Scheduling Analysis with Respect to Hardware Related Preemption Delay[*]

Stefan M. Petters
Institute for Real-Time Computer Systems
Prof. Dr.–Ing. Georg Färber
Technische Universität München, Germany
Stefan.Petters@rcs.ei.tum.de

Abstract

Proofing that all deadlines are met is one of the essential tasks in the development of real-time systems. One method to provide this proof is the response time analysis. Unfortunately state of the art processors and the corresponding real-time operating systems not only incur non negligible task switching times, but also the execution time of the preempted task itself is prolonged due to the built-in acceleration techniques of the processors. In order to cover this while avoiding excessive pessimism a new approach is presented in this paper. A central focus lies on a simple application of the method.

1 Motivation

The domain of real-time systems has broadened vastly in the last few years. One reason for this trend is the penetration of every day life by computer systems. Typical examples may be cash dispensers, vacuum cleaners or the up to 50 processors integrated in modern cars. Not all of these applications have real-time properties and even less are life critical, but e.g. for the customer of a new car it is not acceptable if his newly bought product stops working on the motor-way due to a deadline miss in the engine control unit.

For these real-time systems more and more general purpose processors are deployed. One of the major reasons for this is the unsurpassed cost/performance ratio of these processors which unfortunately stems from several built-in acceleration techniques like caches, speculative execution and instruction reordering which are almost impossible to model exactly and complicate the real-time analysis considerably. This holds true not only for the estimation of the worst case execution time (WCET) but also for the real-time analysis itself which has to cover the increased execution time after a task τ_i has been preempted by another task τ_j of higher priority, due to the effect that the working sets of the task τ_i e.g. in the cache is displaced by task τ_j.

On WCET analysis a considerable amount of papers for modern processors have been published, but for the schedulability analysis only two groups have recent publications on this subject e.g. [1] and [2]. The schedulability analysis presented in this paper provides an analysis method which fits to the WCET analysis approach in [3].

The following section will provide an overview of the basic foundations the method presented Section 3 is build upon. Section 4 compares the approach with previously available work.

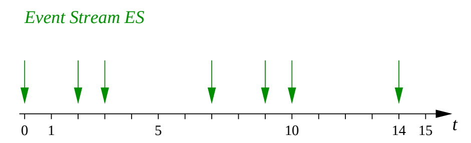
2 Foundations

2.1 Model of Embedding Process

In [4] a method for describing the environment a real-time system is embedded in is presented. The release of tasks is characterised as event streams. Each event corresponds to one release of task τ_i and $ES(T)$ returns the maximum number of events in interval T. Figure 1 depicts a sample event stream and its notation.

An event tuple consist of a periodicity z_r and and an interval a_r. The example in Figure 1 corresponds to the following statements. There is at most 1 interrupt within in-

[*]The work presented in this paper is supported by the *Deutsche Forschungsgemeinschaft* as part of a research programme on “Rapid Prototyping for Embedded Hard Real–Time Systems” under Grant Fa 109/11-3.



$$ES: \quad \left\{ \begin{pmatrix} z_r \\ a_r \end{pmatrix} \right\}_{r=1}^{s} = \left\{ \begin{pmatrix} 7 \\ 0 \end{pmatrix}, \begin{pmatrix} 7 \\ 1 \end{pmatrix}, \begin{pmatrix} 7 \\ 3 \end{pmatrix} \right\} \tag{1}$$

Figure 1. Event Stream Example[5]

terval 0 i.e. no simultaneous interrupts. Within any interval of 1 time unit chosen of the the stream there will be at most 2 interrupts. At most 3 interrupts will be issued within any interval of 3 time units and the pattern is repeated no sooner then every 7 time units.

It has to be noted that z_r is not required to be identical for all tuples belonging to one event stream. For simplicity reasons the restriction $0 < z_r < \infty$ is introduced. The worst case number of releases of task τ_i within the interval T can be computed by the following equation:

$$E_i(T) = \sum_{r=1}^{s} \begin{cases} 0 & : \quad T < a_r \\ \left\lfloor \frac{T-a_r}{z_r} + 1 \right\rfloor & : \quad T \geq a_r \end{cases} \tag{2}$$

In order to satisfy the restriction that $E_i(T)$ returns the maximum number of events which can potentially happen in interval T at least one a_r in a event stream description has to be equal to zero leading to $E_i(0) \geq 1$.

