# Deadline spanning: A graph based approach

Stefan M. Petters
Embedded, Real-Time, and Operating Systems Program
National ICT Australia Ltd.*
School for Computer Science and Engineering
University of New South Wales
Sydney, Australia
smp@nicta.com.au

## Abstract

*Microkernel based systems tend to depend heavily on IPC. This paper adresses the problem of a system response spanning more than one task in an embedded real-time system. The approach is based on a mix of classical response time analysis equations and a graph based approach to estimate the impact of different parts of the system on the time needed by the system to respond. This approach has the major advantage of being intuitive.*

## 1 Introduction

In real-time system the response to a trigger is considered to be associated with a deadline, by which the response has to be completed for a correct operation of the embedded system. Such a deadline usually spans a set of processes. We will uniformly use the term task for these processes, without excluding threads. Splitting the deadline for an activity into deadlines for individual task increases the constraints in the schedulability analysis and hence potentially increases pessimism of the analysis. In this paper we consider this in the context of a microkernel-based environment and more specificly the Pistachio [6] implementation of the L4 API.

The L4 kernel based on the initial work of Liedtke and others [4] exhibits excellent performance while retaining all the main benefits associated with a microkernel. These are the small size, taylorability to specific needs, analysability in terms of functional and temporal behaviour [2], and minimising the scope of deliberate attacks on or critical errors within the system. Because of their structure, systems developed for a microkernel architecture are usually heavily inter-process communication (IPC) dependent, adding to the observation above of deadlines spanning multiple tasks.

This raises the issue of what the critical instant is. Due to the IPC dependent structure of the system, the critical instant is different to the classical interpretation of the release of all tasks at the same time. The response time of an individual task becomes of at least secondary relevance. The prime concern is the response time of the activity, which is only loosly related to that of a single task of the activity. The activity can be seen as a logical thread of control through the system. The temporal dependencies between tasks also provide a means to reduce the impact of blocking to be considered when looking at response times under the assumption that critical sections are encapsulated in server tasks.

*National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

## 2 Related Work

The closest work to what we present in this paper is that of Tindell [8]. The major drawback of Tindell's approach is the individual analysis of all tasks, which results in bad scalability of the analysis. Furthermore the assumption of fixed offsets used to compute worst case response times becomes impractical in the the context of a probabilitic approach.

Kolloch has shown a graph based approach in [5]. In his work he mapped a specification in SDL to a communicating task system. The messages in this system would transport the end-to-end deadline of an activity. These transported deadlines were then used to drive the earliest-deadline-first scheduler. Despite its advantage of having a uniform priority for all tasks within one system response, the implementation of the EDF scheduler adds significant overhead. A two level approach was described by Feng and Liu in [3]. There tasks are divided into mandatory and non-mandatory tasks. The non-mandatory tasks are used to improve on the results of the mandatory tasks. In that respect no hard real time requirements are placed onto the non-mandatory tasks.

## 3 Graph Based Response Time Analysis

### 3.1 L4 Pistachio Kernel

Pistachio is an high performance implementation of the L4 microkernel API. It slightly deviates from microkernel idea of providing mechanisms, but not policies. Pistachio implements a flexible scheduling policy, which is partially mirrored and implemented in the IPC path. The baseline is a priority-based round-robin mechanism. The scheduler would pick the tasks with the highest priority and execute those for time slices based on round robin. However, overlaid is a mechanism to work effectively in the presence of heavy IPC, which is typical for microkernel-based systems. Communication between tasks is considered to be synchronous only. This has been chosen for Pistachio as it removes a lot of overhead in maintaining message queues. Multiple IPC requests

are sorted in FIFO order. Despite being considered inferior to a priority based ordering, we argue that the potential additional blocking time justifies the performance gain achieved by the simple ordering. A form of priority ceiling protocol is implemented by encapsulating critical sections inside a server task opposed to using a semaphore with the associated overhead of implementing the priority ceiling protocol in the kernel. The server task is assigned the same priority as the highest priority task calling this server task. A side effect of this implementation of priority ceiling is the fact that in the case of multiple tasks waiting to enter the critical sections lead to the server task serving all requests, before yielding to the task with the highest priority of those served.

