International Journal of Computer and Communication Technology

Volume 4 | Issue 4 Article 8

October 2013

Reduction of Context Switches due to Task Synchronization in Uniprocessor and Multiprocessor Platform

Arya Paul
Department of Electrical and Electronics Engineering Amrita Viswa Vidyapeetham Ettimadai, Coimbatore, India, aryapaul114u@gmail.com

Anju S. Pillai
Dept. of Electrical and Electronics Engineering Amrita School of Engineering, Coimbatore, India, s_anju@cb.amrita.edu

Reduction of Context Switches due to Task Synchronization in Uniprocessor and Multiprocessor Platform

Arya Paul[1], Anju.S.Pillai[2]
Department of Electrical and Electronics Engineering
Amrita Viswa Vidyapeetham
Ettimadai, Coimbatore, India
E-mail : aryapaul114u@gmail.com[1],s_anju@cb.amrita.edu[2]

***Abstract*—The problem of frequent context switches in multitasking is a real scheduling overhead which wastes extra CPU cycles, memory and causes much delay in scheduling. This paper focuses on reducing the context switches that result due to blocking when jobs are required to synchronize. The Priority Ceiling Protocol (PCP) is used to synchronize the tasks in uniprocessor as well as multiprocessor platforms. The jobs are scheduled using Earliest Deadline First (EDF) policy. The simulation results show that the context switches are reduced by about 20% on an average using our technique of avoiding context switches due to blocking.**

Keywords-context switches; synchronize; priority ceiling protocol; uniprocessor; multiprocessor

I. INTRODUCTION

In real time systems, the correct output of the system depends not only on the computational results but also on the time at which output is generated. In a multitasking system, where the CPU switches among different tasks, predictability and schedulability are the two criteria that need to be met. Predictability is in terms of tasks meeting their timing requirements and schedulability is with respect to the resource utilization at which timing requirements can be met. In real life scenario, not all the processes are independent. They usually depend on each other for sharing some data. Data inconsistency may occur because of concurrent access to shared data by tasks. There are several synchronization policies or protocols to maintain data consistency.

In priority based scheduling, a task with a higher priority can preempt a task with a lower priority. The necessary context associated with the currently running task needs to be saved in memory and has to be retrieved when the task resumes execution later. Preemptive scheduling offers several advantages over non-preemptive scheduling such as flexibility, improved processor utilization etc. But it results in extra CPU cycles and memory requirements due to unnecessary context switches. Context switches, if not properly controlled, may lead to reduced responsiveness, unnecessary delays, energy wastage and extra memory requirements. All these can lead to high overhead in real time embedded systems. When tasks are required to synchronize, more context switches can occur. For example, a task A has to give the CPU control to another task B if A is denied the access to shared resource that task B holds. Such preemption may cause high energy consumption which is a real threat to scheduling.

Our present work focuses on reducing the preemptions resulting from synchronization of tasks, both in uniprocessor and multiprocessor. The method does not allow a high priority task to preempt a low priority task if it will be blocked by the low priority task in the future. The synchronization protocol used here is Priority Ceiling Protocol. The PCP is proved to prevent deadlock and uncontrolled priority inversion that may occur when tasks are synchronized.

II. RELATED WORK

The scheduling problem of periodic real time tasks was first studied by Liu and Layland in 1973 [1]. They found out a necessary and sufficient condition for scheduling periodic tasks. Mok [6] showed that the problem of deciding whether it is possible to schedule a set of periodic tasks when tasks share resources is NP-hard. The two protocols (Basic Priority Inheritance Protocol and Priority Ceiling Protocol) that belong to the family of priority inheritance protocols were first investigated in [2]. The Priority Ceiling Protocol is proved to prevent deadlock and reduce priority inversion duration to atmost one critical section [2]. Another optimal policy, called Optimal Mutex Policy was introduced in [7]. This policy provides necessary and sufficient condition for avoiding deadlocks and limiting the blocking to a single critical section. The synchronization in multiprocessors using priority ceiling protocol was introduced in [5]. The multiprocessor priority ceiling protocol in [5] bounds the duration of blocking and prevents deadlocks.

There were several works that dealt with reducing the number of preemptions. In [8], the aim was to reduce the preemptions in a fixed priority schedule without synchronization. Their method changed the attributes of task instances to reduce preemptions.