2.2 Response Time Analysis

The scheduling analysis method presented in this papers bases on an approach that has been developed independently by the real-time research group in York e.g. in [6], Harter in [7] and Joseph and Pandya in [8]. The method computes the worst case response time R_i of each task separately and compares these with their corresponding deadlines D_i. If each response time is less or equal the corresponding deadline, the system is guaranteed to meet all deadlines. In order to achieve this, the well known assumption is made that in the worst case is provided if all tasks are released at the same time with the least possible interval for all future releases. Furthermore the following restrictions are required:

- The deadline of a task is less or equal to the minimal distance between two consecutive releases of the task.
- Each task has a unique fixed priority with the exception of priority changes due to the priority inversion avoidance protocol.
- To avoid priority inversion due to resource contention an appropriate protocol like the priority inheritance protocol [9] or one of its derivates has to be used.

In an iterative approach the worst case response time R_i of task τ_i is computed. In each iteration the approach tries to allocate in a time window w_i the execution time C_i of the task τ_i, the blocking time B_i due to resource contention and the interference of tasks with higher priority than task τ_i. The interference of a task with higher priority is composed of the number of releases of the task within the current execution window which is tagged N_j in the equation below multiplied by the tasks execution time C_j. The computation of a new time window w_i^{n+1} is depicted below:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in \mathbf{H}_i} \underbrace{\left\lceil \frac{w_i^n}{T_j} \right\rceil}_{N_j} * C_j \tag{3}$$

where $\mathbf{H}_i$ is the set of tasks with higher priority than task τ_i and T_j is the minimum distance between two releases of task τ_j. The value for the initial time window w_i^1 is set to $C_i + B_i$, but other values might be chosen to minimise the number of iterations necessary.

The iteration terminates when either $w_i^{n+1} = w_i^n$, thus the worst case response time $R_i := w_i^n$, or when response time exceeds the deadline (i.e. $w_i^{n+1} > D_i$), thus task τ_i can not be guaranteed to finish before its deadline. A more detailed explanation and the proof of correctness of the algorithm can be read in [6] and [1].

2.3 Accounting for Penalties

In [1] five possible ways of accessing this preemption delay are referred:

1. The time to refill the entire cache.
2. The time to refill the cachelines displaced by the preempting task.
3. The time to refill the cachelines used by the preempted task.
4. The time to refill the maximum number of useful cachelines the preempted task may hold in cache at the worst case instant a preemption may arise. Useful lines are those that are potentially used again by this task like those referred by Lim et al. in [10].

5. The time to refill the intersection of lines between preempting and preempted task like described by Lee et al. in [2].

Busquets et al. present in [1] a method for incorporating the penalty according to number 1 and 2 of the list. In real world applications on modern processors this is very pessimistic. For approach number 1 the large caches of modern processors cause a considerable overestimation. Approaches number 2 and 3 have the drawback that the set of cachelines touched by a task during its execution usually exceeds by far the number of cachelines which are useful to the tasks execution at any point of time. The method following number 4 on the list needs considerable analysis on code and data for modern architectures, but often the necessary values can be derived from the WCET analysis part. Approach number 5 is rather challenging on a modern architecture since the approach has not only to cover the extrinsic interference of several levels cache but also effects like the branch prediction and other acceleration techniques.

Busquets et al. use the following equation for their response time analysis. It differs only from Equation 3 by an additional γ_j which corresponds to the extrinsic interference described above:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in \mathbf{H}_i} \left\lceil \frac{w_i^n}{T_j} \right\rceil * (C_j + \gamma_j) \tag{4}$$

In contrast to the work in [1] the approach presented in this paper the extrinsic interference is covered by considering the worst case additional preemption delay which may be imposed by the preempted task and not the worst case delay imposed by the preempting task. This additional preemption delay is comprised of penalties for the acceleration techniques like branch predictions and pipelines and the maximum number of useful cachelines the task may hold in cache at the worst case instant a preemption arises which corresponds to number 4 in the previously provided list. This additional preemption delay suffered by task τ_i is summarised in δ_i. The different term indicates that δ_i is dependent on the usage of cache and the other acceleration techniques by the preempted task while γ_j denotes the cache usage of the preempting task.