For a real-time application the use of explicit time slice donation and semaphores has to be ruled out. However, since the atomic operation can be provided using server tasks, this seems to be a minor restriction.

### 3.2 Activities

Activities are in general described as system response to a system trigger and involves the execution of the ISR, a set of tasks, some of which might be optional or alternative (either one or another subset) and a set of operating system calls. If an interrupt service routine (ISR), a task, or an OS system call is part of more than one activity, we will use copies of the instances in our analysis model.

Any activity $A_j$ performed by the system is considered to happen with a minimum inter-arrival time $T_j$ and subject to a deadline $D_j$. Besides this, an activity $A_j$ is associated with an worst case execution time (WCET) $C_j$ and a base priority $p_j$. The base priority of an activity is defined as the lowest priority of any task of this activity. From an analysis point of view it makes little difference whether system calls or ISR are tasks with high priorities or OS services.

### 3.3 Taskmodel

Each task communication has to be distinguished between the following types. If the communication between two tasks is paired, i.e. one task sends an IPC to the other and receives a response, we consider this IPC sequence to be a call. A task may do multiple calls, some of which might be mutually exclusive. System calls are considered in similar fashion to calls to other tasks. Another case is a split in the control flow, which means a task sends an IPC to another task and thus produces fine-grained concurrency. By making multiple send operations, an arbitrary number of threads of control may be started. In the case of multiple sends these may be mutually exclusive. Several of these split control flows may be joined by a task waiting for input of those control flows. Finally some tasks will work as IPC sinks. Usually those are driving actuators interfacing with the embedding system. A task may be of more than one type; e.g. making calls and operating as IPC sink.

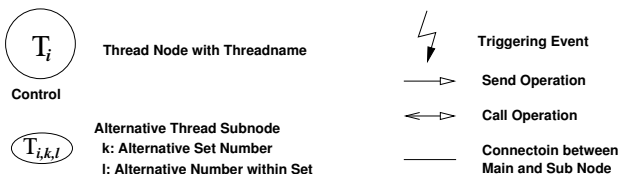


**Figure 1. Legend**

### 3.4 Task Precedence Graph

The work of Kolloch [5] was based on SDL, which needs to address the state of an SDL process. The notion of calls does not exist in this model. As a result we introduce our own task precedence graph (TPG) syntax in Figure 1.

As noted previously some communication from one task may be mutually exclusive. In order to avoid excessive overestimation of response times it is considered essential to take this into account. In this context the main task node contains the WCET of the shared bit of code, which might be in multiple sections (represented by a single value), calls and send operations, and links to alternative subnodes. These alternative subnodes are divided into groups containing mutually exclusive subnodes. In the case of such a subnode again containing mutually exclusive parts this is handled by expanding the subnodes. Joint subnodes are duplicated so each alternative subnode represents exactly one alternative without further subdivision. This generates more complexity than necessary, but the discussion of obtaining the WCET for activities in Section 3.5 becomes more clear. The subactivities formed by this are again considered to be of the lowest priority of any task involved in it.

A task, which has several receive only operations will be called joining task or more specifically of mutual exclusive joining task or constructive joining task. Mutual exclusive joining tasks happen, when a task may either receive one or the other input to the same receive call. Similar to server tasks these are duplicated. Constructive joining tasks need all input IPCs to operate and thus connect to all sending tasks. Additionally there could be an optional joining receive case. In this case alternatives alongside mutual exclusive send operations can be used. However, this is not considered to be a prime problem. An example TPG can be found in Figure 2 in Section 4.

### 3.5 Construction of the WCET of an Activity

Opposed to [5] the system is priority based and can not as straightforward clip trees from the task precedence graph. This is driven by the fact that the priorities are not necessarily as uniform as in the case of the message based EDF scheduling algorithm (cf. Section 2). It might be that an alternative subactivity being longer than other mutually exclusive subactivities, actually has a lower priority and thus may be subject to more preemption by other activities. We assume that each activity has a deadline which is less or equal to the minimum inter-arrival time of triggering events.

