

SCHEDULABILITY ANALYSIS OF REAL-TIME SYSTEMS UNDER DUAL-PRIORITY SCHEDULING

C. K. ANGELOV I. E. IVANOV I. J. HARATCHEREV

*Advanced Control Systems Laboratory, Technical University of Sofia,
1756 Sofia, Darvenica. E-mail: acsl@vmei.acad.bg*

Abstract: This paper presents schedulability analysis which has been developed in order to precisely estimate task response times, taking into account kernel execution effects. The analysis takes into consideration specific features of the *HARTEX* hard real-time kernel, and in particular: asynchronous event-driven operation; integrated scheduling of hard and soft real-time tasks (as well as tasks and resources) and advanced task management, whereby kernel operations are executed in constant time, independent of the number of tasks involved. The developed test has been incorporated into a system specification and analysis tool supporting *HARTEX*-based real-time applications.

Keywords: hard real-time systems, dual-priority scheduling, synchronisation protocols, completion time test

1. INTRODUCTION

Analytical verification of temporal correctness is a major issue, which has been addressed by modern Fixed-Priority Scheduling Theory. This problem can be solved by determining task response time estimated from the moment of task arrival: a task is schedulable iff $R_i \leq D_i$, where R_i and D_i are the task response time and deadline.

The above formulation gives both necessary and sufficient conditions for task and system schedulability. On the other hand, it is relatively easy to implement using a numerical procedure known as the *completion time test* [4, 8]. The latter has been originally derived under simplifying assumptions in the context of single computer real-time systems and periodic task workloads.

The use of this test for practical purposes requires further development of the underlying method in order to take into account kernel operation and kernel execution effects. There have been several attempts to solve this problem. Katcher *et al.* have developed schedulability analysis in the context of both synchronous (tick-driven) and asynchronous (event-driven) kernels [7]. However, it has been criticized for being too pessimistic, i.e. estimated kernel overhead is significantly greater than experimentally measured overhead. A more accurate analysis has been developed by Burns *et al.* but it is applicable to systems operating under tick-driven kernels [5].

In both cases the analysis presumes a conventional kernel design wherein kernel queues are implemented as linked lists. Unfortunately, linked lists processing incurs a considerable and *largely varying* overhead, e.g. simultaneous release of multiple periodic tasks at superperiod boundaries, see e.g., experimental data given in [5]. A radical solution to this problem has been proposed, whereby linked lists have been substituted for Boolean vectors, resulting in constant execution time of kernel operations independent of the number of tasks involved [2]. This approach has been used to implement the *HARTEX* real-time kernel [1].

A previous paper [3] has presented schedulability analysis which can be used to evaluate real-time task behaviour in the context of the *HARTEX* kernel, taking into account specific features of that system, and in particular:

- Asynchronous event-driven operation
- Interruptible and preemptable execution of kernel routines
- Advanced task management techniques featuring Boolean vectors and parallel (bitwise) vector processing.

This paper extends the above analysis to take into account integrated (dual-priority) scheduling of hard and soft real-time tasks executed under the *HARTEX* kernel. The paper is structured as follows. The next section presents basic analysis of task schedulability developed within modern real-time scheduling theory, as well as augmented analysis capturing task behaviour under dual-priority scheduling. That analysis is elaborated in the subsequent section, taking into consideration *HARTEX* operation and kernel execution effects.

2. PROBLEM FORMULATION

2.1. Estimating task response times under fixed-priority scheduling

Task schedulability can be assessed via the so called *completion time test*, which is used to estimate task response times under a worst-case scenario with respect to higher-priority tasks interference [4, 8].

Task response time can be evaluated through the following expression:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{T_j} \right\rceil C_j, \quad (1)$$

where C_i is task execution time and B_i is task blocking time, which is bounded by synchronisation protocols such as the *Conditional Ceiling Priority* protocol used in *HARTEX* [1]. In that case $B_i = \max(tcr_1, tcr_2, \dots, tcr_k)$, where tcr_i , $i = 1 \div k$ denote the duration of critical regions of lower-priority tasks that can block task τ_i through the use of a shared resource. Specifically, a task can experience either direct blocking by lower priority tasks or indirect (push-through) blocking which is inherent to the above mentioned protocol, as well as blocking caused by non-preemptable sections of kernel code executed within lower-priority tasks.

