Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption

Reinder J. Bril[1,2], Wim F.J. Verhaegh[2], and Johan J. Lukkien[1]

[1] *Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands*
[2] *Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands*
{*r.j.bril,j.j.lukkien*}*@tue.nl,* {*reinder.bril,wim.verhaegh*}*@philips.com*

Abstract

In this paper, we present equations to determine the exact worst-case response times of periodic tasks under fixed-priority scheduling with deferred preemption (FPDS) and arbitrary phasing. We show that the worst-case response time analysis is not uniform for all tasks. Our exact analysis is based on a dedicated conjecture for an ε*-critical instant, and uses the notion of* worst-case occupied time.

1 Introduction

Based on the seminal paper of Liu and Layland [10], many results have been achieved in the area of worst-case analysis for fixed-priority preemptive scheduling (FPPS). Arbitrary preemption of real-time tasks has a number of drawbacks, though. In particular in systems using cache memory, e.g. to bridge the speed gap between processors and main memory, arbitrary preemptions induce additional cache flushes and reloads. As a consequence, system performance and predictability are degraded, complicating system design, analysis and testing [4, 5, 8, 11]. Although fixed-priority non-preemptive scheduling (FPNS) may resolve these problems, it generally leads to reduced schedulability compared to FPPS. Therefore, alternative scheduling schemes have been proposed between the extremes of arbitrary preemption and no preemption. These schemes are also known as deferred preemption or co-operative scheduling [2], and are denoted by fixed-priority scheduling with deferred preemption (FPDS) in the remainder of this paper.

Worst-case response time analysis of periodic real-time tasks under FPDS and arbitrary phasing has been addressed in a number of papers [2, 3, 4, 8]. Those papers present a single equation for the worst-case response time analysis for all tasks, i.e. their approach is uniform for all tasks. In this paper, we will show that the exact worst-case response time analysis is not uniform for all tasks. Our analysis is based on a dedicated theorem for an ε-critical instant, and uses a notion that has already been used implicitly in [12] to determine slack, and which we term *worst-case occupied time*. For space considerations, we only discuss results; proofs will appear elsewhere.

This paper is organized as follows. Section 2 describes worst-case analysis of periodic tasks under FPPS and arbitrary phasing. For completeness reasons, we start with a recapitulation of worst-case response times . We subsequently address worst-case occupied times. In Section 3, we present our results for FPDS and arbitrary phasing. In Section 4, we briefly compare our results with those presented in the literature, and show that the application of existing results yields values that are either too optimistic or too pessimistic in specific situations.

2 Worst-case analysis for FPPS

2.1 Basic model

We assume a single processor and a set Γ of n periodic, independent tasks $\tau_1, \tau_2, \ldots, \tau_n$. Each task τ_i is characterized by a (*release*) *period* $T_i \in \mathbb{R}^+$, a (*worst-case*) *computation time* $C_i \in \mathbb{R}^+$, and a (*relative*) *deadline* $D_i \in \mathbb{R}^+$. In this paper, we assume that a task's deadline does not exceed its period, i.e. $D_i \leq T_i$ for each i. A release of a task is also termed a job. The release of task τ_i at time $\varphi_i \in \mathbb{R}$ with $0 \leq \varphi_i < T_i$ serves as a reference release. Time φ_i is also termed the *phasing* of task τ_i, and $\varphi = (\varphi_1, \ldots, \varphi_n)$ is called the phasing of the task set Γ. We assume that we do not have control over the phasing φ, for instance since the tasks are released by external events, so we assume that any arbitrary phasing may occur. This assumption is common in real-time scheduling literature [6, 7, 10]. We also assume other standard basic assumptions [10], i.e. tasks are ready to run at the start of each period and do no suspend themselves, tasks will be preempted instantaneously when a higher priority task becomes ready to run, a job of task

τ_i does not start before its previous job is completed, and the overhead of context switching and task scheduling is ignored. Finally, we assume that the tasks are given in order of decreasing priority, i.e. task τ_1 has highest priority and task τ_n has lowest priority.

