

# Adapting Preemptive Scheduling Analysis to cope with Non-Preemption and Inserted Idle-Time

L. Almeida, J. A. Fonseca

{lda,jaf}@det.ua.pt

Dep. de Electrónica e Telecomunicações / IEETA

Universidade de Aveiro

P-3810-193 Aveiro, Portugal

## Abstract

*Non-preemptive scheduling is known for its lower efficiency in meeting temporal constraints when compared to preemptive scheduling. However, it is still used in certain cases such as in small multi-tasking kernels for embedded systems based on simple microprocessors and, mainly, in message scheduling over serial broadcast buses. Both cases are typically found in control applications requiring the periodic execution (or transmission) of a set of tasks (or messages) with low jitter. This paper shows how such low jitter can be achieved using inserted idle-time in order to circumvent the typical blocking caused by non-preemptive scheduling. A schedulability analysis for the case of fixed priorities is also presented which is based on existing analysis for preemptive scheduling. Current work is being developed for the case of dynamic priorities.*

## 1. Introduction

Non-preemptive scheduling is known for its lower efficiency in meeting temporal constraints when compared to preemptive scheduling. In fact, non-preemption causes higher blocking factors that can easily lead to non-schedulability. When applied to periodic tasks, it also imposes a well known limitation to the tasks' execution time, i.e. the largest task execution time must be shorter than the shortest task relative deadline.

However, it is still used in certain cases such as in small multi-tasking kernels for embedded systems based on simple microprocessors and, mainly, in message scheduling over serial broadcast buses. In the former case, it has the advantage of a lower run-time overhead and simple resource management, in the latter, its use is imperative to support a coherent serial transfer of each message.

Many applications of both one case or the other are in the control of physical devices normally requiring the periodic execution (or transmission) of a set of tasks (or messages) with low jitter. One of the sources of jitter is the blocking caused by non-preemptive scheduling but this can be reduced, or even eliminated, by adequately using inserted idle-time, period transformation and relative phasing. In

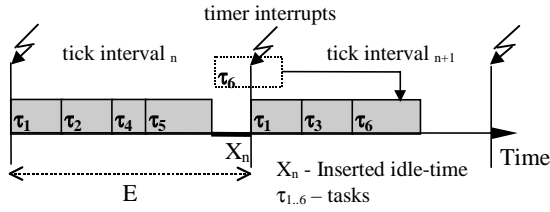
particular, this paper deals with the use of inserted idle-time to delay the release of any lower priority task (message) whenever a higher priority one will become ready before the former one would finish. This is easily achieved in synchronised systems where the release of tasks (messages) is performed by means of a periodic timer interrupt. This paper shows how it can be achieved and presents a schedulability analysis of periodic non-preemptive tasks with fixed priorities using inserted idle-time. This analysis is based on existing analysis available for preemptive scheduling of tasks with fixed priorities.

## 2. Related work

It is known that non-preemptive scheduling causes a considerable schedulability penalty and many systems which are schedulable preemptively cannot be timely scheduled with non-preemption [1]. Nevertheless, when tasks' execution times are small compared to the respective periods then non-preemption is not generally a serious drawback in terms of schedulability. In this case it is known that non-preemptive scheduling approaches preemptive scheduling in terms of the level of schedulability [2].

Nevertheless, well-known results deduced for preemptive scheduling cannot be used directly in a non-preemption situation. Either some degree of adaptation must be done on the existing preemptive analysis or new analysis must be derived. For example, Jeffay *et al.* [3] use a new analysis to prove that the well-known EDF algorithm is also optimal for non-preemptive task scheduling. On the other hand, Vasques [4] suggests the use of an adequate blocking term in usual preemptive RM analysis to account for the effects of non-preemption. The maximum such blocking that a task can suffer equals the duration of the longest task among those with lower priority. Furthermore, this blocking also has a negative impact on tasks' release jitter which can be a drawback for example in control applications.

A way to reduce non-preemption blocking and improve schedulability is to use idle-time insertion [5]. It consists in delaying long non-preemptive tasks in order to wait for the release of tasks with shorter deadlines and higher priority so



**Figure 1. Inserting idle-time to prevent blocking.**

that they can execute first. Otherwise such tasks would miss their deadlines.

