Priority Assignment in Fixed Priority Pre-emptive Systems with Varying Context Switch Costs

Robert I. Davis
University of York, UK
Email: rob.davis@york.ac.uk

Sebastian Altmeyer
University of Amsterdam (UvA), Netherlands
Email: altmeyer@uva.nl

Alan Burns
University of York,UK
Email: alan.burns@york.ac.uk

Abstract

This paper discusses the open problem of priority assignment for systems using Fixed Priority Preemptive Scheduling (FPPS), where the context switch costs are dependent on whether or not the preempting and the preempted tasks belong to the same process and hence share the same address space. Schedulability tests for such systems are not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm. We pose the question: Can optimal (or close to optimal) priority assignments be found efficiently, ideally in a polynomial number of schedulability tests, avoiding the need to check all $n!$ possible priority orderings?

I. Introduction

The relevant safety standards (IEC61508, DO-178C, ISO26262) for electronics, avionics, and automotive systems require that either all applications are developed to the standard required for the highest criticality application, or that independence between different applications is achieved in both the spatial and temporal domains. One approach to ensuring spatial isolation is to make use of the concepts of *processes* and *threads*, where each process has a separate memory address space. Using a single process for all high criticality applications provides a single memory address space, easing the costs of interaction between high criticality tasks, which are implemented as threads within that process. Low criticality applications and their tasks can similarly be mapped to another distinct process and its threads. This ensures that tasks in low criticality applications cannot corrupt the data or memory space used by high criticality applications. Alternatively, individual applications may each be mapped to a distinct process, providing spatial isolation between applications of the same criticality.

The use of processes and threads gives rise to varying context switch costs [1]. Switching threads within a process (i.e. the context switch between tasks of the same application) has a low cost, since this involves switching only the resources unique to the threads, for example the processor state (program counter, stack pointer, processor registers etc.), which can typically be done in a very short time and may be assisted by hardware support. By contrast, switching between processes (i.e. the context switch between tasks of different criticality applications) may have a much higher cost. It involves switching the resources related to the processes. In particular, switching the memory address space, and can also involve operations on the caches [2], and the Translation Lookaside Buffer (TLB), making process switches a much more costly operation.

In a recent paper [3] at RTAS 2018, Davis et al. derived three flavors of schedulability analysis for FPPS accounting for differing context switch costs, referred to as *simple*, *refined*, and *multi-set* analysis. Here we focus on the *refined* analysis.

II. System Model

The system model assumed is an extension of the classical sporadic task model. We are interested in tasks executing under FPPS on a single processor. Each of the n tasks $(\tau_1, \tau_j, \ldots, \tau_n)$, is assigned a unique priority. Each task is characterized by its relative deadline D_i, worst-case execution time C_i, and minimum inter-arrival time or period T_i. Tasks are assumed to have constrained deadlines ($D_i \leq T_i$). A task is schedulable if its worst-case response time R_i is less than or equal to its deadline ($R_i \leq D_i$). Each task is assumed to belong to an application mapped to a specific process and hence a specific address space. A_i indicates the address space that task τ_i is mapped to. If tasks τ_i and τ_j belong to the same process and address space, then $A_i = A_j$, otherwise $A_i \neq A_j$. We assume that a context switch from one task τ_i to another τ_j has a large cost C^C if it involves switching process and address space (i.e. when $A_i \neq A_j$), and a small cost C^S otherwise.

III. Analysis

The simple analysis presented in [3] makes the assumption that all context switches incur the large context switch cost. It is thus equivalent to the standard response time analysis for FPPS [4], [5] with the large context switch cost subsumed into each task's WCET. In reality, however, the context switch time depends on both the preempting task and the preempted task. Taking this information into account, the standard analysis is *refined* in [3] as follows. Note we assume that the first job in the busy period always experiences a large context switch time, since the previously running job may be associated with a different process and address space. (We assume that soft real-time tasks may run in a background process at the lowest priority).

$$R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j}) \text{ where } \gamma_{i,j} = \begin{cases} C^C & \text{if } \exists h \in \text{aff}(i,j) | A_h \neq A_j \\ C^S & \text{otherwise} \end{cases} \quad (1)$$

Where $\text{hp}(i)$ denotes the set of tasks with priorities higher than that of task τ_i, and $\text{aff}(i,j)$ denotes the set of *affected* tasks that can execute between the release and completion of task τ_i and also be preempted by higher priority task τ_j, these tasks have

priorities higher than or equal to that of task τ_i, but lower than that of task τ_j. The term $\gamma_{i,j}$ equates to a large context switch time only if there is some task τ_h that can execute during the busy period (i.e. the response time) of task τ_i, be preempted by task τ_j, and belongs to a different process and address space to τ_j.

