Mälardalen University Press Doctoral Thesis
No.91

# Synchronization Protocols for a Compositional Real-Time Scheduling Framework

Moris Behnam

November 2010


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

ISSN 1651-4238
ISBN 978-91-86135-95-9
Printed by Mälardalen University, Västerås, Sweden

# Abstract

In this thesis we propose techniques to simplify the integration of subsystems while minimizing the overall amount of CPU resources needed to guarantee the schedulability of real-time tasks. In addition, we provide solutions to the problem of allowing for the use of logical resources requiring mutual exclusion.

The contribution of the thesis is presented in three parts. In the first part, we propose a synchronization protocol, called SIRAP, to facilitate sharing of logical resources in a hierarchical scheduling framework. In addition, we extend an existing synchronization protocol, called HSRP, such that each subsystem can be developed independently. The performance of the proposed protocols is evaluated by extensive simulations. In the second part, we present an efficient schedulability analysis that exploits the lower scheduling overhead introduced by each of the proposed protocols. Finally, in the third part, we propose new methods and algorithms that find the optimal system parameters (e.g., optimal resource ceiling), that minimize the amount of CPU resources required to ensure schedulability, when using the proposed synchronization protocols in a hierarchical scheduling framework.

The motivation of this work comes from an emerging industrial trend in embedded software development to integrate multiple applications (subsystems) on a small number of processors. The purpose of this integration is to reduce the hardware related costs as well as the communication complexity between processors. In this setting a large number of industrial applications face the problem of preserving their real-time properties after their integration onto a single processor. An additional motivation is that temporal isolation between the applications during runtime may be required to prevent failure propagation between different applications.

Specifically, we propose a hierarchical scheduling framework that allows for a simplified integration of subsystems. The framework preserves the essential temporal characteristics of the subsystems, both when running in isolation

as well as when they are integrated with other subsystems. In this thesis, we assume a model where a system consists of a number of subsystems. The subsystems can interact with each other using shared logical resources. The framework ensures that the individual subsystem respects its allocated share of the processor. The difficulty lies in allowing two or more subsystems to share logical resources, which introduces an additional complexity in the schedulability analysis and also increases the system load.

To the memory of my mother

# Acknowledgment

This thesis would not been possible without the help of my supervisors Thomas Nolte and Mikael Sjödin and the collaboration with Reinder Bril and Insik Shin. Thomas, thank you very much for the supporting, encouraging, helping and always finding time to guide me. I would like to thank Mikael Sjödin for his advices and invaluable input to my research. I would like to express my special gratitude to Reinder J. Bril for the successful collaboration and for his constructive comments and discussions. A special thanks goes to Insik for all the intensive discussions and fruitful cooperation. I would like to say how much I have appreciated working with Thomas, Reinder, Insik and Mikael, and I have learned a lot from them.

I would like to express my gratitude to all my co-authors for your collaboration, guidance, discussions and nice results, I really enjoyed working with you. Thanks to Farhang Nemati, Mikael Åsberg, Rui Santos, Martijn van den Heuvel.

I would like to thank the PROGRESSers; Hans Hansson for his great leading of the PROGRESS/MRTC center, and Ivica Crnkovic, Christer Norström, Sasikumar Punnekkat, Paul Pettersson, Jan Gustafsson, Andreas Ermedahl, Kristina Lundqvist, Cristina Seceleanu, and Jukka Mäki-Turja.

I would like to acknowledge Radu Dobrin and Antonio Cicchetti for reviewing my thesis and providing valuable feedback.

I would also like to thank my colleagues at the department for the nice time that we had in the department and during conference trips, project trips and PhD schools. I wish to give many thanks to Hüseyin Aysan, Andreas Häjertström, Séverine Sentilles, Aneta Vulgarakis, Marcelo Santos, Stefan Bygde, Yue Lu, Jagadish Suryadevara, Aida Causevic, Rafia Inam, Kathrin Dannmann, Ana Petricic, Sara Dersten, Adnan Causevic, Nikola Petrovic, Holger Kienle,

Federico Ciccozzi, Saad Mubeen, Mehrdad Saadatmand, Johan Kraft, Juraj Feljan, Luka Lednicki, Leo Hatvani, Josip Maras, Tiberiu Seceleanu, Etienne Borde, Thomas Leveque, Andreas SG Gustavsson, Batu Akan, Fredrik Ekstrand, Jörgen Lidholm, Giacomo Spampinato, Markus Bohlin, Jan Carlson, Stefan Cedergren, Barbara Gallina, Andreas Johnsen, Eun-Young Kang, Dag Nyström, Peter Wallin, Johan Fredriksson and Daniel Sundmark.

Special thanks goes to the administrative staff, in particular Gunnar Widforss, Malin Rosqvist, Åsa Lundkvist, Else-Maj Silén, Susanne Fronnå and Carola Ryttersson.

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

I would like to thank Nathan Fisher for hosting me at Wayne State University. I would also like to thank Pradeep Hettiarachchi for helping me during my visit to Wayne State University, it was very kind of you.

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, and the Swedish Research Council.

Moris Behnam
Västerås, November, 2010

# List of Publications

**Publications included in this thesis**

- **Paper A** Moris Behnam, Insik Shin, Thomas Nolte, Mikael Sjödin, *SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems*, In Proceedings of the $7^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07), pages 279-288, October, 2007.
- **Paper B** Moris Behnam, Insik Shin, Thomas Nolte, Mikael Sjödin, *Overrun Methods and Resource Holding Times for Hierarchical Scheduling of Semi-Independent Real-Time Systems*, IEEE Transactions on Industrial Informatics, vol 6, nr 1, pages 93-104, February, 2010.
- **Paper C** Moris Behnam, Thomas Nolte, Mikael Åsberg, Reinder J. Bril, *Overrun and Skipping in Hierarchically Scheduled Real-Time Systems*, In Proceedings of the $15^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), pages 519-526, August, 2009.
- **Paper D** Moris Behnam, Thomas Nolte, Reinder J. Bril, *Bounding the Number of Self-Blocking Occurrences of SIRAP*, In Proceedings of the $31^{th}$ IEEE International Real-Time Systems Symposium (RTSS'10), December, 2010.
- **Paper E** Moris Behnam, Thomas Nolte, Reinder J. Bril, *Schedulability Analysis of Synchronization Protocols Based on Overrun Without Payback for Hierarchical Scheduling Frameworks Revisited*, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-237/2010-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, October, 2010.

- **Paper F** Moris Behnam, Thomas Nolte, Reinder J. Bril, *Refining SIRAP with a Dedicated Resource Ceiling for Self-Blocking*, In Proceedings of the $9^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'09), pages 157-166, October, 2009.

- **Paper G** Insik Shin, Moris Behnam, Thomas Nolte, Mikael Sjödin, *Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources*, In Proceedings of the $29^{th}$ IEEE International Real-Time Systems Symposium (RTSS'08), pages 209-220, December, 2008.

**Publications relevant to the thesis but not included**

A) Journal & Conferences:

- Moris Behnam, Thomas Nolte, Nathan Fisher, *On Optimal Real-time Subsystem-Interface Generation in the Presence of Shared Resources*, In Proceedings of the $15^{th}$ IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'10), September, 2010.

- Thomas Nolte, Insik Shin, Moris Behnam, Mikael Sjödin, *A Synchronization Protocol for Temporal Isolation of Software Components in Vehicular Systems*, IEEE Transactions on Industrial Informatics, vol 5, nr 4, pages 375-387, November, 2009.

- Mikael Åsberg, Moris Behnam, Farhang Nemati, Thomas Nolte, *Towards Hierarchical Scheduling in AUTOSAR*, In Proceedings of the $14^{th}$ IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'09), pages 1181-1188, September, 2009.

- 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 $13^{th}$ IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), pages 575-582, September, 2008.

- Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin, *An Overrun Method to Support Composition of Semi-Independant Real-Time Components*, In Proceedings of the $32^{nd}$ Annual IEEE International Com-

puter Software and Applications Conference (COMPSAC'08), pages 1347-1352, July, 2008.

B) Workshops:

- Rui Santos, Paulo Pedreiras, Moris Behnam, Thomas Nolte, Luis Almeida, *Hierarchical Server-based Traffic Scheduling in Ethernet Switches*, In Proceedings of the $3^{rd}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'10), December, 2010.
- Farhang Nemati, Moris Behnam, Thomas Nolte, *Independently developed Systems on Multi-cores with Shared Resources*, In Proceedings of the $3^{rd}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'10), December, 2010.
- Rui Santos, Moris Behnam, Thomas Nolte, Luis Almeida, Paulo Pedreiras, *Schedulability Analysis for Multi-level Hierarchical Server Composition in Ethernet Switches*, In Proceedings of the $9^{th}$ International Workshop on Real-Time Networks (RTN'2010), pages 44-49, July, 2010.
- Mikael Åsberg, Moris Behnam, Thomas Nolte, Reinder J. Bril, *Implementation of Overrun and Skipping in VxWorks*, In Proceedings of the $6^{th}$ International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'10), pages 45-52, July, 2010.
- Martijn van den Heuvel, Reinder J. Bril, Moris Behnam, *Extending an HSF-enabled Open Source Real-Time Operating System with Resource Sharing*, In Proceedings of the $6^{th}$ International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'10), pages 70-80, July, 2010.
- Farhang Nemati, Moris Behnam, Thomas Nolte, *Multiprocessor Synchronization and Hierarchical Scheduling*, In Proceedings of the 2009 International Conference on Parallel Processing (ICPP'09) Workshops, pages 58-64, September, 2009.
- Reinder J. Bril, Ugur Keskin, Moris Behnam, Thomas Nolte, *Schedulability Analysis of Synchronization Protocols Based on Overrun Without Payback for Hierarchical Scheduling Frameworks Revisited*, In Proceedings of the $2^{nd}$ Workshop on Compositional Theory and Technology

for Real-Time Embedded Systems (CRTS'09), pages 24-32, December, 2009.

- Moris Behnam, Nathan Fisher, *Subsystem-Interface Generation in the Presence of Shared Resources*, In Proceedings of the $2^{nd}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09), pages 16-32, December, 2009.

- Moris Behnam, Thomas Nolte, Mikael Åsberg, Insik Shin, *Synchronization Protocols for Hierarchical Real-Time Scheduling Frameworks*, In Proceedings of the $1^{st}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08), pages 53-60, November, 2008.

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

- Moris Behnam, Thomas Nolte, Insik Shin, *A Hierarchical Approach for Reconfigurable and Adaptive Embedded Systems*, In Proceedings of the $1^{st}$ Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'08), pages 51-54, April, 2008.

# Notes for the Reader

This thesis contains two parts. The first part is an introductory part included in chapters 1-5. The second part is a collection of seven papers (A-G) in chapters 6-12. The seven papers are structured in 3 sections as follows:

- Hierarchical scheduling and synchronization (papers A-C).
- Schedulablity analysis (papers D-E).
- Algorithms for efficient CPU resource usage (papers F-G).

Note that throughout the seven papers, there are some differences in notations, indexes and terminologies. For instance, in some papers we use the term *resource holding time* and in other papers we use *resource locking time* for the same thing. In addition, in some papers we assume that tasks are sorted according to their priorities, in the order of increasing priority, and in other papers we assume that they are sorted in the order of decreasing priority. Therefore it is important to read and follow the corresponding system model of each paper, respectively. Finally, it is recommended to read all included papers before reading chapter 4 (Summary, Conclusions and Future Work) for a better understanding of this chapter.

# Swedish Summary

Målet med avhandlingen är att förenkla integration av delsystem och samtidigt minimera processorkraften som krävs för att delsystemen ska hinna med att utföra alla uppgifter på ett tillfredsställande sätt. I den här avhandlingen fokuserar vi på problemet med att tillåta användandet av logiska resurser tillsammans med hierarkisk schemaläggning och vi föreslår ett nytt synkroniseringsprotokoll för detta. Vi föreslår även nya algoritmer och analyser som på ett resurseffektivt sätt kan minimera processorbelastning vid användandet av synkroniseringsprotokoll för hierarkiska schemaläggare.

En tydlig trend inom många programvaruintensiva industriella tillämpningsområden, till exempel bilindustrin och flygindustrin, är att integrera flera delsystem på ett mindre antal processorer. Syftet med denna integrering är dels att minska olika typer av kostnader, samt att minska komplexiteten framförallt med avseende förenkling av kommunikation mellan delsystem som efter integrering inte längre behöver ske över fysiska nätverk.

Många applikationer/delsystem har realtidskrav och ett problem som uppstår vid integration är att garantera tidsmässiga egenskaper hos dessa applikationer även efter integrering. När integrering sker så finns det en risk att applikationerna kommer att störa varandra på olika sätt. Delsystemens integrering kräver en tidsmässing isolering mellan applikationer/delsystem under körning för att förhindra att en applikation orsakar fel i andra applikationer.

Vi föreslår ett hierarkiskt schemaläggningsramverk som möjliggör en förenklad integrationsprocess av delsystem. Detta ramverk bevarar väsentliga tidsmässiga egenskaper hos delsystemen, både när dessa kör isolerat och när de är integrerade tillsammans med andra delsystem.

I denna avhandling utgår vi ifrån en modell där ett system består av ett antal delsystem. Delsystemen kan interagera med varandra med hjälp av delade logiska resurser. Ramverket ser till att de enskilda delsystemen respekterar sin tilldelade andel av processorn. Svårigheten ligger i att tillåta två eller flera

delsystem att dela logiska resurser, vilket introducerar en extra komplexitet i schemaläggningsanalysen och dessutom ökar processorns belastning.

# Contents

# I

# Thesis

# Chapter 1

# Introduction

In this thesis we address the challenges of allowing sharing of logical resources between tasks that are scheduled by a Hierarchical Scheduling Framework (HSF). Given this HSF, our aim is to provide an efficient compositional integration framework, in terms of CPU resources required to preserve temporal behavior for independently developed applications (subsystems) executing on a single processor.

**Motivation** The complexity of embedded systems is increasing exponentially due to requirements on advanced functionality. For example in the automotive domain, functionality that was realized by mechanical subsystems is often partially or completely replaced by embedded systems (for example engine control, anti-lock braking, etc.). Also new and advanced functionalities are required to be added (for example collision avoidance system, car to car communication, steer by wire, brake by wire, etc.).

To deal with the high complexity of embedded systems, systems are today developed as a set of independent subsystems often by different suppliers. In the final development stages, these subsystems are integrated to produce the final product. Traditionally, in many software intensive industrial application domains, such as automotive and avionics, each subsystem is assigned to one or more dedicated Electronic Control Units (ECUs). In order to provide isolation between subsystems during runtime, different subsystems are not allowed to be executed on the same ECU. However, with the increase of functionality, this approach significantly increases the complexity of the embedded systems in terms of requiring a high number of ECUs, with complex communication so-

lutions in between ECUs. To reduce complexity and cost of these systems, one current trend is to integrate more software subsystems into a lower number of processors [1]. One example can be integrating both the engine control and the gearbox subsystems in one ECU. However, many subsystems have real-time requirements which raise the problem of guaranteeing the timing behavior of these subsystems also after integrating them in a single processor. In addition, temporal isolation between the subsystems during runtime is required to prevent one application from causing a failure of another subsystem.

