

Mälardalen University Press Licentiate Thesis
No.94

Hierarchical Real-Time Scheduling and Synchronization

Moris Behnam

October 2008



School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

Copyright © Moris Behnam, 2008
ISSN 1651-9256
ISBN 978-91-86135-09-6
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

The Hierarchical Scheduling Framework (HSF) has been introduced to enable compositional schedulability analysis and execution of embedded software systems with real-time constraints. In this thesis, we consider a system consisting of a number of semi-independent components called subsystems, and these subsystems are allowed to share logical resources. The HSF provides CPU-time to the subsystems and it guarantees that the individual subsystems respect their allocated CPU budgets. However, if subsystems are allowed to share logical resources, extra complexity with respect to analysis and run-time mechanisms is introduced.

In this thesis we address three issues related to hierarchical scheduling of semi-independent subsystems. In the first part, we investigate the feasibility of implementing the hierarchical scheduling framework in a commercial operating system, and we present the detailed figures of various key properties with respect to the overhead of the implementation.

In the second part, we studied the problem of supporting shared resources in a hierarchical scheduling framework and we propose two different solutions to support resource sharing. The first proposed solution is called SIRAP, a synchronization protocol for resource sharing in hierarchically scheduled open real-time systems, and the second solution is an *enhanced overrun mechanism*.

In the third part, we present a resource efficient approach to minimize system load (i.e., the collective CPU requirements to guarantee the schedulability of hierarchically scheduled subsystems). Our work is motivated from a trade-off between reducing resource locking times and reducing system load. We formulate an optimization problem that determines the resource locking times of each individual subsystem with the goal of minimizing the system load subject to system schedulability. We present linear complexity algorithms to find an optimal solution to the problem, and we prove their correctness.

To the memory of my mother

Acknowledgment

This thesis would not been possible without the help of my supervisors Prof. Mikael Sjödin and Dr. Thomas Nolte and the collaboration with Dr. Insik Shin. I would like to thank Mikael Sjödin for his advices and invaluable input to my research. Thomas, thank you very much for the supporting, encouraging, helping and always finding time to guide me.

A special thank goes to Insik for all the intensive discussions and fruitful cooperation. I would like to say how much I have appreciated working with Thomas, Insik and Mikael, and I have learned a lot from them.

I want to thank the PROGRESSers; Prof. Hans Hansson for his great leading of the PROGRESS center, and Prof. Ivica Crnkovic, Prof. Christer Norström, Prof. Sasikumar Punnekkat, Prof. Paul Pettersson, Dr. Jan Gustafsson, Dr. Andreas Ermedahl and Dr. Cristina Seceleanu.

Also, I would like to thank Prophs'ers (PROGRESS PhD students) Hüseyin Aysan, Andreas Hjertström, Séverine Sentilles, Farhang Nemati, Aneta Vulgarakis, Marcelo Santos, Stefan Bygde, Yue Lu and also the new PhD students Mikael Åsberg, Jagadish Suryadevara, Aida Causevic. We had a lot of fun especially when we arranged the social activities and student surprise for the PROGRESS trips and also when I participated with some of you in PhD schools and conferences.

Many thanks go to Dr. Damir Isovici for informing me about the PhD position and for the very nice recommendation letter that I received from him when I applied for that position.

I would also like to thank the my colleagues at the department for the nice time that I had in the department and special thank goes to the administrative staff, in particular Harriet Ekwall and Monica Wasell for their help in practical

issues.

I would like to express my special gratitude to Dr. Reinder J. Bril at Eindhoven University of Technology, for our collaboration and his constructive comments and discussions.

During my PhD studies, I have participated in 7 conferences, 3 PhD schools and 3 project trips in 7 different countries. Related to this, I would like to thank Dr. Johan Fredriksson and Dr. Daniel Sundmark for being great travel companions.

Finally, my deepest gratitude goes to my wife Rasha and my kids Dany and Hanna for all their support and love.

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

Moris Behnam
Västerås, October, 2008

Contents

I	Thesis	1
1	Introduction	3
1.1	Contributions	5
1.2	Outline of thesis	7
2	Background	9
2.1	Real-time systems	9
2.2	System model	10
2.2.1	Subsystem model	10
2.2.2	Task model	11
2.2.3	Shared resources	11
2.3	Scheduling algorithms	11
2.3.1	Online scheduling	12
2.3.2	Offline scheduling	13
2.4	Logical resource sharing	13
2.4.1	Stack resource policy	14
2.4.2	Resource holding time	14
3	Real-Time Hierarchical Scheduling Framework	17
3.1	Hierarchical scheduling framework	17
3.2	Virtual processor model	18
3.3	Schedulability analysis	19
3.3.1	Local schedulability analysis	19
3.3.2	Global schedulability analysis	20
3.4	Subsystem interface calculation	20

4	Hierarchical Scheduling with Resource Sharing	23
4.1	Problem formulation	23
4.2	Supporting logical resource sharing	25
4.2.1	BWI	25
4.2.2	HSRP	26
4.2.3	BROE	27
4.2.4	SIRAP	28
4.3	Subsystem interface and resource sharing	28
5	Conclusions	31
5.1	Summary	31
5.2	Future work	32
6	Overview of Papers	35
6.1	Paper A	35
6.2	Paper B	36
6.3	Paper C	36
6.4	Paper D	37
	Bibliography	39

II Included Papers 43

7	Paper A:	
	Towards Hierarchical Scheduling in VxWorks	45
7.1	Introduction	47
7.2	Related work	48
7.3	System model	49
7.4	VxWorks	50
7.4.1	Scheduling of time-triggered periodic tasks	51
7.4.2	Supporting arbitrary schedulers	52
7.5	The USR custom VxWorks scheduler	52
7.5.1	Scheduling periodic tasks	52
7.5.2	RM scheduling policy	54
7.5.3	EDF scheduling policy	55
7.5.4	Implementation and overheads of the USR	56
7.6	Hierarchical scheduling	57
7.6.1	Hierarchical scheduling implementation	58
7.6.2	Example	63

7.7	Summary	64
	Bibliography	67
8	Paper B:	
	SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems	71
8.1	Introduction	73
8.2	Related work	74
8.3	System model	76
8.3.1	Hierarchical scheduling framework	76
8.3.2	Shared resources	77
8.3.3	Virtual processor model	77
8.3.4	Subsystem model	79
8.4	SIRAP protocol	80
8.4.1	Terminology	80
8.4.2	SIRAP protocol description	81
8.5	Schedulability analysis	83
8.5.1	Local schedulability analysis	83
8.5.2	Global schedulability analysis	85
8.5.3	Local resource sharing	86
8.6	Protocol evaluation	86
8.6.1	WCET within critical section	87
8.6.2	Task priority	87
8.6.3	Subsystem period	89
8.6.4	Multiple critical sections	91
8.6.5	Independent abstraction	91
8.7	Conclusion	94
	Bibliography	95
9	Paper C:	
	Scheduling of Semi-Independent Real-Time Components: Overrun Methods and Resource Holding Times	99
9.1	Introduction	101
9.2	Related work	102
9.2.1	Hierarchical scheduling	102
9.2.2	Resource sharing	102
9.3	System model and background	103
9.3.1	Resource sharing in the HSF	103
9.3.2	Virtual processor models	104