3 Real-Time Analysis

For the response time analysis with respect to the usage of acceleration techniques as described in the previous section Equation 3 has to extended considerably. For the following theorem and the corresponding proof a few terms not yet defined in this paper have to be introduced:

$\Delta_{i,j}(R_i)$ is the worst case additional preemption delay suffered by task τ_i by releases of task τ_j in the interval R_i.

P_j^n is the worst case number of preemptions of task τ_i by task τ_j which are not covered at start of iteration n.

$\mathbf{S}_{i,j}^n$ is the set of tasks which potentially suffer preemption by τ_j instead of τ_i and have not been covered at start of iteration n.

$\mathbf{L}_k$ is the set of of tasks with lower priority than task τ_k.

$\Theta_{i,j}^n$ The cumulative preemption delay caused by task τ_k on task τ_i after $n-1$ iterations.

$\max(\dots)$ provides the maximum value of it's arguments.

$\min(\dots)$ provides the minimum value of it's arguments.

For the system including the additional extrinsic interference δ_i and non equidistant task releases, where $E_j(T)$ expresses the worst case number of releases of τ_j in the interval T, the following theorem is stated.

Theorem 3.1: *Given a set of real-time tasks scheduled by a fixed priority preemptive policy with priority inheritance for priority inversion protection in a system where task τ_i suffers a worst case penalty of δ_i for every preemption, then there either exists a worst case response time value for R_i for each task τ_i which makes the equation system (Equation 5 to 9) depicted below true or such a task is not schedulable.*

$$R_i = C_i + B_i + \sum_{\tau_j \in \mathbf{H}_i} (E_j(R_i) * C_j + \Delta_{i,j}(R_i)) \tag{5}$$

$\Delta_{i,j}(R_i)$ is computed iteratively by the following formula.

Initialisation:

$$\begin{aligned} P_{i,j}^1 &= E_j(R_i) \\ \mathbf{S}_{i,j}^1 &= \tau_i \cup \mathbf{H}_i \cap \mathbf{L}_j \\ \Theta_{i,j}^1 &= 0 \end{aligned} \tag{6}$$

Iterative process:

$$\begin{aligned} k : \delta_k &= \max(\{\delta_l \mid \tau_l \in \mathbf{S}_{i,j}^n\}) \\ \mathbf{S}_{i,j}^{n+1} &= \mathbf{S}_{i,j}^n \setminus \tau_k \\ \Theta_{i,j}^{n+1} &= \Theta_{i,j}^n + \min(P_{i,j}^n, E_j(R_k) * E_k(R_i)) * \delta_k \\ P_{i,j}^{n+1} &= P_{i,j}^n - E_k(R_j) * E_k(R_i) \end{aligned} \tag{7}$$

Terminal condition:

$$P_{i,j}^{n+1} \quad \leq \quad 0 \tag{8}$$

After the iterative formula has terminated:

$$\Delta_{i,j}(R_i) := \Theta_{i,j}^{n+1} \tag{9}$$

Proof: Due to the dependency on the worst case response times of tasks which have a higher priority than task τ_i it is necessary that the real-time analysis is conducted in priority order starting with the highest priority. As previously stated the iterative response time analysis formula in Equation 3 is taken as a basis. To simplify the following proof it is assumed that the final response time of a task is already determined (i.e. $R_i = w_i^{n+1} = w_i^n$).

$$R_i = C_i + B_i + \sum_{\tau_j \in \mathbf{H}_i} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j \tag{10}$$