The hierarchical scheduling framework has been introduced to enable compositional schedulability analysis of systems with real-time constrains to simplify schedulability analysis of complex systems [2]. It offers many additional interesting features that can solve the problem of guaranteeing temporal requirements during the integration of independently developed applications in a single processor. The HSF provides means for decomposing a complex software system into well-defined parts (subsystems). Each subsystem is associated with an abstract notion of its total CPU resource requirements. This abstract notion, manifested by the *subsystem timing interface*, is used during subsystem design time for various kinds of analysis, and during runtime to guarantee correct allocation of CPU resources to the system. In this thesis we refer to this kind of *interface-based hierarchical scheduling* as the *Hierarchical Scheduling Framework (HSF)*. The main feature of the HSF is that it provides CPU partitioning between different subsystems. Thus, subsystems can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification, unit testing, independent development etc. Finally, since subsystems can be developed independently, the HSF facilitates reusability of subsystems in systems that have real-time constrains.

Integrating different subsystems in a single processor implies that these subsystems will not only share the CPU resources, but they may also be in direct competition for other types of resources (such as flash memory, a memory map of a peripheral device, data structures etc.). Many of these resources may be accessed in a non-preemptable manner (using mutual exclusion). Resources that are shared by tasks (in a non-preemptable manner) from different subsystems are called global shared resources, and synchronization protocols should be used to synchronize the access to these shared resources. However, traditional synchronization protocols such as the Priority Inheritance Protocol (PIP) [3], the Priority Ceiling Protocol (PCP) [4], and the Stack Resource Policy (SRP) [5], give rise to a problem of excessive blocking of subsystems due to budget depletion during global shared resource access (more details will be explained in Chapter 3). More appropriate protocols are needed for hierarchical

scheduling frameworks.

In this thesis, our overall goal is to propose a HSF and corresponding synchronization protocols that together are able to fulfill the following requirements:

- The HSF should support sharing of logical resources between subsystems while preserving temporal predictability.
- The HSF should support independent development of subsystems. This requirement enables parallel development of subsystems, as different suppliers can develop different subsystems without revealing the internal details of each subsystem. In addition, this requirement facilitates reuse of software legacy systems/subsystems; systems that have been developed for a long time possibly not complying with any particular system model.
- The HSF should use CPU-resources efficiently. This requirement can be achieved by minimizing *system load*, the collective CPU needed to guarantee the schedulability of the entire framework. This requirement is a very important since fulfilling the first two requirements, increases the systems load (this will be explained in more details in Chapter 3).

## 1.1 Contributions

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

### 1.1.1 Hierarchical scheduling and synchronization

As mentioned above, traditional synchronization protocols such as PIP, PCP and SRP can not handle the problem of resource sharing in hierarchical scheduling frameworks. Hence, more advanced protocols are needed for this kind of systems.

- In paper A we present Subsystem Integration and Resource Allocation Policy (SIRAP); a synchronization protocol for hierarchical scheduling. In addition, we present a simple schedulability analysis that bounds the timing behavior of SIRAP.
- In paper B we develop a schedulability analysis of an existing synchronization protocol HSRP, such that it allows for independent analysis of

individual subsystems. To distinguish between the original analysis of HSRP and the proposed analysis, we use the term *Overrun* to refer to the proposed analysis.

- Finally, in paper C we present a comparative evaluation of the *Overrun* and SIRAP by means of simulation. We apply the protocols on the HSF and we use the same system settings allowing for a fair comparison. The simulation results indicate when one protocol is better than the other and how system/subsystem parameters should be selected in order to operate efficiently.

### 1.1.2 Schedulability analysis

Supporting global shared resource among subsystems is a major challenge as it increases the complexity of the system analysis considerably. Due to this complexity, the schedulability analysis of both SIRAP and *Overrun* (also HSRP) are based on some simplifying assumptions which make them easier. The consequence of these simplifying assumptions is that the analysis may become very pessimistic, potentially requiring more CPU resources than what is actually needed. Therefore we aim at reducing the potential pessimism in the schedulability analysis of SIRAP and HSRP by introducing tighter analysis.

- In paper D we show that the schedulability analysis associated with the SIRAP protocol can be pessimistic if the number of shared resources and/or number of resource accesses is high. We present two different schedulability analysis approaches for SIRAP. The results obtained from simulation analysis show that the new approaches can decrease the CPU resources allocated to each subsystem significantly compared with the original schedulability analysis.

- In paper E we show that the existing schedulability analysis of the *Overrun* (without payback) is pessimistic[1]. We present a tighter analysis that reduce the required CPU resource demand. In addition we evaluate the improvements that the new analysis can achieve compared with the traditional analysis. Depending on the system parameters, a significant improvement in the CPU resource usage can be achieved when using the new analysis. However, the time complexity of the new analysis is higher than the existing analysis presented in paper B.

[1] The pessimism in the scheduability analysis is also included in the original analysis of HSRP.

### 1.1.3 Algorithms for efficient CPU resource usage

It is required that the HSF should use the CPU-resources efficiently, i.e., given a particular system configuration, the system load should be minimized. However, it may not be straightforward to find the optimal subsystems parameters that generate the minimum system load without violating the requirement of independent subsystem development, i.e., without knowledge about temporal behavior of other subsystems that will be integrated on the same CPU. By taking this contradiction between allowing for independent development of subsystems, and minimizing system load, into account, we propose approaches and algorithms that can decrease the CPU resources demand.

- For SIRAP, we show that it is possible to reduce the allocated CPU resource needs for a subsystem by manipulating the ceiling of resources in paper F. Based on this, we propose an algorithm that selects the optimal resource ceiling value per global shared resource that will be used during self-blocking, resulting in the lowest CPU resources allocation needs for that subsystem.
- For the *Overrun*, and considering the requirement of subsystem independent development, we propose a two-step approach to find an optimal solution to the system load minimization problem in paper G. In the first step, and for each subsystem in isolation, an algorithm is proposed to derive a set of interface candidates. In the second step, during system integration, another algorithm is used to select one candidate for each subsystem that minimizes the system load.

## 1.2 Outline of thesis

The outline of this thesis is as follows: in Chapter 2 we explain and define the basic concepts of real-time systems, and the terms that will be used throughout this thesis. In Chapter 3 we describe the hierarchical scheduling framework, we address the problem of allowing global shared resource between subsystems and we present some solutions for this problem. In Chapter 4 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 5 and we present these papers in Chapters 6-12.

# Chapter 2

# Background and System Model

In this chapter we present some basic concepts concerning real-time systems, as well as the system model 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 its functionality, but also on timeliness, i.e., the system should produce correct results at correct instances of time [6]. 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, e.g., in 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 their execution behaviors, real-time tasks are modeled as

either *periodic*, *sporadic* or *aperiodic* tasks:

- Periodic tasks have a fixed inter-arrival time called period, i.e., a periodic task become ready to execute every predefined period.
- Sporadic tasks have known minimum inter-arrival time while the maximum inter-arrival time can be infinity.
- Aperiodic tasks are triggered at arbitrary times, with no known minimum inter-arrival time.

### 2.1.1 Scheduling algorithms

In a single processor, the CPU can not be assigned to more than one task at any given point in 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 preemptive, otherwise it is called a non-preemptive scheduling algorithm.

Real-time scheduling falls in two basic categories; online scheduling (the order of task execution is determined during runtime) and offline scheduling (a schedule is created before runtime)[7].

For online scheduling, the order of task execution is determined during runtime according to task priorities. The priorities of tasks can be static which means that the priorities of tasks will not change during runtime. This type of scheduling algorithm is called Fixed Priority Scheduling (FPS) and both Rate Monotonic (RM) scheduling [8] and Deadline Monotonic (DM) [9] are examples of this type of scheduling. In RM, the priorities of the tasks are assigned according to their periods, while for DM the priority of a task is assigned based on its deadlines. The task priorities can be dynamic which means that they can change during runtime, and Earliest Deadline First (EDF) [8] is an example of such a scheduler. For EDF, the task that has earlier deadline among all ready tasks will execute first.

### 2.1.2 Logical resource sharing

A logical *resource* is any software structure that can be used by a task to advance its execution [10]. For example a resource can be a data structure, flash memory, a memory map of a peripheral device. If more than one task uses 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 non-preemptable resource. For preemptive scheduling algorithms, sharing logical resources cause a problem called *priority inversion* [4]. The priority inversion problem happens when a job with high priority must 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) [3], the Priority Ceiling Protocol (PCP) [4] and the Stack Resource Policy (SRP) [5], 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.

**Stack resource policy** To describe how SRP works, we first define some terms that are used with SRP.

- *Preemption level*. Each task has a preemption level which is a static value proportional to the inverse of task relative deadline for the EDF scheduling. For RM/DM the preemption level equals to the priority of the task.
- *Resource ceiling*. Each shared resource is associated with a resource ceiling which equals to the highest preemption level of all tasks that use the resource.

- *System ceiling*. System ceiling is a dynamic parameter that changes 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 $\tau_i$ 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 $\tau_i$ is greater than the current subsystem ceiling.

## 2.2 System model

In this thesis, our focus is on a two-level hierarchical scheduling framework where a system $\mathcal{S}$, executing on a single processor, consists of one or more subsystems $S_s \in \mathcal{S}$. The hierarchical scheduling framework can be generally represented as a two-level tree of nodes, where each node represents a subsystem with its own scheduler for scheduling internal tasks, and CPU time is allocated from a parent node to its children nodes, as illustrated in Figure 2.1. Each subsystem $S_s$ consists of a set of tasks and a scheduler as shown in Figure 2.1. During runtime, the system level scheduler (global scheduler) selects which subsystem will access the CPU resources. Once a subsystem is assigned the processor, the corresponding subsystem scheduler (local scheduler) selects which task that will be executed.

In this thesis, tasks from different subsystems are allowed to access logical shared resources. Let $\mathcal{R}_s$ denote the global shared resources accessed by $S_s$. Let us also define resource holding time $X_{sk}$ as the maximum time that a task of $S_s$ may lock a resource $R_k \in \mathcal{R}_s$. Finally, let $\mathcal{X}_s = \{X_{sk}\}$ denote the set of maximum resource holding times $\mathcal{X}_s = \{X_{sk}\}|\forall R_k \in \mathcal{R}_s$.

Shin and Lee [2] proposed the notion of subsystem timing interface that abstract the collective temporal requirements of each subsystem, based on the periodic resource model and assuming that tasks do not share global shared resources. The periodic resource model $\Gamma_s(P_s, Q_s)$ includes $P_s$ as a subsystem period and $Q_s$ as a subsystem budget (which represents a periodic CPU resources allocation time). To consider the problem of global shared resources in the HSF, we extend the subsystem timing interface by including a third parameter on it, i.e., a resource holding time $\mathcal{X}_s$. Note that, we assume that each subsystem is associated with a subsystem timing interface and it is used to perform the composability test of the subsystems. Moreover, the subsystem timing interface is used by the global scheduler during runtime, to assign CPU resources to the subsystem.

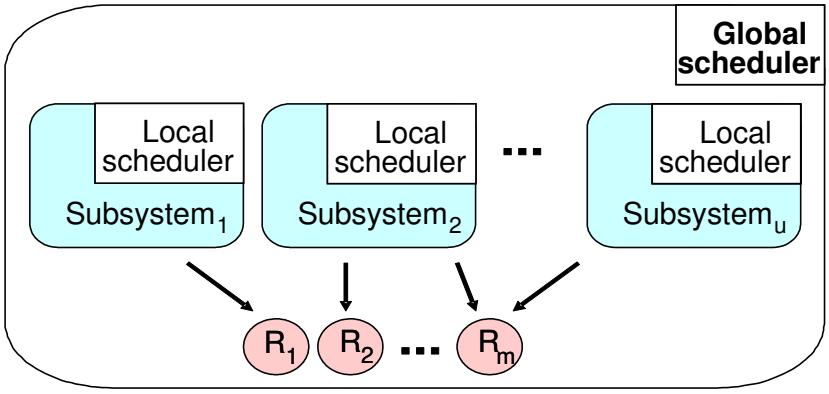


Figure 2.1: Two-level hierarchical scheduling framework with resource sharing.

For the task model, 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 the minimum separation time between its successive jobs, $C_i$ is the worst-case execution time, and $D_i$ is a relative deadline ($C_i \leq D_i \leq T_i$). Each task is allowed to access one or more logical shared resources and each element $c_{i,j} \in \{c_{i,j}\}$ is a *critical section length* that represents the worst-case execution time of $\tau_i$'s access to a global shared resource $R_j$.

In this thesis, we assume that both local and global schedulers use the fixed priority preemptive scheduling policy (FPS), however, most of the results of this thesis are not limited to FPS and can be extended to include other paradigms such as EDF (papers A and B present analysis for EDF as well). Additionally, we focus only on two synchronization protocols that handle the problem of sharing global shared resources, i.e., SIRAP and *Overrun* (without payback), as these protocols have similarities in their analysis (use the periodic resource model) and implementation (use the periodic server[1]). Finally, the SRP protocol is assumed to be used within a subsystem to arbitrate the access of shared resources by tasks.

Note that throughout the seven included papers in this thesis, there are some differences in notations, indexes and terminologies. Therefore it is important to read and follow the corresponding system model of each paper, respectively.

[1] A periodic server is a server that executes as a periodic task.

# Chapter 3

# A Real-Time Hierarchical Scheduling Framework with Logical Resource Sharing

In this chapter, we will first describe the analysis of the HSF. Then we will explain the problem of accessing global shared resources in a hierarchical scheduling framework, and we present some protocols that can handle this problem.

Over the years, there has been a growing attention to using hierarchical scheduling for real-time systems. Deng and Liu [11] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [12] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [13, 14] presented schedulability analysis techniques for the EDF-based global schedulers. Mok *et al.* [15, 16] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [2] 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 [17, 18, 19] and under EDF scheduling [2, 20]. Being central to this thesis, the virtual periodic resource model is presented in detail in this chapter. Easwaran *et al.* [21] introduced the Explicit Deadline Periodic (EDP) virtual processor model. A

common assumption shared by all above studies is that tasks are independent, i.e., tasks are not allowed to share logical resources.

In this thesis, we relax the assumption of independent tasks by addressing the challenge of enabling efficient compositional integration for independently developed subsystems interacting through sharing of global shared resources. In particular, we extend the HSF proposed by Shin and Lee [2] enabling sharing of global shared resources.

## 3.1 HSF schedulability analysis

In the following sections, we will explain how to evaluate the subsystem timing interface and we will show how to verify the composability of the system. We will first explain the virtual processor resource model and the local schedulability analysis that are used to evaluate the subsystem timing interface; the subsystem's abstract notion of respurce requirement needed to ensure correct timing.