III. UNIPROCESSOR PRIORITY CEILING PROTOCOL

The priority ceiling protocol is the most effective protocol that uses priority inheritance. The protocol uses the binary semaphore concept to achieve synchronization. Each and every shared resource is protected by a lock or a semaphore. A task can access the shared resource only if the semaphore is available or, in other words, the resource is not locked by any other tasks. Using the Basic Priority Inheritance Protocol, tasks can be synchronized efficiently, but deadlocks may occur in certain situations. The deadlocks and the uncontrolled priority inversion delay can be prevented using Priority Ceiling Protocol. A priority ceiling is defined for each semaphore, which is the priority of the highest priority task that can lock this semaphore. Under PCP, a task will lock a semaphore only if its priority is higher than any of the priority ceilings of the semaphores currently locked by other tasks [2]. The only additional information needed to implement this protocol is the ceilings of all semaphores in the system. Generally, a task gets blocked if it needs to lock a semaphore and its priority is lower than the ceiling of some locked semaphore. This task can resume only when the blocking task releases its semaphore [2].

IV. REDUCTION OF PREEMPTIONS DUE TO BLOCKING

Unnecessary context switches may occur due to blocking of a task. Context switch occurs, when a high priority task gets blocked and it gives the control of the CPU to a low priority one.

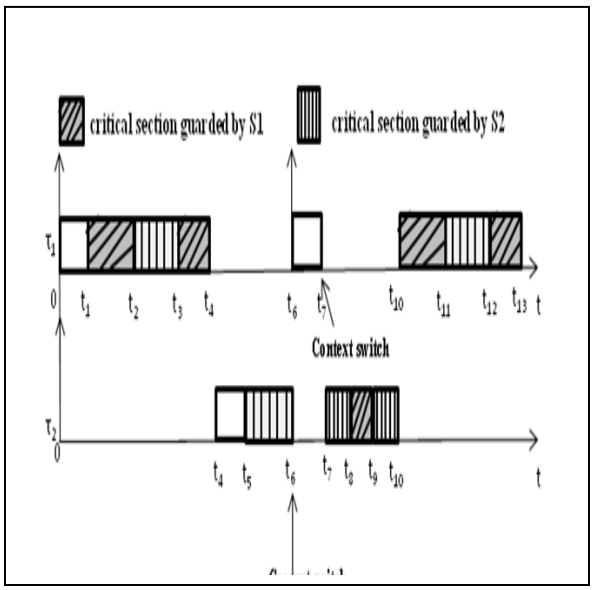


Figure 1. Context switches due to blocking in uniprocessor synchronization.

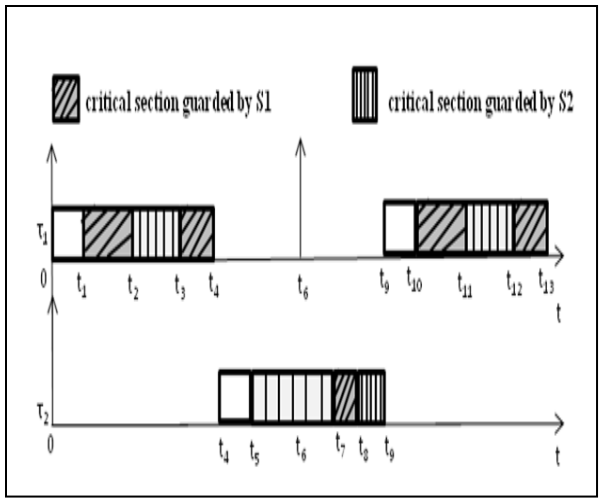


Figure 2. Avoidance of context switches.

Later when the low priority task leaves its critical section the control of CPU is given back to the high priority task. Thus every blocking causes two context switches. We can avoid this situation if we do not allow a high priority task to preempt a low priority task if the high priority task will be blocked in the future. A task A is allowed to preempt the currently running task B on the CPU if and only if the priority of A is higher than that of B and none of the semaphores it will attempt to lock is currently locked by other tasks [3]. Hence a task will not be running on the CPU if it will be blocked later by a low priority task. Thus unnecessary context switches due to blocking can be avoided. The main advantage of this concept is that the task completion times will not get affected [3].