In a first step the number of releases of task τ_i has to be matched to the model of the embedding process described in the Section 2.1. Since $E_i(T)$ provides the maximum number of releases of task τ_i in the interval T, $\left\lceil \frac{R_i}{T_j} \right\rceil$ can be simply replaced, leading to the following equation:

$$R_i = C_i + B_i + \sum_{\tau_j \in \mathbf{H}_i} E_j(R_i) * C_j \tag{11}$$

Now the additional extrinsic interference has to be introduced into Equation 11. Assuming the response time of task τ_2 is investigated and τ_2 suffers preemption of task τ_1 with higher priority. Whenever τ_2 is preempted by τ_1, τ_2 has to pay the additional preemption delay δ_2. Thus with respect to task τ_2 the Equation 11 would be extended to:

$$R_2 = C_2 + B_2 + E_1(R_2) * (C_1 + \delta_2) \tag{12}$$

A slightly different picture is to be drawn if the preemption delay of task τ_3 has to be covered. The preemption of task τ_3 by τ_2 is investigated which is analogous to the preemption of τ_2 by τ_1. But the preemption of task τ_3 by τ_1 has to handled differently. In this case two scenarios have to be drawn which are both depicted in Figure 2.

Within the interval defined by the execution window of task τ_3 task τ_1 is released twice and task τ_2 once. The first occurrence of task τ_1 preempts τ_3, while in the second case τ_2 is preempted. In the second release of task τ_3, this task is preempted twice by τ_1 and once by τ_2. In order to account for the worst case preemption delay the maximum of

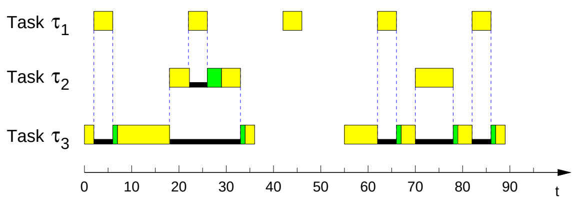


Figure 2. Preemption of task τ_3 by Tasks τ_2 and τ_1

preemption delays has to be taken into account. Thus the Equation 12 needs to be extended by the term $\max(\delta_2, \delta_3)$ for one preemption induced in R_3 by task τ_1 and twice δ_3 for one preemption of task τ_1 and of τ_2 each.

$$\begin{aligned} R_3 \;=\; & C_3 + B_3 + E_2(R_3) * (C_2 + \delta_3) + E_1(R_3) * C_1 + \\ & \max(\delta_2, \delta_3) * E_1\,(R_2) * E_2\,(R_3) + \\ & \delta_3 * (E_1\,(R_3) - E_1\,(R_2) * E_2\,(R_3)) \end{aligned} \tag{13}$$

It needs no mathematical proof to see that the complexity of this formula would increase vastly if extended to more than three tasks. On the other hand, the additional preemption delay to be taken into account for task follows simple rules which will be explained in detail below. Thus the additional preemption delay is abstracted and replaced by the variable $\Delta_{i,j}(R_i)$ resulting in the following form where $\Delta_{i,j}(R_i)$ represents the additional preemption delay suffered by task τ_i due to the preemption by task τ_j:

$$R_i = C_i + B_i + \sum_{\tau_j \in \mathbf{H}_i} (E_j(R_i) * C_j + \Delta_{i,j}(R_i)) \tag{14}$$

For the determination of $\Delta_{i,j}(R_i)$ let us consider first which possible scenarios have to be dealt with whenever task τ_j potentially preempts task τ_i. The release of task τ_j may only preempt tasks of lower priority than task τ_j which form the set $\mathbf{L}_j$. For the additional preemption delay of task τ_i only tasks with higher priority than task τ_i and τ_i itself have to be considered. Thus the set $\mathbf{S}_{i,j}$ of tasks which have to be taken into account for the additional preemption delay for task τ_i due to preemption by task τ_j is comprised of $\tau_i \cup \mathbf{H}_i \cap \mathbf{L}_j$.