### 3.1.1 Virtual processor model

The notion of real-time virtual processor (resource) model was first introduced by Mok *et al.* [15] 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 [2] proposed the periodic virtual processor model $\Gamma(P_s, Q_s)$, where $P_s$ is a subsystem period ($P_s > 0$) and $Q_s$ is a subsystem budget ($0 < Q \leq P$). The supply bound function $\mathtt{sbf_s(t)}$ (shown in Figure 3.1) of the periodic resource model is computed as follows;

$$\mathtt{sbf_s(t)} = \begin{cases} t - (k+1)(P_s - Q_s) & \text{if } t \in V^k \\ (j-1)Q_s & \text{otherwise,} \end{cases} \tag{3.1}$$

where $k = \max\Big(\lceil \big(t - (P_s - Q_s)\big)/P_s \rceil, 1\Big)$ and $V^k$ denotes an interval $[(k+1)P_s - 2Q_s, (k+1)P_s - Q_s]$ in which the subsystem $S_s$ receives budget. To guarantee a minimum CPU resource supply, the worst-case budget provision is considered in Eq. (3.1) assuming that I) tasks are released at the same

time when the subsystem budget depletes (at time $t = 0$ in Figure 3.1), II) the first budget was supplied at the beginning of the period, and III) all following budgets will be supplied at then end of the subsystem period. By considering the worst-case budget provision, a subsystem assumes that the maximum interference from higher priority subsystems will occurs which delays the budget supply to the end of the subsystem period, and by this it does not require any information from other subsystems that will be integrated. Note that, depending on the parameters of the subsystem, the assumption of the worst-case budget provision may make the analysis pessimistic (se Section 6.6.5).

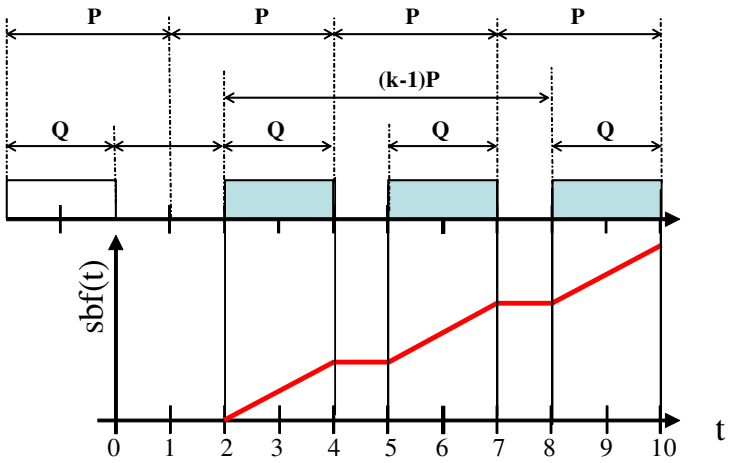


Figure 3.1: The supply bound function of a periodic virtual processor model $\Gamma(P, Q)$ for $k = 3$.

### 3.1.2 Local schedulability analysis

The local schedulability analysis is used to check whether a given periodic resource model $\Gamma(P, Q)$ (subsystem period and budget) can guarantee the temporal requirements of the subsystem internal tasks. The local schedulability analysis is as follows [2]:

$$\forall \tau_i, 0 < \exists t \leq D_i \;\; \texttt{rbf}_{\texttt{FP}}(i, t) \leq \texttt{sbf}(t), \tag{3.2}$$

where $\texttt{rbf}_{\texttt{FP}}(i,t)$ denotes the *request bound function* of a task $\tau_i$ which computes the maximum cumulative execution requests that could be generated from the time that $\tau_i$ is released up to time $t$. $\texttt{rbf}_{\texttt{FP}}(i,t)$ is computed as follows

$$\texttt{rbf}_{\texttt{FP}}(i,t) = b_i + C_i + \sum_{\tau_k \in \texttt{hp}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{3.3}$$

where $\texttt{hp}(i)$ is the set of tasks with priorities higher than $\tau_i$ and $b_i$ is the maximum blocking time from lower priority tasks.

### 3.1.3 System composability

For a system $S$ that consists of a set of $m$ subsystems $S_1, S_2, ...S_m$, the subsystems are composable if each subsystem receives sufficient execution satisfying its timing interface [2], in other words, the subsystems are schedulable under the given global (system level) scheduler. As long as the periodic resource model is used in evaluating the subsystem timing interface, each 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 worst case execution time of a task. Moreover, resource holding times can be modeled as critical section execution times. Then the schedulability analysis used for simple periodic tasks with resource sharing can be used. Hence, the general condition for global schedulability is,

$$\forall S_s \ \exists t : 0 < t \leq P_s, \ \ \texttt{RBF}_s(t) \leq t, \tag{3.4}$$

where $\texttt{RBF}_s(t)$ denotes the request bound function of a subsystem $S_s$ and it is computed as follows,

$$\texttt{RBF}_s(t) = Q_s + B_s + \sum_{S_k \in \texttt{HP}(s)} \left\lceil \frac{t}{P_k} \right\rceil \cdot Q_k, \tag{3.5}$$

where $\texttt{HP}(i)$ is the set of subsystems with priorities higher than that of $S_s$ and $B_s$ is the maximum blocking time (resource holding time) imposed to a subsystem $S_s$, when it is blocked by lower-priority subsystems (suppose that $S_j$ imposes the maximum blocking on $S_s$ then $B_s = X_j$).

### 3.1.4 Subsystem interface evaluation

Looking at Eq. 3.5, the composability of a system is increased when decreasing the subsystem utilization $Q_s/P_s$. Finding optimal values for $P_s$ and $Q_s$ that

minimize the overall processor requirement of the system, without providing information about other subsystems, is rather complex and requires extensive search algorithms [22] given a HSF without sharing of global shared resources. The problem becomes even more complex when allowing for global shared resources, inherent in the dependency that is created between subsystems as a result of sharing global shared resources. To simplify this problem, similar to the work presented in [2], we assume that the subsystem period $P_s$ is given for each subsystem and we are required to evaluate the minimum subsystem budget needed to guarantee the schedulability of all internal tasks. To evaluate the smallest subsystem budget $Q_s$, we use the *FPMINIMUMCAPACITY* algorithm presented in [23]. Furthermore, the algorithm presented in [2], which is based on linearizing the supply bound and the request bound functions, can be used to provide faster but less accurate results.

To be able to apply global schedulability analysis (compositional analysis), the resource holding time of the global shared resources should be provided in the subsystem timing interface (see Eq. 3.5). The resource holding time of a task $\tau_i$ accessing a global shared resource $R_j$ includes the critical section execution time of $\tau_i$ accessing $R_j$ and the maximum interference from higher priority tasks within the same subsystem [24]. Since the resource holding times of all global shared resources should be included in the timing interface, then depending on the number of global shared resources, the subsystem timing interface may contain a lot of information. One way to increase the level of abstraction of the timing interface (providing less information) is to use one value for resource holding time equal to the maximum value among all elements of $\mathcal{X}_s$. For this case, the subsystem timing interface will include less information at the cost of the global schedulability analysis being less accurate.

## 3.2 Global resource sharing

In this section we will focus on the problem of supporting global shared resources such that tasks from different subsystems share logical resources. The logical resources that are shared by tasks from the same subsystem are called *local shared resources* and the works presented in [12, 17, 25] show that using existing synchronization protocols such as SRP can handle the problem of sharing local resources without any modification. However, for global shared resources, new synchronization protocols are required. First we explain the problem of supporting global shared resources followed by discussing some solutions.

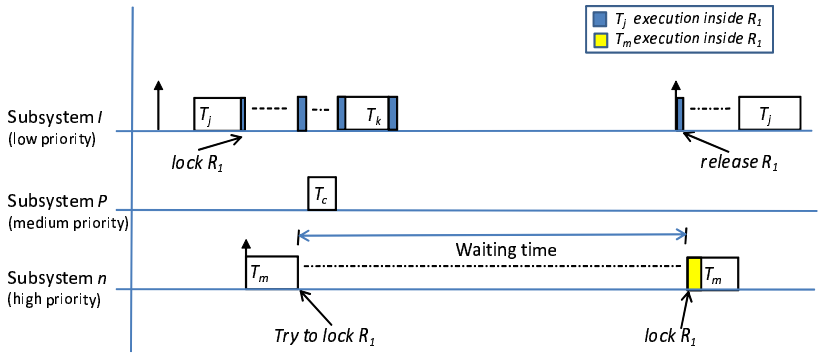


Figure 3.2: Task preemption while running inside a critical section.

### 3.2.1 Problem formulation

When a task $\tau_j$ locks a shared resource $R_k$, all other tasks that want to access the same resource $R_k$ will be blocked until $\tau_j$ 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 SRP used with non-hierarchical scheduling, can not without modification handle the problem of sharing global shared resources in a hierarchical scheduling framework. To explain the reason, suppose $\tau_j$ that belongs to a subsystem $S_I$ is holding a logical resource $R_1$, the execution of the task $\tau_j$ can be preempted while $\tau_j$ is executing inside the critical section of the resource $R_1$ (see Fig 3.2) due to the following reasons:

1. **Intra subsystem preemption**, a higher priority task $\tau_k$ within the same subsystem preempts the task $\tau_j$.

2. **Inter subsystem preemption**, a ready task $\tau_c$ that belongs to a subsystem $S_P$ preempts $\tau_j$ when the priority of subsystem $S_P$ is higher than the priority of subsystem $S_I$.

3. **Budget depletion inside a critical section**, if the budget of the subsystem $S_I$ depletes, the task $\tau_j$ 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 SRP protocol 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. Moreover, if tasks are from different subsystems (inter task preemption) then priorities of tasks belonging to different subsystems are independent. Still, the priorities of the subsystems can be used to solve this problem, and SRP can be used between subsystems such that, if a task that belongs to a subsystem locks a global shared resource, then this subsystem blocks all other subsystems if any of their internal tasks want to access the same global shared resource. However, SRP can not handle case number 3, i.e., budget depletion inside a critical section. Budget depletion can cause a problem if it happens while a task $\tau_j$ of a subsystem $S_I$ is executing inside the critical section of a global shared resource $R_1$. If another task $\tau_m$, belonging to another subsystem, is waiting for the same resource $R_1$, this task must wait until $S_I$ is replenished so $\tau_j$ can continue executing until it releases the lock on resource $R_1$. This waiting time exposed to $\tau_m$ can be potentially very long, causing $\tau_m$ to miss its deadline.

## 3.3 Supporting global resource sharing

Four different mechanisms have been proposed to enable resource sharing in the context of hierarchical scheduling. The mechanisms use different methods to bound the waiting time of tasks that share the same global shared resources. Three of them use the SRP protocol to synchronize access to a global shared resource, while the forth mechanism uses an extended version of PIP.

If SRP is used in a HSF then the SRP's associated terms *resource* and *system* ceiling should be extended as follows:

*Resource ceiling*: With each global shared resource $R_k$, two types of resource ceilings are associated; an *internal* resource ceiling ($rc_{sk}$) for local scheduling and an *external* resource ceiling ($RX_k$) for system level scheduling.

*System/subsystem ceiling*: The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling (i.e. highest priority) of a currently locked resource in the system/subsystem.

Solving the problem of budget depletion inside a critical section can be done following one of the three approaches:

- Preventing a task from locking a shared resource if its subsystem does not has enough remaining budget (*skipping* and *deadline shifting* mechanisms).

- Adding extra resources to the budget of each subsystem to prevent the budget depletion inside a critical section (*overrun* mechanism).
- Using the budget of other subsystems when their internal tasks want to access an already locked global shared resource (*bandwidth* inheritance).

The following sections explain different synchronization protocols using these three approaches.

### 3.3.1 SIRAP

The Subsystem Integration and Resource Allocation Policy (SIRAP) [26] protocol is based on the skipping mechanism to prevent depletion of the budget during global shared resource access. SIRAP uses the periodic resource model and its mechanism works as follows; when a task $\tau_i$ tries to access a global shared resource $R_k$, SIRAP checks the remaining budget before granting the access to the global shared resource; if there is sufficient remaining budget to lock and release $R_k$ before budget depletion (i.e., the currently un-consumed budget is greater than the maximum time that $\tau_i$ may lock $R_k$), then the task enters the critical section, and it updates both system and subsystem ceiling. If there is insufficient remaining budget, SIRAP takes the following actions:

- **Self-blocking**: the job of $\tau_i$ is blocked until the next following budget replenishment so, at that time, there will be enough budget to lock and release the shared resource $R_k$.
- **Subsystem ceiling**: the subsystem ceiling will be updated to bound the minimum required subsystem budget, while the system ceiling is only updated when a global shared resource is accessed so that tasks from other subsystems can access the global shared resource $R_k$.
- **Budget replenishment**: at the time instant when the subsystem budget is replenished, the state of the job of task $\tau_i$ will be changed to ready such that it can execute and lock the global shared resource.

The reason of updating the subsystem ceiling during the self-blocking is to bound the minimum required budget. If tasks with priority lower than that of $\tau_i$ are allowed to execute while $\tau_i$ is in the self-blocking state, it may cause additional self-blocking. This is unpractical as the subsystem budget should be big enough to finish all self-blocking within one budget supply. In paper F we have proposed an algorithm that finds the best value of subsystem ceiling

during the self-blocking state in order to decrease the minimum subsystem budget.

Figure 3.3 illustrates an example of a self-blocking occurrence during the execution of subsystem $S_s$. A job of a task $\tau_i$ tries to lock a global shared resource $R_k$ at time $t_2$. It first determines the remaining subsystem budget $Q^r$ (which is equal to $Q^r = Q_s - (Q^1 + Q^2)$, i.e., the subsystem budget left after consuming $Q^1 + Q^2$). Next, it checks if the remaining budget $Q^r$ is greater than or equal to the maximum resource locking time ($X_{ik}$) of the job access to $R_k$, i.e., if ($Q^r \geq X_{ik}$). In Figure 3.3, this condition is not satisfied, so $\tau_i$ blocks itself and is not allowed to execute before the next replenishment period ($t_3$ in Figure 3.3).

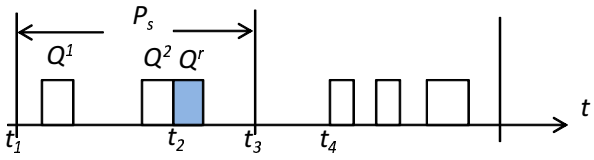


Figure 3.3: An example illustrating self-blocking.

SIRAP uses the periodic resource model to abstract the timing requirements of each subsystem. The effect of using SIRAP on the schedulability analysis appears in the local schedulability analysis in Eq. (3.2) either on $\texttt{rbf}_{\texttt{FP}}(i,t)$ or $\texttt{sbf}(t)$ (papers A and D).