For all activities the following process has to be carried out. To distinguish it from its subactivities, we will call this *main activity*. A more detailed desription of this process may be found in [7], which is an extended version of this paper. For all the subactivities, the WCET is summed up. Each group of mutually exclusive subactivities, which

exhibits the same priority for all its subactivities, is reduced to the subactivity with the largest WCET. The groups which have been dealt like this, are considered mandatory part in computing the WCET of main activity. In most cases this will effectively remove any mutually exclusive subactivity groups, which might exist.

The remaining subactivity groups are reduced by removing any subactivity from that group, which exhibit either a priority higher then the subactivity having the largest WCET within the group or a WCET which is smaller than the WCET of the subactivity with the lowest priority. Next any mutually exclusive subactivity groups for which the priority of the subactivity group are the same or higher than the lowest priority of the remaining elements of the main activity are reduced. The reduction is again achieved by removing all subactivities, which do not have the largest WCET withing the group.

Of each group the subactivities with the minimal and maximal priority is chosen. The groups are compared whether the subactivity with the minimal priority is greater than the subactivity with maximal priority in the other. Groups in which all subactivities are higher priority than another group are also reduced according the WCET and moved as mandatory element to the main activity.

All remaining groups need to be completely explored; i.e. any subtask of one group needs to be considered within the context of the other groups. This leads to a list of tuples of WCET and lowest used priority for each activity. Further reduction would at this stage be possible, but due to practical irrelevance we will not discuss it in this paper.

### 3.6 Critical Instant

In this context the critical instant deserves special consideration. The base assumption is that the trigger events of all activities are independent, meaning that the activities may be started with any phasing. As the priorities of activities change with the control passed through the tasks it is worthwhile to ponder to what degree one activity may influence another. Only tasks with a proirity greater or equal to the base priority of the activity under consideration may prolong the response time of this activity. In the case of a single control flow within the influencing activity only one chain of tasks with a greater or equal priority will have to be accounted for. In the case of a split of the control flow an ordering of execution may be established and thus the control flow serialised in terms of influence analysis, unless the split subactivities have the same priority.

In the case of a split of the control flow, either a chain of tasks of the activitie's single control-flow portion may contribute, or one chain of tasks per split of the control flow. Further dependencies may be used, like fixed ordering of the execution of these tasks, but this is beyond the rigid mathematical representation. It has to be noted that the start and end tasks of an activity need to be considered together.

### 3.7 Response Time Analysis

For the response time of an activity, we need to consider the impact of the other activities on the activity under consideration. In a first step we discuss the case for an activity which has a single WCET and priority tuple describing it. As discussed in the previous section any task with a priority higher than the minimum priority of the activity under consideration could have an impact.

The priority of the activity under consideration $A_f$ will be weighed against the priorities of all other tasks. Any task or subtask with a lower priority than this will be removed from the appropriate activity diagrams for this analysis. Besides the trivial cases of no nodes or all nodes of an activity contributing to the response time of another, the case of the part activities, the WCET will be estimated using the algorithm described in Section 3.5 applied on all part activities of an activity and the largest of those will be denoted $C_l^*$. In the case of split control flow, either one part activity of the single control flow portion or one part activity per split control flow needs to be taken into account for $C_l^*$.