The concept of reducing the context switches due to blocking can be incorporated into the priority ceiling protocol when tasks are scheduled on an EDF basis as follows: A task P is allowed to preempt the currently running task on the CPU if and only if the following two conditions are satisfied.

1) The absolute deadline of P is less than that of running task.

2) The preemption level of P is greater than the ceiling of the semaphore (S) which has got the highest priority ceiling of all the semaphores currently locked by other tasks in the system [3].

The concept of preemption level was introduced by Baker in [4]. The preemption level of each task is chosen as inversely proportional to its deadline. If the second condition is not satisfied, P is said to be blocked on semaphore S and by the task P' which holds the lock on S. At the same time P' will inherit the priority of P until it completes the critical section that caused the block on P. When P' completes the critical section, it resumes its previous priority and the highest priority task will be selected to run [3]. This method reduces the number of

context switches due to blocking taking the advantage that the PCP eliminates deadlock and multiple blocking.

A. *Schedulability Analysis in Uniprocessor Synchronization*

A set of periodic tasks can be scheduled by EDF using priority ceiling protocol if the following condition [4]

$\forall i: i = 1,...,n,\ (C_1 / T_1) + \ldots + (C_i / T_i) + (B_i / T_i) \leq 1$ (1)

is satisfied, where n is the number of periodic tasks, C_i and T_i are the execution time and period of task τ_i, B_i is the blocking duration of task τ_i which is the maximum of the length of the critical sections guarded by those semaphores which can be locked by tasks having lower priority than task τ_i and whose ceilings are greater than or equal to the priority of task τ_i.

B. *Example Task Set*

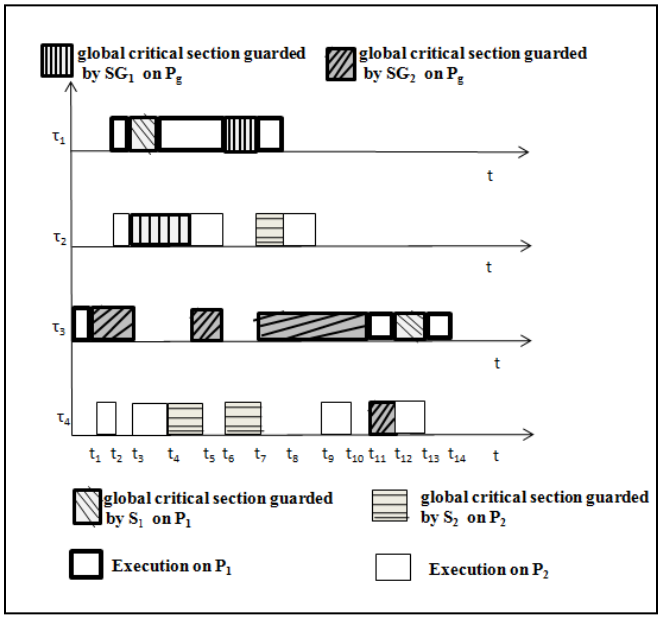
The context switch that occurs due to blocking of tasks in a uniprocessor is shown in Fig. 1 which uses priority ceiling protocol for synchronization. There are two periodic tasks τ_1 and τ_2 each of which has got two critical sections guarded by semaphores S_1 and S_2. Task τ_1 has got higher priority than task τ_2. Let the preemption level of task τ_1 be 2 and task τ_2 be 1. Then the priority ceiling of both semaphores is 2. At time tick t_5 task τ_2 locks semaphore S_2 and at tick t_6 the high priority task τ_1 arrives. The task τ_1 preempts τ_2 at tick t_6. At tick t_7 task τ_1 requires to lock the semaphore S_1 but the preemption level of τ_1, which is 2, is not greater than the priority ceiling of currently locked semaphore S_2, which is also equal to 2. Task τ_1 gets blocked by τ_2. So another context switch occurs at tick t_7. At tick t_{10} task τ_2 releases semaphore S_2 and task τ_1 locks S_1 and executes.

In Fig. 2, the two context switches that occur due to blocking are avoided. At tick t_6 task τ_1 will not preempt τ_2 because the preemption level of τ_1 is not greater than the ceiling of currently locked semaphore S_2. At tick t_9 task τ_2 releases S_2 and task τ_1 will take control of CPU.