9.3.3	Stack resource policy (SRP)	105
9.3.4	System model	106
9.4	Schedulability analysis	106
9.4.1	Local schedulability analysis	107
9.4.2	Subsystem interface calculation	107
9.4.3	Global schedulability analysis	107
9.5	Overrun mechanisms	108
9.5.1	Basic overrun	108
9.5.2	Enhanced overrun	110
9.6	Comparison between basic and enhanced overrun mechanisms	111
9.6.1	Subsystem-level comparison	112
9.6.2	System-level comparison	113
9.7	Computing resource holding time	114
9.8	Summary	116
	Bibliography	119

10 Paper D:

	Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources	123
10.1	Introduction	125
10.2	Related work	126
10.3	System model and background	127
10.3.1	Virtual processor models	127
10.3.2	System model	128
10.3.3	Stack Resource Policy (SRP)	129
10.4	Resource sharing in the HSF	130
10.4.1	Overrun mechanism	130
10.4.2	Schedulability analysis	131
10.5	Problem formulation and solution outline	132
10.6	Interface candidate generation	134
10.6.1	ICG algorithm	138
10.7	Interface selection	140
10.7.1	Description of the ICS algorithm	140
10.7.2	Correctness of the ICS algorithm	143
10.8	Overrun mechanism with payback	149
10.9	Conclusion	150
	Bibliography	153

I

Thesis

Chapter 1

Introduction

Hierarchical scheduling has shown to be a useful approach in supporting modularity of real-time software [1] by providing temporal partitioning among applications. In hierarchical scheduling, a system can be hierarchically divided into a number of subsystems that are scheduled by a global (system-level) scheduler. Each subsystem contains a set of tasks that are scheduled by a local (subsystem-level) scheduler. The Hierarchical Scheduling Framework (HSF) allows for a subsystem to be developed and analyzed in isolation, with its own local scheduler. At a later stage, using a global scheduler such as Fixed Priority Scheduling (FPS), Earlier Deadline First (EDF) or Time Division Multiple Access (TDMA), it allows for the integration of multiple subsystems without violating the temporal properties of the individual subsystems. The subsystem integration involves a system-level schedulability test, verifying that all timing requirements are met. This approach by isolation of tasks within subsystems, and allowing for their own scheduler, has several advantages including [2]:

- It allows for the usage of the best scheduler (e.g., FPS, EDF or TDMA) that fit the requirements of each subsystem.
- By keeping a subsystem isolated from other subsystems, and keeping the subsystem local scheduler, it is possible to re-use a complete subsystem in a different application¹ from where it was originally developed.

¹Assuming that the timing parameters of the internal tasks of the subsystem will not be changed when the subsystem is re-used in a different application.

- Hierarchical scheduling frameworks naturally support *concurrent development* of subsystems.

Over the years, there has been a growing attention to HSFs for real-time systems. Deng and Liu [3] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [4] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [5, 6] presented schedulability analysis techniques for the EDF-based global schedulers. Mok *et al.* [7, 8] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [1] introduced the periodic virtual processor model (to characterize the periodic CPU allocation behaviour), and many studies have been proposed on schedulability analysis with this model under fixed-priority scheduling [9, 10, 11] and under EDF scheduling [1, 12]. Being central to this thesis, the virtual periodic resource model is presented in detail in Chapter 3. More recently, Easwaran *et al.* [13] introduced Explicit Deadline Periodic (EDP) virtual processor model. However, a common assumption shared by all above studies is that tasks are independent.

In this thesis we address the challenges of enabling efficient compositional integration preserving temporal behavior for independently developed semi-independent subsystems (i.e., subsystems are allowed to synchronize by the sharing of logical resources) in open systems where subsystems can be developed independently. Efficient compositional integration means that the system should require as little CPU-resources as possible, allowing more subsystems to be integrated in a single processor. Achieving efficient compositional integration makes the HSF a cost-efficient approach applicable for a wide domain of applications, including, automotive, automation, aerospace and consumer electronics.

There have been studies on supporting resource sharing within subsystems [9, 4] and across subsystems [14, 15, 16] in HSFs. Davis and Burns [14] proposed the Hierarchical Stack Resource Policy (HSRP) supporting global resource sharing on the basis of an overrun mechanism. The schedulability analysis associated with the HSRP does not support independent subsystem development (i.e., when performing schedulability analysis for internal tasks of a subsystem using HSRP, information about other subsystems should be provided). Fisher *et al.* [16] proposed the BROE server in order to handle sharing of logical resources in a HSF. A detailed description of these protocols and a comparison between our proposed protocol and these protocols is

presented in Chapter 4.

Our overall goal of this thesis is to propose a scheduling framework and synchronization protocols that are able to fulfill the following requirements;

- With acceptable implementation overhead, it should be possible to implement the HSF in commercial real-time operating systems.
- The framework should support sharing of logical resources between subsystems while preserving the timing predictability and thereby allowing for temporal requirements of the system.
- No knowledge about the parameters of other subsystems is required when developing a subsystem, even in the case when there are dependencies between subsystems (semi-independent subsystems) inherent in the sharing of logical resources.
- The HSF should use the CPU-resources efficiently by minimizing the collective CPU requirement (i.e., system load) necessary to guarantee the schedulability of an entire framework.

1.1 Contributions

The contributions presented in this thesis can be divided into three parts:

Implementation Over the years, there has been a growing attention to HSFs for real-time systems. However, up until now, those studies have mainly worked on various aspects of HSFs from a theoretical point of view. To our knowledge, there are very few studies that focus on the implementation of HSF, especially looking at what can be done with commercial operating systems.

We present our work towards a full implementation of the hierarchical scheduling framework in the VxWorks commercial operating system without changing or modifying the kernel of the operating system. Moreover, to show the efficiency of the implementation, we measure the overheads imposed by the implementation as a function of number of subsystems and number of tasks for both FPS and EDF local and global schedulers.

Supporting shared resources Allowing tasks from different subsystems to share logical resources imposes more complexity for the scheduling of subsystems. A proper synchronization protocol should be used to prevent unpredictable timing behavior of the real-time system. Since there are dependencies

between subsystems through sharing of logical resources, using the protocol with the HSF should not require any information from other subsystems when developing a subsystem in order to not violate the requirement of developing subsystems independently (support open systems).

We present the SIRAP protocol, a novel approach to allow synchronization of semi-independent hierarchically scheduled subsystems. We present the deduction of bounds on the timing behaviour of SIRAP together with accompanying formal proofs and we evaluate the cost of using this protocol in terms of the extra CPU-resources that is required by the usage of the protocol.

In addition to SIRAP, we extend the schedulability analysis of HSRP [14] so that it allows for independent analysis of individual semi-independent subsystems. And also, we propose an enhanced overrun mechanism that gives two benefits (compared with the old version of overrun mechanism): (1) it may increase schedulability within a subsystem by providing CPU allocations more efficiently, and (2) it can even accept subsystems which developed their timing requirements without knowing that the proposed modified overrun mechanism would be employed in the system.

Efficient CPU-resources usage As mentioned previously, one of the requirements that the proposed framework should provide, is to minimize the system load. This can be achieved by finding optimal subsystem timing interfaces (specifies the collective temporal requirements of a subsystem) that minimize the system load. Supporting shared resources across subsystems produces interference among subsystems which imposes more CPU demands for each subsystem and makes the problem of minimizing the system load more complex.

