FP/FIFO Feasibility Conditions with Kernel Overheads for Periodic Tasks on an Event Driven OSEK System

Franck Bimbard
Cedric/CNAM
292 Rue St Martin FR-75141
75007 Paris Cedex 03
bimbard@ece.fr

Laurent George
ECE, LACSC
53, rue de Grenelle
75007 Paris
lgeorge@ieee.org

Abstract

In this paper we show how to take into account kernel overheads in classical real-time feasibility conditions for Fixed Priority (FP) scheduling where tasks having the same fixed priority are scheduled FP/FIFO. We consider the periodic task model with arbitrary deadlines and an event driven OSEK kernel. The feasibility conditions are based on the worst case response time computation of the tasks. We identify the sources of kernel overheads that influence the response time of the tasks. In such a system the overheads are due to the context switching that activates/terminates and reschedules tasks and to the granularity of the periodic timer used to implement the periodic task model. We show how to take into account those overheads in the classical FP/FIFO feasibility conditions. We compare the worst case response time obtained with kernel overhead to the response time obtained on a real event driven OSEK implementation. We show that the kernel overheads cannot be neglected and that the theoretical results are valid and can be used for a real-time dimensioning.

1 Introduction

Fixed Priority (FP) preemptive scheduling of periodic tasks in real-rime systems has been extensively studied in the last thirty years. The characteristics of a periodic task τ_i are defined in table 1.

Symbol	Description
C_i	The worst case execution time (WCET)
T_i	The period of the task
D_i	The deadline constraint (a task released at time t must be executed by $t + D_i$)
P_i	The priority of the task (priority 0 is the lowest priority)

Table 1. Characteristics of a periodic task τ_i

The starting point for preemptive FP scheduling is in [6] that proposed a simple polynomial time sufficient feasibility condition for the Rate Monotonic (RM) algorithm. The Feasibility Conditions (FC) have then been extended by [4] in the case where $\forall i, D_i \leq T_i$ and by [8], [3] for tasks with no obvious relation between D_i and T_i (arbitrary deadlines). We consider in this paper arbitrary deadlines. The feasibility conditions are based on the worst case response time computation for any periodic task. [7] show that the feasibility conditions obtained for FP can be improved considering that tasks with the same fixed priority are scheduled FIFO. This scheduling, denoted FP/FIFO is the one considered in this paper.
The scheduling model used in the FC is the event driven model. We can notice that classical FC either do not consider kernel overheads or the preemption cost is included as an extra duration in the WCET of the tasks, leading to imprecise FC. For the time driven model, [8] showed how to take into account the cost of the scheduler. The scheduler behaves as periodic tasks with a preemption cost that can be included in the feasibility conditions. Yet, in the event driven model, the solution to increase the durations of the tasks to take into account kernel overheads can be very pessimistic as it always considers a worst case maximum number of preemptions for a task.

We propose in this paper to study the sources of kernel overheads for the preemptive FP/FIFO scheduling in an event driven kernel. We show how to integrate the overheads of the kernel in the classical theoretical feasibility conditions. Based on those feasibility conditions, it is possible to design and prove an OSEK real-time system before implementation. Thus, providing real-time dimensioning.

Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing
0-7695-2561-X/06 $20.00 © 2006 IEEE

We consider an event driven implementation of OSEK. OSEK standard has been initiated in 1993 by several german companies like BMW, Bosch, Daimler-Benz, Opel, and Siemens. The objectives were to save money, with a standard OS and to increase the software compatibility between manufacturers by using standard interfaces for all processors and network protocols. The OSEK operating system offers the necessary functionality to support event driven scheduling. Yet, the current approach used for system dimensioning leads to overestimate the overhead of the operating system, without a precise analysis of the operating system leading to a pessimistic dimensioning. e.g. developers generally limit the CPU of the tasks to allocate the rest of the CPU to the operating system without a good characterization of the OS. In this paper, we propose to characterize the overheads of an OSEK kernel to propose a deterministic system dimensioning.