### 3.3.2 HSRP

The Hierarchical Stack Resource Policy (HSRP) [25] extends the SRP protocol to handle the sharing of global shared resources problem in hierarchical scheduling frameworks. HSRP is based on the overrun mechanism and it works as follows: when the budget of a subsystem expires and a job of task $\tau_i$ that belong to the subsystem has not released the lock of a global shared resource, the subsystem overruns its budget and the job continues its execution until it releases the locked resource. When a job accesses a global shared resource its priority is increased to the highest local priority to prevent any preemption from other tasks that belong to the same subsystem during the access of the shared resource. SRP is used at the global level to synchronize the execution of subsystems. Each global shared resource has a ceiling equal to the maximum priority of the subsystem that its internal tasks may access that resource. Two versions of the overrun mechanism have been presented: 1) The overrun mechanism with payback which works as follows: whenever overrun happens in a subsystem, the budget of the subsystem will be decreased by the amount of the overrun time in its next execution instant as shown in Figure 3.4a. 2) In the second version which is called overrun mechanism without payback, no further actions will be taken after the event of an overrun (see Figure 3.4b). Selecting which of these two mechanisms gives better results, in terms of task response times depends on the system parameters. 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 paper B the analysis of HSRP has been extended to support compositional scheduling based on the periodic resource model. In addition, and to generalize the local analysis, SRP is assumed to be used locally and globally. The effect of using *Overrun* on the schedulability analysis is added to the global schedulability analysis in Eq. (3.5).

### 3.3.3 The BROE server

The Bounded-delay Resource Open Environment (BROE) server [29] extends the Constant Bandwidth Server (CBS) [30] in order to handle the problem of sharing logical resources in a hierarchical scheduling framework. The analysis associated with the BROE server supports independently developed subsystems (open environment). BROE uses the bounded-delay resource model [15] to characterize the CPU allocations for each subsystem. Because of using the

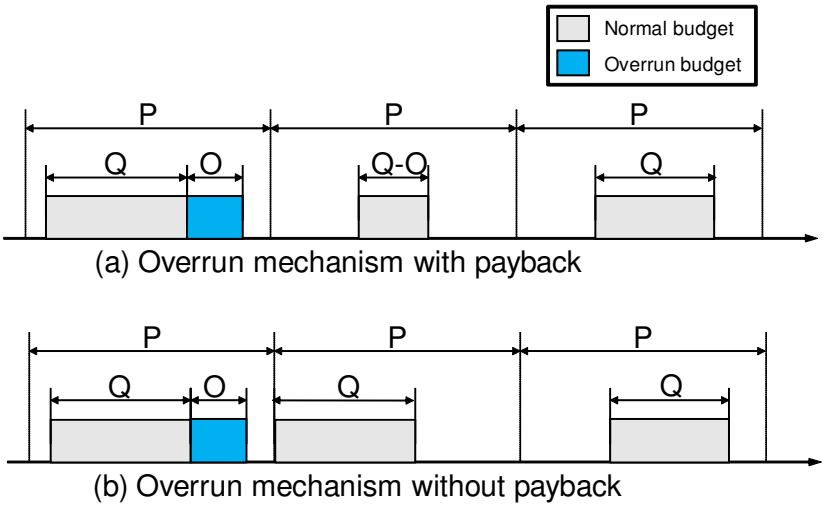


Figure 3.4: Overrun mechanism.

bounded-delay resource model, the timing interface of a subsystem includes server speed, delay tolerance and resource holding time. These parameters are used both during design time, to verify the composability of the system (global schedulability analysis), as well as during runtime, when the global scheduler allocates the required CPU for each subsystem. The BROE server is based on the deadline shifting mechanism and it uses the SRP protocol to arbitrate access to global shared resources. In order to prevent budget depletion inside critical sections, the subsystem performs a budget check before accessing a global shared resource. If the subsystem remaining budget is sufficient, it allows the task to lock the global shared resource. Otherwise it postpones its current deadline and replenishes its budget (according to certain rules that guarantee the correctness of the CBS servers execution and the CPU resource model supply) to be able to lock and release the global shared resource safely. Finally, as BROE extends the CBS server, only the EDF scheduling algorithm can be used in the global (system level) scheduling.

### 3.3.4 BWI

The BandWidth Inheritance protocol (BWI) [31] 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 is executing inside a lower priority server blocks another task executing in a higher priority server, the blocking task will be added to the higher priority server. When the task releases the shared resource, it will be discarded from the high priority server. For schedulability analysis, each server should include a characterization of 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 a critical section can not be predicted. In addition, the scheduling algorithm does not require any a-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 worst-case interference from all tasks that belong to other servers and access global shared resources will be added to the budget of each server to guarantee the timing requirements of the tasks. Hence, BWI becomes pessimistic in terms of CPU-resource usage for systems that have hard real-time tasks accessing global shared resources. In this thesis we consider systems with hard real-time requirements, hence we will exclude further discussion of BWI.

## 3.4 Isolation between subsystems

One of the key advantages of hierarchical scheduling is that it provides an isolation between subsystems during runtime, i.e., the execution of tasks that belong to a subsystem will not affect the execution of tasks that belong to other subsystems in an unpredictable manner. A full isolation between subsystems can be achieved for independent subsystems where tasks do not share global shared resources. However, sharing of global shared resources makes it more difficult to guarantee isolation between subsystems during runtime. There are two reasons that may violate the isolation between subsystems:

- The execution time of a job inside a critical section may exceed its estimated worst-case execution time. This may increase the resource holding time and may affect the global schedulability of the system. As a result, the other subsystems may not get enough CPU resources and their

tasks may miss their deadlines. To solve this problem, a runtime mechanism should be used to make sure that a task will never exceed its critical section execution time when it accesses a global shared resource.

- A task may corrupt the data of a global shared resource which may affect all subsystems that share the same global shared resource [29]. If the corrupted data is detectable by the system then a backup strategy may be used to restore the latest correct data. Otherwise it is impossible to achieve a full isolation between subsystems and it will be the responsibility of the designer/developer to avoid the use of such global shared resources.

## 3.5 Comparing SIRAP, HSRP and BROE

This section compares HSRP, BROE, and SIRAP, looking at some theoretical properties, implementation complexity and overhead.

### 3.5.1 Theoretical comparison

A detailed systematic comparison may not be possible/fair between the three protocols SIRAP, HSRP and BROE, since each protocol has different assumptions, settings and goals. For example, in both HSRP and BROE it is assumed that subsystem parameters (period and budget) are given and it is required to verify the schedulability of the system, while the goal of the analysis associated with SIRAP is to find optimal/suboptimal subsystem parameters that increase the composability of the system. In addition, the schedulability algorithms used in the three approaches are different; HSRP uses only FPS in both subsystem and system level, while BROE uses only EDF and SIRAP uses either FPS or EDF in the local and global level. Another difference between the protocols is the resource supply model; BROE uses the bounded-delay resource model while SIRAP uses the periodic resource model, and HSRP uses the periodic resource model implicitly. Furthermore, HSRP uses SRP in the global level while locally it uses a simple non-preemptive approach when a task accesses a global shared resource. For BROE and SIRAP, SRP is used locally and can also be used globally depending on the required level of abstraction of the subsystem timing interface. Finally, the analysis associated with HSRP does not support independent subsystem development, while this is supported by SIRAP and BROE.

In this thesis, we aim at defining a common framework in which we can compare the mechanisms used by the protocols, such that we can compare the protocols and/or properties of the protocols.

One of the properties that can be used to compare efficiency of the protocols is the *system load*, which is a measure of the amount of collective CPU requirements necessary to guarantee the schedulability of an entire framework. By minimizing the system load, more subsystems can be integrated in a single processor, which makes the framework cost-efficient and applicable for a wider range of applications.

In spite of the differences between the three protocols, a theoretical high level comparison can be carried out based on their schedulability analysis. Comparing the schedulability analysis of SIRAP and HSRP, it is not possible to prove that one of the protocols is more efficient than the other. The reason is that their efficiency depends on the subsystem parameters as well as on the parameters of the shared resources (even between the two types of overrun mechanisms presented in [25] is not easy to find which of them that requires less system load). In paper C we have compared SIRAP and *Overrun* by means of simulation analysis using the same assumptions and settings. The results of this comparison confirmed our conclusion that the efficiency of the mechanisms depends exclusively on the system parameters.

BROE seems to be more efficient than the other two as it does not add direct effect on the local and global schedulability analysis. However, BROE uses the bounded-delay resource model which may affect both the local and global schedulability analysis compared with the periodic resource model used by SIRAP and HSRP. The periodic resource model provides more CPU resources than the bounded-delay resource model. As a result, it may affect the local schedulability analysis and it may require larger subsystem budget compared with the case of using periodic resource model. In the global schedulability analysis, because of using the bounded-delay resource model, it is only possible to use an approximated schedulability analysis with BROE, while for the other protocols, exact schedulability analyses can be applied which give tighter (less pessimistic) results. Considering this effect, there can be some cases when SIRAP or HSRP give better results than BROE and of course the performance also depends on the system parameters.

### 3.5.2 Implementation complexity and overhead

Both SIRAP and HSRP rely on using the periodic server to implement each subsystem, and a server is assigned $Q_s$ budget every period $P_s$. Implementing

the BROE server is done relying on an EDF global scheduler together with a modified version of CBS. Comparing the two types of servers, the implementation of the periodic server is easier and has less runtime overhead than the implementation of the CBS server (CBS has more states). Comparing the SIRAP and HSRP implementations using the periodic server, the implementation of HSRP impose more runtime overhead than SIRAP as it requires to change the behavior of the global scheduler when an overrun occurs. SIRAP does not require any change in the execution of the periodic server during runtime. An implementation of both SIRAP and HSRP has been presented in [32, 33], and the results show that the primitives that are used to implement HSRP impose more runtime overhead than using the SIRAP primitives. In addition, the number of scheduler calls will be higher for HSRP than SIRAP, which increases the runtime overhead. In [34] the implementation of all three protocols including BROE is presented and the results show that BROE imposes the highest runtime overhead compared with the other protocols to solve the problem of budget depletion. The reason for this is that BROE may change the deadline and the replenishment time of the server more often and such operations are relatively expensive, in terms of runtime overhead, as they require to arrange the ready queue of the server. On the other hand, SIRAP requires that the resource holding times of all tasks accessing any global shared resource should be provided during runtime, while the other two protocols may require the maximum resource locking time for each subsystem. This may require more memory space storing the resource holding times, which is one of the disadvantages of SIRAP compared with the other protocols (HSRP and BROE). One way to decrease the memory space for SIRAP is to consider the maximum value of resource holding times per global shared resource or among all global shared resources. However, this should be taken into account in the local schedulability analysis which makes the results of SIRAP less accurate (more pessimistic).

# Chapter 4

# Summary, Conclusions and Future Work

In this thesis we have addressed the problem of supporting global shared resources in the context of hierarchical scheduling. For this purpose, we have presented a novel synchronization protocol called Subsystem Integration and Resource Allocation Policy (SIRAP), which provides temporal isolation between subsystems that share logical resources. We have evaluated the overhead introduced by SIRAP through a simulation study. To decrease this overhead, the results of the study showed that the subsystem period should be chosen as small as possible, while taking into account that the resource holding time may increase the subsystem utilization for small subsystem periods, and the overhead of context-switch will increase when selecting a smaller subsystem period.

In addition, we have extended the analysis of HSRP allowing subsystems to be developed independently. Also, we have proposed a new version of the overrun mechanism that in certain cases performs better than the other two overrun mechanisms proposed by HSRP. We have compared the three versions of overrun mechanisms based on their schedulability analysis and we have shown which parameters have greater effect on the system overhead. For example, to decrease this overhead, I) the resource holding time should be as small as possible, II) the subsystem period should be much greater than the resource holding time, and III) the subsystem period should be less than the smallest internal task period.

Furthermore, we have evaluated and compared the performance of SIRAP

and *Overrun* by means of simulation analysis. The results of the simulation studies showed that in general, it is not trivial to evaluate which protocol is better than the other and the performance is extremely dependent on the subsystem and task parameters. In general, our evaluation shows that the *Skipping* mechanism used by SRIAP can perform better than the two mechanisms of *Overrun* if the task periods are much larger than their corresponding subsystem period. On the other hand, for a high difference between subsystem periods, the version of *Overrun* with payback can give better results. For equal or close to equal subsystem periods, the version of *Overrun* without payback performs better.

Based on the evaluation results, we could identify some sources of pessimism in the analysis of SIRAP and HSRP, and we have proposed tighter and more complex analysis. For SIRAP, we have proposed two different tighter analyses; one based on adding the effect of using SIRAP on the request bound function, and the other is based on adding the effect of using SIRAP on the supply bound function. The simulation studies showed that both proposed analyses can significantly decrease the CPU resource requirement when the number of accesses to a shared resource of a subsystem is high.

For *Overrun* (without payback), we have presented a tighter analysis, more accurately considering the limited preemptions from higher priority subsystems during overrun. In spite of the higher complexity of the new analysis, it can achieve a significant improvement in the CPU resource usage in some cases, especially when the ratio between resource holding time and subsystem period is high, which makes the performance of the traditional *Overrun* without payback very low.

Finally, we have studied the relationship between certain subsystem parameters that have great effect on the performance of the protocols, and we have proposed algorithms and design approaches that manipulate the subsystem parameters in order to improve the performance of the protocols in terms of requiring less CPU resource.

For SIRAP, we have presented an algorithm that finds the best internal ceilings of the global shared resources during the self-blocking state of a subsystem, and thereby minimizing the CPU resource requirement. We have shown through a simulation study that using the algorithm can decrease the overhead of using SIRAP in terms of requiring less CPU resource, specially for subsystems that have high subsystem periods.

For *Overrun* (without payback), we have identified a tradeoff between reducing resource holding time and increasing subsystem budget and its effect on the system load. We have showed that it is not possible to use this trade-

off during the development phase of a subsystem to minimize the system load, without providing the timing parameters of all other subsystems of the system. To effectively explore such tradeoff, we have presented a two step approach, where in the first step, we have proposed an optimal algorithm to generate a set of timing parameters (interfaces) during the development phase of a subsystem. In the second step and during the integration phase of subsystems, we have proposed an algorithm that can select the best interface from each subsystem to minimize the system load of the system.

## 4.1 Conclusions

To conclude, supporting global resource sharing is complex and may significantly increase the overhead when using a synchronization protocol. For instance, when using a synchronization protocol, and if the parameters of the subsystem are not selected carefully, it is not possible to guarantee schedulability of a subsystem even if its corresponding task set utilization is low. It is obvious that decreasing certain parameters can decease the extra overhead of using synchronization protocols, such as decreasing the subsystem period, subsystem budget and resource holding times. However, unfortunately, the relationship between these parameters are so complex that decreasing one parameter may increase another parameter. Moreover selecting the optimal parameter depends on other subsystems. In this thesis we tried to provide some guidelines and methods for the designers of the subsystems to select the best synchronization protocol and the best subsystem parameters to decrease the overhead as much as possible.

### 4.1.1 Discussion

In the following text, we will discuss the applicability of each single contribution that we have presented, and also the possibility of generalizing and combining them.