V. MULTIPROCESSOR TASK SYNCHRONIZATION

In multiprocessor scheduling, when tasks share resources, the concept of blocking differs from uniprocessor scheduling. When a task in one processor needs to wait for the execution of another task assigned to another processor, the former task is said to experience remote blocking. Tasks assigned to different processors may access the same semaphore. Such semaphores are called global semaphores and the critical section guarded by global semaphore is called global critical section [5]. The semaphore accessed by tasks allocated to a single processor is called local semaphore and the critical section guarded by local semaphore is called local critical section [5].

The global critical section of a task can be executed on a processor other than the host processor of the task. Local critical sections and non-critical sections are executed on the host processor of the task. All the global critical sections associated with the same global semaphore must be bound to the same processor [5]. Thus each global semaphore is



bound to a synchronization processor and all the tasks that need to lock the global semaphore have to execute their

Figure 3. Multiprocessor synchronization.

Corresponding global critical section on the synchronization processor. The processors that execute global critical section are called synchronization processors and which execute application tasks only are called application processors. A synchronization processor can also execute application tasks.

When a task executes global critical section, it needs to migrate to its synchronization processor, during which the host processor is free and can be used to run a low priority job. When the task exits its global critical section, it migrates back to its host processor to execute non-critical section. The global critical section needs to run at a priority greater than the highest assigned priority to tasks.

The multiprocessor priority ceiling protocol [5] is as follows:

1) The priority ceiling protocol is made to run on the set of tasks and the set of local semaphores assigned to application processors.

2) The global critical section is made to run at its assigned priority in the synchronization processor.

3) The priority ceiling protocol is made to run on the global critical sections, set of application tasks (if any) and the set of global and local semaphores bound to synchronization processors.

A. *Schedulability Analysis in Multiprocessor Synchronization*

The set of inequalities in (1) can be tested for each processor in the multiprocessor synchronization. Let n be

the number of tasks bound to a processor P, C_k and T_k be the execution time and period of task τ_k bound to P, G_i be the number of global critical sections that a task τ_i executes before its completion [5]. The term B_i is the maximum duration that a task τ_i bound to P can be blocked. The blocking time B_i is the sum of the following terms [5]:

1) Duration of G_i+1 local critical sections on P that can block task τ_i.

2) Duration of G_i global critical sections of lower priority tasks on synchronization processor if τ_i accesses a global semaphore.

3) Duration of global critical sections of higher priority tasks bound to processors other than P if τ_i accesses a global semaphore.

4) Duration for the deferred execution requests of higher priority tasks on P.

B. Example Task Set

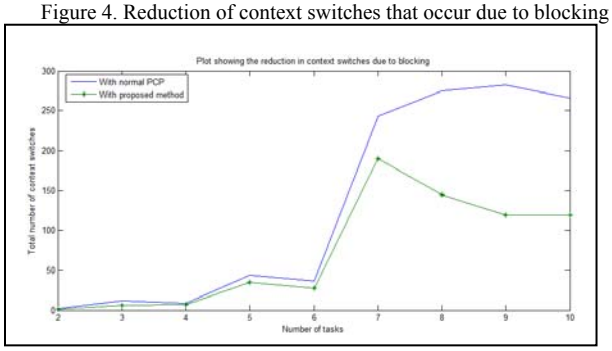
In Fig. 3 there are 4 tasks τ_1, τ_2, τ_3 and τ_4 in which τ_1 and τ_3 are bound to application processor P_1 and τ_2 and τ_4 are bound to application processor P_2. The local semaphores S_1 and S_2 are bound to $P_1$and P_2 respectively. Let the global semaphores SG_1 and SG_2 be bound to synchronization processor P_g. The local priority ceilings of local semaphores S_1 and S_2 are the priorities or preemption levels (P') of tasks τ_1 and τ_2 respectively. Let us define P'_b (base priority ceiling) = P'_1+1. The priority ceilings of SG_1 and SG_2 are assigned as $P'_b + P'_1$ and $P'_b + P'_3$ respectively. The global critical section of task τ_i is assigned a priority of $P'_b + P'_i$.

