

# Hard Real-Time Programming is Different \*

Peter Puschner  
Institut für Technische Informatik  
Technische Universität Wien, A1040 Wien, Austria  
Email: peter@vmars.tuwien.ac.at

## Abstract

*The performance requirements imposed on (hard) real-time code resp. non real-time code differ. As a consequence, conventional coding strategies as used to develop non real-time software are not suited for hard real-time code. This paper shows why non real-time coding is not suited for hard real-time systems and presents WCET-oriented programming as a strategy that avoids these shortcomings. It further discusses components of an infrastructure that support the WCET-oriented development of hard real-time code.*

## 1 Introduction

The theory and practice of processor scheduling provides a rich set of scheduling algorithms, each of them suitable for scheduling the tasks of applications with specific requirements, e.g., real-time responsiveness. If a real-time software engineer develops software for a hard real-time application, he or she chooses an adequate scheduling algorithm out of a number of existing real-time schedulers.

When it comes to programming, the answer to the question of what is the right coding strategy for hard real-time code is not so straightforward. No dedicated theory of hard real-time programming has evolved to this date. On the contrary, to a large extent real-time programmers use the same algorithms and programming techniques that have proven to be effective for non real-time applications.

In this paper we demonstrate the different performance criteria that apply to real-time and non real-time code, respectively. We show that as a consequence of this, hard real-time code development must use a different coding strategy than non real-time or soft real-time programming. Further we provide a list of items required for efficient hard real-time code development.

---

\*This work has been supported by the IST research project "High-Confidence Architecture for Distributed Control Applications (NEXT TTA)" under contract IST-2001-32111.

## 2 Performance Criteria

The performance requirements imposed on non real-time respectively real-time code are usually quite diverse. In a system that does not need to fulfil timing requirements, a high throughput, i.e., short average execution time, is desirable.

In a hard real-time system, in contrast, it is crucial that all time-critical tasks meet their deadlines under all anticipated circumstances [3, 1]. To guarantee the latter and still keep the resource needs reasonable, real-time tasks need to have a short worst-case execution time (WCET).

Like hard real-time tasks, soft real-time tasks have deadlines. They may, however, miss these deadlines occasionally. If a deadline miss may be due to overload only and the task must always finish within its time budget, then the task is effectively a hard real-time task. On the other hand a task may be allowed to overrun its budget with a certain probability. In that case, like in the non real-time case, the tasks performance criterium is a statistical one.

For the sake of brevity we only focus on hard real-time and non real-time tasks (as representatives for tasks with probabilistic performance criteria) in the following.

## 3 Shortcomings of Non Real-Time Coding

Non real-time programmers typically aim at a good average performance to allow for a high throughput. Therefore, the primary performance goal of non real-time programmers is the speed optimization for the most probable (i.e., frequent) scenarios. In order to be able to favour the frequent cases the code tests the properties of input-data sets and chooses the actions to be performed during an execution based on input data.

Using input-data dependent control decisions is an effective way to achieve short execution times for the favoured input-data sets. This approach is therefore suitable for optimizing the average execution time. In contrast to this, a programming style that is based on input-data dependent con-

control decisions adversely affects the quality of the achievable WCET. This is due to the following reasons:

- *Tests to identify the current input data and branching:* Even if an input-data set is not among the “favoured” inputs it has to be tested at the points where the control flow between favoured and non favoured inputs splits. Also, the respective branching statements have to be executed. While the fast code makes up for the cost of the control decisions in the case of favoured inputs, the execution time of the input-data tests and branching statements add up to the execution time without compensation for all other data.
- *Information-theoretical imbalance:* Every functionality on a defined input-data space and available data memory has a specific complexity. The overall problem complexity determines the number and types of operations needed to solve the problem for the given input-data space. Performance oriented, non real-time programming spreads this overall complexity unequally over the input-data scenarios. As the complexity inherent to a problem is constant, a cost reduction for some part of the input-data space necessarily causes higher costs for the rest of the inputs. Again, this impairs the achievable WCET.

Input-data dependent control decisions are the consequence of traditional performance-optimization patterns. The strategy followed in such optimizations – looking for solutions that have the shortest expected completion time – seems to be quite similar to the optimization patterns we use in every-day life. In the following we show that we have to apply a completely different and not so common optimization strategy if we aim at optimizing the worst-case completion time.

#### 4 Programming for the Worst Case

We observe that it is the different treatment of scenarios, i.e., favouring certain input-data sets over others, that causes an increased WCET. In order to write code that has a good WCET the shortcomings of the traditional programming style have to be avoided. We have to use a coding style that we call WCET-oriented programming [2]:

*WCET-oriented programming* (i.e., programming that aims at generating code with a good WCET) tries to produce code that is free from input-data dependent control flow decisions or, if this cannot be completely achieved, it restricts operations that are only executed for a subset of the input-data space to a minimum.

Note that in some applications it is impossible to treat all inputs identically. This can be due to the inherent semantics of the given problem or the limitations of the programming language used.

The pieces of code resulting from WCET-oriented programming are characterized by competitive WCETs due to the small number of tests (and branches) on input data and the minimal information-theoretical imbalance. Further, WCET-oriented programming keeps the total number of different execution paths through a piece of code low, thus making WCET analysis easier and less error-prone.

#### 5 Support for WCET-Oriented Programming

WCET-oriented programming needs a way of thinking that is quite different from the solution strategies we normally use. As a consequence, it produces unconventional algorithms that may not look straightforward at the first sight.

To establish a good practice of writing hard real-time code, the WCET-oriented programming approach needs a supportive infrastructure. In particular the following are needed:

- New WCET-oriented algorithms for typical problems found in hard real-time application software,
- Code libraries for the distribution and reuse of WCET-oriented code,
- Programming strategies and guidelines for a systematic development of hard real-time code,
- Tool support for WCET-oriented coding (e.g., specific programming editors and code analysis software to identify input-data dependencies).

We claim that once these supportive items will be available it will be easy to develop code that meets hard real-time requirements. Like in the area of processor scheduling, software developers will then be provided with the adequate means to develop code that is well-suited for hard real-time applications.

#### References

- [1] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.
- [2] P. Puschner. Algorithms for dependable hard real-time systems. In *Proc. 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Jan. 2003.
- [3] J. A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19, Oct. 1988.