Starting from the two considered protocols, SIRAP and *Overrun*, SIRAP handles the problem of budget depletion (explained in section 3.2) within the subsystem level. For *Overrun*, the budget depletion problem is considered in the global level by overrunning the given budget of a subsystem. This difference has many consequences on the analysis and implementation of each protocols. For instance, in SIRAP any scheduler in the global level providing SRP (if the global scheduler is of preemptive type) can be used without modifi-

cation. In addition, no communication between the local and global schedulers is required which provides better isolation between local and global schedulers. For *Overrun*, the global scheduler should be informed by the local scheduler when an overrun occurs, and this requires extending the implementation of the global scheduler. Since the effect of using SIRAP is handled locally within the subsystem level, its overhead in terms of requiring extra CPU resources, is added into the local schedulability analysis. This makes it possible for SIRAP to consider the behavior of tasks including the frequency of accessing global shared resources and the critical section execution time, in the local analysis to optimize the local schedulability analysis. This possibility has been used in the tighter analysis presented in paper D. For *Overrun*, and since its overhead should be considered in the global schedulability analysis, the task information should be included in the subsystem timing interface to be able to optimize the global scheduler considering the frequency of accessing global shared resources and the critical section execution time.

To decrease the overhead introduced by the proposed synchronization protocols, we have presented tighter analyses and algorithms that may be used during runtime and/or off-line depending on the type of the system. In general, systems can be classified as static and dynamic systems. Static systems are developed off-line and they do not change during runtime, while for dynamic systems, subsystems and/or tasks may be added or removed during runtime. We refer to the systems that their subsystems can be added or removed as Subsystem Level Dynamic (SLD). Usually, an admission controller is used to verify the possibility of adding a new subsystem without violating the schedulability of the subsystems. The admission controller applies the compositional analysis (global schedulability analysis) whenever a new subsystem is added. If tasks are allowed to be added or removed from a subsystem during runtime, then the systems is called Task Level Dynamic (TLD). Whenever a task is added to a subsystem, an admission controller re-computes the timing interface of the subsystem, and then it applies the compositional analysis.

For static systems all analysis is done off-line. Hence, the presented tighter analyses and algorithms for both SIRAP and *Overrun* (without payback) are applied during the design time of subsystems. For SLD, the subsystem analysis is performed off-line while the compositional analysis is done during runtime. The new analysis and the algorithm presented for SIRAP (in paper D and paper F) are applied at the subsystem level, i.e., they can be applied off-line during the design of the subsystem. On the other hand, the tighter analysis and the algorithm (presented in paper E and paper G) for *Overrun* (without payback) should be done at the system level, i.e., they have to be performed

online. Note that the presented analyses and the algorithms give better result than the original analysis, however, they increase the computation complexity of the analysis during runtime. To decrease the computation complexity during runtime, the admission controller may first apply a simple and less accurate analysis for faster response, and if the system is not composable it may use the more advanced analysis and algorithms. Considering TLD systems, all proposed analysis and algorithms should be applied during runtime.

Furthermore, let us discuss the possibility of combining the tighter analysis and the proposed algorithm for each proposed protocol. For *Overrun*, combining the tighter analysis and the algorithms of the two step approach significantly increases the computation complexity since the computation complexity of evaluating the system load when considering the tighter analysis is relatively high. For SIRAP, combining the presented algorithm and the new analysis may not increase the computation complexity since they do not affect each other.

Finally, considering the possibility of generalizing the two step approach (presented in Paper G) which was proposed to explore the tradeoff between decreasing resource holding time and subsystem budget, using *Overrun* (without payback). A similar tradeoff can be founded in SIRAP, however, because of high dependencies between some parameters used in the analysis of SIRAP (subsystem budget and resource holding time), the algorithms may not be able to find optimal settings. Nevertheless, we could prove in [35] that the two proposed algorithms can be used without modification for BROE as well.

## 4.2 Future work

The work presented in this thesis has left and opened some issues that would be interesting to investigate in the future.

- **Multi-processor**: The work presented in this thesis is suitable only for systems executed on a single processor while multi-processor architectures are becoming more attractive for real-time applications. Recently, there has been some focus on extending the hierarchical scheduling approach to multi-processor platforms [36, 37]. However, the problem of sharing logical resources has not been considered. It would be very interesting to generalize this work by including support for sharing of logical resources. As an initial step in this area, we have proposed a synchronization protocol for hierarchically scheduled multi-core systems in [38]. The presented protocol groups dependent tasks that directly or

indirectly share mutually exclusive resources into independent components. Within a component, dependent tasks use classical uni-processor synchronization protocols, such as SRP. The components are then scheduled on the cores by a global scheduler.

- **Multi-level HSF**: Another interesting work will be investigating the efficiency of using the synchronization protocols in multi-level hierarchical scheduling frameworks, since we only consider a two-level hierarchical scheduling framework. In [39] we present a schedulability analysis algorithm for FTT Ethernet enabled switches. The framework is a multilevel hierarchical scheduling framework and it uses the skipping mechanism to avoid pre-emption during message transmission which makes the analysis similar to the analysis of SIRAP.

- **Resources**: In this thesis, we have only focused on optimizing the CPU resource usage. However, considering other types of resources can be interesting and important, e.g., network, memory, power consumption, etc.

- **Approximation algorithms**: For task level dynamic systems TLD, the subsystem interface parameters should be recalculated during runtime, which may require faster algorithms. An efficient approximation algorithm has been proposed for the HSF [23] which significantly decreases the computation time of the local schedulability test, assuming that tasks are independent. A similar algorithm has been extended to be used with BROE in [40] and a simulation study showed that the pessimism generated from the approximation is very low. It would be interesting to generalize the use of this algorithm with SIRAP and HSRP.

- **Implementation**: Most of the recent implementations of the HSF presented in [32, 33, 41] do not separate the execution of the global scheduler and the local schedulers, and all these schedulers should be supported by the operating system. The motivation behind this way of implementation is that it is easier and more efficient to implement both schedulers together. One of the drawbacks of such an implementation is that the subsystems become platform dependent and if a subsystem uses a special scheduler, then the platform should support it. It would be interesting to implement a platform independent local schedulers within their subsystems and compare this approach with our existing implementations.

- **BROE**: We have compared the performance of SIRAP and *Overrun* by means of simulation analysis. We are planning to extend the comparison to include the BROE server. As a first step, we have adapted the analysis of BROE in [35, 40] to be compatible with our framework and the assumptions that we have considered when we compared between SIRAP and *Overrun*.
- **Subsystem period:** Finding optimal values for $P_s$ and $Q_s$ that minimize the overall processor requirement of the system, without providing information about other subsystems, is very complex and it requires extensive search algorithms [22] even for a hierarchical scheduling framework without resource sharing. Adding the problem of logical resource sharing makes it even more complex. It is an interesting research direction that could be investigated.

# Chapter 5

# Overview of the Papers

## 5.1 Paper A

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

**Contribution** The basic idea of this paper was suggested by Moris Behnam. The work was mainly 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. All authors contributed to the writing

of the paper.

## 5.2 Paper B

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Sjödin, *Overrun Methods and Resource Holding Times for Hierarchical Scheduling of Semi-Independent Real-Time Systems*, IEEE Transactions on Industrial Informatics, vol 6, nr 1, pages 93-104, February, 2010.

**Summary** The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework to enable compositional schedulability analysis of embedded software systems with real-time properties. In this paper a software 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, in the paper, the proposed methods allow non-intrusive configuration and tuning of subsystem timing-behaviour via subsystem interfaces for selecting scheduling parameters. This paper considers three methods to handle overruns due to resource sharing between subsystems in the HSF. For each one of these three overrun methods corresponding scheduling algorithms and associated schedulability analysis are presented together with analysis that shows under what circumstances one or the other is preferred. The analysis is generalized to allow for both Fixed Priority Scheduling (FPS) and Earliest Deadline First (EDF) scheduling. Also, a further contribution of the paper is the technique of calculating resource-holding times within the framework under different scheduling algorithms; the resource holding times being an important parameter in the global schedulability analysis.

**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. All authors contributed to the writing of the paper.

## 5.3 Paper C

Moris Behnam, Thomas Nolte, Mikael Åsberg, Reinder J. Bril, *Overrun and Skipping in Hierarchically Scheduled Real-Time Systems*, In Proceedings of the $15^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), pages 519-526, August, 2009.

**Summary** Recently, two SRP-based synchronization protocols for hierarchically scheduled real-time systems based on Fixed- Priority Preemptive Scheduling (FPPS) have been presented, i.e., HSRP [9] and SIRAP [4]. Preventing depletion of budget during global resource access, the former implements an overrun mechanism, while the latter exploits a skipping mechanism. A theoretical comparison of the performance of these mechanisms revealed that none of them was superior to the other, as their performance is heavily dependent on the systems parameters. To better understand the relative strengths and weaknesses of these mechanisms, this paper presents a comparative evaluation of the depletion prevention mechanisms overrun (with or without payback) and skipping. These mechanisms are investigated in detail and the corresponding system load imposed by these mechanisms is explored in a simulation study. The mechanisms are evaluated assuming FPPS and a periodic resource model [23]. The periodic resource model is selected as it supports locality of schedulability analysis, allowing for a truthful comparison of the mechanisms. Given system characteristics, guiding the design of hierarchically scheduled real-time systems, the results of this paper indicate when one mechanism is better than the other and how a system should be configured in order to operate efficiently.

**Contribution** The paper was based on ideas of Moris and Reinder and Thomas. Moris has done most of the work including simulations and the analysis of the results. All authors contributed to the writing of the paper.

## 5.4 Paper D

Moris Behnam, Thomas Nolte, Reinder J. Bril, *Bounding the number of self-blocking occurrences of SIRAP*, In Proceedings of the $31^{th}$ IEEE International Real-Time Systems Symposium (RTSS'10), December, 2010.

**Summary** In this paper we have developed a new schedulability analysis for hierarchically scheduled real-time systems executing on a single processor

using SIRAP; a synchronization protocol for inter subsystem task synchronization. We have shown that it is possible to bound the number of selfblocking occurrences that should be taken into consideration in the schedulability analysis of subsystems, and correspondingly developed and proved correctness of two novel schedulability analysis approaches for SIRAP. An evaluation suggests that this new schedulability analysis can decrease the analytical subsystem utilization significantly.

**Contribution** The paper was based on ideas of Moris and Reinder. Moris has done most of the work including the schedulability analysis and the evaluation. All authors have contributed to the writing of the paper.

## 5.5 Paper E

Moris Behnam, Thomas Nolte, Reinder J. Bril, *Schedulability Analysis of Synchronization Protocols Based on Overrun Without Payback for Hierarchical Scheduling Frameworks revisited*, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-237/2010-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, October, 2010.

**Summary** In this paper, we show that both global as well as local schedulability analysis of synchronization protocols based on the stack resource protocol (SRP) and overrun without payback for hierarchical scheduling frameworks based on fixed-priority pre-emptive scheduling (FPPS) are pessimistic. We present improved global and local schedulability analysis, illustrate the improvements by means of examples, and show that the improved global analysis is both uniform and sustainable. We evaluate the improved global and local schedulability analysis based on an extensive simulation study and compare the results with the existing analysis.

**Contribution** The paper is based on an idea of Reinder. The analysis was developed by Reinder but Moris was responsible for evaluating the new analysis and finding the parameters that affect the improvement of the new analysis. All authors have contributed to the writing of the paper.

## 5.6 Paper F

Moris Behnam, Thomas Nolte, Reinder J. Bril, *Refining SIRAP with a Dedicated Resource Ceiling for Self-Blocking*, In of the $9^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'09), pages 157-166, October, 2009.

**Summary** In recent years, several synchronization protocols for resource sharing have been presented for use in a Hierarchical Scheduling Framework (HSF). An initial comparative assessment of existing protocols revealed that none of the protocols is superior to the others and that the performance of a protocol heavily depends on system parameters. In this paper, we aim at efficiency improvements of the synchronization protocol SIRAP and its associated schedulability analysis, where efficiency refers to calculated CPU resource needs. The contribution of the paper is threefold. Firstly, we present an improvement of the schedulability analysis for SIRAP, which makes SIRAP more efficient. Secondly, we generalize SIRAP by distinguishing separate resource ceilings for self-blocking and resource access. Using a separate resource ceiling for self-blocking enables a reduction of the interference from lower priority tasks, which can result in efficiency improvements. The efficiency improvement depends on both subsystem characteristics and the value selected for the resource ceiling for self-blocking, however. The third contribution of this paper is therefore an algorithm that given a subsystem selects for each globally shared resource an optimal value in terms of efficiency for its resource ceiling for self-blocking. The efficiency improvement gained by the algorithm compared to the original SIRAP approach is evaluated by means of simulation.

**Contribution** The paper was based on ideas of Moris and Reinder. Moris was responsible for developing the algorithm and evaluating its performance. All authors have contributed to the writing of the paper.

## 5.7 Paper G

Insik Shin, Moris Behnam, Thomas Nolte, Mikael Sjödin, *Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources*, In Proceedings of the $29^{th}$ IEEE International Real-Time Systems Symposium (RTSS'08), pages 209-220, December, 2008.

**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 recourses (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.

**Contribution** The paper was based on ideas of Moris and Insik. Moris was responsible for developing the algorithms and proving their correctness and optimality formally. Moris was also involved in the discussions and witting of the other parts of the paper. All authors have contributed to the writing of the paper.

# Bibliography


[1] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. From a federated to an integrated automotive architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):956–965, 2009.

[2] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[3] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON'87)*, pages 909–916, November 1987.

[4] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, December 1988.

[5] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[6] J.A. Stankovic and K. Ramamritham, editors. *Hard Real-Time Systems Tutorial*. 1988.

[7] 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.

[8] 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.

[9] 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.

[10] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,* $2^{nd}$ *ed.* Springer, 2005.

[11] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[12] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the* $20^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[13] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the* $6^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[14] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the* $21^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[15] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the* $7^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[16] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* $23^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[17] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the* $4^{th}$ *ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[18] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[19] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 26$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[20] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, December 2007.

[21] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 129–138, December 2007.

[22] A. Easwaran. *Compositional Schedulability Analysis Supporting Associativity, Optimality, Dependency and Concurrency.* PhD thesis, Computer and Information Science, 2007.

[23] F. Dewan and N. Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In *Proceedings of the 16$^{th}$ IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*, pages 247–256, April 2010.

[24] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the* 15$^{th}$ *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[25] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[26] 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 7$^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[27] M. Behnam, T. Nolte, and R. J. Bril. Refining SIRAP with a dedicated resource ceiling for self-blocking. In *Proceedings of the $9^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'09)*, pages 157–166, October 2009.

[28] M. Behnam, T. Nolte, and R. Bril. Bounding the number of self-blocking occurrences of SIRAP. In *Proceedings of the $31^{th}$ IEEE international Real-Time Systems Symposium (RTSS'10)*, December 2010.

[29] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the $28^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[30] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the* $19^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'98)*, pages 4–13, December 1998.

