2));
RR4 = Rand (Min4, Max4 + abs (Max1 - RR1) + abs (Max2-RR2) + abs (Max3 - RR3)).

This scheme gives the processor utilization between minimum utilization of 46.875% and 100%, with the average utilization of 85% in our simulations and the resource value of T_1 is unchanged.

4. Methods and simulations

Our rule for resource modification operates during the simulation and at each period when the task executes all the units of its resource, but one. And, when the remainder of the resource for a task equals 1, the scheduler assigns a new resource to the task, as per the rules defined above.

For the dynamic priority driven scheduling algorithms of EDF and LLF, we constructed our simulations based on successive schemes:

- EDF with random resource on periodic, sporadic and Poisson tasks;
- EDF with static resource on periodic, sporadic and Poisson tasks;
- LLF with random resource on periodic, sporadic and Poisson tasks;
- LLF with static resource on periodic, sporadic and Poisson tasks.

The results we get with these schemes are shown in Table 3–Table 6, showing the average values we got from the simulations as per the scheme sequence above.

Table 3. Results of EDF with Random Resource on periodic, sporadic and Poisson tasks.

Alg	Type: Random Resource										
Earliest Deadline First	Arr Pt		Periodic			Sporadic			Poisson		
	Param		SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
	Res		20%	23	3.2	40%	21.5	3.7	30%	22.1	4.2
	Average Wcet	T_1	1			1.5			2.2		
		T_2	8.9			9			10.7		
		T_3	9.6			9			12.1		
		T_4	23.4			17.1			26.4		

On looking at these results, we make following inferences:

1) Both algorithms are unstable with the changing resources, and the total schedulability of the system is lower for all arrival patterns because of the worst case execution time values of each task is worse than those under static resource conditions. We can add the overhead of computations to assign newer resource values to tasks. The context switch and preemption numbers are slightly reduced but there is not a significant change overall.

2) EDF algorithm works better than the LLF algorithm under Poisson Task arrivals. This situation is not changed under random resource need conditions, as seen from Table 4 and Table 6 with schedulability rates of 30% and 0%.

Table 4. Results of EDF with Static Resource on periodic, sporadic, Poisson tasks.

Alg	Type: Static Resource										
Earliest Deadline First		Arr	Periodic			Sporadic			Poisson		
		Param	SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
		Res	100%	31	2	100%	28	6.6	30%	24.7	7.2
	Average Wcet	T_1	1			1			1		
		T_2	4			4			8		
		T_3	8			7.7			19.2		
		T_4	31			19.4			36.7		

Table 5. Results of LLF with Random Resource on periodic, sporadic and Poisson tasks.

Alg	Type: Random Resource										
Least Laxity First		Arr	Periodic			Sporadic			Poisson		
		Param	SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
		Res	20%	25	3.8	20%	21.9	3.2	20%	22.7	7.7
	Average Wcet	T_1	1			1.3			1.7		
		T_2	9.9			8.9			10.4		
		T_3	9.6			9.6			14.3		
		T_4	23.3			17.4			24.2		

3) As we tried to understand the reason why schedulability lowers under random resource need conditions, we ended up with the reason that one or two tasks, despite their often higher priority, became stuck and did not complete within the time and resources allocated: under static resource conditions one can know beforehand if the tasks will fail or not considering the total utilization of the system, while this cannot be guessed for random resource need conditions.

Table 6. Results of LLF with Static Resource on Periodic, Sporadic, Poisson Tasks.

Alg	Type: Static Resource										
Least Laxity First	Arr Pt		Periodic			Sporadic			Poisson		
	Param		SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
	Res		100%	35	6	100%	27.6	5.7	0%	24.7	6.6
	Average Wcet	T_1	1			1			1		
		T_2	4			4			8.5		
		T_3	14			8			21.8		
		T_4	31			22.1			33.5		

To overcome these issues and make the random resource condition a predictable one we developed a new scheduler, one able to provide steady scheduling schemes with bounded processor utilizations as well as able to make better use of resources by incrementing the total processor load without deadline misses.

4.1. Scheduler providing steady and bounded utilization with resource modification

Our scheduler provides a “Steady and Bounded Utilization with Resource Modification” (SBU) and is applied together with the EDF and LLF dynamic priority driven scheduling algorithms.

It works algorithmically as follows.