Out of set $\mathbf{S}_{i,j}$ the task τ_k has to be chosen which suffers the largest individual additional preemption delay δ_k whenever it is preempted. In the next step one has to consider how often task τ_k may be preempted by task τ_j in its execution window R_k. This has been solved before in Equation 2. Thus the number of preemptions suffered in the worst case by task τ_k during its response time R_k by task τ_j is $E_j(R_k)$. Combined with the worst case number of preemptions task τ_i suffers by τ_k during one execution of τ_i, which is $E_k(R_i)$, it can be computed how many times task τ_k might suffer preemption instead of task τ_i. Thus the additional preemption delay $\theta_{i,j,k}$ can be computed by Equation 15:

$$\theta_{i,j,k} = E_j(R_k) * E_k(R_i) * \delta_k \tag{15}$$

In the next iteration the remaining preemptions of $E_j(R_i)$ not covered by Equation 15 have to be considered. The number of preemptions not already considered is present in $P^n_{i,j}$. For this the second worst case task is chosen out of the set $\mathbf{S}_{i,j} \setminus \tau_k$. In order to avoid to account for more the $P^1_{i,j}$ preemptions by task τ_j the number of preemptions covered in iteration n may not exceed $P^n_{i,j}$. Thus $\min(P^n_{i,j}, E_j(R_k) * E_k(R_i))$ are covered within iteration n. The process is repeated until all preemptions by task τ_j are covered. In order to represent the limitations outlined above Equation 15 has to be modified to:

$$\theta_{i,j,k} = \min\left(P^n_{i,j}, E_j(R_k) * E_k(R_i)\right) * \delta_k \tag{16}$$

In order to simplify the iterative process and avoid the final summation of all $\theta_{i,j,k}$ the equation can be transformed to:

$$\Theta^{n+1}_{i,j} = \Theta^n_{i,j} + \min(P^n_{i,j}, E_j(R_k) * E_k(R_i)) * \delta_k \tag{17}$$

The steps presented in Equations 14 to 17 can be summarised in the following equations which correspond to Equations 5 to 9:

$$R_i = C_i + B_i + \sum_{\tau_k \in \mathbf{H}_i} (E_k(R_i) * C_j + \Delta_{i,j}(R_i)) \tag{18}$$

With $\Delta_{i,j}(R_i)$ computed iteratively:

Initialisation:
The number of preemptions which have to be covered by the analysis is computed by Equation 19. The starting set of tasks to be considered for the preemption of task τ_i by task τ_k is depicted in Equation 20. Equation 21 sets the initial value of the preemption delay to zero.

$$P^1_{i,j} = E_j(R_i) \tag{19}$$
$$\mathbf{S}^1_{i,j} = \tau_i \cup \mathbf{H}_i \cap \mathbf{L}_j \tag{20}$$
$$\Theta^1_{i,j} = 0 \tag{21}$$

Iterative process:
Equation 22 displays the selection of the index of the task which comprises the worst case individual preemption delay in set $\mathbf{S}^n_{i,j}$. The set is then reduced by the found task τ_k in Equation 23. The preemption penalty covered by task τ_k is computed in Equation 24 while in Equation 25 the preemptions not covered after this step is computed.

$$k : \delta_k = \max(\{\delta_l \mid \tau_l \in \mathbf{S}^n_{i,j}\}) \tag{22}$$
$$\mathbf{S}^{n+1}_{i,j} = \mathbf{S}^n_{i,j} \setminus \tau_k \tag{23}$$
$$\Theta^{n+1}_{i,j} = \Theta^n_{i,j} + \min(P^n_{i,j}, E_j(R_k) * E_k(R_i)) * \delta_k \tag{24}$$
$$P^{n+1}_{i,j} = P^n_{i,j} - E_j(R_k) * E_k(R_i) \tag{25}$$

Terminal condition:
The iteration is finished when all preemptions have been covered. This is presented in Equation 26.

$$P^{n+1}_{i,j} \quad \leq \quad 0 \tag{26}$$

Due to the step from Equation 16 to Equation 17 $\Theta^{n+1}_{i,j}$ holds the value $\Delta_{i,j}(R_i)$ after the last iteration. Which can than be put into Equation 14.

$$\Delta_{i,j}(R_i) := \Theta^{n+1}_{i,j} \tag{27}$$

Finally the termination of the iterations has to be proven. Starting point for this is that set $\mathbf{S}^1_{i,j}$ has a limited number of elements, thus eventually task τ_i is picked and leads Equation 25 to be:

$$P^{n+1}_{i,j} = P^n_{i,j} - E_j(R_i) * E_i(R_i) \tag{28}$$

Where $E_i(R_i) \geq 1$ (cf. Equation 2) and $P^n_{i,j} \leq P^1_{i,j} = E_j(R_i)$ (cf. Equation 19). Assuming the minimum value for $E_i(R_i)$ and the maximum value for $P^1_{i,j}$ when task τ_i is chosen in Equation 22 Equation 25 can be transformed to:

$$P^{n+1}_{i,j} \quad \leq \quad E_j(R_i) - E_j(R_i) \tag{29}$$
$$\rightsquigarrow \quad P^{n+1}_{i,j} \quad \leq \quad 0 \tag{30}$$

Equation 30 corresponds to the terminal condition in Equation 26. Thus the termination of the iterative process is proven.