The interference of higher priority tasks is modeled by the third term of (1), where $hp(i)$ is the set of higher priority tasks with execution times C_j , and each of them is executed k_j times,

$k_j = \left\lceil \frac{R_i}{T_j} \right\rceil$, during the time window $[0-R_i]$, thus preempting

the execution of task τ_i .

The following assumptions have been made when deriving equation (1):

- The task load consists of periodic or pseudoperiodic (sporadic) tasks with periods T_i , $i = 1, 2, 3, \dots$
- Tasks are scheduled for execution using fixed priority preemptive scheduling. Furthermore, task priorities are determined in accordance with execution dynamics using either rate-monotonic or in the general case - deadline monotonic scheduling, assuming that $D_i \leq T_i$.
- Higher-priority task interference is determined for a worst case corresponding to the so-called *critical instant* whereupon task τ_i is released simultaneously with all higher-priority tasks.

Equation (1) has no analytical solution but it can be used to calculate task response time through a recurrence relation:

$$R_i^{(n+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil \cdot C_j, \quad (2)$$

assuming that $R_i^{(0)} = C_i$. This relation is iteratively computed until $R_i^{(n+1)} = R_i^{(n)}$ or alternatively, until $R_i^{(n+1)} > D_i$. In the latter case the task is deemed unschedulable.

2.2. Estimating task response times under dual-priority scheduling

A similar approach can be used when determining task response times in the context of dual priority scheduling [6], which is also supported by *HARTEX*. In this case the following assumptions are made:

- An augmented fixed priority scheduling discipline is used where time-critical tasks have dual priorities, i.e. a lower-band and an upper band priority, whereas non time-critical tasks are assigned middle-band priorities. Time-critical tasks are initially released at their lower-band priorities and subsequently, they may be preempted by non-time critical (middle-band priority) tasks up until a priority promotion instant t_{pr} . The latter is defined by an offset Y_i measured from the arrival of task τ_i . At that instant task priority is promoted to its upper-band value and subsequently, the task can be preempted only by other time-critical tasks having higher promoted priorities.
- The worst case scenario in terms of higher-priority task interference is defined as follows:
 - a) The system is fully loaded resulting in a situation in which time-critical tasks are entirely executed at their upper-band priorities;
 - b) The priorities of task τ_i and all higher (upper-band) priority tasks are simultaneously promoted at $t = 0$ (i.e. tasks τ_j , $j = 1, 2, \dots, i$ are released at instants $-Y_j$).

The last two assumptions define the notion of “*critical instant*” in the context of dual priority scheduling. It has been shown that the completion time test remains valid under this scenario [6]. In that case task response time is given as $R_i = Y_i + w_i$ where w_i is task completion time after priority

promotion which is again evaluated using the completion time test (2). Accordingly, the offset Y_i must be chosen in such a way as to guarantee that the task deadline will not be violated.

It must be pointed out however, that unlike the original worst-case scenario, the above definition has no meaningful interpretation, i.e. this is an artificially derived situation that is unlikely to happen in a realistic environment. This scenario may cause some difficulties when trying to capture kernel behaviour. Moreover, it introduces a “hen & egg” problem arising from the fact that task response time is defined in terms of task offset and the latter is defined in terms of task response time. Indeed, it is postulated that a task is schedulable if $R_i \leq D_i$ and $R_i = Y_i + w_i$, where Y_i is the task priority promotion offset and w_i is task completion time for the worst case scenario defined above. On the other hand, $Y_i = D_i - R_{i(Y=0)} \Rightarrow Y_i = D_i - w_i$, since $R_{i(Y=0)} = w_i$ (according to the analysis developed in [6]). Substituting Y_i for its equivalent expression we arrive at the relation:

$$R_i = D_i - w_i + w_i \leq D_i \Rightarrow D_i \leq D_i,$$

which is an apparent tautology.