2.2 Recapitulation of worst-case response times

The worst-case response time of a task is the length of the longest interval from a task's release till its completion. To determine worst-case response times under arbitrary phasing, it suffices to consider only so-called *critical instants* [10]. For FPPS, critical instants are given by time points at which all tasks have a simultaneous release. From this notion of critical instants, Joseph and Pandya [6] have derived that the worst-case response time R_i of a task τ_i is given by the smallest $x \in \mathbb{R}^+$ that satisfies

$$x = C_i + \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil C_j. \qquad (1)$$

Assuming a critical instant at time zero, the factor $\left\lceil \frac{x}{T_j} \right\rceil$ in (1) gives the worst-case number of preemptions that an execution of task τ_i suffers from task τ_j in an interval $[0,x)$. To calculate worst-case response times, we can use an iterative procedure based on recurrence relationships [1].

Table 1 provides an example with characteristics of a task set Γ consisting of three tasks, and the worst-case response times under FPPS. We used the superscript P to denote FPPS in Table 1. Similarly, we will use superscripts D and N later to denote FPDS and FPNS, respectively.

Figure 1 shows a timeline with the executions of these three tasks and a critical instant at time zero.

2.3 Worst-case occupied time

The worst-case occupied time of a task is closely related to its worst-case response time. The worst-case occupied time O_i of a task τ_i is defined as the length of the longest possible interval from a release of that task during which the processor is *occupied* with the execution of C_i of that job and executions of higher priority tasks till the moment in time that the same job could start an additional bit of computation. Hence, O_i includes preemptions *and* aligning successive executions of higher priority tasks. For $D_i \leq T_i$, the worst-case occupied time of τ_i is the worst-case response time of that task extended with all aligning successive executions of higher priority tasks. Note that we only consider a single job of τ_i for the worst-case occupied time, making our notion different from the notion of *busy period* [9].

Given this above definition, we may simply derive the worst-case occupied times for Γ from Figure 1. For the highest priority task τ_1, O_1 equals R_1, i.e. $O_1 = 2$. For task τ_2, O_2 extends R_2 with the aligning execution of task τ_1, i.e. $O_2 = R_2 + C_1 = 7$. Note that the aligning job of task τ_2 itself at time seven is not included. Finally, O_3 of task τ_3 extends R_3 with the aligning executions of both τ_1 and τ_2, i.e. $O_3 = R_3 + C_1 + C_2 = 33$. Note that O_3 of task τ_3 is larger than its deadline D_3 and its period T_3.

To determine worst-case occupied times under arbitrary phasing, we can use a similar approach as for worst-case response times. However, rather than considering releases of higher priority tasks in an interval $[0,x)$, we need to consider releases of those tasks in an interval $[0,x]$, i.e. including those at time x. As already explained in [12], this can be done by using the floor rather than the ceiling function, and adding a term 1, i.e. the worst-case occupied time O_i of a task τ_i is given by the smallest $x \in \mathbb{R}^+$ that satisfies

$$x = C_i + \sum_{j<i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor + 1 \right) C_j. \qquad (2)$$

Similarly to worst-case response times, we can use an iterative procedure based on recurrence relationships to calculate worst-case occupied times.

Unlike the worst-case response time, the worst-case response time is well defined for a computation time of zero. In this case, the worst-case occupied time is equal to the *worst-case start time*, i.e. the length of the longest interval from a release of a task till the start of its execution. For our leading example, the worst-case start time S_1^{P} of task τ_1 equals 0, $S_2^{\mathrm{P}} = 2$, and $S_3^{\mathrm{P}} = 12$; cf. Figure 1.

2.4 Concluding remarks

In the remainder, we will parameterize R_i and O_i of task τ_i with C_i when needed. As illustrated in [7], blocking can

	T_i	D_i	C_i	R_i^{P}	R_i^{D}	R_i^{N}
τ_1	5	4	2	2	4	6
τ_2	7	7	1 + 2	5	7	13
τ_3	30	30	2 + 2	28	21	16

Table 1. Task characteristics and worst-case response times under FPPS, FPDS, and FPNS.