4 Results

The approach of Lee et al. presented in [2] focuses on the computation of the additional preemption delay by intersecting cachelines of preempted and preempting tasks. In order to provide this a detailed analysis of possible schedules is made which is then solved by an integer linear programming approach. Since this approach provides an integrated approach for WCET and schedulability analysis and

Task	Frequency [Hz]	Period [ms]	Computation Time [ms]	Cache Load [ms]	Scheduling [ms]	Test System
1	31	32.26	2	0.50	0.10	1
2	17	58.82	4	1.00	0.11	1
3	12	83.33	5	2.50	0.11	2
4	10	100.00	7	2.75	0.12	2
5	7	142.86	9	3.25	0.13	1,2
6	6	166.66	10	4.25	0.14	2
7	5	200.00	13	5.25	0.14	1,2
8	3	333.33	21	2.25	0.16	1

Table 1. Sample Task System

is limited to instruction cache effects, the adaption to uptodate processors is nearly impossible.

A quantitative comparison is only possible with the work of Busquets et al. in [1]. As in [1] the PN series of the hartstone benchmark [11] is taken as basis for the comparison test. The PN series is a set of 5 periodic tasks with different, non-harmonic inter-arrival times. Due to the age of the test the processor performance given in the benchmark does not correspond with modern processors and some relevant data is not specified. For the test no code is executed, but simply both schedulability analysis are exercised.

Since the blocking time is equivalent in both approaches, it is not separately considered. Additionally a single triggering tuple according to Section 2.1 will be used, thus reducing it to simple periodic tasks and thus making the approaches compatible for comparison.

The time needed to load the number of cachelines the task utilises is specified with the term *cache load*. The value *scheduling* contains the time needed for a task switch and a penalty for the interference of the other acceleration techniques like branch prediction or pipeline.

Table 4 provides the basis for the following investigation. For the Equation 4 of Busquets et al. in the extrinsic interference γ_j is comprised of the sum of the cache load and the scheduling penalty for the approach in this paper the penalty δ_i is given by:

$$\delta_i = scheduling_i + cacheload_i * usabilityfactor \quad (31)$$

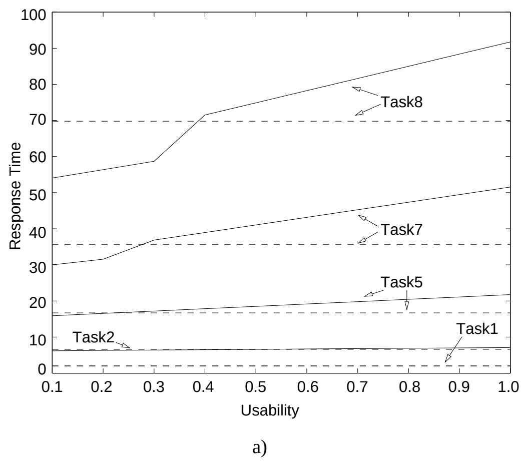
It has to be noted, that this method is only used for comparison of the approaches. In the final application δ_i has to be provided by the WCET analysis. The Figures 3 a) and b) give an overview of the response times of the tasks of the sample systems with varying usability of the cache contents. The dashed lines corresponds to the response time of the tasks computed with the method of Busquets et al. [1]. For Figure 3 a) only the task set consisting of the tasks 1, 2, 5, 7 and 8 is considered which corresponds to the setting in the hartstone PN series.