At time t_3 task τ_1 attempts to lock S_1.The local priority ceiling protocol grants the lock since there is no other locked semaphore local to P_1. Task τ_2 attempts to lock SG_1. Using the priority ceiling protocol on P_g, since the assigned priority of global critical section of τ_2 is greater than the global priority ceiling of currently locked semaphore SG_2 the global critical section of τ_2 is allowed to preempt global critical section of τ_3 on P_g.

The context switches occurring due to blocking in multiprocessor synchronization can be reduced in the same way as in uniprocessor synchronization. In multiprocessor scheduling, the context switch that occurs when a task in one processor gets blocked by another task in the same processor is considered and reduced by the above proposed method.

VI. SIMULATION RESULTS

Figure 4. Reduction of context switches that occur due to blocking



The simulation has been done in Matlab. Fig. 4 shows the reduction in context switches when the proposed method is used. The above solid line in Fig. 4 represents the total number of context switches that occur using normal PCP and the below line represents the number of context switches that occur when the proposed method is used. The results show that about 10 to 20% of context switches can be reduced using this method.

The simulation results obtained when tasks are synchronized and scheduled in multiprocessors are shown in Fig. 5. There are three processors, five tasks and three semaphores. The host processor of task 1 and 2 is processor 1 and that of 3 and 4 is processor 2. The host processor of task 5 is processor 3. Tasks 1 and 2 are scheduled on processor 1 as shown in Fig. 5(a). The shaded portions represent the critical sections.

The semaphore S_1 and S_2 are global semaphores. The global priority ceilings of these semaphores are 11 and 9 respectively. S_3 is a local semaphore and its local priority ceiling is 3. All tasks access S_1 and the critical sections guarded by S_1 are executed on processor 3 as shown in Fig. 5 (c). Tasks 3 and 4 are scheduled on processor 2 as shown in Fig. 5(b). Processor 3 acts as a synchronization processor.

In processor 2, task 3 preempts task 4 at time tick 1 as shown in Fig. 5(b). But at tick 3, task 3 needs to lock semaphore S_2. Since the preemption level of task 3 is not greater than the local priority ceiling of S_3, task 3 is denied the access to the lock. Task 4 preempts task 3 and later when task 4 releases S_3, task 3 acquires lock and begins to run.

In processor 2, task 3 does not preempt task 4 at time tick 1 as shown in Fig. 5(d) since the preemption level of task 3 is not greater than the local priority ceiling of locked semaphore S_3.

Thus the context switches occurring due to blocking in multiprocessor synchronization is reduced in the same way as in uniprocessor synchronization.

The task set used for synchronization in multiprocessors is shown in Table I.

TABLE I: TASK SET USED FOR SYNCHRONIZATION IN MULTIPROCESSORS

Task	Execution Time	Period	Preemption level	Semaphores used
1	4	10	5	{ S_1}
2	3	20	4	{ S_1}
3	10	30	3	{S_1, S_2, S_3}
4	8	60	2	{ S_1, S_3}
5	35	120	1	{ S_1, S_2}