In section 2, we recall the principles of an OSEK kernel. We then describe in section 3 the environment used and the sources of kernel overheads. We identify different sources of overheads. First, the time granularity chosen in the periodic timer used for the periodic model implementation. A wrong choice of granularity may lead to a smaller or higher period used by the kernel w.r.t a periodic task, that must be taken into account in the feasibility conditions. Second, we focus on the activation/termination and context switch overheads. In section 4, we show how to integrate the identified kernel overheads in the theoretical FC of FP/FIFO. In section 5, we propose to compare the theoretical worst case response times with kernel overheads to the experimental results obtained with a real OSEK implementation showing that our analysis is relevant for system dimensioning. Finally, we conclude.

2 OSEK characteristics

In subsection 2.1, task management in OSEK is exposed. The scheduling policy of OSEK is detailed in subsection 2.2. Then, the alarm mechanism, used to implement the periodic task model is described in subsection 2.3.

2.1 Task management

Two different task concepts are provided by the OSEK operating system: basic tasks, and extended tasks. Extended tasks are distinguished from basic tasks by being allowed to wait for events for communications between tasks and ressources management. The OSEK operating system is responsible for saving and restoring task context in conjunction with task state transitions whenever necessary. We are interested in this paper in the overheads due to the switching task mechanism and to the alarms treatment used to implement the periodic task model. We therefore focus on basic tasks which have three possible states (see also figure 1):

- **Running:** In the running state, the CPU is assigned to the running task, so that its instructions can be executed. Only one task can be in this state at any time, while all the other states can be adopted simultaneously by several tasks.
- **Ready:** All functional prerequisites for a transition into the running state exist, and the task only waits for election of the processor.
- **Suspended:** In the suspended state, the task is passive and can be activated.

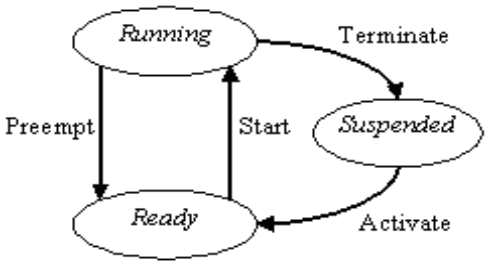


Figure 1. Basic task state model

We now describe the transitions between the states exposed in the figure 1:

Transition	Former state	New state	Description
Activate	Suspended	Ready	A new task is set into the ready state by the service ActivateTask.
Start	Ready	Running	A ready task selected by the scheduler is executed.
Preempt	Running	Ready	The scheduler decides to start another task. The running task is put in the ready state.
Terminate	Running	Suspended	The running task completes and self-suspends by the service TerminateTask.

Table 2. States and status transitions for basic tasks

In the OSEK operating system, a task can complete by calling the service TerminateTask. The OSEK

operating system also provides the service ChainTask to ensure that a prescribed task activation is performed just after the completion of the running task. This service is not used for periodic tasks and therefore will no further be considered. Hence, in this paper, ending the task without a call to TerminateTask is strictly forbidden. Task activation is performed using the operating system service ActivateTask. After activation the task is ready to execute. The following figure illustrates the interactions between two tasks suspended at time 0 and the evolution of their states with time. Task τ_1 is set to the running state and later preempted by a task τ_2, of higher priority.

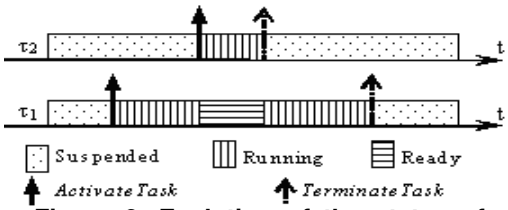


Figure 2. Evolution of the states of two basic tasks

2.2 Scheduling policy

In the OSEK operating system, there are three different scheduling policies: full preemptive, non preemptive, and mixed preemptive. In the later case, a system is composed of both preemptive and non-preemptive tasks. In this paper, we consider Full preemptive scheduling as it maximizes the kernel overheads we want to study. Full preemptive scheduling means that a running task may be put into the ready state, as soon as a higher priority task gets ready. The preempted task context is saved so that it can be resumed at the situation where it was preempted.