We identify a tradeoff between reducing the time that a subsystem can block other subsystems when accessing a shared resource (locking time which is a part of subsystem timing interface) and decreasing the system load. Selecting the optimal subsystem interface for a subsystem requires information from other subsystems that the subsystem will interact with. However, the required information may not be available during the development stage of the subsystem and in this case we may not be able to select the optimal interface. To solve the problem of selecting an optimal interface for each subsystem, we propose a two-step approach towards the system load minimization problem. In the first step, a set of interface candidates, that have a potential to produce an optimal system load, is generated for each subsystem in isolation. In the second step, one interface will be selected for each subsystem from its own candidates to find the minimum resulting system load. We provide one algorithm for each step and we also prove the correctness and the optimality of the

provided algorithms formally.

1.2 Outline of thesis

The outline of this thesis is as follows: in Chapter 2 we explain and define the basic concepts for real-time systems and the terms that will be used throughout this thesis and in addition we present the system model. In Chapter 3 we describe the hierarchical scheduling framework and the associated schedulability analysis assuming that the subsystems are fully independent. In Chapter 4 we address the problem of allowing dependency through sharing logical resource between subsystem and we present some solutions for this problem. In Chapter 5 we present our conclusion and suggestions for future work. We present the technical overview of the papers that are included in this thesis in Chapter 6 and we present these papers in Chapters 7-10.

Chapter 2

Background

In this chapter we present some basic concepts concerning real-time systems, as well as some methods that will be used in the next chapters.

2.1 Real-time systems

A real-time system is a computing system whose correctness relies not only on the functionality, but also on timeliness, i.e., the system should produce correct results at correct instances of time. Real-time systems are usually constructed using concurrent programs called *tasks* and each task is supposed to perform a certain functionality (for example reading a sensor value, computing output values, sending output values to other tasks or devices, etc). A real-time task should complete its execution before a predefined time called *deadline*.

Real-time tasks can be classified according to their timing constraint to either *hard* real-time tasks or *soft* real-time tasks. For hard real-time tasks, all tasks should complete their execution before their deadlines otherwise a catastrophic consequence may occur. However, for soft real-time tasks, it is acceptable that deadlines are missed which may degrade the system performance, for example consider a mobile phone where missing some deadlines will decrease the quality of the sound. Many systems contain a mix of hard and soft real-time tasks.

A real-time task consists of an infinite sequence of activities called jobs, and depending on the way of task triggering, real-time tasks are modeled as either an *aperiodic task* or a *sporadic task* or a *periodic task*:

- Aperiodic tasks are triggered at arbitrary times, with no known minimum inter-arrival time.
- Sporadic tasks have known minimum inter-arrival time.
- Periodic tasks have a fixed inter-arrival time called period.

Depending on the task model, each task is characterized by timing parameters including task period (periodic task), worst case execution time, deadline, etc.

2.2 System model

In this thesis we focus on scheduling of a single node. Each node is modeled as a system \mathcal{S} which consists of one or more subsystems $S_s \in \mathcal{S}$. The scheduling framework is a two-level hierarchical scheduling framework as shown in Fig 2.1. During run-time, the system level scheduler (Global scheduler) selects which subsystem that will access the CPU-resources.

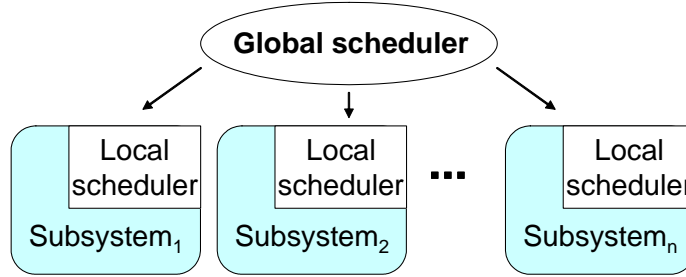


Figure 2.1: Two-level hierarchical scheduling framework with resource sharing.

2.2.1 Subsystem model

A subsystem S_s consists of a task set and a scheduler. Once a subsystem is assigned the processor, the corresponding local scheduler will select which

task that will be executed. Each subsystem S_s is associated with a periodic processor model (abstraction) $\Gamma_s(P_s, Q_s)$, where P_s and Q_s are the subsystem period and budget respectively. This abstraction $\Gamma_s(P_s, Q_s)$ specifies the collective temporal requirements of a subsystem and it is used as an interface between the subsystem and the global scheduler (we refer to this abstraction as *subsystem timing interface*).

2.2.2 Task model

In this thesis, we consider a deadline-constrained sporadic hard real-time task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$ where T_i is a minimum separation time between its successive jobs, C_i is a worst-case execution time requirement for one job, D_i is a relative deadline ($C_i \leq D_i \leq T_i$) by which each job must have finished its execution. Each task is allowed to access one or more logical resources and each element $c_{i,j}$ in $\{c_{i,j}\}$ is a *critical section execution time* that represents a worst-case execution time requirement within a critical section of a global shared resource R_j .

2.2.3 Shared resources

The presented hierarchical scheduling framework allows sharing of logical resource between tasks in a mutually exclusive manner. To access a resource R_j , a task must first lock the resource, and when the task no longer needs the resource it is unlocked. The time during which a task holds a lock is called a critical section time. Only one task at a time may be inside a critical section corresponding to a specific resource. A resource that is used by tasks in more than one subsystem is denoted a *global shared resource*. A resource only used within a single subsystem is a *local shared resource*. We are concerned only with global shared resources and will simply denote them by shared resources.

2.3 Scheduling algorithms

In a single processor, the CPU can not be assigned to more than one task to be executed at the same time. If a set of tasks are ready to execute then a scheduling criterion should be used to define the execution order of these tasks. The scheduling criterion uses a set of rules defined by a scheduling algorithm to determine the execution order of the task set. If all tasks complete their execution before their deadlines then the schedule is called a feasible schedule and

the tasks are said to be schedulable. If the scheduler permit other tasks to interrupt the execution of the running task (task in execution) before completing of its execution then the scheduling algorithm is called a preemptive algorithm, otherwise it is called a non-preemptive scheduling algorithm.

Real-time scheduling algorithms fall in two basic categories; online schedule and off-line schedule [17].

2.3.1 Online scheduling

For online scheduling, the order of task execution is determined during run-time according to task priorities. The priorities of tasks can be static which means that the priorities of tasks will not change during run-time. This type of scheduling algorithm is called Fixed Priority Scheduling (FPS) and both Rate Monotonic (RM) scheduling [18] and Deadline Monotonic (DM) [19] use this type of scheduling. The task priorities can be dynamic which means that they can change during run-time, and Earlier Deadline First (EDF) [18] is an example of such scheduler.

RM and DM scheduling algorithms In RM, the priorities of the tasks are assigned according to their periods; the priority of a task is proportional to the inverse of the task period such that the task with shorter period will have higher priority than the tasks with longer period. The priority of a task is fixed during the run time. The RM scheduling algorithm assumes that tasks periods equals to tasks deadlines. Another FPS algorithm is DM which is similar to RM but the priority depends on the task relative deadlines instead of periods.

The schedulability analysis for each task using RM or DM is as follows [20];

$$\forall \tau_i \in \Gamma, 0 < \exists t \leq D_i \text{ dbf}(i, t) \leq t. \quad (2.1)$$

where Γ is the set of tasks that will be scheduled and D_i is the relative deadline of the task τ_i and $\text{dbf}(i, t)$ is evaluated as follows;

$$\text{dbf}(i, t) = C_i + \sum_{\tau_k \in \text{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil C_k, \quad (2.2)$$

where C_i is the worst case execution time of the task τ_i and T_i is the task period and $\text{HP}(i)$ is the set of tasks with priority higher than that of τ_i .