[31] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization' in reservation-based real-time systems. *Transactions on Computers*, 53(12):1591–1601, December 2004.

[32] M. Åsberg, M. Behnam, T. Nolte, and R. J. Bril. Implementation of overrun and skipping in vxworks. In *Proceedings of the $6^{th}$ International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'10)*, pages 45–52, July 2010.

[33] M. van den Heuvel, R. Bril, and M. Behnam. Extending an hsf-enabled open source real-time operating system with resource sharing. In *Proceedings of the $6^{th}$ International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'10)*, pages 71–81, July 2010.

[34] M. van den Heuvel, R. Bril, and J. Lukkien. Protocol-transparent resource sharing in hierarchically scheduled real-time systems. In *Proceedings of the $15^{th}$ International Conference on Emerging Technologies and Factory Automation (ETFA'10)*, September 2010.

[35] M. Behnam, T. Nolte, and N. Fisher. On optimal real-time subsystem-interface generation in the presence of shared resources. In *Proceedings of the $15^{th}$ International Conference on Emerging Technologies and Factory Automation (ETFA'10)*, September 2010.

[36] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the 20$^{th}$ Euromicro Conference on Real-Time Systems conference (ECRTS'08)*, pages 181–190, July 2008.

[37] E. Bini, G. Buttazzo, and M. Bertogna. The Multi Supply Function Abstraction for Multiprocessors. In *Proceedings of the 15$^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, pages 294–302, August 2009.

[38] F. Nemati, M. Behnam, and T. Nolte. Multiprocessor synchronization and hierarchical scheduling. In *Proceedings of the 1$^{st}$ IEEE International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice (XRTS'2009) in conjunction with (ICPP'09)*, pages 58–64, September 2009.

[39] R. Santos, M. Behnam, T. Nolte, L. Almeida, and P. Pedreiras. Schedulability analysis for multi-level hierarchical server composition in ethernet switches. In *Proceedings of the 9$^{th}$ International Workshop on Real-Time Networks (RTN'2010)*, pages 44–49, July 2010.

[40] M. Behnam and N. Fisher. Subsystem-interface generation in the presence of shared resources. In *Proceedings of the 2$^{nd}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09)*, pages 16–23, December 2009.

[41] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R.J. Bril. Towards hierarchical scheduling in VxWorks. In *Proceedings of the 4$^{th}$ International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'08 )*, pages 63–72, July 2008.

# II

# Included Papers

# Chapter 6

# Paper A: SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin

In Proceedings of the $7^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07), pages 279-288, October, 2007.

**Abstract**

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.

## 6.1 Introduction

In many industrial sectors integration of electronic and software subsystems (to form an integrated hardware and software system), is one of the activities that is most difficult, time consuming, and error prone [1, 2]. Almost any system, with some level of complexity, is today developed as a set of semi-independent subsystems. For example, cars consist of multiple subsystems such as antilock braking systems, airbag systems and engine control systems. In the later development stages, these subsystems are integrated to produce the final product. Product domains where this approach is the norm include automotive, aerospace, automation and consumer electronics.

It is not uncommon that these subsystems are more or less dependent on each other, introducing complications when subsystems are to be integrated. This is especially apparent when integrating multiple software subsystems on a single processor. Due to these difficulties inherent in the integration process, many projects run over their estimated budget and deadlines during the integration phase. Here, a large source of problems when integrating real-time systems stems from subsystem interference in the time domain.

To provide remedy to these problems we propose the usage of a real-time scheduling framework that allows for an easier integration process. The framework will preserve the essential temporal properties of the subsystem both when the subsystem is executed in isolation (unit testing) and when it is integrated together with other subsystems (integration testing and deployment). Most importantly, the deviation in the temporal behaviour will be bounded, hence allowing for predictable integration of hard real-time subsystems. This is traditionally targeted by the philosophy of open systems [3], allowing for the independent development and validation of subsystems, preserving validated properties also after integration on a common platform.

In this paper we present the Subsystem Integration and Resource Allocation Policy (SIRAP), which makes it possible to develop subsystems individually without knowledge of the temporal behaviour of other subsystems. One key issue addressed by SIRAP is the resource sharing between subsystems that are only semi-independent, i.e., they use one or more shared logical resources.

**Problem description** A software system $\mathcal{S}$ consists of one or more subsystems to be executed on one single processor. Each subsystem $S_s \in \mathcal{S}$, in turn, consists of a number of tasks. These subsystems can be developed independently and they have their own local scheduler (scheduling the subsystem's tasks). This approach by isolation of tasks within subsystems, and allowing for

their own local scheduler, has several advantages [4]. For example, by keeping a subsystem isolated from other subsystems, and by keeping the subsystem local scheduler, it is possible to re-use a complete subsystem in a different application from where it was originally developed.

However, as subsystems are likely to share logical resources, an appropriate resource sharing protocol must be used. In order to facilitate independent subsystem development, this protocol should not require information from all other subsystems in the system. It should be enough with only the information of the subsystem under development in isolation.

**Contributions** The main contributions of this paper include the presentation of SIRAP, a novel approach to subsystem integration in the presence of shared resources. Moreover, the paper presents the deduction of bounds on the timing behaviour of SIRAP together with accompanying formal proofs. In addition, the cost of using this protocol is thoroughly evaluated. The cost is investigated as a function of various parameters including: cost as a function of the length of critical sections, cost depending on the priority of the task sharing a resource, and cost depending on the periodicity of the subsystem. Finally, the cost of having an independent subsystem abstraction, which is suitable for open systems, is investigated and compared with dependent abstractions.

**Organization of the paper** Firstly, related work on hierarchical scheduling and resource sharing is presented in Section 6.2. Then, the system model is presented in Section 6.3. SIRAP is presented in Section 6.4. In Section 6.5 schedulability analysis is presented, and SIRAP is evaluated in Section 6.6. Finally, the paper is summarized in Section 6.7.

## 6.2 Related work

**Hierarchical scheduling** For real-time systems, there has been a growing attention to hierarchical scheduling frameworks [5, 6, 3, 7, 8, 9, 10, 11, 12, 13, 14].

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 [8] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [9, 15] presented schedulability analysis techniques for the EDF-based global schedulers.

Mok *et al.* [16] proposed the bounded-delay resource partition model for a hierarchical scheduling framework. Their model can specify the real-time guarantees that a parent component provides to its child components, where the parent and child components have different schedulers. Feng and Mok [7] and Shin and Lee [14] presented schedulability analysis techniques for the hierarchical scheduling framework that employs the bounded-delay resource partition model.

There have been studies on the schedulability analysis with the periodic resource model. This periodic resource model can specify the periodic resource allocation guarantees provided to a component from its parent component [13]. Saewong *et al.* [12] and Lipari and Bini [10] introduced schedulability conditions for fixed-priority local scheduling, and Shin and Lee [13] presented a schedulability condition for EDF local scheduling. Davis and Burns [6] evaluated different periodic servers (Polling, Deferrable, and Sporadic Servers) for fixed-priority local scheduling.

**Resource sharing** When several tasks are sharing a logical resource, typically only one task is allowed to use the resource at a time. Thus the logical resource requires mutual exclusion of tasks that uses it. To achieve this a *mutual exclusion protocol* is used. The protocol provides rules about how to gain access to the resource, and specifies which tasks should be blocked when trying to access the resource.

To achieve predictable real-time behaviour, several protocols have been proposed including the Priority Inheritance Protocol (PIP) [17], the Priority Ceiling Protocol (PCP) [18], and the Stack Resource Policy (SRP) [19].

When using SRP, a task may not preempt any other tasks until its priority is the highest among all tasks that are ready to run, and its preemption level is higher than the system ceiling. The preemption level of a task is a static parameter assigned to the task at its creation, and associated with all instances of that task. A task can only preempt another task if its preemption level is higher than the task that it is to preempt. Each resource in the system is associated with a resource ceiling and based on these resource ceilings, a system ceiling can be calculated. The system ceiling is a dynamic parameter that changes during system execution.

The duration of time that a task lock a resource, is called Resource Holding Time (RHT). Fisher *et al*. [20, 21] proposed algorithms to minimize RHT for fixed priority and EDF scheduling with SRP as a resource synchronization protocol. The basic idea of their proposed algorithms is to increase the ceiling of resources as much as possible without violating the schedulability of the

system under the same semantics of SRP.

Deng and Liu [3] proposed the usage of non-preemptive global resource access, which bounds the maximum blocking time that a task might be subject to. The work by Kuo and Li [8] used SRP and they showed that it is very suitable for sharing of local resources in a hierarchical scheduling framework. Almeida and Pedreiras [5] considered the issue of supporting mutually exclusive resource sharing within a subsystem. Matic and Henzinger [11] considered supporting interacting tasks with data dependency within a subsystem and between subsystems, respectively.

More recently, Davis and Burns [22] presented the Hierarchical Stack Resource Policy (HSRP), allowing their work on hierarchical scheduling [6] to be extended with sharing of logical resources. However, using HSRP, information on all tasks in the system must be available at the time of subsystem integration, which is not suitable for an open systems development environment, and this can be avoided by the SIRAP protocol presented in this paper.

## 6.3 System model

### 6.3.1 Hierarchical scheduling framework

A hierarchical scheduling framework is introduced to support CPU time sharing among applications (subsystems) under different scheduling services. Hence, a system $\mathcal{S}$ consists of one or more subsystems $S_s \in \mathcal{S}$. The hierarchical scheduling framework can be generally represented as a two-level tree of nodes, where each node represents a subsystem with its own scheduler for scheduling internal tasks (threads), and CPU time is allocated from a parent node to its children nodes, as illustrated in Figure 6.1.

The hierarchical scheduling framework provides *partitioning* of the CPU between different subsystems. Thus, subsystems can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification and unit testing.

The hierarchical scheduling framework is also useful in the domain of open systems [3], where subsystems may be developed and validated independently in different environments. For example, the hierarchical scheduling framework allows a subsystem to be developed with its own scheduling algorithm internal to the subsystem and then later included in a system that has a different global level scheduler for scheduling subsystems.

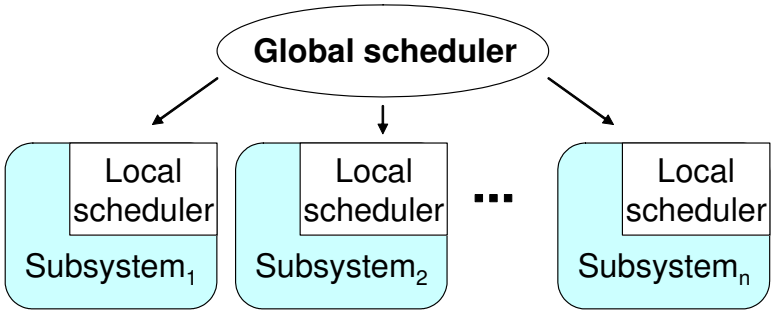


Figure 6.1: Two-level hierarchical scheduling framework.

### 6.3.2 Shared resources

For the purpose of this paper a shared (logical) resource, $r_i$, is a shared memory area to which only one task at a time may have access. To access the resource 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*. Only one task at a time may lock each 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*. In this paper we are concerned only with global shared resources and will simply denote them by shared resources. Management of local shared resources can be done by using any synchronization protocol such as PIP, PCP, and SRP.

### 6.3.3 Virtual processor model

The notion of real-time virtual processor (resource) model was first introduced Mok *et al.* [16] 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 amounts 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 [13] proposed the periodic virtual processor model $\Gamma(\Pi, \Theta)$, where $\Pi$ is a period ($\Pi > 0$) and $\Theta$ is a periodic allocation time ($0 < \Theta \leq \Pi$). The capacity $U_\Gamma$ of a periodic virtual processor model $\Gamma(\Pi, \Theta)$ is defined as $\Theta/\Pi$. The periodic virtual processor model $\Gamma(\Pi, \Theta)$ is defined to characterize the following property:

$$\text{supply}_\Gamma\big(k\Pi, (k+1)\Pi\big) = \Theta, \quad \text{where } k = 0, 1, 2, \ldots, \tag{6.1}$$

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 $[t_1, t_2)$.

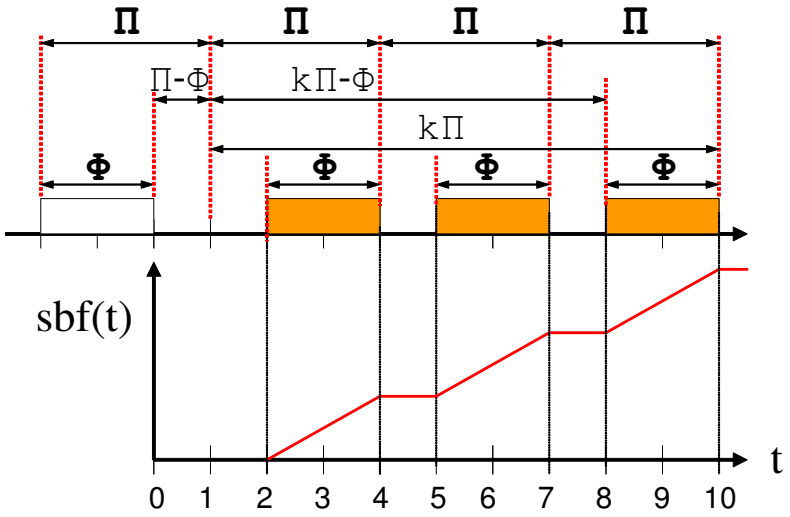


Figure 6.2: The supply bound function of a periodic virtual processor model $\Gamma(\Pi, \Theta)$ for $k = 3$.

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

$$\texttt{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(\Pi - \Theta) & \text{if } t \in [(k+1)\Pi - 2\Theta, \\ & (k+1)\Pi - \Theta], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{6.2}$$

where $k = \max\Big(\big\lceil\big(t - (\Pi - \Theta)\big)/\Pi\big\rceil, 1\Big)$. Here, we first note that an interval of length $t$ may not begin synchronously with the beginning of period $\Pi$. That is, as shown in Figure 6.2, the interval of length $t$ can start in the middle of the period of a periodic model $\Gamma(\Pi, \Theta)$. We also note that the intuition of $k$ in Eq. (6.2) basically indicates how many periods of a periodic model can overlap the interval of length $t$, more precisely speaking, the interval of length $t - (\Pi - \Theta)$. Figure 6.2 illustrates the intuition of $k$ and how the supply bound function $\mathtt{sbf}_\Gamma(t)$ is defined for $k = 3$.

### 6.3.4 Subsystem model

A subsystem $S_s \in \mathcal{S}$, where $\mathcal{S}$ is the whole system of subsystems, consists of a task set and a scheduler. Each subsystem $S_s$ is associated with a periodic virtual processor model abstraction $\Gamma_s(\Pi_s, \Theta_s)$, where $\Pi_s$ and $\Theta_s$ are the subsystem period and budget respectively. This abstraction $\Gamma_s(\Pi_s, \Theta_s)$ is supposed to specify the collective temporal requirements of a subsystem, in the presence of global logical resource sharing.