2.3 Alarm mechanism

The alarm mechanism allows implementing the periodic task model. Each alarm has two parameters: the time where it starts for the first time, and its period. This mechanism uses the OSEK time base to count the time which is different from the CPU time.

The OSEK time granularity has a duration T_{tick}, multiple of the internal clock cycle of the processor. The use of a timer enables the processor to create a periodic interrupt. The CPU load due to this interrupt is discussed in subsection 3.2.

3 Kernel overheads

Because several OSEK versions exist we have to describe our development environment. Our OSEK operating system is based on the OSEK-OS-specification version 2.2 provided by Vector Corp [2]. Our target device is a dsPIC30F6014 which is provided by Microchip and excited by a quartz at 7,3728 MHz. The integrated Phase Lock Loop multiplies this frequency by 16. According to the structure of dsPIC, each instruction requires 4 quartz oscillations. Thus, the internal cycle is equal to $\frac{4}{16\times7372800} \approx 33.91ns$. Different measurements have to be made in order to determine the kernel overheads of OSEK and to validate our tests and experimentations. In subsection 3.1, we explain our measurement methods. Then, we describe the OSEK's overheads in subsection 3.2. Overheads measurements are shown in subsection 3.3.

3.1 Measurements methods

We have done two kinds of measurements. We first determine the influence of the Tick Time on both the worst case response times and the real values of the periods chosen by the kernel. Then we study the influence of the kernel on the worst case response times of the tasks.

The worst case response times of the tasks depends on the WCET of the tasks.Each task is only composed of a simple empty loop which corresponds to "$for(i = 0; i < EndLoop; i++)$;". To determine the WCET, we use a standard simulation tool MPLab, which is also provided by Microchip Corp., depending on the value of "*EndLoop*" constant. Thus, no uncertainty is introduced in the execution times of the tasks (we only want to measure the kernel overheads, not the WCET uncertainty).

We have developed a software which automatically measures several the worst case response time of several activations for each task in the worst case scenario corresponding to lemma 2 . This software uses a 16-bit timer to make his measurements which are stored in RAM. Once all measurements are made, these ones are transmitted via a serial port at 115200 bauds. Thus, the transmission does not influence the obtained results.

3.2 OSEK's overheads

Like any operating system, OSEK needs to generate its own time base, called the Tick Time, based on an interrupt having a period T_{tick}. The Tick Time

is the only interrupt source in this study. As described in subsection 2.3, this Tick Time is used to manage the alarms in charge of implementing the periodic task model. Under certain conditions, this management can add a CPU load which cannot be neglected. In paragraph 3.2.1, an experiment is done to show how this Tick Time influences the response time of a single task. In addition, because the switching task mechanism creates another CPU load, it can also affect the response times of the tasks when it is too frequent. In paragraph 3.2.2, an experiment is done to show how this mechanism can also influence the response time of a task.

3.2.1 Tick Time

To illustrate the Tick Time influence, we consider the following example where a single task is run by the system. This task has a period equal to $100ms$ and a duration equal to $50ms$. We present its response time (r) for a given Tick Time period T_{tick}:

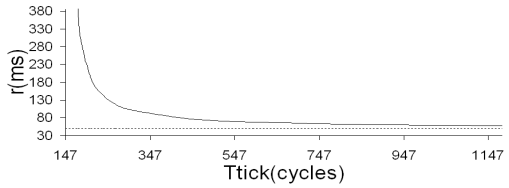


Figure 3. Comparison between the WCET (dotted curve) to the measured response time (continuous curve) of the task

The overhead of the OSEK operating system increases when T_{tick} decreases. Consequently, the task response time increases in the same way. As we can see on figure 3, the response time is multiplied by at least 2 or more when the Tick Time is below 295 cycles. The response time is strongly increased when the Tick Time is equal to 147 cycles.