Parameters:

$T_i(i \in [1, n])$, HP, U, $W(t)$, $B(t)$, $W_{rk}(t)$, $RoR(RestofResource)$

Algorithm:

```
For all tasks in the task set
  For tasks having resources greater than 1
    If Rest of Resource (RoR) for a task equals to 1
      If the Resource Need ($W_{rk}$ (t)) for the task is greater than 0
        If the Available (Idle) Time (B(t)) is greater than or equals to Resource Need
          If [Processor Availability (B(t)) - Resource Need ($W_{rk}$ (t))] is
          greater than half of the Resource Need
            Increment $R_i$
          Else
            $R_i$ remains the same
        Else If the Available (Idle) Time (B(t)) is smaller than Resource Need
          Decrement $R_n$ (n is the other tasks in the task set) until they reach back
          to their initial values
End all
```

We consider the Processor Availability for each task between the simulation time and the simulation range. If the Available Time is suitable for a task to increment (1.5 times more than the Resource Need) the resource allocated to the task is incremented. And if we end up with a situation that the available time is not enough for that task to accomplish, the resource incrementation made for other tasks in the task set are taken back and decremented to provide more available time for the task which is about to finish its activation. When we applied this algorithm to the task sets with utilizations less than 1, we got the results that processor utilization extends and shrinks according to the dynamic situations as we mentioned. We remark the fact that the task resources never fall under the initial resource values, and the decrementation is made just for the incremented resources.

The results with the same task set are shown in Table 7 and Table 8.

Table 7. Results of EDF with SBU on Periodic, Sporadic, Poisson Tasks.

Alg	Steady And Bounded Utilization										
Earliest Deadline First		Arr	Periodic			Sporadic			Poisson		
		Param	SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
		Res	100%	63	4	90%	52.4	10.6	10%	50.2	9.4
	Average Wcet	T_1	1			2.4			3		
		T_2	4			5.1			14.4		
		T_3	8			8			25.3		
		T_4	31			24.2			41.7		

If we analyze these newer results for the new scheduler, we can make following inferences:

1) The schedulability is much better than that of random resource scheduler under periodic and sporadic tasks. The schedulability values for SBU with EDF are 100% and 90% under periodic and sporadic tasks, respectively. This schedulability rate is much better considering the schedulability rates of 20% and 40% under periodic and sporadic task arrival patterns respectively for Random Resource Scheduler. The results for Poisson Task arrivals are 10% to 30% for “SBU with EDF” and “Random Resource with EDF,” respectively. When we consider values for LLF algorithm, the results are 100% and 80% under periodic and sporadic arrival patterns which are much better than the results of 20% and 20% for the same sequence with Random Resource Scheduler. As seen from the results, for Poisson Task arrival pattern, no improvements are obtained as the schedulability rates of 0% to 20% for “SBU with LLF” and “Random Resource with LLF,” respectively.

2) EDF and LLF algorithms with SBU Scheduler do not solve the lower schedulability problem under Poisson Task arrivals.

3) We can make the inference that our scheduler works well under Random Resource conditions, taking the schedulability of the system to higher rates by extending and shrinking the task resources.

Table 8. Results of LLF with SBU on Periodic, Sporadic, Poisson Tasks.

Alg	Steady And Bounded Utilization										
Least Laxity First	Arr		Periodic			Sporadic			Poisson		
	Param		SCH	CS	PR	SCH	CS	PR	SCH	CS	PR
	Res		100%	74	14	80%	52.7	12.7	0%	74	28.9
	Average Wcet	T_1	1			3.6			3.1		
		T_2	6			5			15.1		
		T_3	15			9			24.2		
		T_4	31			20.4			37.7		

Another important property of our scheduler is that it provides a better utilization by utilizing the unused time in the scheduling scheme. For a better explanation of this issue, we have chosen a task set having a total processor utilization of 75%. Task set is seen in Table 9.

Table 9. Tasks set for testing utilization incrementation parameter for SBU.

Task	Period	Resource	Utilization
T_1	4	1	0.25
T_2	8	2	0.25
T_3	16	2	0.125
T_4	32	4	0.125
Total Utilization			0.75

The hyper period for this task set is 32 ($LCM(4, 8, 16, 32)$) and we used a simulation range of 0—128, which is 4 times greater than the hyper period.

With EDF applied to this task set we had the total processor utilization of 75% and schedulability rate of 100%, but we have 25% of the processor resources unused meaning that 32 units of time in 128 time units which are our simulation time range.

We wanted to get the total processor utilization to some values greater than existing algorithms while providing the same schedulability rate. As seen from Table 10, Figure 1 and Figure 2; with our scheduler we managed to accomplish the issues we wanted. We got to the results of the same schedulability rates (100%) for each simulation while providing a total processor utilization of more than 90% in each algorithm and for each simulation range which is much better than the existing algorithms processor utilization.