EDF scheduling algorithm In this scheduling algorithm, the task that has earlier deadline among all tasks that are ready to execute, will execute first. The priority of the task is dynamic and can be changed during run-time depending on the deadline of the task instant and other released tasks ready for execution. The schedulability test for a set of tasks that use EDF is shown in Eq. (2.3) [21] which includes the case when task deadlines are allowed to be less than or equal to task periods.

$$\forall t > 0, \sum_{\tau_i \in \Gamma} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i \leq t \quad (2.3)$$

2.3.2 Offline scheduling

In offline scheduling, a schedule is created before run-time. The scheduling algorithm can take into consideration the timing constraints of real-time tasks such as execution time, deadline, precedence relation (if a task should execute always before another task), etc. The resulting execution sequence is stored in a table and then dispatched during run-time. Finding a feasible schedule using offline scheduling should be done up to the hyper-period (LCM) of task periods, and then, during the run-time, this hyper-period is repeated regularly.

2.4 Logical resource sharing

A *resource* is any software structure that can be used by a task to advance its execution [22]. For example a resource can be a data structure, flash memory, a memory map of a peripheral device. If more than one task use the same resource then that resource is called *shared resource*. The part of task's code that uses a shared resource is called critical section. When a job enters a critical section (starts accessing a shared resource) then no other jobs, including the jobs of higher priority tasks, can access the shared resource until the accessing job exits the critical section (mutual exclusion method). The reason is to guarantee the consistency of the data in the shared resource and this type of shared resource is called nonpreemptable resource. For preemptive scheduling algorithms, sharing logical resources cause a problem called *priority inversion*. The priority inversion problem happens when a job with high priority wants to access a shared resource that is currently accessed by another lower priority job, so the higher priority job will not be able to preempt the lower priority job. The higher priority job will be blocked until the lower priority job releases

the shared resource. The time that the high priority job will be blocked can be unbounded since other jobs with intermediate priority that do not access the shared resource can preempt the low priority job while it is executing inside its critical section. As a result of the priority inversion problem, the higher priority job may miss its deadline. A proper protocol should be used to synchronize the access to the shared resource in order to bound the waiting time of the blocked tasks. Several synchronization protocols, such as the Priority Inheritance Protocol (PIP) [23], the Priority Ceiling Protocol (PCP) [24] and the Stack Resource Policy (SRP) [25], have been proposed to solve the problem of priority inversion. We will explain the SRP protocol in details, a protocol central for this thesis, suitable for RM, DM, and EDF scheduling algorithms.

2.4.1 Stack resource policy

To describe how SRP [25] works, we first define some terms that are used with SRP.

- *Preemption level.* Each task τ_i has a preemption level which is a static value and proportional to the inverse of task relative deadline $\pi_i = 1/D_i$, where D_i is a relative deadline of task τ_i .
- *Resource ceiling.* Each shared resource R_j is associated with a resource ceiling which equal to the highest preemption level of all tasks that use the resource R_j ; $rc_j = \max\{\pi_i | \tau_i \text{ accesses } R_j\}$.
- *System ceiling.* System ceiling is a dynamic parameter that change during execution. The system ceiling is equal to the currently locked highest resource ceiling in the system. If at any time there is no accessed shared resource then the system ceiling would be equal to zero.

According to SRP, a job J_i generated by task τ_i can preempt the currently executing job J_k only if J_i is a higher-priority job of J_k and the preemption level of τ_i is greater than the current subsystem ceiling.

2.4.2 Resource holding time

For a set of tasks that uses the SRP protocol, the duration of time that a task τ_i locks a shared resource, is called *resource holding time* [26, 27] which equals to the maximum task execution time inside a critical section plus the interference (preemption inside the critical section) of higher priority tasks that have preemption level greater than the ceiling of locked resource. The resource holding

time can be computed depending on the scheduling algorithm in use, as shown below;

Under FPS scheduling the resource holding time h_j of a shared resource R_j is [26];

$$W_j^{FPS}(t) = cx_j + \sum_{k=rc_j+1}^n \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \quad (2.4)$$

where cx_j is the maximum worst-case execution time inside the critical section of all tasks that access resource R_j and n is the number of tasks.

The resource holding time h_j is the smallest positive time t^* such that

$$W_j^{FPS}(t^*) = t^*. \quad (2.5)$$

Under EDF scheduling the resource holding time h_j of a shared resource R_j is [27];

$$W_j^{EDF}(t) = cx_j + \sum_{k=rc_j+1}^n \left(\min \left(\left\lceil \frac{t}{T_k} \right\rceil, \left\lfloor \frac{D_i - D_k}{T_k} \right\rfloor + 1 \right) \right) \cdot C_k, \quad (2.6)$$

The resource holding time h_j is the smallest positive time t^* such that

$$W_j^{EDF}(t^*) = t^*. \quad (2.7)$$

An algorithm to decrease the resource holding time without violating the schedulability of the system under the same semantics as that of SRP, was presented in [26, 27]. The algorithm works as follows; it increases the resource ceiling of each shared resource to the next higher value (higher preemption level than the ceiling of the resource) in steps and in each step it checks if the schedule is still feasible or not. If the schedule is feasible then it continues increasing the ceiling of the resource until either the schedule becomes infeasible or the ceiling of the task equals to the maximum preemption level. The minimum resource holding time of a resource R_j is obtained when its resource ceiling equal to the maximum preemption level of the task set. Note that the resource holding time is a very important parameter for the hierarchical scheduling framework, as will be shown in Chapter 4.

Chapter 3

Real-Time Hierarchical Scheduling Framework

In this chapter, we will describe the HSF assuming that all tasks are fully independent, i.e., tasks are not allowed to share logical resources. While in the next chapter we will consider the problem of accessing global shared resources.

3.1 Hierarchical scheduling framework

One of the important properties that the HSF can provide is the isolation between subsystems during design time and run-time such that the subsystems are separated functionally for fault containment and for compositional verification, validation and certification. The HSF guarantees independent execution of the subsystems and it prevents one subsystem from causing a failure of another subsystem through providing the CPU-resources needed for each subsystem.

Each subsystem specifies the amount of CPU-resources that are required to schedule all internal tasks through its timing interface. And the global scheduler will provide the required CPU-resources for all subsystems as specified by the timing interfaces of the subsystems.

In the following sections, we will explain how to evaluate the subsystem timing interface and also show how to verify whether the global scheduler can supply the subsystems with required resources using global schedulability analysis.

Given a subsystem timing interface, it is required to check if the interface

can guarantee that all hard real-time tasks in the subsystem will meet their deadlines using this interface. This check is done by applying local schedulability analysis. But before presenting the local schedulability analysis, we will explain the virtual processor resource model which will be used in the local schedulability analysis.

3.2 Virtual processor model

The notion of real-time virtual processor (resource) model was first introduced by Mok *et al.* [7] to characterize the CPU allocations that a parent node provides to a child node in a hierarchical scheduling framework. The *CPU supply* of a virtual processor model refers to the amount of CPU allocations that the virtual processor model can provide. The *supply bound function* of a virtual processor model calculates the minimum possible CPU supply of the virtual processor model for a time interval length t .