We now, examine the maximum absolute error obtained on the considered periodic task for a given value of T_{tick}. As explained in subsection 2.3, the periodic model is based on an alarm mechanism which depends on the Tick Time. That is why, the period is more precise when Tick Time is multiple of it. The actual period, denoted T_i^* for a task τ_i chosen by OSEK kernel is as follows:

$$T_i^* = \left(1 + \lfloor \frac{T_i - T_{tick}/2}{T_{tick}} \rfloor\right) T_{tick}$$

We can notice that its error never exceeds the duration $T_{tick}/2$. Our OSEK implementation uses a 16-bit timer to generate the Tick Time. Hence T_{tick} cannot exceed $(65535 \times 33,91ns)\ 2222\mu s$. Finally, in this interval, the Tick Time should be the greatest value multiple of the greatest common divisor between all periods of the tasks to cancel the imprecision $T_{tick}/2$.

3.2.2 Switching task mechanism

Now, we consider a preemptable task of WCET $50ms$. This task is interrupted by a higher priority task which is empty. The Tick Time is constant and equal to 2949 cycles. Consequently, the Tick Time constantly increases the response time of $3,5ms$. Thus, when the period of the higher priority task decreases, we observe the deviation which is only due to the switching task mechanism on figure 4.

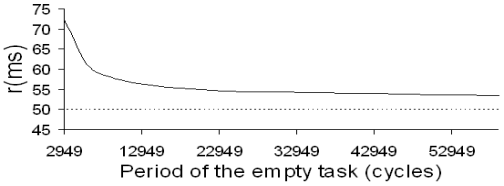


Figure 4. WCET (dotted curve) vs measured response time (continuous curve) of the task

We can see that the difference between theoretical and real durations increases when the period of the higher priority task decreases. In other words, switching task mechanism is non-negligible. We show how to take it into account in section 4.

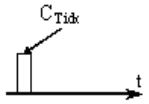
3.3 Overheads measurements

Now that the overheads are identified, the objective is to measure them in order to integrate them into the feasibility conditions. In subsection 3.3.1, we begin with an illustration of the measured durations. Then, durations of the overheads are given in subsection 3.3.2.

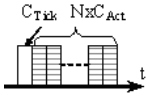
3.3.1 Illustration of the measured events

In this subsection, we determine the durations of the overheads previously exposed. Each time a Tick Time occurs, there are three possible scenarios:

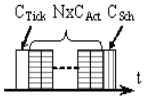
The Tick Time only manages the alarms but no task must be activated



The Tick Time manages the alarms and N tasks must be activated



The Tick Time manages the alarms, N tasks must be activated and one of them must be set to the running state (potentially stopping the execution of a running task)



The two last scenarios illustrate the cases where alarms occur and activate their associated task.

3.3.2 Measurements

The table 3 gives the notations used for each overheads:

Symbol	Description
C_{tick}	The execution time of the alarms management that occurs every T_{tick}.
C_{act}	The execution time required to activate a task. The state of an activated task is set to ready.
C_{sched}	The execution time required to schedule the task (if any) that has just been activated and that has the highest priority among all the tasks in the ready state.
C_{term}	The execution time to terminate the task and reschedule.

Table 3. Notations of the measurements

The following table 4 gives the results of our measurements obtained for 5 tasks (the number of tasks considered in section 5) with the software exposed in section 3.1. These values are the worst case values obtained with our tools.

Symbol	WCET (cycles)
C_{tick}	180
C_{act}	570
C_{sched}	420
C_{term}	450

Table 4. Execution times of kernel overheads

Notice that the previous execution times overheads may depend on the number of tasks. We consider the worst case execution times obtained for $n = 5$ tasks in our experiments (see section 5.

4 Real-time analysis with kernel overheads

We now recall classical results in the uniprocessor context for real-time scheduling. We recall that we consider preemptive scheduling in this paper.

- A task is said to be non-concrete if its first release time is not known a priori. In this paper, we only consider non concrete first request times, as the first activation request times are supposed to be unpredictable.