This technique can also be used to reduce the release jitter of high priority tasks. For example, consider a synchronous system where tasks become ready synchronously with a timer interrupt. Moreover, consider that a task is not released if it cannot finish before the next interrupt. Then, when the interrupt comes, such task is rescheduled together with the remaining ones. Whenever this occurs, a small amount of idle-time (implicitly inserted) may appear just before the interrupt (fig. 1).

This technique has been proposed and discussed by the authors [6] for message scheduling in synchronous fieldbus systems. Basically, two effects are referred concerning idle-time insertion: avoidance of potential preemption instants, which makes it irrelevant whether the scheduling is preemptive or non-preemptive, and a bounded waste of bus-time (CPU in this case), corresponding to the inserted idle-time. The former effect is responsible for the avoidance of blocking and guarantees a jitter-free periodic release of the highest priority task (fig. 1). The latter effect invalidates the direct applicability of the existing analysis for fixed priorities preemptive scheduling (e.g [7]), leading to the need for a new analysis.

### 3. Task model

The analysis presented in the paper considers a set  $\Gamma$  of  $N$  independent periodic non-preemptive tasks. Each task  $\tau$  is an infinite succession of instances which are periodically activated. Nevertheless, no instance is released for execution until the previous one is terminated.

The parameters that characterise a generic task  $\tau_i$  are the computation time  $C_i$ , the period  $T_i$ , the relative deadline  $D_i$  which is equal to or shorter than the period, an initial phasing  $\Phi_i$  expressing the activation instant of the first instance, and a fixed priority  $P_i$  which can be, but not necessarily, derived from the period, deadline or value in the context of the application. This is formalised in expression (1).

$$\Gamma \equiv \{ \tau_i ( C_i , T_i , D_i , \Phi_i , P_i ) , i = 1..N \} \quad (1)$$

The instances of each task are activated by a timer interrupt which has a period of  $E$ . The periods of all tasks are

expressed as integer multiples of  $E$ , i.e.  $\forall_{i=1..N} T_i = k_i * E$ ,  $k_i \geq 1$ . The same constraint applies to the initial phasing  $\Phi_i$ . Furthermore, it will be assumed that each task must be short enough to execute within  $E$ , i.e.  $\forall_{i=1..N} C_i < E$ . In typical application areas for which this analysis is suited, this assumption is not restrictive, e.g. in fieldbus communication systems where typical messages are very short, in the control of small autonomous robots using reflex behaviour approaches where typical tasks are short.

In the analysis that follows, the task set  $\Gamma$  will be assumed to be ordered by decreasing priority, i.e.  $\forall_{i,j=1..N} i < j \Rightarrow P_i > P_j$ .

### 4. Worst-case phasing

In order to determine the worst-case scenario concerning the interference caused by higher priority tasks to the release of lower priority ones, three aspects must be taken into account. Firstly, that tasks are activated synchronously with the tick interrupt and that the initial phasings are also expressed as integer multiples of the tick duration. Thus, no task becomes ready for execution in the middle of a tick interval. Secondly, that tasks are not released for execution unless they can complete within the current tick interval thus leading to a possible implicit idle-time insertion. Both previous aspects make it irrelevant whether the scheduling is preemptive or not (fig. 1) since potential preemption instants are avoided. Thirdly, that the order by which ready tasks are executed respects their relative priorities, i.e. a lower priority task cannot execute if there is a higher priority one, ready for execution. Thus, whenever a portion of idle-time is inserted no lower priority tasks can execute in that portion of time even if they fit in.

The previous three aspects altogether lead to a situation which is equivalent to the scheduling of fixed priorities preemptive tasks in terms of worst-case scenario. In fact, the interference felt by each task is then maximised when it is activated together with all higher priority tasks. Thus, to assess the schedulability of a given task set it suffices to consider all tasks in phase, i.e. all activated at a given instant in time known as the critical instant (e.g.  $t=0$ ), and check if the first instance of each task meets its deadline.