**Task model** We consider a periodic task model $\tau_i(T_i, C_i, \mathcal{X}_i)$, where $T_i$ and $C_i$ represent the task's period and worst-case execution time (WCET) respectively, and $\mathcal{X}_i$ is the set of WCETs within critical sections belonging to $\tau_i$. Each element $x_{i,j}$ in $\mathcal{X}_i$ represents the WCET of a particular critical section $cx_{i,j}$ executed by $\tau_i$. Note that $C_i$ includes all $x_{i,j} \in \mathcal{X}_i$.

The set of critical sections cover for the following two cases of multiple critical sections within one job:

1. sequential critical sections, where $\mathcal{X}_i$ contains the WCETs of all sequential critical sections, i.e. $\mathcal{X}_i = \{x_{i,1}, ..., x_{i,o}\}$ where $o$ is the number of sequential shared resources that task $\tau_i$ may lock during its execution.

2. nested critical sections, where $x_{i,j} \in \mathcal{X}$ being the length of the outer critical section.

Note that in the remaining paper, we use $x_i$ rather than $x_{i,j}$ for simplicity when it is not necessary to indicate $j$.

**Scheduler** In this paper, we assume that each subsystem has a fixed-priority preemptive scheduler for scheduling its internal tasks.

## 6.4 SIRAP protocol

### 6.4.1 Terminology

Before describing the SIRAP protocol, we define the terminology (also depicted in Figure 6.3) that are related to hierarchical logical resource sharing.

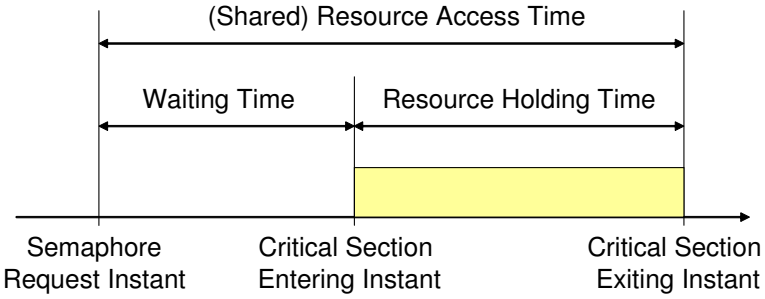


Figure 6.3: Shared resource access time.

- *Semaphore request instant:* an instant at which a job tries to enter a critical section guarded by a semaphore.
- *Critical section entering (exiting) instant:* an instant at which a job enters (exits) a critical section.
- *Waiting time:* a duration from a semaphore request time to a critical section entering time.
- *Resource holding time:* a duration from a critical section entering instant to a critical section exiting instant. Let $h_{i,j}$ denote the resource holding time of a critical section $cx_{i,j}$ of task $\tau_i$.
- *(Shared) resource access time:* a duration from a semaphore request instant to a critical section exiting time.

In addition, a context switch is referred to as *task-level context switch* if it happens between tasks within a subsystem, or as *subsystem-level context switch* if it happens between subsystems.

### 6.4.2 SIRAP protocol description

The subject of this paper is to develop a synchronization protocol that can address global resource sharing in hierarchical real-time scheduling frameworks, while aiming at supporting independent subsystem development and validation. This section describes our proposed synchronization protocol, SIRAP (Subsystem Integration and Resource Allocation Policy).

**Assumption** SIRAP relies on the following assumption:

- The system's global scheduler schedules subsystems according to their periodic virtual processor abstractions $\Gamma_s(\Pi_s, \Theta_s)$. The subsystem budget is consumed every time when an internal task within a subsystem executes, and the budget is replenished to $\Theta_s$ every subsystem period $\Pi_s$. Similar to traditional server-based scheduling methods [23], the system provides a run-time mechanism such that each subsystem is able to figure out at any time $t$ how much its remaining subsystem budget $\Theta_s$ is, which will be denoted as $\Theta'_s(t)$ in the remaining of this section.

The above assumption is necessary to allow run-time checking whether or not a job can potentially enter and execute a whole critical section before a subsystem-budget expire. This is useful particularly for supporting independent abstraction of subsystem's temporal behavior in the presence of global resource accesses.

In addition to supporting independent subsystem development, SIRAP also aims at minimizing the resource holding time and bounding the waiting time at the same time. To achieve this goal, the protocol has two key rules as follows:

R1 When a job enters a critical section, preemptions from other jobs within the same subsystem should be bounded to keep its resource holding time as small as possible.

R2 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.

The first rule R1 aims at minimizing a resource holding time so that the waiting time of other jobs, which want to lock the same resource, can be minimized as well. The second rule R2 prevents a job $J_i$ from entering a critical section $cx_{i,j}$ at any time $t$ when $\Theta'(t) < h_{i,j}$. This rule guarantees that when the budget of a subsystem expires, no task within the subsystem locks a global shared resource.

**SIRAP : preemption management** The SRP [19] is used to enforce the first rule R1. Each subsystem will have its own system ceiling and resources ceiling according to its jobs that share global resources. According to SRP, whenever a job locks a resource, other jobs within the same subsystem can preempt it if the jobs have higher preemption levels than the locked resource ceiling, so as to bound the blocking time of higher-priority jobs. However, such task-level preemptions generally increase resource holding times and can potentially increase subsystem utilization. One approach to minimize $h_{i,j}$ is to allow no task-level preemptions, by assigning the ceiling of global resource equal to the maximum preemption level. However, increasing the resource ceiling to the maximum preemption level may affect the schedulability of a subsystem. A good approach is presented in [20], which increases the ceiling of shared global resources as much as possible while keeping the schedulability of the subsystem.

**SIRAP : self-blocking** When a job $J_i$ tries to enter a critical section, SIRAP requires each local scheduler to perform the following action. Let $t_0$ denote the semaphore request instant of $J_i$ and $\Theta'(t_0)$ denote the subsystem's budget at time $t_0$.

- If $h_{i,j} \leq \Theta'(t_0)$, the local scheduler executes the job $J_i$. The job $J_i$ enters a critical section at time $t_0$.
- Otherwise, i.e., if $h_{i,j} > \Theta'(t_0)$, the local scheduler delays the critical section entering of the job $J_i$ until the next subsystem budget replenishment. This is defined as *self-blocking*. Note that the system ceiling will be equal to resource ceiling at time $t_0$, which means that the jobs that have preemption level greater than system ceiling can only execute during the self blocking interval[1]. This guarantees that when the subsystem of $J_i$ receives the next resource allocation, the subsystem-budget will be enough to execute job $J_i$ inside the critical section[2].

[1] With simple modifications to the SRP protocol, the execution of tasks can be allowed within the self blocking interval if they do not access global resources even though their preemption levels are less than the system ceiling. However this is off the point of this paper.

[2] The idea of self-blocking has been also considered in different contexts, for example, in CBS-R [23] and zone based protocol (ZB) [24]. Our work is different from those in the sense that CBS-R used a similar idea for supporting soft real-time tasks, and ZB used it in a pfair-scheduling environment, while we use it for hard real-time tasks under hierarchical scheduling. This difference inherently requires the development of different schedulability analysis, including Eqs. (6.5), (6.6), and (6.7).

## 6.5 Schedulability analysis

### 6.5.1 Local schedulability analysis

Consider a subsystem $S_s$ that consists of a periodic task set and a fixed-priority scheduler and receives CPU allocations from a virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$. According to [13], this subsystem is schedulable if

$$\forall \tau_i, 0 < \exists t \leq T_i \;\; \mathtt{dbf}_{\mathtt{FP}}(i,t) \leq \mathtt{sbf}_{\Gamma}(t). \tag{6.3}$$

The goal of this section is to develop the demand bound function $\mathtt{dbf}_{\mathtt{FP}}(i,t)$ calculation for the SIRAP protocol. $\mathtt{dbf}_{\mathtt{FP}}(i,t)$ is computed as follows;

$$\mathtt{dbf}_{\mathtt{FP}}(i,t) = C_i + I_S(i) + I_H(i,t) + I_L(i), \tag{6.4}$$

where $C_i$ is the WCET of $\tau_i$, $I_S(i)$ is the maximum self blocking for $\tau_i$, $I_H(i,t)$ is the maximum possible interference imposed by a set of higher-priority tasks to a task $\tau_i$ during an interval of length $t$, and $I_L(i)$ is the maximum possible interference imposed by a set of lower-priority tasks that share resources with preemption level (ceiling) greater than or equal to the priority of task $\tau_i$.

The following lemmas shows how to compute $I_S(i)$, $I_H(i,t)$ and $I_L(i)$.

**Lemma 1.** *Self-blocking imposes to a job $J_i$ an extra processor demand of at most $\sum_{j=1}^{o} h_{i,j}$ if a job access multiple shared resources.*

*Proof.* When the job $J_i$ self-blocks itself, it consumes the processor of at most $h_{i,j}$ units being idle. If the job access shared resources then the worst case will happen when the job block itself whenever it tries to enter a critical section. □

**Lemma 2.** *A job $J_i$ can be interfered by a higher-priority job $J_j$ that access shared resources, at $t$ time units for a duration of at most $\lceil \frac{t}{T_j} \rceil (C_j + \sum_{k=1}^{o} h_{j,k})$ time units.*

*Proof.* Similar to classical response time analysis [25], we add $\sum_{k=1}^{o} h_{j,k}$ to $C_j$ which is the worst case self blocking from higher priority tasks, the lemma follows. □

**Lemma 3.** *A job $J_i$ can be interfered by only one lower-priority job $J_j$ by at most $2 \cdot max(h_{j,k})$, where k=1,...,o.*

*Proof.* A higher-priority job $J_i$ can be interfered by a lower-priority job $J_j$. This occurs only if $J_i$ is released after $J_j$ tries to enter a critical section but before $J_j$ exits the critical section. When $J_i$ is released, only one job can try to enter or be inside a critical section. That is, a higher-priority job $J_i$ can then be interfered by at most a single lower-priority job. The processor demand of $J_j$ during a critical section period is bounded by $2 \cdot max(h_{j,k})$ for the worst case. The lemma follows. □

From Lemma 1, the self-blocking $I_S(i)$ is given by;

$$I_S(i) = \sum_{k=1}^{o} h_{i,k} \tag{6.5}$$

According to Lemma 2 and taking into account the interference from higher priority tasks, $I_H(i,t)$ is computed as follows;

$$I_H(i,t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil (C_j + \sum_{k=1}^{o} h_{j,k}). \tag{6.6}$$

The maximum interference from lower priority tasks can be evaluated according to Lemma 3 according to;

$$I_L(i) = \max_{j=i+1,\ldots,n} (2 \cdot \max_{k=1,\ldots,o} (h_{j,k})). \tag{6.7}$$

Based on Eq. (6.5) and (6.6) and (6.7), the processor demand bound function is given by Eq. (6.4).

The resource holding time $h_{i,j}$ of a job $J_i$ that access a global resource is evaluated as the maximum critical section execution time $x_{i,j}+$ the maximum interference from the tasks that have preemption level greater than the ceiling of the logical resource during the execution $x_{i,j}$. $h_{i,j}$ is computed [20] using $W_{i,j}(t)$ as follows;

$$W_{i,j}(t) = x_{i,j} + \sum_{l=ceil(x_{i,j})+1}^{u} \lceil \frac{t}{T_l} \rceil C_l, \tag{6.8}$$

where $ceil(x_{i,j})$ is the ceiling of the logical resource accessed within the critical section $x_{i,j}$, and $C_l$, $T_l$ are the worst case execution time and the period of job that have higher preemption level than $ceil(x_{i,j})$, and $u$ is the maximum ceiling within the subsystem.

The resource holding time $h_{i,j}$ is the smallest time $t_i^*$ such that $W_{i,j}(t_i^*) = t_i^*$.

### 6.5.2 Global schedulability analysis

Here, issues for global scheduling of multiple subsystems are dealt with. For a subsystem $S_s$, it is possible to derive a periodic virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$ that guarantees the schedulability of the subsystem $S_s$ according to Eq. (6.3).

The local schedulability analysis presented for subsystems is not dependent on any specific global scheduling policy. The requirements for the global scheduler, are as follows: i) it should schedule all subsystems according to their virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$, ii) it should be able to bound the waiting time of a task in any subsystem that wants to access global resource.

To achieve those global scheduling requirements, preemptive schedulers such as EDF and RM together with the SRP [19] synchronization protocol can be used. So when a subsystem locks a global resource, it will not be preempted by other subsystems that have preemption level less than or equal to the locked resource ceiling. Each subsystem, for all global resources accessed by tasks within a subsystem, should specify a list of pairs of all those global resources and their maximum resource holding times $\{(r_1, H_{r_1}), ..., (r_p, H_{r_p})\}$. However it is possible to minimize the required information that should be provided for each subsystem by assuming that all global resources have the same ceiling equal to the maximum preemption level $\hat{\pi}_s$ among all subsystems. Then for the global scheduling, it is enough to provide virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$ and the maximum resource holding times among all global resources $\hat{H}_s = max(H_{R_1}, ..., H_{R_p})$ for each subsystem $S_s$. On the other hand, assigning the ceiling of all global resources to the maximum preemption level of the subsystem that access these resources is not as efficient as using the original SRP protocol, this since we may have resources with lower ceiling which permit more preemptions from the higher preemption level subsystems.

Under EDF global scheduling, a set of $n$ subsystems is schedulable [19] if

$$\forall k_{k=1,...,n}(\sum_{i=1}^{k} \frac{\Theta_i}{\Pi_i}) + \frac{B_k}{\Pi_k} \leq 1, \tag{6.9}$$

where $B_k$ of subsystem $S_k$ is the duration of the longest resource holding time among those belonging to subsystems with preemption level lower than $\pi_k$.

For RM global scheduling, the schedulability test based on tasks' response time is

$$W_i = \Theta_i + B_k + \sum_{j=1}^{i-1} \left\lceil \frac{W_i}{\Pi_j} \right\rceil (C_j). \tag{6.10}$$

It is also possible to use a non-preemptive global scheduler together with the SIRAP protocol. In this case, no subsystem-level context switch happens when there is a task inside a critical section. That is, whenever a task tries to lock a global resource, it is guaranteed that the global resource is not locked by another task from other subsystems. This way provides a clean separation between subsystems in accessing global shared resources. Then, we can achieve a more subsystem abstraction, i.e., subsystems do not have to export information about their global shared resource accesses, for example, which global shared resources they access and the maximum resource holding time. In fact, it will require more system resources to schedule subsystems under non-preemptive global scheduling rather than under preemptive global scheduling. Hence, we can see a tradeoff between abstraction and efficiency. Exploring this tradeoff is a topic of our future work.

