Modelling and Analysis Suite for Real Time Applications (MAST 1.3.7)

Analysis Techniques used in MAST

By: Michael González Harbour mgh@unican.es
José Carlos Palencia Gutiérrez palencij@unican.es
José Javier Gutiérrez García gutierjj@unican.es

1. Introduction

The MAST toolset contains several schedulability analysis tools capable of analysing single processor and distributed systems scheduled with fixed priority, EDF, and EDF within priorities scheduling policies. The tools are based on different scheduling analysis techniques published in the literature:

- *Classic RM Analysis*. This analysis implements the classic exact response time analysis for single-processor fixed-priority systems first developed by Harter [3] and Joseph and Pandya [7], and later extended by Lehoczky to handle arbitrary deadlines [9] and by Tindell to handle jitter [26]. It corresponds to Technique 5, “Calculating response time with arbitrary deadlines and blocking”, in [8].
- *Varying Priorities Analysis*. This analysis implements the response time analysis for single processor fixed priority systems in which tasks may explicitly change their priorities, developed by González, Klein and Lehoczky [4]. It corresponds to Technique 6, “Calculating response time when priorities vary”, in [8].
- *EDF Monopocessor Analysis*. This analysis implements the exact response time analysis for single-processor EDF systems first developed by Spuri [23]. In the MAST implementation we use the EDF Within Priorities (see below), because there may be interrupt service routines (modelled as fixed priority tasks) in addition to the EDF tasks.
- *EDF Within Priorities Analysis*. This analysis is a mixture of the response time analysis for fixed priority systems [8][9][26] and for EDF [23]. It is capable of analysing systems with hierarchical schedulers, in which the underlying primary scheduler is based on fixed priorities, and there may be other EDF (secondary) schedulers scheduling tasks at a given priority level. It was developed by González and Palencia [5].
- *Holistic Analysis*. This analysis extends the response time analysis to multiprocessor and distributed systems. It was first developed for fixed priority systems by Tindell and Clark [26][27] and refined by Palencia et al [12]. Later, Spuri [24] extended it to EDF systems. It is not an exact analysis, because it makes the assumption that tasks of the same transaction are independent.
- *Offset Based Analysis*. This is a response time analysis for multiprocessor and distributed systems that greatly improves the pessimism of the holistic analysis by taking into account that tasks of the same transaction are not independent, through the use of offsets. Offset based analysis for fixed priorities was first introduced by Tindell [28] and then extended to distributed systems by Palencia and González [13]. It was later extended to

EDF systems by Palencia and González [15]. Although it provides much better results than the holistic analysis, it is not an exact method because the exact analysis is intractable.

- *Optimized Offset Based Analysis*. This is an enhancement of the offset based analysis for fixed priority systems in which the priorities of the tasks of a given transaction are used together with the precedence relations among those tasks to provide a tighter estimation of the response times. It was developed by Palencia and González [14], and later enhanced by Redell [18].

The MAST toolset is able to automatically calculate the blocking times caused by mutual exclusion synchronization. The model includes shared resources with the immediate ceiling [2], priority inheritance [21], and stack resource [2] synchronization protocols. It also allows mixtures of these protocols and their use in multiprocessor systems [16][17], with some restrictions that are described below. For the basic priority inheritance, Rodríguez and García [19] showed that most implementations do not strictly follow the rules in [21], and that the amount of blocking is usually higher than that predicted by the theory. In MAST we take this into account when calculating the blocking times due to the use of the priority inheritance protocol.

The MAST toolset also contains tools to automatically assign priorities and other scheduling parameters. Priority assignment tools are provided for single-processor and distributed systems. In single-processor systems, if deadlines are within the periods the optimum deadline monotonic priority assignment developed by Leung and Layland is used [10]. The Liu and Leyland classic rate monotonic priority assignment for the case of deadlines equal to the periods [11] is know to be a special case of the deadline monotonic assignment. When deadlines are larger than the task periods, the optimum priority assignment developed by Audsley is used [1]. This technique is based on the iterative use of the schedulability analysis tools for different solutions, until a schedulable solution is obtained.

In multiprocessor and distributed systems the problem of assigning priorities is much harder, as there are strong interrelations between the response times in the different resources. We provide two heuristic solutions based on iteratively applying the schedulability analysis tools. The first one is based on the use of the simulated annealing optimization techniques first used by Tindell, Burns, and Wellings for assigning priorities [1]. The second heuristics, which usually provides better and faster results is the HOPA algorithm developed by Gutiérrez and González [6].