As can easily be seen up to a usability of 30 % of the contents of cache the new approach is superior for the given task system, while the approach degrades when a utilisation of more then 40 % is reached. Analysis of a number of programs have shown, that a typical amount of useful data in the cache is about 30 to 40 percent. For the second test system only tasks 3, 4, 5, 6 and 7 are taken, thus simulating a set of more uniform requirements. Figure 3 b) depicts the response times for this set of tasks. For this system the break even point of the two approach is reached later at approximately 70 %.

For the approach described in [3] the usability of the cache is less due to the method of estimating the WCET. Thus it also depends on the method of WCET estimation which has to provide the data regarding cache usage, which of both methods is superior.

5 Conclusion

The presented approach provides an alternative method for computing the worst case response time of tasks. As has been shown the choice of method depends strongly on the system under test and on the method used to provide the penalties for cache preemption. Fortunately the complexity of the given approach is only slightly greater than in [1] and much less then the work in [2]. Thus it is possible to imple-



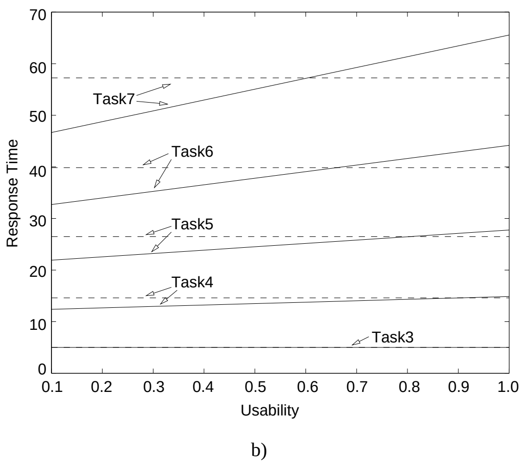


Figure 3. Response Times of the Sample Task Systems with Varying Usability

ment the method in [1] and the approach presented in this paper within one tool and then choose individually for each preemption pair (τ_j preempts τ_i) which approach provides the lesser impact due to the fact that both approaches provide an upper bound on the preemption delay. Future work in this area will focus on extending the approach to arbitrary deadlines.

References

[1] J. V. Busquets-Mataix, D. Gil, P. Gil, and A. Wellings, “Techniques to increase the schedulable utilization of cache-based preemptive real-time systems,” *Journal of System Architecture*, vol. 46, pp. 357–378, 2000.

[2] C. Lee, J. Hahn, Y. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim, “Enhanced analysis of cache-related preemption delay in fixed-priority preemptive scheduling,” *IEEE Transactions on Software Engineering*, 1998.

[3] S. M. Petters, “Bounding the execution of real–time tasks on modern processors,” in *Proc. of the 7th Int. Conf. on Real–Time Computing Systems and Applications*, (Cheju Island, South Korea), Dec. 12–14 2000.

[4] K. Gresser, “An event model for deadline verification of hard real–time systems,” in *Proc. Fifth Euromicro Workshop on Real Time Systems*, (Finland), pp. 118–123, IEEE, June 1993.

[5] S. M. Petters, A. Muth, T. Kolloch, T. Hopfner, F. Fischer, and G. Färber, “The REAR framework for emulation and analysis of embedded hard real–time systems,” *Design Automation for Embedded Systems*, vol. 5, pp. 237–250, Aug. 2000.

[6] A. Burns, “Preemptive priority based scheduling: An appropriate engineering approach,” in *Advances in Realtime Systems*, pp. 225–248, Prentice–Hall International, Inc., 1994.

[7] P. K. Harter, “Response times in level-structured systems,” *ACM Trans. Computer Systems*, vol. 5, pp. 232–248, Aug. 1987.

[8] M. Joseph and P. K. Pandya, “Finding response times in a real-time system,” *The Computer Journal (British Computer Society)*, vol. 29, pp. 390–395, Oct. 1986.

[9] R. Rajkumar, *Synchronization in Real–Time Systems. A Priority Inherintance Approach*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

[10] S.-S. Lim *et al.*, “An accurate worst case timing analysis for RISC processors,” *IEEE Transactions on Software Engineering*, vol. 21, Nr. 7, pp. 593–603, July 1995.

[11] N. H. Weiderman and N. I. Kamenoff, “Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems,” *Journal of Realtime Systems*, vol. 4, pp. 353–382, 1992.