However, this is just a formal problem which is relevant to the definition of the task promotion offset and it does not violate the validity of the analysis developed in [6]. This problem can be overcome through modified analysis, as follows:

- Task response time $R_{i(Y=0)}$ is evaluated under a worst case scenario, i.e. a critical instant such that:
 - (1) Task τ_i and all higher upper-band priority tasks are released at $t = 0$;
 - (2) The priority of task τ_i and all higher upper-band periodic tasks are simultaneously promoted at $t = 0$;
 - (3) Hard sporadic tasks of higher priority are also released at $t = 0$.

Note that the above formulation is essentially equivalent to the original definition of the critical instant [4, 8]. On the other hand, it takes into account a *HARTEX* limitation, i.e. dual-priority scheduling is provided only for hard periodic tasks. Sporadic tasks are always released at upper-band priorities in order to reduce undesirable effects caused by task response/release jitter, in the context of distributed event-driven transactions [1].

- The task will be schedulable under the worst case scenario if $R_{i(Y=0)} \leq D_i$. In this case it is possible to evaluate task laxity L_i (if any), where $L_i = D_i - R_{i(Y=0)}$. Thus, it is possible to safely define the task promotion offset, i.e. $Y_i = L_i$.
- The above definition of task schedulability (and task promotion offset) guarantees that task τ_i will be schedulable when periodic tasks are executed with non-zero offsets. This can be easily shown:

Indeed, under that definition enough time has been reserved for the execution of τ_i even in the worst case when hard periodic task priorities are simultaneously promoted and even more, higher priority sporadic tasks are released at that moment too (which is highly unlikely). In case of non-zero offsets, the task τ_i will be executed for time w_i' measured from the moment of priority promotion, where $w_i' < R_{i(Y=0)}$ since higher-priority task interference for other task phasings is always *smaller* than the

interference defined for the critical instant assumed. Therefore, it can be argued that:

$$w_i' < R_{i(Y=0)} \Rightarrow R_i = Y_i + w_i' = D_i - R_{i(Y=0)} + w_i' = D_i - \varepsilon/\varepsilon > 0 \Rightarrow R_i < D_i \quad (3)$$

Hence, the task will always be schedulable .

It has been shown above that the completion time test (2) is applicable in the context of common fixed-priority scheduling as well as dual priority scheduling. However, the basic test is still not adequate for practical purposes because it does not take into account kernel execution effects. Recent research has demonstrated that the latter can greatly influence task execution and task schedulability analysis [5, 7]. Therefore, the basic completion time test (2) may be used only for a preliminary (rough) assessment of task schedulability.

The next section presents extended schedulability analysis taking into account *HARTEX* execution effects. It must be noted that the modified definition of the worst case scenario introduced above describes a realistic situation which facilitates reasoning about kernel behaviour, i.e. the way timing events are generated and the associated overhead (see below).

3. ESTIMATION OF TASK RESPONSE TIMES TAKING INTO ACCOUNT KERNEL EXECUTION EFFECTS

Application tasks are executed in an operational environment provided by the real-time kernel. Hard real-time tasks are usually activated by timing events in the context of periodic execution sequences. However, it is also possible to sporadically activate hard tasks upon the occurrence of external events. Soft real-time tasks are usually activated by external events.

Servicing a mixed load of periodic and aperiodic (sporadic) tasks is best carried out by an asynchronous event-driven kernel such as *HARTEX*, providing for minimal event-reaction times as well as the elimination of kernel-induced task release jitter. In such a system terminal interrupts (i.e. interrupts requesting task execution) invoke the kernel which subsequently activates the requested task if its priority is higher than current task priority.