The MAST toolset is able to determine not only whether the system is schedulable or not, but also how far it is from being schedulable, or how much capacity is available until the system becomes unschedulable. It does so by calculating slacks, which are defined as the percentage by which we can increase the execution times of some operations while keeping the system schedulable (for positive slacks) or the percentage by which we have to decrease the execution times to make the system schedulable (for negative slacks). A slack of zero means that the system is just schedulable, and that even the smallest increment in the execution times would lead to non schedulability. There are different kinds of slacks provided:

- *System slack*: affects all the operations in the system
- *Processing resource slack*: affects only the operations executed in a given processing resource
- *Transaction slack*: affects only the operations used in a given transaction

- *Operation slack*; affects only one single operation

The slacks are calculated by modifying the worst-case execution times and repeating the analysis using a binary search to find the point in which the system becomes unschedulable (or schedulable if it wasn't). The slack is calculated with a 1% precision to limit the amount of times the analysis is repeated to around 20 times.

In addition to these tools, some additional analysis techniques were needed to develop the MAST toolset, because it allows combinations of scheduling policies and synchronization protocols that did not have a global treatment in the published analysis techniques that we know about. This document describes these additional extensions that were specifically developed for MAST.

2. Calculating Blocking Times

In addition to the preemption and self execution effects, the response time of a given task has to include the effects of blocking delays caused by lower priority tasks. These delays are caused by mutual exclusion synchronization, by the temporary execution of high priority sections by the low priority tasks, or by non-preemptible sections, which behave as high priority sections of the highest possible priority. Mutual exclusion synchronization can use one of the three protocols defined: priority inheritance (PIP) [21], immediate priority ceiling (IPC) [2], and stack resource policy (SRP) [2]. Analysis of priority inheritance resources takes into account the POSIX implementation, which is known [19] to have a worse blocking time than the implementation described in the original paper.

In this section we describe how to calculate the blocking times when all these effects are combined, and also when different combinations of synchronization protocols are used. In order to simplify the analysis we have made the following restrictions in the current implementation of the MAST tools:

- SRP resources are used only by scheduling servers with EDF scheduling parameters.
- PIP resources are used only by scheduling servers with fixed priority scheduling parameters.

Therefore, the only protocol that can be used to share information among EDF and fixed priority tasks is the immediate priority ceiling. This is also the only protocol allowed for global shared resources, i.e., those that can be used from different processing resources.

2.1 Blocking by high priority sections and preemption rules

When analysing the blocking time of a given task segment, one of the possible blocking effects is caused by the execution of a high priority section of a lower priority task. Because only one of these sections may be executing in the processing resource of the task under analysis when it is released, we take the maximum of all these high priority sections of lower priority tasks [2]:

- local critical sections with Immediate Ceiling Protocol and with a ceiling higher than the priority of the task under analysis
- global critical sections in the same processor and with Immediate Ceiling Protocol and with a ceiling higher than the priority of the task under analysis
- sections with overridden scheduling parameters of priority higher than the task's priority

- non-preemptible sections

In [5] we show that in the presence of the SRP protocol, each task may only be delayed by either an SRP or an IPC critical section, but not by both. Blocking is cased by local critical sections of tasks of the same priority level (in the case of the EDF-within priorities) with a preemption level higher than that of the task under analysis. Therefore:

$$B_i^{ceil} = max(CS_{kj}, CS_{lm})$$
$$\forall k, \forall j | (S_j = FP) \wedge (P_j < P_i) \wedge (Ceil(CS_{kj}) \geq P_i)$$
$$\forall l, \forall (m \neq i) | (Lev(CS_{lm}) \geq Lev_i)$$

Where $Lev(CS_{lm})$ is the preemption level of critical section l of task m, and Lev_i is the preemption level of task i. No context switch overhead is added in these cases [2].

2.2 Blocking by Global Critical Sections

Restrictions and assumptions:

- Global critical sections always use the immediate priority ceiling protocol.
- The ceiling has been set high enough to minimize the remote blocking. In particular, we limit remote blocking to be caused only by simple tasks (belonging to a transaction with a single initial high priority segment), that do not use shared resources. This allows us modelling interrupt service routines that cause remote blocking, and that cannot be disabled during the critical section execution.
- We will assume that remote blocking is small, and thus we will not consider the advantage that the task suspension has for lower priority tasks. In any case we should not consider it, because in the worst case suspension would not occur.