Shin and Lee [1] proposed the periodic virtual processor model $\Gamma(P, Q)$, where P is a period ($P > 0$) and Q is a periodic allocation time ($0 < Q \leq P$). The capacity U_Γ of a periodic virtual processor model $\Gamma(P, Q)$ is defined as Q/P . The periodic virtual processor model $\Gamma(P, Q)$ is defined to characterize the following property:

$$\text{supply}_\Gamma(kP, (k+1)P) = Q, \quad \text{where } k = 0, 1, 2, \dots, \quad (3.1)$$

where the supply function $\text{supply}_{R_s}(t_1, t_2)$ computes the amount of CPU allocations that the virtual processor model R_s provides during the interval $[t_1, t_2]$.

For the periodic model $\Gamma(P, Q)$, its supply bound function $\text{sbf}_\Gamma(t)$ is defined to compute the minimum possible CPU supply for every interval length t as follows:

$$\text{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(P - Q) & \text{if } t \in [(k+1)P - 2Q, \\ & (k+1)P - Q], \\ (k-1)Q & \text{otherwise,} \end{cases} \quad (3.2)$$

where $k = \max\left(\lceil (t - (P - Q))/P \rceil, 1\right)$. Here, we first note that an interval of length t may not begin synchronously with the beginning of period P . That is, as shown in Figure 3.1, the interval of length t can start in the middle of the period of a periodic model $\Gamma(P, Q)$. We also note that the intuition of k in Eq. (3.2) basically indicates how many periods of a periodic model can

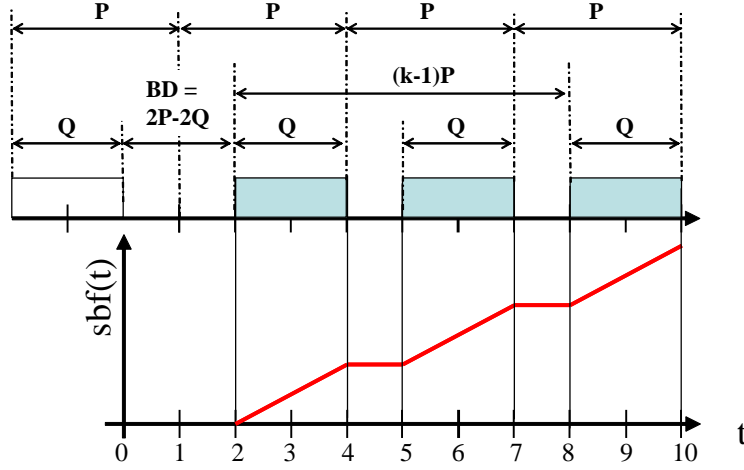


Figure 3.1: The supply bound function of a periodic virtual processor model $\Gamma(P, Q)$ for $k = 3$.

overlap the interval of length t , more precisely speaking, the interval of length $t - (P - Q)$. Figure 3.1 illustrates the intuition of k and how the supply bound function $\text{sbf}_\Gamma(t)$ is defined for $k = 3$.

3.3 Schedulability analysis

This section presents the schedulability analysis of the HSF, starting with local schedulability analysis needed to calculate subsystem interfaces, and finally, global schedulability analysis.

3.3.1 Local schedulability analysis

Let $\text{dbf}_{\text{EDF}}(i, t)$ denote the demand bound function of a task τ_i under EDF scheduling [28], i.e.,

$$\text{dbf}_{\text{EDF}}(i, t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i. \quad (3.3)$$

The local schedulability condition under EDF scheduling is then ([1])

$$\forall t > 0 \quad \sum_{\tau_i \in \Gamma} \text{dbf}_{\text{EDF}}(i, t) \leq \text{sbf}(t), \quad (3.4)$$

Let $\text{dbf}_{\text{FP}}(i, t)$ denote the demand bound function of a task τ_i under FPS [20], i.e.,

$$\text{dbf}_{\text{FP}}(i, t) = C_i + \sum_{\tau_k \in \text{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \quad (3.5)$$

where $\text{HP}(i)$ is the set of tasks with higher priorities than that of τ_i . The local schedulability analysis under FPS can then easily be extended from the results of [25, 1] as follows:

$$\forall \tau_i, 0 < \exists t \leq D_i \quad \text{dbf}_{\text{FP}}(i, t) \leq \text{sbf}(t). \quad (3.6)$$

3.3.2 Global schedulability analysis

The global scheduler schedules subsystems in a similar way as scheduling simple real-time periodic tasks. The reason is that we are using the periodic resource model to abstract the collective timing temporal requirements of subsystems, so the subsystem can be modeled as a simple periodic task where the subsystem period is equivalent to the task period and the subsystem budget is equivalent to the task execution time. Depending on the global scheduler (if it is EDF, RM or DM), it is possible to use the schedulability analysis methods used for scheduling periodic tasks (presented in section 2.3) in order to check the global schedulability.

3.4 Subsystem interface calculation

Using HSF, a subsystem S_s is assigned fraction of CPU-resources which equals to Q_s/P_s . It is required to decrease the required CPU-resources fraction for each subsystem as much as possible without affecting the schedulability of its internal tasks. By decreasing the required CPU-resources for all subsystems, the overall CPU demand required to schedule the entire system (system load) will be decreased, and by doing this, more applications can be integrated in a single processor.

To evaluate the minimum CPU-resources fraction required for a subsystem S_s and given P_s , let $\text{calculateBudget}(S_s, P_s)$ denote a function that calculates

the smallest subsystem budget Q_s that satisfies Eq. (3.4) and Eq. (3.6). Hence, $Q_s = \text{calculateBudget}(S_s, P_s)$. The function is a searching function similar to the one presented in [1] and the resulting subsystem timing interface is (P_s, Q_s) .

Chapter 4

Hierarchical Scheduling with Resource Sharing

In this chapter we extend the HSF that was presented in the previous chapter and allow tasks from different subsystems to share global resources. We are concerned only with global shared resources while managing of local shared resources can be done by using several existing synchronization protocols such as PIP, PCP, and SRP (see [9, 14, 4] for more details).

First, we explain the problem of supporting logical resources followed by discussing some solutions. Later, we show the effect of supporting sharing of global shared resources on the system load required to schedule the entire system.

4.1 Problem formulation

When a task access a shared resource, all other tasks that want to access the same resource will be blocked until the task that is accessing the resource releases it. To achieve a predictable real-time behaviour, the waiting time of other tasks that want to access a locked shared resource should be bounded. The traditional synchronization protocols such as PIP, PCP and SRP that are used with non-hierarchical scheduling, can not without modification, handle the problem of sharing global resources in hierarchical scheduling framework. To explain the reason, suppose a task τ_j that belongs to a subsystem S_I is holding a logical resource R_1 , the execution of the task τ_j can be preempted while

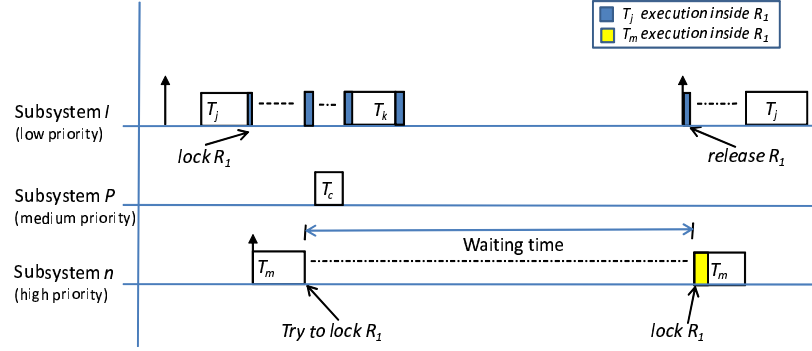


Figure 4.1: Task preemption while running inside a critical section.