In this context, effective task response time can be estimated as:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{T_j} \right\rceil \cdot C_j + I_{em} \quad (4)$$

Here, C_i is the effective task execution time and:

$$C_i = t_{preempt} + t_{init} + E_i + t_{exit}, \quad (5)$$

where $t_{preempt}$ is the execution time of the task manager *preempt_current_task* function; t_{init} is the time which is required to (optionally) initialize auxiliary system timers; E_i is task execution time, and t_{exit} is the execution time of the system primitive *task_exit* which is invoked at the end of task execution (for more details, see [1]).

In the above expression I_{em} is the event management overhead:

$$I_{em} = I_{tem} + I_{aem} \quad (6)$$

where I_{tem} is the timing events management overhead and I_{aem} is the external (aperiodic) events management overhead.

3.1. Timing events management overhead

Timing event management overhead is defined as follows:

$$I_{tem} = k_1 C_{clock} + k_2 C_{ptr} + k_3 C_{pro} \quad (7)$$

where the first term represents clock interrupt processing overhead, the second term - time manager overhead due to timing events releasing periodic tasks and the third term - time manager overhead due to timing events causing periodic task priority promotion, within the interval $[0-R_i]$. These overheads are determined as multiples of corresponding processing times where C_{clock} is the execution time of the basic clock interrupt processing routine, C_{ptr} is the execution time of the time manager when releasing one or more periodic tasks, and C_{pro} is the execution time of the time manager needed to promote the priority of one or more periodic tasks.

It must be emphasized that these execution times are *constant* values (unlike other kernels) as a result of sophisticated task management techniques featuring Boolean vectors and bitwise vector processing of kernel data structures. In particular, multiple tasks can be released simultaneously in constant time independent of the number of tasks involved (n), whereas in conventional kernels the complexity of that operation is $O(n^2)$ [7]. Likewise, priority promotion time is a constant value, independent of the number of tasks whose priorities are being promoted.

The above feature reduces considerably the complexity of the analysis and the estimation of kernel overhead becomes straightforward.

Coefficients k_1 , k_2 and k_3 are defined as follows.

- k_1 is the total number of clock interrupts in the interval $[0-R_i]$. If the real-time clock is implemented in software,

$$k_1 = \left\lceil \frac{R_i}{T_{clock}} \right\rceil \quad (8)$$

- k_2 is the total number of periodic task release events in the same interval. An upper bound of k_2 is given by the following expression:

$$k_2 = \sum_{j \in pt} \left\lceil \frac{R_j}{T_j} \right\rceil, \quad (9)$$

where pt is the set of periodic tasks.

However, the above formula gives a somewhat pessimistic estimate of the corresponding time manager overhead, because it does not take into account simultaneous release of multiple tasks at certain time instants within the interval $[0-R_i]$, which follows from the worst-case task interference scenario assumed.

- k_3 is the total number of timing events causing priority promotion for the corresponding periodic tasks in the interval $[0-R_i]$:

$$k_3 = \sum_{j \in pt} \left\lceil \frac{R_i}{T_j} \right\rceil, \quad (10)$$

by analogy with coefficient k_2 . Once again, this expression gives an upper bound, i.e. a pessimistic estimate of the number of priority promotion events, as it does not take into account possible coincidence of priority promotion instants for some periodic tasks.

However, the upper bounds given by expressions (9) and (10) are useful for deriving the corresponding upper bounds of task response times. Furthermore, upper bounds of task response times can be used to safely derive realistic, i.e. non-zero priority promotion offsets for periodic tasks.

3.2. External events management overhead

External events management overhead is modeled by the following expression:

$$I_{aem} = \sum_{j \in at} \left\lceil \frac{R_i}{T_j} \right\rceil C_{isr}^j, \quad (11)$$

i.e. it represents external (aperiodic) interrupt processing time during the interval $[0-R_i]$ where at is the set of aperiodic tasks; C_{isr}^j is the execution time of the j -th interrupt handler, which takes into account basic interrupt processing time as well as the time needed to execute the corresponding task invocation primitive:

$$C_{isr}^j = C_{int}^j + t_{sysop}^j, \quad (12)$$

and $t_{sysop}^j = t_{enqueue} + t_{intexit}$, assuming that aperiodic tasks are requested for execution by means of the *enqueue (task)* primitive and the interrupt handler is exited via the *interrupt_exit* primitive [1].