Blocking by global critical sections in the same processor as the task under analysis is considered as part of the high priority segment blocking. Therefore, we only need to take into account critical sections from other processors using the same resource. At first, it seems that we should use only one such critical section per processor, given that priority ceiling is being used. However, if a critical section from one processor has caused blocking, it is possible for that processor to continue executing while another critical section of another processor is executing, thus entering a third critical section after the second one has finished. Therefore, for the task under analysis we need to take into account the effects of global critical sections for each global critical section.

Consequently, we consider two situations:

A. Applicable critical sections belong to only one processor. In this case, only one such critical section may affect, and it will be calculated as

$$B_i^{res-k} = max_{CS_j \in res-k}(Resp(CS_j))$$

where $Resp(CS_j)$ is the response time of critical section j, given by:

$$Resp(CS_j) = \left\{ max(R > 0) | R = CS_j + \sum_{i \in hl(j)} C_i^h + \sum_{i \in h(j)} \left\lceil \frac{R}{T_i} \right\rceil \cdot C_i \right\}$$

where $HL(j)$ is the set of tasks that start at a priority strictly higher than Pj, but then go to low priority; $H(j)$ is the set of tasks that start at a priority strictly higher than Pj; and C_j^h is the length of the H segment of the corresponding task.

B. The applicable critical sections belong to more than one processor. In this case, all of them could potentially cause remote blocking, and thus we consider all of them.

$$B_i^{res-k} = \sum_{CS_j \in res-k} RespCS_j$$

Enhancements possible but not yet implemented:

- Model the remote critical section as a remote procedure call only as a mean of calculating the remote blocking for non-simple tasks.
- Study other cases in which not all the considered critical sections can block the same task.

2.3 Blocking caused by priority inheritance critical sections

Assumptions

- We only consider local critical sections in fixed priority tasks; global critical sections are forced to use immediate ceiling locking, with the appropriate global priority protection. EDF tasks will use the SRP or IPC protocols.

Suppose that the task segment under analysis, i, has k critical sections using distinct resources $r1..rk$. If a task segment uses the same resource more than once, it will only suffer one blocking from other critical sections with that resource, because to be delayed, the resource mush have already been acquired by the lower priority task, and then it is not possible for it to run again until segment i finishes [21].

The task may be blocked once at each critical section, by a critical section of another lower priority task that is holding the lock [21]. For each critical section l, the delay is equal to the longest critical section on rl from lower priority tasks. Since each lower priority task can influence with at most one critical section, all combinations may not be possible.

In addition, a task may get push-through blocking, by critical sections of lower priority tasks that have a ceiling higher than or equal to that of the task i. In the original priority inheritance protocol only one such blocking per lower priority tasks may occur (because after the critical section the lower priority task cannot run until task i has completed its execution). However, in some implementations of the priority inheritance protocols (for example some POSIX implementations) the task under analysis may get delayed by several lower priority tasks [19].

In all implementations only one blocking can occur per shared resource, as the resource must have been acquired before the task started to run.

Because a critical section of a lower priority task with enough ceiling can always affect the task segment under analysis, either as push-through blocking or as direct blocking, if the original priority inheritance protocol was used we count the blocking time using the following equation:

$$B_i^{pip} = min(\sum_{r_1 \ldots r_k} max(CS_{r_l,j} | (ceil(CS_{r_l,j}) \geq P_i)), \sum_{t \in lp(i)} max(CS_{t,j} | (ceil(CS_{t,j}) \geq P_i)))$$

But because of the possibility of different implementations, the MAST toolset uses only the first term:

$$B_i^{pip} = \sum_{r_1 \ldots r_k} max(CS_{r_l,j} | (ceil(CS_{r_l,j}) \geq P_i))$$

A possible enhancement would be to explore less pessimistic approaches by considering that not all combinations from lower priority tasks may occur, as shown in [19].

2.4 Total blocking

The total blocking is the addition of all the blocking effects mentioned

$$B_i = B_i^{ceil} + \sum_{res-k \in globalres} B_i^{res-k} + B_i^{pip}$$

3. Calculating Context Switch Overheads

For each task segment that exists between potentially blocking points we need to consider two context switches, both for the own's task analysis, as well as for the analysis of lower priority tasks [20]. The easiest is to add the context switches to the execution time of the task.

In addition, we have to take into account the context switches due to synchronization. The effects that the context switch overhead of a task that is suspended has on lower priority tasks is equal to two context switches per critical section considered. This has to be added to the normal two context switches that a task has, as it represents overhead for lower priority tasks.

- For local shared resources using the immediate ceiling or the SRP protocols there are no context switch activities due to synchronization [2].
- For local shared resources using the priority inheritance protocol the amount of context switches due to synchronization is double the number of critical sections that influence blocking [21].
- For global shared resources using the global immediate priority ceiling, the amount of context switches is double the number of blockings that the task may suffer [16][17].