For any task τ_i,

- $hp(i)$ denotes the set of tasks having a strictly higher priority than τ_i.
- $lp(i)$ denotes the set of tasks having a strictly lower priority than τ_i but τ_i.
- $sp(i)$ denotes the tasks having the same priority as τ_i.
- Time is assumed to be discrete (task arrivals occur and task executions begin and terminate at clock cycles: the parameters used are expressed as multiples of the clock cycles). In [1] it is shown that there is no loss of generality with respect to feasibility conditions restricting the schedules to be discrete, once the task parameters are assumed to be integers (multiples of the clock cycles) i.e. a discrete schedule exists, if and only if a continuous schedule exists.
- Given a non-concrete task set, the **synchronous busy period** is defined as the time interval $[0, L)$ delimited by two distinct idle times in the schedule of the corresponding synchronous release pattern [1]. The value of L does not depend on the scheduling algorithm but only on the task arrival pattern. L is called the length of the synchronous busy period.
- The value of the synchronous busy period with no kernel overheads, denoted L_{no} can be computed using the following recursive equation [3]:

$$\begin{cases} L_{no}^{m+1} = W(L_{no}^{m}) \\ L_{no}^{0} = \sum_{i=1}^{n} C_i \end{cases}$$

  The recursion ends when $L_{no}^{m+1} = L_{no}^{m} = L_{no}$, where for any time t, $W(t) = \sum_{i=1}^{n} \lceil \frac{t}{T_i} \rceil C_i$.

Let τ_i be a task requested at time t in a busy period (the processor in fully busy). Let time 0 be the beginning of the busy period. For such a task we have the following notations:

- An idle time of level $B_{i,t}$, is defined as a time t', such that there are no tasks in $sp(i) \cup \tau_i$ released at a time smaller than or equal to $t \leq t'$ pending at time t'.
- A level $B_{i,t}$ busy period is defined as a time interval $[a, b)$, such that there is no idle time of level $B_{i,t}$, in $[a, b)$ and such that both a and b are idle times of level $B_{i,t}$.

- The worst case level $B_{i,t}$ busy period [7] is the first busy period resulting from the scenario where all tasks $\tau_j \in hp(i) \cup sp(i) \cup lp(i)$ are first requested at time 0 and are then periodic and where task τ_i is periodic from $t_{i,0}$ to t, where $t_{i,0} \in [0, T_i - 1)$.

Notice that this definition of the worst case level $B_{i,t}$ busy period is slightly different from the one proposed by [5] where only tasks in $hp(i) \cup sp(i) \cup \tau_i$ where considered. With kernel overhead, we show in theorem 1 that a task in $lp(i)$ can also influence the worst case response time of a tasks τ_i due to its activation requests that must be taken into account by the kernel. Furthermore, with FIFO scheduling, we have to consider for tasks τ_i all the first request times of task τ_i at time $t_{i,0} \in [0, T_i - 1)$ [7].

- $U_{no} = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is the processor utilization factor, i.e., the fraction of processor time spent in the execution of the task set [6] without kernel overheads. An obvious necessary condition for the feasibility of any task set is $U_{no} \leq 1$ (this is assumed in the sequel).

Lemma 1 *[4], [7], The worst-case response time r_i of a non-concrete periodic task τ_i scheduled FP/FIFO with no kernel overhead is found in the first worst case level $B_{i,t}$ busy period and r_i is the solution of $r_i = \max_{t \in S}(r_{i,t})$, where:*

$$r_{i,t} = w_{i,t} - t$$
$$S = \cup_{\tau_j \in sp(i) \cup \tau_i} \{k \times T_j\} \cap [0, L_{no}[$$
$$w_{i,t} = \sum_{\tau_j \in sp(i) \cup \tau_i} (1 + \lfloor \frac{t}{T_j} \rfloor) C_j + \sum_{j \in hp(i)} \lceil \frac{w_{i,t}}{T_j} \rceil C_j$$

Proof: The influence of the task in $hp(i)$ is the same as the one proposed in [4]. The difference between the equation of r_i proposed in [4] is the term $\sum_{\tau_j \in sp(i) \cup \tau_i} (1 + \lfloor \frac{t}{T_j} \rfloor) C_j$ that takes into account FIFO scheduling for all tasks having the same priority as τ_i. With FIFO, a task requested after time t cannot be executed before τ_i requested at time t. We therefore have to consider only tasks in $[0,t]$. In [7], it is shown that only times t corresponding to the request times of the tasks in $sp(i) \cup \tau_i$ in the synchronous busy period must be considered. ■