Table 10. Resource and utilization incrementation by SBU applied to dynamic priority driven scheduling algorithms of EDF and LLF.

	SIM	ID	U	CS	PR		SIM	ID	U	CS	PR
EDF SBU	32	2	93.7	18	4	LLF SBU	32	2	93.7	18	4
	64	3	95.3	34	5		64	3	95.3	39	7
	128	5	96.1	66	7		128	5	96.1	75	10
	256	10	96.1	127	9		256	10	96.1	144	11
	512	18	96.5	255	17		512	18	96.5	288	19
	SIM	ID	U	CS	PR		SIM	ID	U	CS	PR
EDF	32	8	75.0	18	4	LLF	32	8	75.0	18	4
	64	16	75.0	36	8		64	16	75.0	36	8
	128	32	75.0	72	16		128	32	75.0	72	16
	256	64	75.0	144	32		256	64	75.0	144	32
	512	128	75.0	288	64		512	128	75.0	288	64

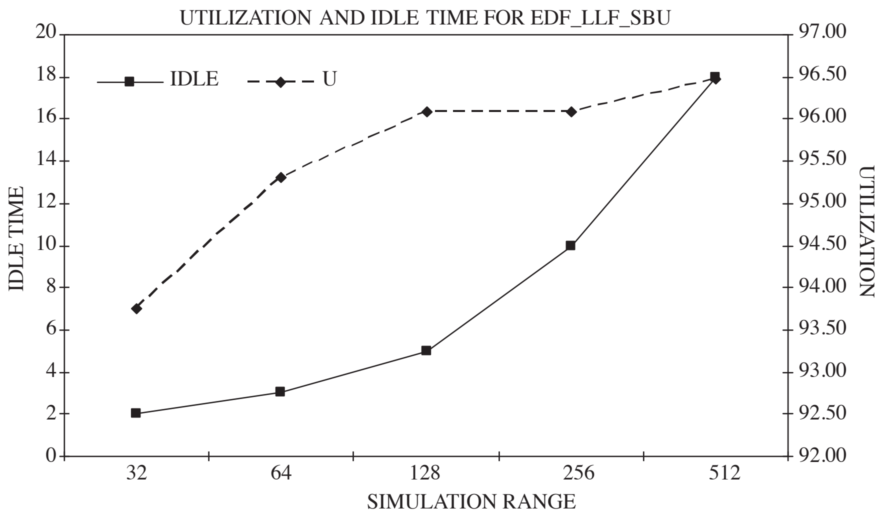

Figure 1. Processor idle time and utilization for the algorithm SBU with the dynamic priority driven scheduling algorithms of EDF and LLF (Note that both algorithms end up with the same results).

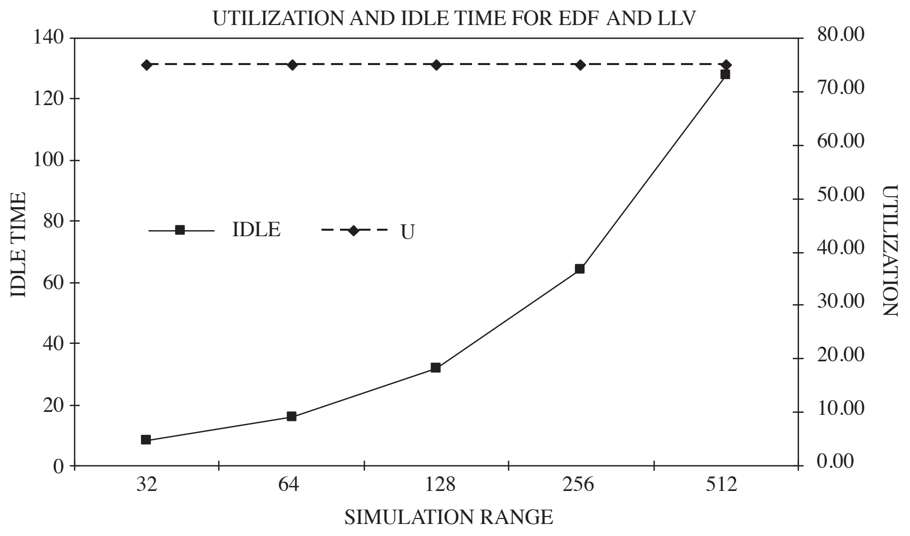

Figure 2. Processor idle time and utilization for the dynamic priority driven scheduling algorithms of EDF and LLF (Note that both algorithms end up with the same results).