Therefore, an additional context switch overhead of $2Cs$ is considered for each segment, as additional execution time, and not as additional blocking time. This may introduce a small amount of pessimism, because the overhead may not be possible for all periods of a task that are considered in the busy period of a lower priority task.

References

[1] N.C. Audsley, “Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times”, Dept. Computer Science, University of York, December 1991.

[2] T.P. Baker, “Stack-Based Scheduling of Realtime Processes”, Journal of Real-Time Systems, Volume 3, Issue 1 (March 1991), pp. 67–99.

[3] P.K. Harter, "Response times in level-structured systems". ACM Transactions on Computer Systems, vol. 5, no. 3, pp. 232-248, Aug. 1984.

[4] M. González Harbour, M.H. Klein, and J.P. Lehoczky. “Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems”. IEEE Trans. on Software Engineering, Vol. 20, No.1, January 1994.

[5] M. González Harbour and J.C. Palencia, “Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities”, Proceedings of the 24th IEEE Real-Time Systems Symposium, Cancun, México, December, 2003.

[6] J.J. Gutiérrez García and M. González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems". Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 124-132, April 1995.

[7] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System". The Computer Journal (British Computing Society) 29, 5, pp. 390-395, October 1986.

[8] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour. “A practitioner’s Handbook for Real-Time Analysis”. Kluwer Academic Pub., 1993.

[9] J.P. Lehoczky. “Fixed Priority Scheduling of Periodic Tasks Sets with Arbitrary Deadlines”. IEEE Real-Time Systems Symposium, 1990.

[10] J. Leung, and J.W. Layland. “On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”. Performance Evaluation 2, 237-50, 1982.

[11] 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, pp. 46-61.

[12] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, M. González Harbour. “On the schedulability analysis for distributed hard real-time systems”. 9th Euromicro Workshop on Real-Time Systems. Toledo, 1997.

[13] J.C. Palencia Gutiérrez and M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets". Proceedings of the 18th. IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

[14] J.C. Palencia, and M. González Harbour, “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems”. Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[15] J.C. Palencia and M. González Harbour, “Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF”. Euromicro conference on real-time systems, Porto, Portugal, June 2003.

[16] R. Rajkumar, L. Sha, and J.P. Lehoczky. “Real-Time Synchronization Protocols for Multiprocessors”. IEEE Real-Time Systems Symposium, December 1988.

[17] R. Rajkumar. “Real-Time Synchronization Protocols for Shared Memory Multiprocessors”. Proceedings of the 10th International Conference on Distributed Computing, 1990.

[18] Redell, O. Response Time Analysis for Implementation of Distributed Control Systems, Doctoral Thesis, TRITA-MMK 2003:17, ISSN 1400-1179, ISRN KTH/MMK/R--03/17--SE, 2003

[19] P. Rodríguez Hernández and J.J. García Reinoso, “Nota sobre la Implementación del mecanismo de herencia”, VI Jornadas de Tiempo Real, Gijón, February 2003 (In Spanish).

[20] L. Sha, T. Ralya, and J.B. Goodenough, “An analytical approach to real-time software engineering”, Software Engineering Institute Annual Technical Review, 1989.

[21] L. Sha, R. Rajkumar, and J.P. Lehoczky. “Priority Inheritance Protocols: An approach to Real-Time Synchronization”. IEEE Trans. on Computers, September 1990.

[22] B. Sprunt, L. Sha, and J.P. Lehoczky. “Aperiodic Task Scheduling for Hard Real-Time Systems”. The Journal of Real-Time Systems, Vol. 1, 1989, pp. 27-60.

[23] M. Spuri. “Analysis of Deadline Scheduled Real-Time Systems”. RR-2772, INRIA, France, 1996.

[24] M. Spuri. “Holistic Analysis of Deadline Scheduled Real-Time Distributed Systems”. RR-2873, INRIA, France, 1996.

[25] K.W. Tindell, A. Burns, and A.J. Wellings. “Allocating Real-Time Tasks. An NP-Hard Problem Made Easy”. Real-Time Systems Journal, Vol. 4, No. 2, May 1992. pp. 145- 166.

[26] K. Tindell, “An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks”. Journal of Real-Time Systems, Vol. 6, No. 2, March 1994.

[27] K. Tindell, and J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”. Microprocessing & Microprogramming, Vol. 50, Nos.2-3, pp. 117-134, April 1994.

[28] K. Tindell, “Adding Time-Offsets to Schedulability Analysis”, Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.