In the general case, basic interrupt processing time for the j -th aperiodic event can be estimated as:

$$C_{int}^j = \sum_{i=1}^{r_j} t_{int}^j, \quad (13)$$

assuming a sequence of r_j interrupts associated with that event, including r_j-1 non-terminal interrupts preceding the terminal interrupt which releases the corresponding aperiodic task. However, in most cases $r_j = 1$.

3.3. An augmented completion time test

A formula defining task completion time can be derived by substituting periodic and aperiodic event management overheads for their equivalent expressions in (6) and subsequently, in the basic expression (4). Unfortunately, the resulting equation has no analytical solution because the unknown quantity R_i is present in both the left-hand side and the right-hand side of the equation. However, this problem can be overcome through a recurrence relation as in the original completion time test:

$$R_i^{(n+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j + \left\lceil \frac{R_i^{(n)}}{T_{clock}} \right\rceil C_{clock} + (C_{ptr} + C_{pro}) \sum_{j \in pt} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil + \sum_{j \in ut} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_{isr}^j \quad (14)$$

The upper bounds of task response times can be used to determine priority promotion offsets for periodic tasks, as follows:

$$\forall task \in pt: R_i < D_i \Rightarrow Y_i = D_i - R_i \quad (15)$$

4. CONCLUSION

A computational test has been derived which can be used to estimate task response time and task schedulability in a realistic context, specified by the *HARTEX* execution environment. This test takes into account specific features of that kernel and in particular - event-driven operation, dual-priority scheduling and *constant* execution time of certain system functions, e.g. the release of periodic and sporadic tasks.

Constant duration of kernel operations will enhance system safety and predictability and it will facilitate real-time system development by reducing the computational and experimental complexity of schedulability analysis.

The presented completion time test has been incorporated into a system configuration and analysis tool - a *Windows NT* application supporting *HARTEX*-based real-time systems development.

Acknowledgement: This research has been partly supported by a Royal Society grant under the project "*Design Methodology for Hard Real-Time Distributed Control Systems*".

REFERENCES

1. Angelov C. K. and I. E. Ivanov (1999). *HARTEX* - a real-time kernel for distributed computer control systems. *Internal Report ACSL-1/2000*. Advanced Control Systems Laboratory, Faculty of Automatics, Technical University of Sofia.
2. Angelov C. K. and I. E. Ivanov (1999). High performance task management for hard real-time systems. *Proceedings of the Technical University of Sofia*, vol. 50-2, pp. 190-197.
3. Angelov C. K. and I. E. Ivanov (1999). Schedulability analysis of hard real-time systems. *Proceedings of the Technical University of Sofia*, vol. 50-2, pp. 198-205.
4. Burns A. (1994). Preemptive priority-based scheduling: an appropriate engineering approach, In: *Advances in Real-Time Systems* (S. H. Son (Ed.)), pp. 225 - 248, Prentice-Hall.
5. Burns A., K. Tindell and A. Wellings (1995). Effective analysis for engineering real-time fixed-priority schedulers. *IEEE Trans. on Soft. Eng.*, vol. 21, pp. 475-480
6. Davis R. and A. Wellings (1995). Dual-priority scheduling. *Proc. of the IEEE Real-Time Systems Symposium*, pp. 100-109.
7. Katcher D., H. Arakawa and J. Strosnider (1993). Engineering and analysis of fixed-priority schedulers. *IEEE Trans. on Soft. Eng.*, vol. 19, pp. 920-934.
8. Klein M. H., T. Ralya, B. Pollak, R. Obenza and M. G. Harbour (1993). A practitioner's handbook for real-time analysis: guide to rate-monotonic analysis for real-time systems. Kluwer Academic Publishers, Boston.