τ_j is executing inside the critical section of the resource R_1 (see Fig 4.1) due to the following reasons:

1. **Inter subsystem preemption**, a higher priority task τ_k within the same subsystem preempts the task τ_j .
2. **Intra subsystem preemption**, a ready task τ_c that belongs to a subsystem S_P preempts τ_j when the priority of subsystem S_P is higher than the priority of subsystem S_I .
3. **Budget expiry inside a critical section**, if the budget of the subsystem S_I expires, the task τ_j will not be allowed to execute until the budget of its subsystem will be replenished at the beginning of the next subsystem period P_I .

The PIP, PCP and SRP protocols can only solve the problem caused by task preemption within a subsystem (case number 1) since there is a direct relationship between the priorities of tasks within the same subsystem. However, if tasks are from different subsystems (intra task preemption) then priorities of tasks belonging to different subsystems are independent of each other, which make these protocols not suitable to be used directly to solve this problem. One way to solve this problem is by using the protocols PIP, PCP and SRP between subsystems such that if a task that belongs to a subsystem lock a global resource, then this subsystem blocks all other subsystems where their internal tasks want to access the same global shared resource.

Another problem of directly applying PIP, PCP and SRP protocols is that of budget expiry inside critical section. The subsystem budget Q_I is said to *expire* at the point when one or more internal (to the subsystem) tasks have executed a total of Q_I time units within the subsystem period P_I . Once the budget is expired, no new tasks within the same subsystem can initiate execution until the subsystem's budget is replenished. This replenishment takes place in the beginning of each subsystem period, where the budget is replenished to a value of Q_I .

Budget expiration can cause a problem, if it happens while a task τ_j of a subsystem S_I is executing within the critical section of a global shared resource R_1 . If another task τ_m , belonging to another subsystem, is waiting for the same resource R_1 , this task must wait until S_I is replenished so τ_j can continue to execute and finally release the lock on resource R_1 . This waiting time exposed to τ_m can be potentially very long, causing τ_m to miss its deadline.

4.2 Supporting logical resource sharing

Several mechanisms have been proposed to enable resource sharing in hierarchical scheduling framework. These mechanisms use different methods to handle the problem of bounding the waiting time of other tasks that are waiting for a shared resource. Most of them use the SRP protocol to synchronize access to a shared resource within a subsystem to solve the problem of inter subsystem preemption, and they also use SRP among subsystems to solve the problem of intra subsystem preemption. Note that the effect of using SRP with both local and global scheduling should be considered during the schedulability analysis.

In general, solving the problem of budget expiry inside a critical section is based on two approaches;

- Adding extra resources to the budget of each subsystem to prevent the budget expiration inside a critical section.
- Preventing a task from locking a shared resource if its subsystem does not have enough remaining budget.

The following sections explain these mechanisms in detail.

4.2.1 BWI

The BandWidth Inheritance protocol (BWI) [29] extends the resource reservation framework to systems where tasks can share resources. The BWI approach

uses (but is not limited to) the CBS algorithm together with a technique that is derived from the Priority Inheritance Protocol (PIP). According to BWI, each task is scheduled through a server, and when a task that executed inside lower priority server blocks another task executed in higher priority server, the blocking task will be added to the higher priority server. When the task releases the shared resource, then it will be discarded from the high priority server. For schedulability analysis, each server should be characterized by an interference time due to adding lower priority tasks in the server. This approach is suitable for systems where the execution time of a task inside critical section can not be evaluated. In addition, the scheduling algorithm does not require any prior knowledge about which shared resources that tasks will access nor the arrival time of tasks. However, BWI is not suitable for systems that consist of many hard real-time tasks. The reason is that the interference (that includes the summation of the execution times inside the critical section) from the lower priority tasks will be added to the budget of a hard real-time task server to guarantee that the task will not miss its deadline. Hence, BWI becomes pessimistic in terms of CPU-resources usage for hard real-time tasks.

4.2.2 HSRP

The Hierarchical Stack Resource Policy (HSRP) [14] extends the SRP protocol to be appropriate for hierarchical scheduling frameworks with tasks that access global shared resources. HSRP is based on the overrun mechanism which works as follows: when the budget of a subsystem expires and the subsystem has a job J_i that is still locking a global shared resource, the job J_i continues its execution until it releases the locked resource. When a job access a global shared resources its priority is increased to the highest local priority to prevent any preemption during the access of shared resource from other tasks that belong to the same subsystem. SRP is used in the global level to synchronize the execution of subsystems that have tasks accessing global shared resources. Each global shared resource has a ceiling equal to the maximum priority of subsystems that has a task accessing that resource. Two versions of the overrun mechanisms have been presented; 1) The overrun mechanism with payback which works as follows, whenever overrun happens in a subsystem S_s , the budget of the subsystem will be decreased by the amount of the overrun time in its next execution instant. 2) In the second version which is called overrun mechanism without payback, no further actions will be taken after the event of an overrun. Selecting which of these two mechanisms that can give better results in terms of task response times depends on the system param-

eters. The presented schedulability analysis does not support composability, disallowing independent analysis of individual subsystems since information about other subsystems is needed in order to apply the schedulability analysis for all tasks. In addition, HSRP does not provide a complete separation between the local and the global schedulers. The local scheduler should inform the global scheduler to let the server continue executing when a budget expiry inside a critical section problem happens and then the local scheduler should inform the global scheduler when its task releases the global shared resource.

4.2.3 BROE

The Bounded-delay Resource Open Environment (BROE) server [16] extends the Constant Bandwidth Server (CBS) [30] in order to handle the sharing of logical resources in a HSF. The BROE server is suitable for open systems since it allows for each application to be developed and validated independently. For each application, the maximum CPU-resources demand is characterized by server speed, delay tolerance (using the bounded-delay resource partition [7]) and resource holding time. These parameters will be used as an interface between the application and the system scheduler so that the system scheduler will schedule all servers according to their interface parameters. The interface parameters will also be used during the admission control of new applications to check if there is enough CPU-resources to run this new application on the processor. The BROE server uses the SRP protocol to arbitrate access to global shared resources and in order to prevent the budget expiration inside critical section problem, the application performs a budget check before accessing a global shared resource. If the application has sufficient remaining budget then it allows its task to lock the global resource otherwise it postpones its current deadline and replenishes its budget (according to certain rules that guarantee the correctness of the CBS servers execution) to be able to lock and release the global resource safely. Comparing the BROE server with HSRP, BROE does not need more resources to handle the problem of budget expiry in the global level while HSRP may require more resources since it uses an overrun mechanism and the overrun time should be taken into account in the global scheduling. However, the only scheduling algorithm that is suitable for the presented version of the BROE server is EDF which is one of the limitations of this approach. In addition, in [16], the authors didn't explain how to evaluate the value of the resource holding time for BROE server (the authors left this issue to a future submission) and how this value may affect the CPU-resources usage locally and globally.