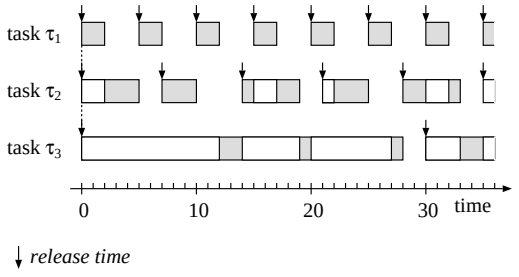

Figure 1. Timeline under FPPS with a critical instant at time zero.

be taken into account when calculating R_i of task τ_i by incorporating a worst-case blocking term B_i in C_i.

3 Worst-case analysis for FPDS

3.1 Refined model

For FPDS, we need to refine our basic model of Section 2.1. Each job of task τ_i is now assumed to consist of $m(i)$ subjobs. The j^{th} subjob of τ_i is characterized by a computation time $C_{i,j} \in \mathbb{R}^+$, where $C_i = \sum_{j=i}^{m(i)} C_{i,j}$. We assume that subjobs are non-preemptable. Hence, tasks can only be preempted at subjob boundaries, i.e. at so-called *preemption points*. For convenience, we will use the term F_i to denote the computation time $C_{i,m(i)}$ of the final subjob of τ_i. Note that when $m(i) = 1$ for all i, we have FPNS as special case.

3.2 Worst-case response times

The non-preemptive nature of subjobs may cause blocking of a task by at most one lower priority task under FPDS. The maximum blocking of task τ_i by a lower priority task is equal to the longest computation time of any subjob of a task with a priority lower than task τ_i, which is given by

$$B_i = \max_{j>i} \max_{1\leq k\leq m(j)} C_{j,k}. \tag{3}$$

To determine worst-case response times under FPDS and arbitrary phasing, we have to revisit critical instants. In this paper, we merely postulate the following conjecture.

Conjecture 1 *An ε-critical instant of a task τ_i under FPDS and arbitrary phasing occurs when that task is released simultaneously with all tasks with a higher priority than τ_i, and the subjob with the longest computation time of all lower priority tasks starts an infinitesimal time $\varepsilon > 0$ before that simultaneous release.*

From this conjecture we conclude that a critical instant for FPDS is a supremum for all but the lowest priority task, i.e. that instant cannot be assumed.

For the analysis, we consider three cases: the highest priority task τ_1, the lowest priority task τ_n, and a medium priority task τ_i (with $1 < i < n$).

Task τ_1 may be blocked, but is never preempted. The worst-case response time R_1^{D} of task τ_1 therefore includes a term B_1, i.e.

$$R_1^{\mathrm{D}} = B_1 + C_1. \tag{4}$$

Note that $B_1 + C_1$ is a supremum, i.e. that value cannot be assumed, but it can be approximated arbitrarily closely. Further note that this latter equation may also be written as $R_1^{\mathrm{D}} = R_1^{\mathrm{P}}(B_1 + C_1)$, or $R_1^{\mathrm{D}} = R_1^{\mathrm{P}}(B_1 + C_1 - F_1) + F_1$. Because $R_1^{\mathrm{P}} = O_1^{\mathrm{P}}$, the equation may even be written as $R_1^{\mathrm{D}} = O_1^{\mathrm{P}}(B_1 + C_1)$, or $R_1^{\mathrm{D}} = O_1^{\mathrm{P}}(B_1 + C_1 - F_1) + F_1$.