### 5. Adapting existing analysis for fixed priority preemptive systems

Although it is not possible to directly apply the typical analysis for fixed priorities preemptive scheduling as referred before, such analysis can still be used if a small adaptation is carried out. This is expressed in the following:

**Theorem:** Consider a set of synchronous tasks  $\Gamma$  as defined in (1) but with  $\Phi_i=0 \forall_{i=1..N}$ , i.e. the first instance of

all tasks becomes ready at instant  $t=0$ .  $E$  stands for the tick duration and the scheduler uses inserted idle-time as well as enforces priority ordering in the execution of tasks. Consider  $X = \max_n(X_n)$  which stands for the maximum inserted idle-time in any tick (fig. 1). Also consider the hypothetical set of independent preemptable tasks  $\Gamma'$  defined as in (2) and scheduled according to the same static priorities assignment but without inserted idle-time.

$$\Gamma' \equiv \{ \tau'_i (C'_i, D_i, T_i, P_i), C'_i = C_i * \frac{E}{E-X} \quad \forall i=1..N \} \quad (2)$$

Then:

Set  $\Gamma'$  is guaranteed to be schedulable by at least one sort of analysis for fixed priority preemptive scheduling  $\Rightarrow$  Set  $\Gamma$  is also schedulable

**Proof:** Define function  $H_i(t)$  as in expression (3), which represents the cumulative demand for CPU time at instant  $t$  by the instances of tasks  $\tau_1$  to  $\tau_i$ . This function contains two terms, the former due to the tasks effective processing and the latter due to the inserted idle-time. Starting from  $t=0^+$ , the value of  $H_i(t)$  raises in steps whenever a new task instance becomes ready. As long as there are uncompleted ready tasks, the demand is higher than the effective CPU time, i.e.  $H_i(t) > t$  (see fig. 2 for task 9 of task set of table 1). However, if the CPU has enough capacity to process all the tasks then  $H_i(t)$  will grow slower than  $t$ . When  $t$  reaches  $H_i(t)$  it means that all the instances of tasks  $\tau_1$  to  $\tau_i$  that became ready in the interval  $[0,t)$  have been completed (the interval  $[0,t)$  is then called the level  $i$  busy interval). Furthermore, such instant  $t$  also corresponds to the response time of the first instance of task  $\tau_i$ , or, more precisely, the worst-case response time ( $Rwc_i$ ) since all tasks are in phase. This is expressed in equation (4).

$$H_i(t) = \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C_j + \sum_{n=1}^{\left\lceil \frac{t}{E} \right\rceil - 1} X_n \quad \forall i=1..N \quad (3)$$

$$Rwc_i = H_i(Rwc_i), \quad \forall i=1..N \quad (4)$$

On the other hand, a similar function,  $H'_i(t)$ , can be established, as in expression (5), for set  $\Gamma'$  which represents the cumulative processor demand at instant  $t$  by the tasks  $\tau'_1$  to  $\tau'_i$ . The worst-case response time ( $Rwc'_i$ ) can also be written as the solution to equation (6).

$$H'_i(t) = \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C'_j \quad \forall i=1..N \quad (5)$$

**Table 1. Set of N tasks** ( $\forall i=1..N \quad D_i = T_i, \Phi_i = 0, P_i = 1/i$ )

i	1	2	3	4	5	6	7	8	9
$C_i$ (ms)	0.21	0.21	0.2	0.2	0.2	0.2	0.2	0.14	0.14
$T_i$ (ms)	1	2	2	2	2	4	4	4	4

$$Rwc'_i = H'_i(Rwc'_i), \quad \forall i=1..N \quad (6)$$

The proof of the theorem consists in showing that  $Rwc'_i > Rwc_i, \forall i=1..N$ . Then, if set  $\Gamma'$  is guaranteed to be schedulable then  $D_i \geq Rwc'_i, \forall i=1..N$ . Consequently,  $\Gamma$  is also schedulable since  $D_i \geq Rwc'_i \geq Rwc_i, \forall i=1..N$ .