Lemma 2 *The worst case response time of a periodic task with kernel overheads is found in the worst case level $B_{i,t}$ busy period.*

Proof: For a task τ_i, the kernel overheads are maximized when the number of activations of tasks in $hp(i)$ and $lp(i)$ are maximized. Leading to the same worst case scenario for tasks in $hp(i)$ as in [4]. Notice that we also have to consider the overheads of the tasks activations (achieved by the activateTask) in $lp(i)$ whose number is maximized when tasks in $lp(i)$ are released as described in the worst case level $B_{i,t}$ busy period. ■

We now extend lemma 1 to take into account the OSEK kernel overheads. The first overhead described in subsection 3.2.1 is due to the tick time value leading for a tasks τ_i to consider the modified period T_i^*. This overhead is considered in the following theorem.

Theorem 1 *The worst case response time r_i of a periodic task τ_i scheduled FP/FIFO with the OSEK kernel overheads is the solution of the following equation: $r_i = \max_{t \in S}(r_{i,t})$, where $r_{i,t} = w_{i,t} - t$, $S = \cup_{t_{i,0}=0}^{T_i - 1} S(t_{i,0})$ such that $S(t_{i,0}) = t_{i,0} + k.T_i, k \in N, k \leq K$, where K is the smallest value such that $w_{i,t_{i,0}+K.T_i} \leq (t_{i,0} + (K+1).T_i)$, and*

$$w_{i,t} = \sum_{\tau_j \in sp(i) \cup \tau_i} (1 + \lfloor \frac{t}{T_j^*} \rfloor)(C_j + C_{term}) + \sum_{\tau_j \in hp(i)} \lceil \frac{w_{i,t}}{T_j^*} \rceil (C_{act} + C_j + C_{term}) + \sum_{\tau_j \in lp(i) \cup sp(i) \cup \tau_i} \lceil \frac{w_{i,t}}{T_j^*} \rceil C_{act} + \max_{\tau_j \in \tau_i \cup hp(i)} \lceil \frac{w_{i,t}}{T_j^*} \rceil C_{sched} + \lceil \frac{w_{i,t}}{T_{tick}} \rceil C_{tick}$$

Proof: We consider a task τ_i released at time t in its worst case worst case level $B_{i,t}$ busy period. Let $w_{i,t}$ be the completion time of τ_i. $w_{i,t}$ is composed of five parts:
The first part is equal to the workload to execute and terminate tasks in $sp(i) \cup \tau_i$ limited to the interval $[0,t]$ due to FIFO scheduling: $\sum_{\tau_j \in sp(i) \cup \tau_i} (1 + \lfloor \frac{t}{T_j^*} \rfloor)(C_j + C_{term})$
For any request of task in $hp(i)$, the kernel must activate, run and terminate the tasks. The activation is taken into account in the second part of the equation. Leading to a workload : $\sum_{\tau_j \in hp(i)} \lceil \frac{w_{i,t}}{T_j^*} \rceil (C_{act} + C_j + C_{term})$
For any task in $lp(i) \cup sp(i) \cup \tau_i$, the scheduler must at least activate the tasks according to their request times and put it in the ready state. Leading to the third part: $\sum_{\tau_j \in lp(i) \cup sp(i) \cup \tau_i} \lceil \frac{w_{i,t}}{T_j^*} \rceil C_{act}$
Considering the last part of the equation. The scheduler is called to save/restore the context of task τ_i every time a task with a priority higher than or equal to τ_i (including τ_i as τ_i can be the task with the highest priority) is run. The maximum number of scheduler calls is bounded by: $\max_{\tau_j \in \tau_i \cup hp(i)} \lceil \frac{w_{i,t}}{T_j^*} \rceil C_{sched}$
We must also take into account the alarm overhead. By assumption, all the alarms are managed by a pe-