Task τ_n may be preempted (at subjob boundaries), but is never blocked. The worst-case response time R_n^{D} of task τ_n can hence be found by calculating the worst-case start time of the final subjob, and adding its computation time F_n. The non-preemptive nature of the other subjobs of τ_n may result in deferred preemptions by higher priority tasks. Although that has an influence on the order of the executions of the subjobs of tasks, it does not influence the total amount of time spent on those executions. The amount of time spent on executions of all but the final subjob of τ_n including the (deferred) preemptions of higher priority tasks is given by $R_n^{\mathrm{P}}(C_n - F_n)$. The final subjob of τ_n may subsequently start after the aligning successive executions of higher priority tasks have completed. Hence, the worst-case start time of the final subjob of task τ_n is given by $O_n^{\mathrm{P}}(C_n - F_n)$, and we arrive at the following equation for R_n^{D}.

$$R_n^{\mathrm{D}} = O_n^{\mathrm{P}}(C_n - F_n) + F_n \tag{5}$$

Note that $O_n^{\mathrm{P}}(C_n - F_n) + F_n$ is a maximum, i.e. that value can be assumed. Further note that for $m(n) = 1$, we get $O_n^{\mathrm{P}}(C_n - F_n) = O_n^{\mathrm{P}}(0)$, which is equal to the worst-case start time S_n^{P} of task τ_n.

Task τ_i with $1 < i < n$, may be both preempted at subjob boundaries by higher priority tasks and blocked by a lower priority task. Similarly to task τ_n, the worst-case response time R_i^{D} of τ_i can be found by calculating the worst-case start time of the final subjob, and by subsequently adding its computation time F_i. Similarly to τ_n, the non-preemptive nature of the other subjobs of τ_i has no influence on the worst-case start time of the final subjob. At first hand, it therefore looks as if the same reasoning applies as for τ_n, and that we can calculate the worst-case start time by means of $O_i^{\mathrm{P}}(B_i + C_i - F_i)$. However, the blocking subjob of the lower priority tasks has to start an infinitesimal time $\varepsilon > 0$ *before* the simultaneous release of τ_i and its higher priority tasks. Hence, the amount of time spent from the release of τ_i on executions of the blocking subjob and all but the final subjob of τ_i including the (deferred) preemptions of higher priority tasks is given by $O_i^{\mathrm{P}}(B_i - \varepsilon + C_i - F_i)$. This latter value is equal to $R_i^{\mathrm{P}}(B_i + C_i - F_i)$ *minus* an infinitesimal time $\varepsilon > 0$. It is exactly this infinitesimal difference, which approaches zero, that allows the final subjob of τ_i to start executing, and that defers potential additional preemptions from higher priority tasks at time $R_i^{\mathrm{P}}(B_i + C_i - F_i)$. The worst-case response time R_i^{D} of τ_i is therefore given by

$$R_i^{\mathrm{D}} = R_i^{\mathrm{P}}(B_i + C_i - F_i) + F_i. \tag{6}$$

Note that $R_i^{\mathrm{P}}(B_i + C_i - F_i) + F_i$ is also a supremum.

As mentioned above, we may rewrite (4) to $R_1^{\mathrm{D}} =$

$O_1^{\mathrm{P}}(B_1+C_1-F_1)+F_1$ as well as to $R_1^{\mathrm{D}}=R_1^{\mathrm{P}}(B_1+C_1-F_1)+F_1$. Hence, (4) is similar to both (5) and (6).

3.3 An example

To illustrate the equations, consider the task characteristics of Table 1. For FPNS, the set is not schedulable because the worst-case response time R_1^{N} of task τ_1 is equal to $B_1+C_1=4+2=6$, which exceeds D_1.

For FPDS, let $m(1)=1$, $m(2)=2$ with $C_{2,1}=1$ and $C_{2,2}=2$, and $m(3)=2$ with $C_{3,1}=C_{3,2}=2$; see Table 1. Using the equations above yields $R_1^{\mathrm{D}}=B_1+C_1=2+2=4$, $R_2^{\mathrm{D}}=R_2^{\mathrm{P}}(B_2+C_2-F_2)+F_2=R_2^{\mathrm{P}}(2+3-2)+2=7$, and $R_3^{\mathrm{D}}=O_3^{\mathrm{P}}(C_3-F_3)+F_3=O_3^{\mathrm{P}}(2)+2=21$. Hence, by splitting both task τ_2 and task τ_3 into two non-preemptive subjobs, the task set becomes schedulable under FPDS.