It is easy to construct examples with three tasks τ_A, τ_B, and τ_C with $A_A = A_C$ and $A_A \neq A_B$ showing that the worst-case response time of task τ_C depends upon the relative priority order of tasks τ_A and τ_B; such an example is given in [3]. This means that Deadline Monotonic (DM) priority assignment [6] is no longer optimal. Further, the analysis is not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [7], since the dependence on the relative priority ordering of higher priority tasks breaks a *necessary* condition for the applicability of Audsley's algorithm [8].

IV. Open Problems

A priority assignment algorithm or policy P is said to be *optimal* with respect to a schedulability test S and a given task model, if and only if there are no task sets that are compliant with the task model that are deemed schedulable by test S using another priority assignment policy, that are not also deemed schedulable by test S using policy P.

We are interested in finding optimal priority assignments for systems with context switch costs that depend on whether the preempting and the preempted task belong to the same process, and so share a common address space. Here, each task belongs to one of two (or more) distinct processes, and the analysis used is the refined test for FPPS given above (or alternatively, the multi-set analysis given in [3]). In each case, an optimal priority assignment could be found by exploring all $n!$ possible priority orderings; however, such an approach becomes intractable even for relatively small task sets (e.g. for $n = 15$, $n! > 10^{12}$). Rather, we are interested in efficient methods of finding an optimal (or close to optimal) assignment; ideally with complexity that is polynomial in the number of schedulability tests.

Priority assignment toolkit and ideas

In prior work on priority assignment for fixed priority systems, a number of techniques have proven useful:

(i) Establishing the properties of a priority ordering by considering if schedulability is maintained when the priorities of particular tasks are swapped, for example swapping tasks that have adjacent priorities but are out of DM order [9].

(ii) Work on Robust Priority Assignment [10] has established certain properties (such as the optimality of DM priority ordering) that hold in the presence of general forms of additional interference. It can be useful to disregard subsets of tasks, representing them only as additional interference, to enable a simpler form of reasoning about the optimal priority ordering of the tasks that remain.

(iii) Simple sufficient tests (using only the large context switch costs) and simple necessary conditions (using only the small context switch costs) that are compatible with Audsley's algorithm can be used to guide priority assignment [11]. These techniques enable partial assignments to be found that are certain to be schedulable, while discarding others that are certain to be unschedulable.

As an initial idea, if we could show that if a schedulable priority ordering exists, then a revised ordering with all of the tasks belonging to each specific process in DM partial order is also schedulable, then reasoning along these lines might reduce the overall problem to one of merging DM partial orders for each process – potentially a much simpler problem.

For more background information on techniques for priority assignment in fixed priority systems, see the review on this topic [9]. Finally, we note that solutions to the problems posed may lead to improved solutions to the more complex problem of priority assignment for FPPS with Cache Related Preemption Delays (CRPD) [12] [13].

References

[1] S. Yamada and S. Kusakabe, "Effect of context aware scheduler on TLB," in *Proceedings IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–8.

[2] J. Whitham, N. C. Audsley, and R. I. Davis, "Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, p. 120, 2014.

[3] R. I. Davis, S. Altmeyer, and A. Burns, "Mixed criticality systems with varying context switch costs," in *Proceedings IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2018.

[4] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[5] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, May 1986.

[6] J.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237 – 250, 1982.

[7] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters 79(1)*, pp. 39–44, May 2001.

[8] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Real-Time Systems, Volume 47, Issue 1*, 2010, pp. 1–40.

[9] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *Journal of Systems Architecture*, vol. 65, pp. 64 – 82, 2016.

[10] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proceedings IEEE International Real-Time Systems Symposium (RTSS)*, Dec 2007, pp. 3–14.

[11] ——, "On optimal priority assignment for response time analysis of global fixed priority pre-emptive scheduling in multiprocessor hard real-time systems," University of York,, Tech. Rep., April 2010.

[12] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[13] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, "Addressing cache related preemption delay in fixed priority assignment," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.