riodic timer of period T_{tick} of duration C_{tick}. Leading to an overhead equals to $\lceil \frac{w_{i,t}}{T_{tick}} \rceil C_{tick}$.
The set S corresponds to all the request times of τ_i that must be checked. The set $S(t_{i,0}) = t_{i,0} + k.T_i, k \in N, k \leq K$ corresponds to a scenario where the first request of task τ_i occurs at time $t_{i,0}$. The value of K corresponds to the activation request of τ_i at time $t_{i,0} + K.T_i$ such that $w_{i,t_{i,0}+K.T_i} \leq t_{i,0} + (K+1).T_i$. $w_{i,t_{i,0}+K.T_i}$ is the end of the level $B_{i,t_{i,0}+K.T_i}$ busy period and the task released at time $t_{i,0} + (K+1).T_i$ is released in a new busy period that does need to be checked (from lemma 2). ■

Remarks:

- The set S is an extension of [8] where, in the case of a task set with tasks having different priorities, we only have to check the set $S(0)$. From the definition of a level $B_{i,t}$ busy period and because of FIFO scheduling, we must check more request times.
- If tasks are scheduled according to Rate Monotonic [4] then, the value $\max_{\tau_j \in \tau_i \cup hp(i)} \lceil \frac{w_{i,t}}{T_j^*} \rceil C_{sched}$ is maximum for the task in $hp(i) \cup \tau_i$ having the smallest period, the one chosen by Rate Monotonic. Hence, theorem 1 leads in this case to the exact worst case response times of the tasks.

Theorem 2 *A sufficient feasibility condition for the scheduling of periodic tasks scheduled with preemptive FP with the OSEK kernel overheads is (where T_α is the period with the smallest value in task set):*

$$\forall \tau_i \in \tau, r_i \leq D_i \quad (1)$$

$$U_{no} + \sum_{j=1}^{n} \left(\frac{C_{act} + C_{term}}{T_j^*} + \frac{C_{sched}}{T_\alpha} + \frac{C_{tick}}{T_{tick}}\right) \leq 1 \quad (2)$$

Proof: Equation (1) is straightforward. r_i is determined from theorem 1. Equation (2) is clearly necessary as the kernel overheads add a duration $C_{act} + C_{term}$ to every tasks and the scheduler is called at least once for every activations of the task with the minimum period leading to a processor utilization $\frac{C_{sched}}{T_\alpha}$. ■

5 Experimentations

In this section, we experiment the previous theoretical results on two task sets. Each of these task sets is composed of five preemptive periodic tasks with arbitrary deadlines. These task sets are described in the following tables. The first task set has been created with a T_{tick} equal to 9997 cycles in order to be more realistic :

Task	C_i (cycles)	D_i (cycles)	T_i (cycles)	P_i
τ_5	299991	49985	69979	2
τ_4	699790	15995200	15995200	1
τ_3	899730	19994000	15995200	1
τ_2	4998500	29991000	31990400	1
τ_1	9997000	63980800	63980800	0

Table 5. Task Set 1 ($T_{tick} = 9997 cycles$)

The second task set has been created with a T_{tick} parameter equal to 796 cycles in order to increase the CPU load due to the Tick Time. This CPU load is, in worst case, equal to $C_{tick}/T_{tick} = 0,23$. In the next task set, only fifo tasks τ_2, τ_3, and τ_4 verify $D_i > T_i$:

Task	C_i (cycles)	D_i (cycles)	T_i (cycles)	P_i
τ_5	15920	31840	318400	2
τ_4	71640	1273600	318400	1
τ_3	79600	2547200	636800	1
τ_2	398000	5094400	1910400	1
τ_1	796000	7641600	7641600	0

Table 6. Task Set 2 ($T_{tick} = 796 cycles$)

We now compare in each task set for any task τ_i the theoretical response time without overhead r_i^0, the theoretical response time with kernel overheads r_i^1 and the measured response time r_i^2 in a real OSEK system.

Task	r_i^0 (cycles)	r_i^1 (cycles)	r_i^2 (cycles)
τ_5	29991	34431	31092
τ_4	11546535	12420108	12201421
τ_3	11546535	12420108	12201384
τ_2	11546535	12420108	12220185
τ_1	31840445	46573406	45871157

Table 7. Comparison between the different response times for task set 1

In the task set 1, we observe that each task is activated once at any time at the maximum.