4 Related Work

We briefly compare our results with those presented in the literature. The schedulability test in [5] is based on utilization bounds, and is therefore typically pessimistic. The worst-case response time analysis presented in [8] is based on a single equation, i.e. it is uniform for all tasks. The blocking effect of (partially) non-preemptive lower priority tasks has been covered in that analysis, but the effect of the non-preemptive nature of the final subjob is not taken into account. The analysis is therefore pessimistic.

The results presented in [2, 3, 4] are very similar to ours. Unlike our approach, their approach is uniform for all tasks. Using our notation, the worst-case response time $\tilde{R}_i^{\mathrm{D}}$ under FPDS and arbitrary phasing presented in [2, 4] is given by

$$\tilde{R}_i^{\mathrm{D}}=R_i^{\mathrm{P}}(B_i+C_i-(F_i-\Delta))+(F_i-\Delta). \qquad (7)$$

According to [4], Δ is an arbitrary small positive value needed to ensure that the final subjob has actually started. Hence, when task τ_i has consumed $C_i-(F_i-\Delta)$, the final subjob has (just) started. When Δ approaches to zero, we may rewrite (7) to

$$\tilde{R}_i^{\mathrm{D}}=O_i^{\mathrm{P}}(B_i+C_i-F_i)+F_i.$$

This result is identical to ours for the highest and lowest priority tasks, but differs from ours for intermediate tasks. For the example presented above, our analysis yields $R_2^{\mathrm{D}}=7$, whereas the analysis presented in [4] yields $\tilde{R}_2^{\mathrm{D}}=9$. In the example, this latter result is too pessimistic; because $\tilde{R}_2^{\mathrm{D}}$ exceeds D_2, the task set would incorrectly be considered non-schedulable. This difference between R_2^{D} and $\tilde{R}_2^{\mathrm{D}}$ can be traced back to Conjecture 1. The analysis in [4] does not take into account that τ_i can only be blocked by a subjob of a lower priority task if that subjob also starts an amount of time Δ *before* the simultaneous release of τ_i and all tasks with a higher priority than τ_i. When this aspect is be taken into account in the analysis of [4], e.g. when B_i is replaced by $B_i-\Delta$ in (7), their result becomes identical to ours.

In the scheduling analysis review presented in [3], the worst-case response time $\tilde{R}_i^{\mathrm{D}}$ ignores the term Δ, i.e.

$$\tilde{R}_i^{\mathrm{D}}=R_i^{\mathrm{P}}(B_i+C_i-F_i)+F_i.$$

This result is identical to ours, except for the lowest priority task. For the example presented above, this results in $\tilde{R}_3^{\mathrm{D}}=R_3^{\mathrm{P}}(C_3-F_3)+F_3=16$, which is too optimistic.

References

[1] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proc. 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, May 1991.

[2] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[3] A. Burns. Defining new non-preemptive dispatching and locking policies for Ada. In *Proc. 6th Ada-Europe International Conference*, pages 328–336, May 2001.

[4] A. Burns and A. Wellings. Restricted tasking models. In *Proc. 8th Int. Real-Time Ada Workshop*, pages 27–32, 1997.

[5] R. Gopalakrishnan and G. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proc. ACM Sigmetrics Conf. on Measurement & modeling of computer systems*, pages 1–12, May 1996.

[6] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[7] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[8] S. Lee, C.-G. Lee, M. Lee, S. Min, , and C.-S. Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In *Proc. ACM Sigplan Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 51–64, June 1998.

[9] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 201–209, December 1990.

[10] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[11] J. Simonson and J. Patel. Use of preferred preemption points in cache-based real-time systems. In *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS'95)*, pages 316–325, April 1995.

[12] S. Thuel and J. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proc. 15th IEEE Real-Time Systems Symposium (RTSS)*, pages 22–33, April 1994.