4.2.4 SIRAP

The Subsystem Integration and Resource Allocation Policy (SIRAP) [15] protocol supports subsystem integration in the presence of shared logical resources. SIRAP can be used in an open systems. It uses a periodic resource model to abstract the timing requirements of each subsystem. Each subsystem is characterized by its period and budget and resource holding time and it is implemented as a simple periodic server. SIRAP uses the SRP protocol to synchronize the access to global shared resources in both local and global scheduling. SIRAP applies a skipping approach to prevent the budget expiration inside critical section which works as follows; when a job wants to enter a critical section, it enters the critical section at the earliest instant such that it can complete the critical section before the subsystem budget expires. This can be achieved by checking the remaining budget before granting the access to the global shared resources, if there is sufficient remaining budget then the job enters the critical section. If there is insufficient remaining budget, the local scheduler delays the critical section entering of the job until the next subsystem budget replenishment. Comparing SIRAP and BROE, both provide better isolation between the global and the local schedulers than HSRP since they solve the problem of budget expiry inside a critical section locally. However, using HSRP, it is not required to include the resource holding time in the interface of subsystems during run-time and its required only for schedulability analysis while the resource holding times are required during run-time for SIRAP and BROE. Both SIRAP and BROE do not need extra resources in the global scheduling level. The SIRAP protocol needs extra resources in the local level scheduling when it increases the resource demand of the subsystem and for BROE it is not clear since the way of evaluating resource holding time was not presented. Another difference between BROE and SIRAP is that the SIRAP protocol uses FPS as a global scheduling algorithm and can be easily adapted to include local and global EDF while BROE can only work with EDF as a global scheduler.

4.3 Subsystem interface and resource sharing

Supporting shared resources across subsystems produces interference among subsystems which imposes more CPU demands for each subsystem. In the local schedulability analysis and because of using SRP locally, the blocking times should be added to the maximum resources demand side in Eq. (3.4) and Eq. (3.6) and this will increase the minimum required subsystem budget Q_s . In the global level and because of using SRP between subsystems, the block-

ing time (resource holding time¹) that a subsystem may block other subsystems should be added to the global schedulability analysis. So for the global schedulability analysis the subsystem interface should include in addition to the subsystem period and budget, the maximum resource holding time for each global shared resource that the internal tasks of the subsystem may access. One way to decrease the amount of information of subsystem interface needed for global schedulability analysis, can be by considering that the subsystem will access all global resources, then it is required to provide the maximum resource holding time of all internal tasks that access the global shared resources. The subsystem timing interface of a subsystem S_s for this case is (P_s, Q_s, H_s) where H_s is the maximum resource holding time of all internal tasks of S_s that access global shared resources. Finally the extra CPU demand that is required to solve the problem of budget expiry inside the critical section depends on the used mechanism.

As mentioned previously, a subsystem can be blocked in accessing a global shared resource, if there is another subsystem locking the resource at the moment. Such blocking imposes more CPU demands, resulting in an increase of the system load. Therefore, subsystems can reduce their resource holding time, for example, using the mechanism presented in [26, 27] by increasing the resource ceiling of the global shared resources locally inside the subsystems, in order to potentially reduce the blocking of other subsystems towards decrease of the system load. However, we have found that decreasing the value of resource holding times may increase the required budget of the same subsystem Q_s and it may increase the system load.

¹In paper D we use the term resource locking time instead of resource holding time to remove any confusion since the term resource holding time was firstly presented in the context of non-hierarchical scheduling.

Chapter 5

Conclusions

5.1 Summary

We have implemented a HSF in a commercial operating system (VxWorks) without changing the kernel of the operating system. Each subsystem has been implemented using periodic servers. As most commercial real-time operating system, VxWorks does not support the periodic activation of tasks. In order to enable periodic activations of tasks and servers, we have used a timer and an interrupt service rutin. We have measured the overhead of the implementation and the results shows that a hierarchical scheduling framework can effectively achieve the clean separation of subsystems in terms of timing interference (i.e., without requiring any temporal parameters of other subsystems) with reasonable implementation overheads.

We have also investigated the problem of supporting sharing of logical resources and we have presented a novel Subsystem Integration and Resource Allocation Policy (SIRAP), which is a synchronization protocol providing temporal isolation between subsystems that share logical resources. Furthermore, we have formally proven key features of SIRAP such as bounds on delays for accessing shared resources. Also we have provided schedulability analysis for tasks executing in the subsystems; allowing for use of hard real-time applications within the SIRAP framework. Naturally, the flexibility and predictability offered by SIRAP comes with some costs in terms of overhead. We have evaluated this overhead through a comprehensive simulation study.

In addition, we have proposed new overrun mechanisms based on the approach presented in [14], for hierarchical scheduling frameworks, that can be

used in the domain of open systems. We have presented both independent local schedulability analysis as well as global schedulability analysis for the proposed overrun mechanism as well as the existing basic overrun. We have presented analysis of when one overrun mechanism is better than the other and the results indicate that in the general case it is not trivial to evaluate which overrun mechanism that is better than the other.

We have focused on assigning the CPU-resources to subsystems in an efficient way such that the resulting system load will be as low as possible. We introduced a tradeoff between decreasing the resource locking time and the system load, and we presented a two-step approach to explore the intra and inter-subsystem aspects of the tradeoff efficiently, towards determining optimal subsystem interfaces constituting the minimum system load.

5.2 Future work

The work presented in this thesis has left and opened some issues that would be interesting to be investigated in the future. Some of the issues that will be presented are general and some others are specific for each paper.

Starting from general issues, in this work we assume that a system is executed in a single processor while many real-time applications are distributed into several processors that communicate through some communication network. Also, complementing single processor systems, other systems are executed in a multi-processor or multi-core architecture. It will be interesting to extend the HSF include the distributed systems and multi-processor systems.

We would also like to include the subsystem context-switch in the schedulability analysis and check whether using non-preemptive global scheduling can be more efficient than preemptive scheduler in terms of CPU-resources usage. Note that a subsystem context-switch has more overhead than a task context-switch because if a subsystem gets preempted by another subsystem then the scheduler should remove the first subsystem and all its associated tasks and add the higher priority subsystem with all ready tasks that belong to the second subsystem, which takes longer time and could be expensive.

Another interesting work will be on supporting shared resources in multi-level hierarchical scheduling frameworks since we only consider a two-level hierarchical scheduling framework. Also we would like to consider other resource models such as the EDP resource model [13]. Finally it is important to test our framework with real applications by doing case studies.

Paper A In the next stage of the implementation of the HSF, we intend to implement synchronization protocols in hierarchical scheduling frameworks, e.g., using SIRAP [15] and HSRP [14]. In addition, our future work includes supporting sporadic tasks in response to specific events such as external interrupts. We also plan to support soft aperiodic tasks in an efficient way to increase the quality of service of the soft tasks. Moreover, we intend to extend the implementation to make it suitable for more advanced architectures including multi-core processors.

Paper B Future work includes investigating the effect of the context-switch overhead on subsystem utilization together with the subsystem period and the maximum value of h_i .

Paper C Future work includes finding the exact schedulability analysis for the enhanced overrun mechanism, since the presented analysis merely gives upper bound. We would like to include the development of local and global schedulability analysis for Fixed Priority Scheduling (FPS), as the current results only consider Earliest Deadline First (EDF). Another interesting issue is to compare the implementation of the enhanced overrun mechanism with other synchronization mechanisms such as BWI [29], BROE server [16] and SIRAP [15].

Paper C In this paper, we considered only Fixed Priority Scheduling (FPS), and we plan to extend our work to EDF scheduling. Furthermore, our future work includes generalizing our framework to other synchronization protocols such as BROE server [16] and SIRAP [15].