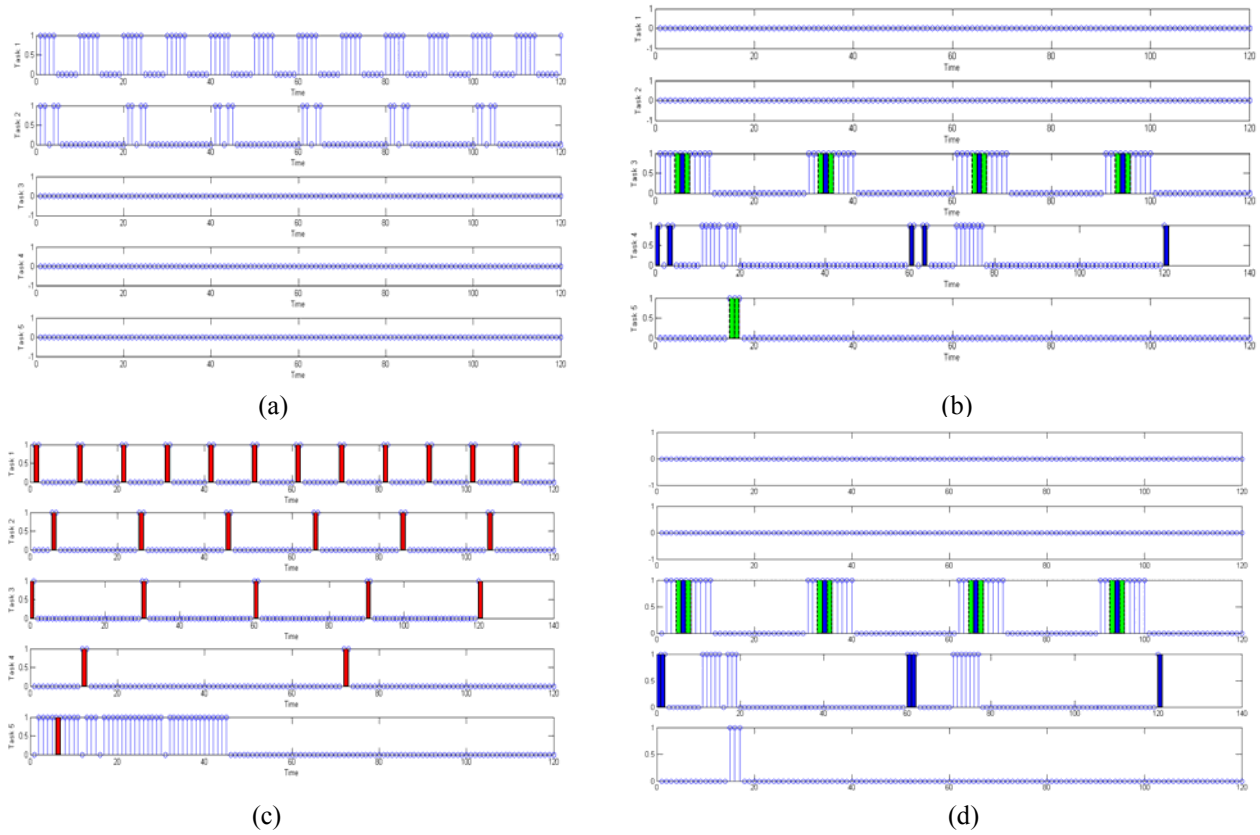


Figure 5. Multiprocessor scheduling : (a) Scheduling on processor 1 (b) Scheduling on processor 2 (c) Scheduling on processor 3 (d) Avoidance of context switches in processor 2.

VII. CONCLUSION

In this paper, we have presented a solution for reducing the number context switches that occur due to blocking of tasks when tasks are required to synchronize. We are able to reduce context switches by about 20% on an average in both uniprocessor as well as multiprocessor platforms. The priority ceiling protocol utilizing binary semaphores is used for synchronization in both uniprocessor and multiprocessor scheduling. The advantage of this concept is that the task completion times are not delayed. Also deadlocks can be prevented and priority inversion delay can be minimized by using priority ceiling protocol.

As future work, the context switches arising due to remote blocking in multiprocessors need to be reduced. Also the cache related problems arising due to context switches need to be analyzed.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Dr. T. N. Padmanabhan Nambiar, Head of Department, Electrical and Electronics Engineering for providing us affectionate guidance.

REFERENCES

[1] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a Hard Real Time Environment," Journal of the ACM, vol.20,no.1,1973.

[2] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," IEEE Transactions on Computers, pp. 1175-1185, Sept. 1990.

[3] Albert M.K. Cheng and Fan Jiang, "An Improved Priority Ceiling Protocol to Reduce Context Switches in Task Synchronization," Technical Report, Computer Science Department, University of Houston, 2005.

[4] T.P. BAKER, "Stack-Based Scheduling of Realtime Processes," The Journal of Real Time Systems, 3, 67--99 (1991).

[5] Ragunathan Rajkumar, Lui Sha and John P.Lehoczky, "Real time synchronization protocols for multiprocessors," Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS), pp. 259-269,1988.

[6] A. K. Mok, "Fundamental design problems of distributed systems for the hard reak time environment," PhD Thesis, M.I.T., 1983.

[7] R. Rajkumar, L. Sha, J. P. Lehoczky, and K. Ramamritham, "An optimal priority inheritance policy for synchronization in real-time systems," Advances in

Real-Time Systems, pp. 249-271, New Jersey: Prentice Hall, 1995.

[8] Radu Dobrin. and Gerhard Fohler, "Reducing the Number of Preemptions in Fixed Priority Scheduling," EuroMicro Conference on Real-Time Systems, 2004.