Scheduling of self-suspending tasks: state of art and new insights

Frédéric Ridouard, Pascal Richard
LISI-ENSMA and University of Poitiers
{frederic.ridouard,pascal.richard}@ensma.fr

1 Introduction

Most of real-time systems contain tasks with self-suspension. A task with a self-suspension is a task that during its execution prepares specifics computations (e.g. In/Out operations or *FFT* on a digital signal processor). The task is self-suspended to execute the specifics computations upon external dedicated processors. External operations introduce self-suspension delays in the behavior of tasks. The task waits until the completion of the external operations to finish its execution. Generally, the execution requirement of external operations can be integrated in the execution requirement of the task. But, if self-suspension delays are large, then such an approach cannot be used to achieve a schedulable system. Thus, self-suspension must be explicitly considered in the task model.
We consider sporadic tasks with self-suspension. Let I be a task system of n tasks. Every occurrence of a task is called a *job*. Every task τ_i ($1 \leq i \leq n$) arrives in the system at time 0, its relative deadline is denoted D_i and its period T_i. We assume that tasks are subjected to constrained-deadlines ($D_i \leq T_i$). Tasks are allowed to self-suspend at most once. Every task τ_i ($1 \leq i \leq n$) has two subtasks (with a maximum execution requirement $C_{i,k}$, $1 \leq k \leq 2$) separated by a maximum self-suspension delay X_i between the completion of the first subtask and the start of the second subtask. Such delays change from one execution to another since they model execution requirements of external operations. Consequently every task τ_i is denoted: $\tau_i : (C_{i,1}, X_i, C_{i,2}, D_i, T_i)$.
Only few positive results have been defined for schedulability analysis of self-suspending tasks. Most of them exploit particular task sets and restrictive assumptions. But to the best of our knowledge, no solution provides simultaneously results for efficiently scheduling and analysing self-suspending tasks. In this note, we present the state of the art about the self-suspending tasks. Then, we present some possible insights to define solutions to this scheduling problem.

2 Known results

It has been already proved in [9, 8] that the feasibility problem of scheduling self-suspending task systems is $\mathcal{NP}$-Hard in the strong sense. We have also shown the presence of scheduling anomalies under fixed priorities and *EDF* for scheduling independent tasks with self-suspension upon an uniprocessor platform when preemption is allowed. We have also proved that classical on-line scheduling algorithms (*EDF*,*LLF*,*RM*,*DM*) are not better than 2-competitive to minimize the maximum response time and not competitive to minimize the number of tardy tasks. Response Time Analysis for fixed-priority scheduling algorithm *RM*, [2, 6] have been proposed for computing response time upper bounds. Finally, we have also shown that it is impossible to define an optimal on-line algorithm to schedule sporadic tasks systems when tasks are allowed to self-suspend (cf. [8]).
In [3], the authors characterize the exact critical instant for self-suspending sporadic tasks. They deduce a pseudo-polynomial response-time tests for analysing the schedulability of such self-suspending tasks. In [5], the periodic tasksets with suspensions, pipelines, and non-preemptive sections are considered. The authors show how to transform such a task system into a periodic taskset with only suspensions. Then, they use prior results [4] to derive tardiness bounds for more complex systems.

From a practical scheduling point of view, in [1] is presented a configurable synchronization protocol for self-supending process sets. In fact, the protocol extends the concept of priority ceilings. Furthermore, an algorithm for computing the corresponding maximum blocking times is presented.

3 New insights

We next present some ideas to develop in order to achieve valuable positive results on scheduling self-suspending tasks:

Particular task characteristics. Most of negative results are based on instance problems in which suspension delays are quite huge. A recurrent problem with the scheduling of self-suspending tasks is the duration of suspension delays since it can be so important that the computation time is negligible. Studying particular task set, while fixing some properties of task parameters usually help to understand the difficulties encountered while solving a general complex problem. For that purpose, we think that it is important to investigate some particular cases such as: suspension delays cannot exceed processing time for each task, suspension delays are all equal a constant, suspension delays are all equal to 1, etc. It is surely one way to achieve some positive results.

Resource augmentation technique. Classical scheduling algorithms are not optimal for scheduling tasks allowed to self-suspend. An important question is: is there a processor speed s so that a classical scheduling algorithm (e.g., *RM* or *EDF*) will lead to a feasible schedule if one exists upon a unit-speed processor (i.e., computed by an optimal off-line scheduling algorithm).

Impact of scheduling anomalies. What is the impact of considering only worst-case execution time and suspension delays while performing a schedulability analysis? Is-it possible to establish simular results as these one obtained by Mok et al. in [7] that analyses the robustness of non-preemptive scheduling for *RM* and *EDF*. They proved that scheduling anomalies can lead to miss at most 50% of deadlines. If such a positive result cannot be achieved for the general self-suspending task scheduling problems, but may be some basic assumptions on task parameters will help (e.g., bounding the ratio $\frac{X_i}{C_i}$) to formulate sufficient conditions for the self-suspending robustness.

The multiprocessor point of view. Self-suspending taskset can in fact be modelled by chains of tasks, where suspension delays are tasks run upon some dedicated processors (e.g. I/O processing devices). We think that known results in the multiprocessor scheduling theory can be used to derive some results for uni-processor scheduling of self-suspending tasks.

References

[1] Y.S. Chen and L.P. Chang. A real-time configurable synchronization protocol for self-suspending process sets. *Real-Time Systems*, 42:34–62, 2009.

[2] I-G. Kim, K-H. Choi, S-K. Park, D-Y. Kim, and M-P. Hong. Real-time scheduling of tasks that contain the external blocking intervals. *Real-Time and Embedded Computing Systems and Applications(RTCSA'95)*, 1995.

[3] K. Lakshmanan and R. (Raj) Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Stockholm, Sweden*, (12–15), April 2010.

[4] C. Liu and J.H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. *Proceedings of the 30th IEEE Real-Time System Symposium*, pages 425–436, 2009.

[5] C. Liu and J.H. Anderson. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems. *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Stockholm, Sweden*, (12–15), April 2010.

[6] Jane W. S. Liu. *Real-Time Systems*, chapter Priority-Driven Scheduling of Periodics Tasks, pages 164–165. Prentice Hall, 2000.

[7] A.K. Mok and W.C. Poon. Non-preemptive robustness under reduced system load. *Proceedings of the 26th IEEE Real-Time System Symposium (RTSS'05)*, 2005.

[8] F. Ridouard and P. Richard. Worst-case analysis of feasibility tests for self-suspending tasks. In *proc. 14th Real-Time and Network Systems, Poitiers*, 2006.

[9] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, 1, December 2004.