The impacting activities will be sorted into two sets in such a way that $\mathbf{H}(A_j)$ contains activities of same or higher priority than $A_j$ and $\mathbf{L}(A_j)$ containing those with lower priority. Similar to Tindell in [8] the response time will be computed iteratively.

$$R_j^n = C_j + \sum_{\forall A_h \epsilon \mathbf{H}(A_j)} \left\lceil \frac{T_h}{R_j^{n-1}} \right\rceil * C_h + \sum_{\forall A_g \epsilon \mathbf{L}(A_j)} C_g^* \quad (1)$$

The proof of this equation is similar to the proofs of most iteratively computed response time analysis approaches and due to space restrictions omitted from this paper. In the case of activities starting and ending with higher priorities and operating in lower ones during the main operation the equation needs to be slightly modified. This is the case for most activities, as they tend to be triggered by interrupts and potentially use a high priority driver to access actuators. In that respect we need to see whether this applies. Assuming we have $C_f^{*S}(p_j)$ and $C_f^{*E}(p_j)$ describing the start and end block WCET for an activity $A_f$ with same or higher priority than $p_j$. Should the start block start with a priority lower than $p_j$ than $C_f^{*S}(p_j) = 0$ and likewise for $C_f^{*E}(p_j)$. We need to sort the contents of $\mathbf{L}(A_j)$ into two sets describing the worst impact of the each activity in this set.

$$\phi = C_f^{*S}(p_j) + C_f^{*E}(p_j) \quad (2)$$

$$\mathbf{M}(A_j) = \left\{ A_f | \left( A_f \epsilon \mathbf{L}(A_j) \right) \wedge \left( \phi \geq C_f^* \right) \right\} \quad (3)$$

$$\mathbf{L}^*(A_j) = \left\{ A_f | \left( A_f \epsilon \mathbf{L}(A_j) \right) \wedge \left( \phi < C_f^* \right) \right\} \quad (4)$$

$$R_j^n = C_j + \sum_{\forall A_h \epsilon \mathbf{H}(A_j)} \left\lceil \frac{T_h}{R_j^{n-1}} \right\rceil * C_h + \sum_{\forall A_g \epsilon \mathbf{L}^*(A_j)} C_g^* + \sum_{\forall A_e \epsilon \mathbf{M}(A_j)} \left( C_f^{*S} + C_f^{*E} \right) \quad (5)$$

## 4 Exemplary Analysis

In this section we go through a small example. The example is derived from the Olympus Attitude and Orbital Control System (AOCS) case study described in [1]. The structure is adapted from the SDL based model of Kolloch in [5]. All

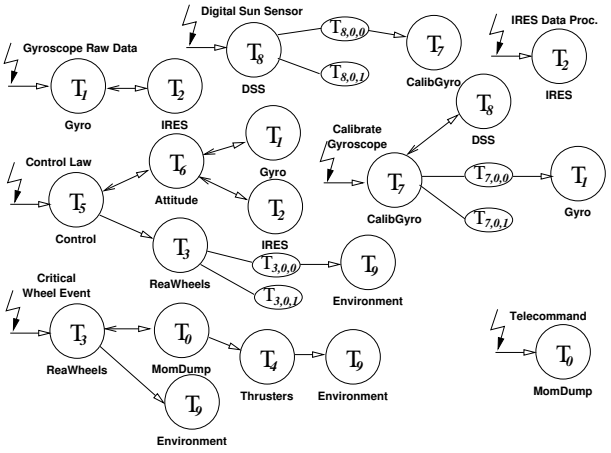


**Figure 2. Task Precedence Graph Examples**

times are given in milliseconds. The translation from Kollochs model leads to suboptimal priority assignements, as many tasks end up with high priorities due to dependencies.

The priorities were assigned in such a way to assign each task the highest priority of any acitvity it is part of. The WCETs in Table 3 have been computed by using the values in Table 2. It has to be noted that the execution times for tasks like $\tau_7$ for the use in for example the digital sun sensor, has been made up of the times for the common part plus the alternative exhibiting no further send operation. Furthermore the part of the triggering interrupts has been neglected to keep the example illustrative. However, their introduction would be straight forward.