5. Conclusions

The dynamic situations that we come across in hard, real-time systems are the ones we need to cope with. To understand the needs of the dynamic systems we have to consider all the parameters and try to respond to the requests of the real world. In our work we made our research on resource issue and tried to answer the resource modification requests of the real world systems. Generating a reasonable task set and applying the random resource issue to the dynamic priority driven scheduling algorithms of EDF and LLF we got the result that the system is unstable under random resource conditions and has lower schedulability.

To overcome that issue and to have a better utilization for systems which are not highly loaded (meaning that having processor utilization below 1.00) we developed a resource modification scheme providing a steady and bounded utilization for the task set. We accomplished to provide a better utilization (over 90%) and 100% schedulability with the dynamic schedulers of EDF and LLF which provide only the initial utilization (if the initial utilization is 75% as in the example, that load does not change with time) with the same schedulability rate.

Our proposed scheme provides a better utilization by extending and shrinking the resource parameter according to the system workload which is calculated in each cycle, calculating the idle time in the scheduling scheme and comparing that with the task's need for completing its activations before deadline. This provides all the tasks to finish before deadline and our scheme never assigns a lower resource value than the initial resources for all tasks.

References

[1] D. Stewart, P. Khosla, Real-Time Scheduling of Sensor-Based Control Systems in Real-Time Programming, Ed. by W. Halang and K. Ramamritham, Tarrytown, New York, Pergamon Press Inc., 1992.

[2] P. Marti, R. Villa, J. M. Fuertes, G.. Fohler, "On real-time control tasks schedulability", 2001.

[3] E. Bini, G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems", IEEE Transactıons on Computers, Vol. 53, No. 11, pp. 1462–1473, 2004.

[4] K. Albers, F. Slomka, "Efficient feasibility analysis for real-time systems with EDF scheduling", Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05), IEEE Computer Society, 2005.

[5] P. M. Alvarez, H. Aydin, D. Mosse, R. Melhem, "Scheduling optional computations in fault-tolerant real-time systems", Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications, pp. 323, 2000.

[6] R. Samet, "Fault-tolerant procedures for redundant computer systems". Int. J. Quality and Reliability Engineering International, Vol. 25, pp. 41–69, 2009.

[7] C. Han, K. G. Shin, J. Wu, "A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults", IEEE Transactions on Computers, Vol. 52, No. 3, pp. 362–372, 2003.

[8] G. Manimaran, C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 11, pp. 1137–1152, 1998.

[9] G. M. A. Lima, A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," IEEE Transactions on Computers, Vol. 52, No. 10, pp.1332–1346, 2003.

[10] J. Locke, "Designing real-time systems," In IEEE International Conference of Real-Time Computing Systems and Applications (RTCSA '97), Taiwan (Invited talk), 1997.

[11] C. L. Liu, 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. 46–61, 1973.

[12] J. L. Lehoczky, et al. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," In Proceedings 10^{th} IEEE Real-Time Systems Symposium, Santa Monica, CA, pp. 166–171, 1989.

[13] S. Lauzac, R. Melhem, D. Mosse, "An improved rate-monotonic admission control and its applications". IEEE Transactions on Computers, Vol. 52 (3), 2003.

[14] Y. Chen, G. Xiong, "Imprecise computation fault-tolerant rate-monotonic scheduling". Computer Science and Engineering College Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02) IEEE, 2002.

[15] S. H. Oh, S. M. Yang, "A modified least-laxity-first scheduling algorithm for real-time tasks," Proceedings of the Fifth International Conference on Real-Time Computing Systems and Applications, pp. 31–36, 1998.

[16] W. Zhang, S. Teng, Z. Zhu, X. Fu, H. Zhu, "An improved least-laxity-first scheduling algorithm of variable time slice for periodic tasks", Proc. 6th IEEE Int. Conf. on Cognitive Informatics (ICCIO07), IEEE, 2007.

[17] J. Y. T. Leung, M. L. Merril, "A note on preemptive scheduling of periodic", Real-Time Tasks Information processing Letters, Vol. 11, 1980.

[18] S. K. Baruah, A. K. Mok, A. K. Rosier, "Preemptively scheduling hard real-time sporadic tasks on one processor", Proceedings of the 11th Real-Time Systems Symposium, pp. 182–190, 1990.

[19] J. Goossens, C. Macq, “Limitation of the hyper-period in real-time periodic task set generation,” 9th Conference RTS embedded systems, pp. 133-148, 2001.

[20] G. Buttazzo, “Rate monotonic vs. EDF: Judgment day real-time systems,” Vol. 29, pp. 5–26, 2005.

[21] L. George, N. Rivierre, M. Spuri, “Preemptive and non-preemptive real-time uniprocessor scheduling,” INRIA Research Report, 2966, 1996.