Task	r_i^0 (cycles)	r_i^1 (cycles)	r_i^2 (cycles)
τ_5	15920	25400	22860
τ_4	581080	783960	753191
τ_3	581080	783960	753498
τ_2	581080	783960	753596
τ_1	2778040	5608300	5088369

Table 8. Comparison between the different response times for task set 2

In the task set 2, the tasks τ_3, and τ_4 may be reactivated during their execution.
Now, we determine two significant ratios in tables 9 to 10 for each task set and for each task to characterize the performance of our theoretical worst case response time with kernel overheads.

Deviation	τ_5	τ_4	τ_3	τ_2	τ_1
$\frac{1-r_i^2/r_i^1}{100}$	9.70	1.76	1.76	1.61	1.51
$\frac{1-r_i^0/r_i^1}{100}$	12.90	7.03	7.03	7.03	31.63

Table 9. Performance of theoretical worst case response time with kernel overheads for task set 1

Deviation	τ_5	τ_4	τ_3	τ_2	τ_1
$\frac{1-r_i^2/r_i^1}{100}$	10.00	3.92	3.89	3.87	9.27
$\frac{1-r_i^0/r_i^1}{100}$	37.32	25.88	25.88	25.88	50.47

Table 10. Performance of theoretical worst case response time with kernel overheads for task set 2

In tables 9 to 10, the first line provides the percentage of deviation between the theoretical response time with kernel overheads and the measured response time obtained with our OSEK system. The deviation is always important for the task τ_5 as it has the smallest execution time. The overheads are accordingly more important.

The deviations obtained for the tasks τ_2, τ_3, and τ_4 scheduled FIFO on their priority level decreases as the T_{tick} parameter increases. Because they depend on each other, those tasks are greatly influenced by the kernel overheads. Because task τ_1 has the lowest priority, it is often preempted by higher-priority tasks. In others words, the task τ_1 is influenced by the overheads of all higher priority tasks. That is why its deviation is always important. In all cases, the deviations are small and enable to use the theoretical approach for a real-time dimensioning. In tables 9 to 10, the second line shows the deviation between the theoretical approach with and without kernel overheads showing that the deviation ranges from 7,03% to 50,47%. Hence, the kernel overheads cannot be neglected and influences significantly the worst case response times of the tasks.

6 Conclusion

In this paper we have studied the impact of kernel overheads in the theoretical feasibility conditions of preemptive FP/FIFO scheduling of periodic tasks. We have considered an event driven OSEK system proposed by Vector Corp. We have identified the sources of kernel overheads and have shown how to integrate them in the worst case response times of the tasks, used by the feasibility conditions. We have shown in our experiments that the overestimation of the theoretical worst case response times does not exceed 10,00% of the real worst case response times and that the feasibility condition without kernel overhead are not valid.

References

[1] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, Vol. 2, pp. 301-324, 1990.

[2] V. Corp. Osek/vdx operating system v. 2.2.3 specification.

[3] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive scheduling real-time uniprocessor scheduling. *INRIA Research Report*, No. 2966, September 1996.

[4] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Comp. Jour.*, 29(5), pp. 390-395,, 1986.

[5] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings 11th IEEE Real-Time Systems Symposium*, pp 201-209, Dec. Lake Buena Vista, FL, USA, 1990.

[6] L. C. Liu and W. Layland. Scheduling algorithms for multi-programming in a hard real time environment. *Journal of ACM*, Vol. 20, No 1, pp. 46-61, January 1973.

[7] S. Martin, P. Minet, and L. George. Improving fixed priority schedulability with dynamic priority as secondary criterion. *Journal of Embedded Computing*, num 4, 2006.

[8] K. Tindell, A. Burns, and A. J. Wellings. An extendible Approach For Analysing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems 6(2)v*, 1994.

Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing
0-7695-2561-X/06 $20.00 © 2006 IEEE