## 5 Conclusion

In this paper we have demonstrated an graph based approach to response time analysis for systems, where the deadline required by the environment spans a whole set of tasks. It takes into account blocking and system calls of the tasks. It allows very efficient analysis of systems incorporating a large number of tasks, while avoiding excessive overestimation. It specifically takes into account where the impact of operating system activities like interrupt service routines is mutually inclusive to blocking.

| Event Type | Period | Deadl. | Priority and Activity No. |
|---|---|---|---|
| Critical Wheel Event | 10000 | 100 | 1 |
| Gyroscope Raw Data | 100 | 100 | 2 |
| IRES Data Processing | 100 | 100 | 3 |
| Telecommands | 190 | 190 | 4 |
| Control Law | 200 | 200 | 5 |
| Digital Sun Sensor | 1000 | 1000 | 6 |
| Calibrate Gyroscope | 1000 | 1000 | 7 |

**Table 1. Periods, Priorities and Deadlines of Activities**

| Task | $\tau_0$ | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_{3,0,0}$ | $\tau_{3,0,1}$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|---|---|---|
| Prio. | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
| $C_j$ | 4 | 10 | 8 | 5 | 2 | 1 | 11 | 12 |
| Task | $\tau_6$ | $\tau_7$ | $\tau_{7,0,0}$ | $\tau_{7,0,1}$ | $\tau_{8,0,0}$ | $\tau_{8,0,1}$ | $\tau_8$ | $\tau_9$ |
| Prio. | 5 | 7 | 7 | 7 | 6 | 6 | 6 | 0 |
| $C_j$ | 6 | 7 | 4 | 3 | 3 | 2 | 10 | 5 |

**Table 2. Priorities and WCETs for all Tasks**

Future work will focus on introduction of semaphores while still maintaining the good estimation of the impact of other activities in the system. Another area of investigation will be work towards probabilistic guarantees.

## References


[1] A. Burns, A. Wellings, C. Bailey, and E. Fyfe. The Olympus attitude and orbital control system: A case study in hard real–time system design and implementation. In *Ada sans frontiers, Proc. of the 12th Ada–Europe Conf.*, Lecture Notes in Computer Science, pages 19–35. Springer–Verlag, 1993.

[2] F. Engel, G. Heiser, I. Kuz, S. M. Petters, and S. Ruocco. Operating systems on SoCs: A good idea? In *Embedded Real-Time Systems Implementation (ERTSI 2004) Workshop*, Lisbon, Portugal, Dec. 2004.

[3] W.-C. Feng and J. W.-S. Liu. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. *IEEE Transactions on Software Engineering*, 23, 1997.

[4] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The performance of $\mu$-kernel-based systems. In *Proceedings of the 16th ACM Symposium on OS Principles*, pages 66–77, St. Malo, France, Oct. 1997.

[5] T. Kolloch. *Scheduling with Message Deadlines for Hard Real-Time SDL Systems*. Dissertation, Institute for Real-Time Computer Systems, Technical University Munich, Germany, 2002.

[6] L4Ka Team. L4Ka::Pistachio kernel. http://l4ka.org/projects/pistachio/ .

[7] S. M. Petters. A graph-based response-time analysis of systems with deadlines spanning multiple tasks. Technical Report NICTA-050801T-1, National ICT Australia, Sydney 2052, Australia, Aug. 2005.

[8] K. Tindell. Adding time-offsets to schedulability analysis. Technical report YCS221 (1994), University of York, Department of Computer Science, York, YO10 5DD, United Kingdom, 1994.


| $A_i$ | WCET | Impact of Activity | | | | | | | $R_i$ |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| 1 | 31 | - | | | 4 | 12 | | | 47 |
| 2 | 18 | 31 | - | 8 | 4 | 22 | | 10 | 97 |
| 3 | 8 | 31 | 18 | - | 4 | 22 | | 10 | 97 |
| 4 | 4 | 31 | | | - | 22 | | | 57 |
| 5 | 48 | 31 | 36 | 16 | 4 | - | | 10 | 145 |
| 6 | 23 | 31 | 36 | 16 | 4 | 48 | - | 33 | 191 |
| 7 | 33 | 31 | 36 | 16 | 4 | 48 | 23 | - | 191 |

**Table 3. WCETs, Impact of other Activities and Response Times for Activities**