Chapter 6

Overview of Papers

6.1 Paper A

Moris Behnam, Thomas Nolte, Insik Shin, Mikael Åsberg, Reinder J. Bril, *Towards Hierarchical Scheduling on top of VxWorks*, In Proceedings of the 4th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'08), pages 63-72, Prague, Czech Republic, July, 2008.

Summary Over the years, we have worked on hierarchical scheduling frameworks from a theoretical point of view. In this paper we present our initial results of the implementation of our hierarchical scheduling framework in a commercial operating system VxWorks. The purpose of the implementation is twofold: (1) we would like to demonstrate feasibility of its implementation in a commercial operating system, without having to modify the kernel source code, and (2) we would like to present detailed figures of various key properties with respect to the overhead of the implementation. During the implementation of the hierarchical scheduler, we have also developed a number of simple task schedulers. We present details of the implementation of Rate-Monotonic (RM) and Earliest Deadline First (EDF) schedulers. Finally, we present the design of our hierarchical scheduling framework, and we discuss our current status in the project.

My contribution The results of this paper was based on the results of a master project under the supervision of Moris Behnam.

6.2 Paper B

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin, *SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems*, In Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT'07), pages 279-288, Salzburg, Austria, October, 2007.

Summary This paper presents a protocol for resource sharing in a hierarchical real-time scheduling framework. Targeting real-time open systems, the protocol and the scheduling framework significantly reduce the efforts and errors associated with integrating multiple semi-independent subsystems on a single processor. Thus, our proposed techniques facilitate modern software development processes, where subsystems are developed by independent teams (or subcontractors) and at a later stage integrated into a single product. Using our solution, a subsystem need not know, and is not dependent on, the timing behaviour of other subsystems; even though they share mutually exclusive resources. In this paper we also prove the correctness of our approach and evaluate its efficiency.

My contribution The basic idea of this paper was suggested by Moris Behnam. The work was done in cooperation with Moris and Insik Shin, and Moris was responsible for the evaluation part of the paper and he was also involved in the schedulability analysis.

6.3 Paper C

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin, *Scheduling of Semi-Independent Real-Time Components: Overrun Methods and Resource Holding Times*, In Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), IEEE Industrial Electronics Society, Hamburg, Germany, September, 2008.

Summary The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework enabling compositional schedulability analysis of embedded software systems with real-time properties. In this paper a system consists of a number of semi-independent components called subsystems. Subsystems are developed independently and later integrated to form a system. To support this design process, our proposed methods allow non-intrusive configuration and tuning of subsystem timing behaviour via subsystem interfaces for selecting scheduling parameters. This paper considers two methods to handle overruns due to resource sharing between subsystems in the HSF. We present the scheduling algorithms for overruns and their associated schedulability analysis, together with analysis that shows under what circumstances one or the other overrun method is preferred. Furthermore, we show how to calculate resource-holding times within our framework.

My contribution The paper is based on an idea of Insik Shin but Moris has done most of the work including the schedulability analysis for enhanced overrun mechanism and the comparison between the enhanced and the basic overrun mechanism, as well as the simplified equation to evaluate the resource holding times with the required proofs.

6.4 Paper D

Insik Shin, Moris Behnam, Thomas Nolte, Mikael Nolin, *Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources*, In Proceedings of the 29th IEEE International Real-Time Systems Symposium (RTSS08), IEEE Press, Barcelona, Spain, December, 2008, (to be appear).

Summary This paper presents algorithms that (1) facilitate system independent synthesis of timing-interfaces for subsystems and (2) system-level selection of interfaces to minimize CPU load. The results presented are developed for hierarchical fixed-priority scheduling of subsystems that may share logical resources (i.e., semaphores). We show that the use of shared resources results in a tradeoff problem, where resource locking times can be traded for CPU allocation, complicating the problem of finding the optimal interface configuration subject to schedulability. This paper presents a methodology where such a tradeoff can be effectively explored. It first synthesizes a bounded set of interface-candidates for each subsystem, independently of the final system, such that the set contains the interface that minimizes system load for any given

system. Then, integrating subsystems into a system, it finds the optimal selection of interfaces. Our algorithms have linear complexity to the number of tasks involved. Thus, our approach is highly suitable for adaptable and reconfigurable systems.

My contribution The paper was based on ideas of Moris and Insik. Moris was responsible for developing the algorithms and prove their correctness and optimality formally. Moris was also involved in the discussions and writing of the other parts of the paper.

Bibliography

- [1] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, Cancun, Mexico, December 2003.
- [2] G. Lipari, P. Gai, M. Trimarchi, G. Guidi, and P. Ancilotti. A hierarchical framework for component-based real-time systems. In *Component-Based Software Engineering*, volume LNCS-3054/2004, pages 253–266. Springer Berlin / Heidelberg, May 2005.
- [3] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the 18th IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, San Francisco, CA, USA, December 1997. IEEE Computer Society.
- [4] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20th IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, Phoenix, AZ, USA, December 1999. IEEE Computer Society.
- [5] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, Washington DC, USA, May-June 2000. IEEE Computer Society.
- [6] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the 21th IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, Orlando, FL, USA, December 2000.

- [7] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 75–84, Taipei, Taiwan ROC, May 2001.
- [8] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the 23th IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, Austin, TX, USA, December 2002.
- [9] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software (EMSOFT '04)*, pages 95–103, Pisa, Italy, September 2004.
- [10] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, Porto, Portugal, July 2003. IEEE Computer Society.
- [11] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, Miami Beach, FL, USA, December 2005.
- [12] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, Washington, DC, USA, December 2007. IEEE Computer Society.
- [13] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 129–138, Washington, DC, USA, 2007. IEEE Computer Society.
- [14] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, Rio de Janeiro, Brazil, December 2006.
- [15] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *Proceedings of the 7th ACM and IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, Salzburg, Austria, October 2007.

-
- [16] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, Washington, DC, USA, December 2007. IEEE Computer Society.
 - [17] J. A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of classical scheduling results for Real-Time Systems. Technical Report UM-CS-1993-023, University of Massachusetts, Amherst, June 1993.
 - [18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, January 1973.
 - [19] J. Y. T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, December 1982.
 - [20] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the 20th IEEE International Real-Time Systems Symposium (RTSS'89)*, pages 166–171, Santa Monica, CA, USA, December 1989. IEEE Computer Society.
 - [21] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2:301–324, 1990.
 - [22] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. Springer, 2005.
 - [23] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the International Conference on Industrial Electronics, Control, and Instrumentation IECON87*, pages 909–916, Cambridge, MA, USA, November 1987.
 - [24] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, Huntsville, AL, USA, December 1988. IEEE Computer Society.
 - [25] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

- [26] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the 15th International Workshop on Parallel and Distributed Real-Time Systems(WPDRTS)*, pages 1–8, Long Beach, CA, USA, March 2007.
- [27] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in EDF-scheduled systems. In *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, Bellevue, WA, USA, 2007.
- [28] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE International Real-Time Systems Symposium(RTSS'90)*, pages 182–190, Lake Buena Vista, Florida, USA, December 1990. IEEE Computer Society.
- [29] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization' in reservation-based real-time systems. *IEEE Transactions on Computers*, 53(12):1591–1601, December 2004.
- [30] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE International Real-Time Systems Symposium (RTSS'98)*, pages 4–13, Madrid, Spain, December 1998. IEEE Computer Society.