To prove that  $Rwc'_i > Rwc_i, \forall i=1..N$ , consider the referred functions  $H_i(t)$  and  $H'_i(t)$  restricted to the interval  $(0, Rwc_i]$  or  $(0, Rwc'_i]$  whichever is longer or until one of them grows beyond the deadline  $D_i$ . This latter situation corresponds to the non-schedulability of either  $\Gamma'$  or  $\Gamma$  which is not relevant to this proof. On the other hand, showing that  $H'_i(t) \geq H_i(t)$  in the interval  $(0, Rwc'_i]$  leads to the desired result.

$$\begin{aligned} H'_i(t) \geq H_i(t) &\Leftrightarrow \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C'_j \geq \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C_j + \sum_{n=1}^{\left\lceil \frac{t}{E} \right\rceil - 1} X_n \\ &\Leftrightarrow \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C_j \geq \sum_{n=1}^{\left\lceil \frac{t}{E} \right\rceil - 1} X_n * \frac{E-X}{X} \end{aligned} \quad (7)$$

By definition  $H'_i(t) \geq t$  in the interval  $(0, Rwc'_i]$  yielding condition (8). Notice that, by the definition of the ceiling and floor functions,  $\lceil x \rceil \geq x \geq \lfloor x \rfloor \geq \lceil x \rceil - 1$ .

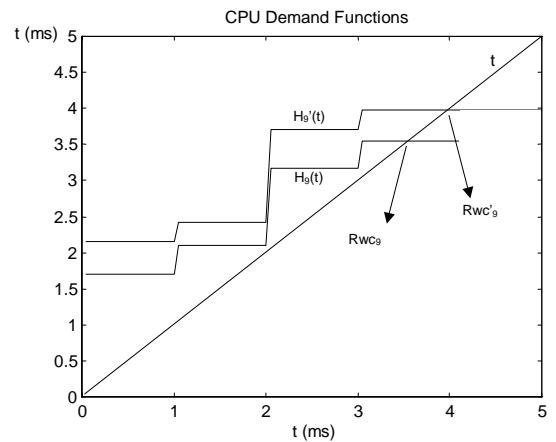
$$\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C_j \geq t \Leftrightarrow \sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil * C_j \geq t * \frac{E-X}{E} = \frac{t}{E} * (E-X) \geq \left( \left\lceil \frac{t}{E} \right\rceil - 1 \right) * (E-X) \quad (8)$$

Taking into account that  $X$  stands for  $\max_n(X_n)$  the following inequality can be written

$$\left( \left\lceil \frac{t}{E} \right\rceil - 1 \right) * X \geq \sum_{n=1}^{\left\lceil \frac{t}{E} \right\rceil - 1} X_n \quad (9)$$

Combining inequalities (8) and (9) proves the condition expressed in (7) and consequently, the theorem (q.e.d.).

The above theorem represents a general result which can be exploited to enlarge the applicability of several analysis



**Figure 2. CPU demand functions for task 9 within the set described in table 1.**

based on fixed-priorities preemptive scheduling to non-preemptive scheduling of periodic and synchronous tasks using inserted idle-time. However, any schedulability assessment resulting from the use of this theorem is always sufficient and not necessary even when the analysis performed on the hypothetical set  $\Gamma'$  is necessary and sufficient. This is due to the fact that the theorem considers  $X$ , the maximum inserted idle-time, and not the effective values  $X_n$ .

A practical difficulty resides in the determination of  $X = \max_n(X_n)$  without actually building the schedule to determine the values of  $X_n$  (the inserted idle-time in the  $n^{\text{th}}$  tick). Hence, an upper bound can be used given by (10).

$$\max_{i=1..N} (C_i) \geq X \geq \max_n (X_n) \quad (10)$$

The enlarged execution times are then given by (11).

$$C_i' = C_i * \frac{E}{E - \max_{j=1..N} (C_j)} \quad \forall_{i=1..N} \quad (11)$$

Two corollaries can now be established concerning the adaptation of two typical analysis: the utilisation-based analysis for rate-monotonic scheduling [8] and the response time-based analysis for fixed-priorities preemptive scheduling [7].

**Corollary 1:** Consider a set of tasks  $\Gamma$  as defined in (1), with  $D_i = T_i \quad \forall_{i=1..N}$  and a priority assignment according to rate-monotonic, i.e.  $T_i < T_j \Rightarrow P_i > P_j \quad \forall_{i,j=1..N}$ . Then:

$U = \sum_{i=1}^N \left( \frac{C_i}{T_i} \right) < N \left( 2^{\frac{1}{N}} - 1 \right) \times \frac{E - X}{E} \quad \Rightarrow$	Set $\Gamma$ is schedulable under any phasing
---	---

This corollary can be easily proved since the hypothetical set  $\Gamma'$  (2) can be scheduled under rate-monotonic if

$$U' = \sum_{i=1}^N \left( \frac{C_i'}{T_i} \right) < N \left( 2^{\frac{1}{N}} - 1 \right)$$

The schedulability of set  $\Gamma$  is then guaranteed by the theorem as long as  $\Gamma'$  is schedulable in the conditions referred. By transforming  $U'$  into  $U = U' * (E - X) / E$  the corollary is then proved (q.e.d.).

**Corollary 2:** Consider a set of tasks  $\Gamma$  as defined in (1), with  $D_i \leq T_i \quad \forall_{i=1..N}$  and any decreasing fixed priority assignment, i.e.  $i < j \Rightarrow P_i > P_j \quad \forall_{i,j=1..N}$ . Then:

$Rwc'_i \leq D_i \quad \forall_{i=1..N} \quad \Rightarrow$	Set $\Gamma$ is schedulable under any phasing
--	---

$Rwc'_i$  is the worst-case response time for an instance of  $\tau'_i$  in  $\Gamma'$  (2) obtained by the usual response time analysis for fixed priority preemptive scheduling [7]. Since when the left-hand side condition is true set  $\Gamma'$  is schedulable, then, by the theorem, corollary 2 is proved (q.e.d.).

## 6. Conclusions and work-in-progress

Although non-preemptive scheduling does not seem very attractive due to the large blocking factors normally associated to it, it is still advantageously used in applications such as small multitasking kernels of embedded systems based on simple microprocessors and message scheduling in fieldbus communication systems.

In this paper it is shown that by implicitly using idle-time insertion in synchronous systems, i.e. those executing periodic tasks, only, activated synchronously with a timer interrupt, it is possible to eliminate the blocking factor associated to non-preemption. The result is a jitter-free release of the highest priority task in the system. If appropriate periods and initial phasing are chosen for the remaining tasks it might be possible to obtain a jitter-free schedule for the whole task set.

Furthermore, the paper shows how the existing analysis for fixed priorities preemptive scheduling can be adapted to cope with inserted idle-time and non-preemption in synchronous systems. The result, nevertheless, is just a sufficient schedulability assessment. Current work is being developed to investigate the possible extension of this result to dynamic priorities.

## References

- [1] Stankovic, J.A. *et al.*. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, **28(6)**, 1995.
- [2] Carderia, C.. *Ordonnancement Temps Réel par Réseaux de Neurones*. PhD Thesis, INPL, Nancy, France, 1994.
- [3] Jeffay, K. , D. Stanat and C.U. Martel. On Non-preemptive Scheduling of Periodic and Sporadic Tasks. *Proc. of IEEE RTSS'91*. San Antonio, USA, 1991.
- [4] Vasques, F.. *Sur l'Intégration de Mécanismes d'Ordonnancement et de Communication dans la Sous-Couche MAC de Réseaux Locaux Temps-Réel*. PhD Thesis, LAAS-CNRS, Université Paul Sabatier, Toulouse, France, 1996.
- [5] Howell, R. and M. Venkatrao. On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Times. *Information and Computation*, **117**, 1995.
- [6] Almeida, L. and José A. Fonseca. Schedulability Analysis in a Real-Time Fieldbus Network. *Proceedings of IFAC SICICA'97*, Annecy, France, June 1997.
- [7] Audsley, N., A. Burns, M. Richardson, K. Tindell and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling. *Software Engineering Journal*, **8(5)**: 285-292, 1993.
- [8] Liu C. L. and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of ACM*, **20(1)**: 46-61, 1973.