### 6.5.3 Local resource sharing

So far, only the problem of sharing global resource between subsystems has been considered. However, many real time applications may have local resource sharing within subsystem as well. Almeida and Pedreiras [5] showed that some traditional synchronization protocols such as PCP and SRP can be used for supporting local resource sharing in a hierarchical scheduling framework by including the effect of local resource sharing in the calculation of $\texttt{dbf}_{\texttt{FP}}$. That is, to combine SRP/PCP and the SIRAP protocol for synchronizing both local and global resources sharing, Eq. (6.7) should be modified to

$$I_L(i) = \max(\max(2 \cdot x_{j,k}), b_i), \quad \text{where } j = i+1, \ldots, n. \tag{6.11}$$

where $b_i$ is the maximum duration for which a task $i$ can be blocked by its lower-priority tasks in critical sections from local resource sharing.

## 6.6 Protocol evaluation

In this section, the cost of using SIRAP is investigated in terms of extra CPU utilization ($U_\Gamma$) required for subsystem schedulability guarantees. We assume

that all global resource ceilings can be equal to the maximum preemption level, which means that no tasks within a subsystem preempt a task inside a critical section, and therefore $h_{i,j} = x_{i,j}$. Supporting logical resource sharing is expected to increase subsystem utilizations $U_\Gamma$. This increment in $U_\Gamma$ depends on many factors such as the maximum WCET within a critical section $x_{i,j}$, the priority of the task sharing a global resource, and the subsystem period $\Pi_s$.

Sections 6.6.1, 6.6.2, and 6.6.3 investigate the effect of those factors under the assumption that task $i$ accesses a single critical section. In Section 6.6.4, this assumption is relaxed so as to investigate the effect of the number of critical sections. Section 6.5 compares independent and dependent abstractions in terms of subsystem utilization.

### 6.6.1 WCET within critical section

One of the main factors that affect the cost of using SIRAP is the value of $x_{i,j}$. It is clear from Eqs. (6.4), (6.6), and (6.7) that whenever $x_{i,j}$ (which equals to $h_{i,j}$) increases, $\texttt{dbf}_{\texttt{FP}}$ will increase as well, potentially causing $U_\Gamma$ to increase in order to satisfy the condition in Eq. (6.3). Figure 6.4 shows the effect of increasing $x_i$ on two different task sets. Task set 1 is sensitive for small changes in $x_i$ whilst task set 2 can tolerate the given range of $x_i$ without showing a big change in $U_\Gamma$. The reason behind the difference is that task set 1 has a task with period very close to $\Pi_s$ while the smallest task period in task set 2 is greater than $\Pi_s$ by more than 4 times. Hence, SIRAP can be more or less sensitive to $x_i$ depending on the ratio between task and subsystem period.

For the remaining figures (Figure 6.5 and 6.6), simulations are performed as follows. We randomly generated 100 task sets, each containing 5 tasks. Each task set has a utilization of 25%, and the period of the generated tasks range from 40 to 1000. For each task set, a single task accesses a global shared resource; the task is the highest priority task, the middle priority task, or the lowest priority task. For each task set, we use 11 different values of $x_i$ ranging from 10% to 50% of the subsystem period.

### 6.6.2 Task priority

From Eqs. (6.4), (6.6) and (6.7), looking how tasks sharing global logical resources affect the calculations of $\texttt{dbf}_{\texttt{FP}}$, it is clear that task priority for these tasks is of importance. The contribution of low priority tasks on $\texttt{dbf}_{\texttt{FP}}$ is fixed to a specific value of $x_i$ (see Eq. (6.7)), while the increase in $\texttt{dbf}_{\texttt{FP}}$ by higher priority tasks depends on many terms such as higher priority task period $T_k$ and

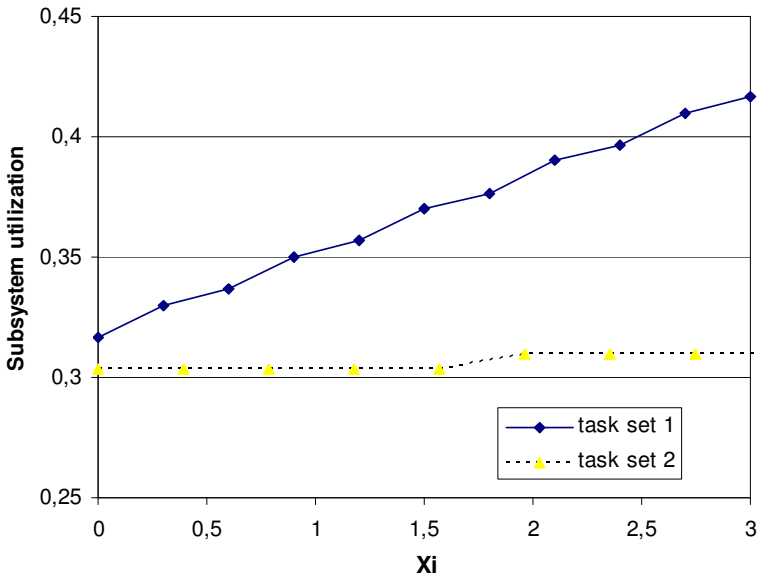


Figure 6.4: $U_\Gamma$ as a function of $x_i$ for two task sets where only the lowest priority task share a resource.

execution time $C_k$ (see Eq. (6.6)). It is fairly easy to estimate the behaviour of a subsystem when lower priority tasks share global resources; on one hand, if the smallest task period in a subsystem is close to $\Pi_s$, $U_\Gamma$ will be significantly increased even for small values of $x_i$. As the value of `sbf` is small for time intervals close to $\Pi_s$, the subsystem needs a lot of extra resources in order to fulfil subsystem schedulability. On the other hand, if the smallest task period is much larger than $\Pi_s$ then $U_\Gamma$ will only be affected for large values of $x_i$, as shown in Figure 6.4.

Figure 6.5 shows $U_\Gamma$ as a function of $x_i$ for when the highest, middle and lowest priority task are sharing global resources, respectively, where $\Pi_s = 15$. The figure shows that the highest priority task accessing a global shared resource needs in average more utilization than other tasks with lower priority. This observation is expected as the interference from higher priority task is larger than the interference from lower priority tasks (see Eq. (6.6) and (6.7)). However, note that in the figure this is true for $x_i$ within the range of [0,5]. If

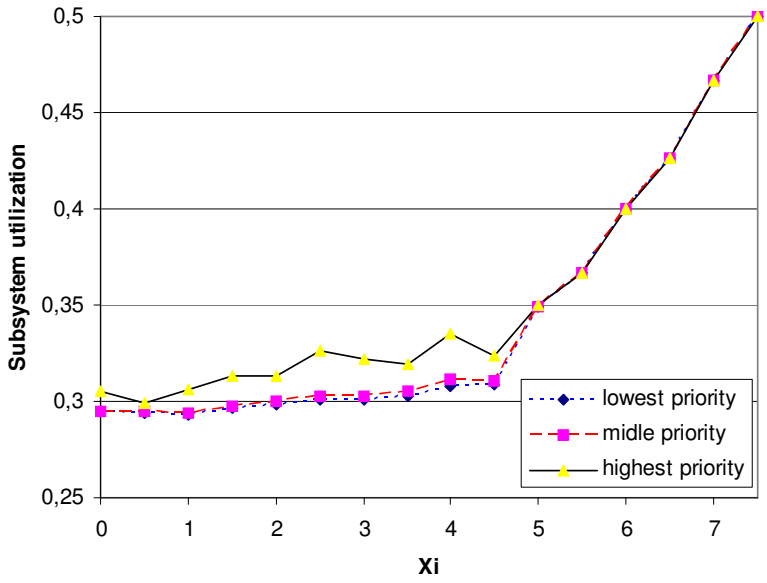


Figure 6.5: Average utilization for 100 task sets as a function of $x_i$, when low, medium and high priority task share a resource respectively, $\Pi_s = 15$.

the value of $x_i$ is larger than 5, then $U_\Gamma$ keeps increasing rapidly without any difference among the priorities of tasks accessing the global shared resource. This can be explained as follows. When using SIRAP, the subsystem budget $\Theta_s$ should be no smaller than $x_i$ to enforce the second rule R2 in Section 6.4.2. Therefore, when $x_i \geq 5$, $\Theta_s$ should also become greater than 5 even though subsystem period is fixed to 15. This essentially results in a rapid increase of $U_\Gamma$ with the speed of $x_i/15$.

### 6.6.3 Subsystem period

The subsystem period is one of the most important parameters, both in the context of global scheduling and `sbf` calculations for a subsystem. As $\Pi_s$ is used in the `sbf` calculations, $\Pi_s$ will have significant effect on $U_\Gamma$ (see Eq. (6.3)).

Figure 6.6 compares average subsystem utilization for different values of subsystem period, i.e., for $\Pi_s = 20$ and $\Pi_s = 40$ for the same task sets. Here,

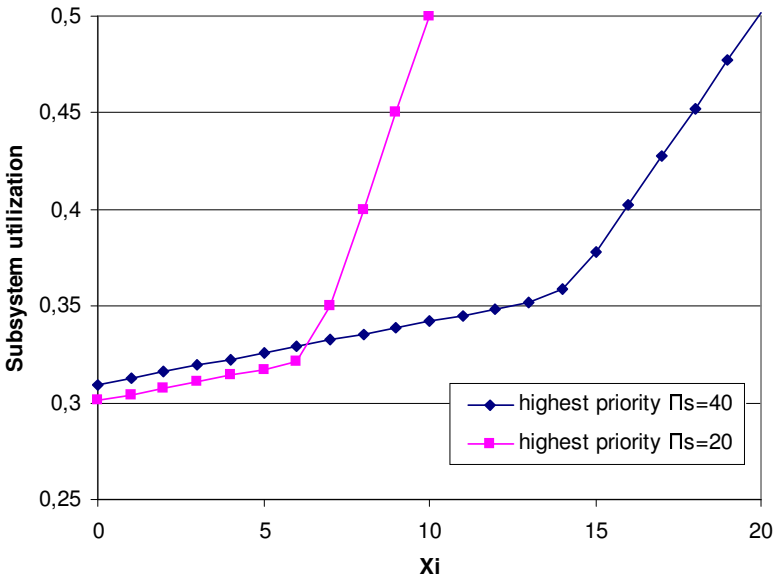


Figure 6.6: Average utilization for 100 task sets as a function of $x_i$, when only the highest priority tasks share a resource and the subsystem period is $\Pi_s = 20$ and $\Pi_s = 40$.

only the highest priority task accesses a global shared resource. It is interesting to see that the lower value of $\Pi_s$, i.e, $\Pi_s = 20$, results in a lower subsystem utilization when $x_i$ is small, i.e., $x_i \leq 6$, and then a higher subsystem utilization when $x_i$ gets larger from $x_i = 6$. That is, $x_i$ and $\Pi_s$ are not dominating factors one to another, but they collectively affect subsystem utilization. It is also interesting to see in Figure 6.6 that the subsystem utilization of $\Pi_s = 40$ behaves in a similar way by increasing rapidly from $x_i = 14$.

Hence, in general, $\Pi_s$ should be less than the smallest task period in a subsystem, as in hierarchical scheduling without resource sharing, the lower value of $\Pi_s$ gives better results (needs less utilization). However, in the presence of global resources sharing, the selection of the subsystem period depends also on the maximum value of $x_i$ in the subsystem.

### 6.6.4 Multiple critical sections

We compare the case when a task $i$ accesses multiple critical sections (MCS) with the case when a task $j$ accesses a single critical section (SCS) within duration $x_j = \sum_{k=1}^{o} x_{i,k}$ according to the demand bound function calculations in Eq. (6.4). The following shows the effect of accessing MCS by a task on itself and on higher and lower priority tasks;

- Self blocking, Eq. (6.5) shows that both accessing MCS and SCS by a task gives the same result.
- Higher priority task, the effect from higher priority task accessing MCS or SCS can be evaluated by Eq. (6.6). $I_H$ will be the same for both cases also.
- Lower priority task, Eq. (6.7) shows that $I_L$ for MCS is less than SCS case because in MCS the maximum of $x_{i,j}$ will be less than $x_i$ for SCS.

We can conclude that the required subsystem utilization for MCS case will be always less than or equal to the case of SCS having $x_j = \sum_{k=1}^{o} x_{i,k}$, which means that our proposed protocol is scalable in terms of the number of critical sections.

### 6.6.5 Independent abstraction

In this paper, we have proposed a synchronization protocol that supports independent abstraction of a subsystem, particularly, for open systems. Independent abstraction is desirable since it allows subsystems to be developed and validated without knowledge about temporal behavior of other subsystems. In some cases, subsystems can be abstracted *dependently* of others when some necessary information about all the other subsystems is available. However, dependent abstraction has a clear limitation to open systems where such information is assumed to be unavailable. In addition, dependent abstraction is not good for dynamically changing systems, since it may be no longer valid when a new subsystem is added. Despite of the advantages of independent abstraction vs. dependent abstraction, however, one may wonder what costs look like in using independent abstraction in comparison with using dependent abstraction. In this section, we discuss this issue in terms of resource efficiency (subsystem resource utilization).

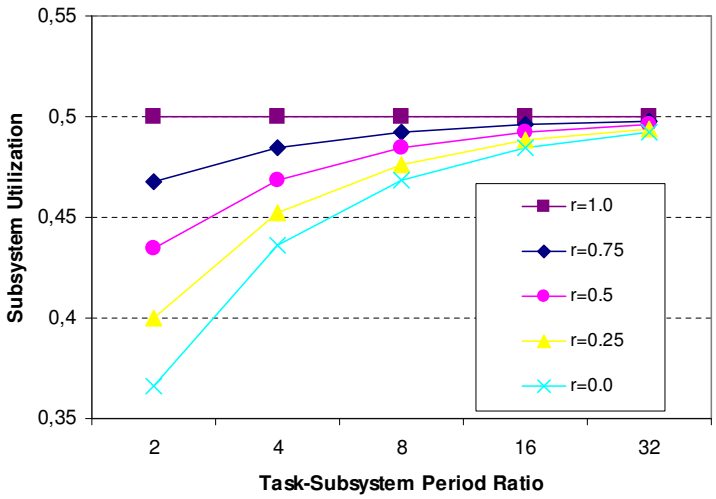


Figure 6.7: Comparison between independent and dependent abstractions in terms of subsystem utilization.

One of the key differences between independent and dependent abstractions is how to model a resource supply provided to a subsystem, more specifically, how to characterize the longest *blackout duration* during which no resource supply is provided. Under independent abstraction, the longest blackout duration is assumed to be the worst-case (maximum) one. Whereas, it can be exactly identified by some techniques [6, 26] under dependent abstraction. This difference inherently yields different subsystem resource utilizations, as illustrated in Figure 6.7. Before explaining this figure, we need to establish some notions and explain how to obtain this figure.

We first extend the periodic resource model $\Gamma(\Pi, \Theta)$ by introducing an additional parameter, *blackout duration ratio (r)*. We define $r$ as follows. Let $L_{min}$ and $L_{max}$ denote the minimum and maximum possible blackout duration, and

$$L_{min} = \Pi - \Theta \text{ and } L_{max} = 2(\Pi - \Theta).$$

When exactly computed, the longest blackout duration can then be represented as $r \cdot (L_{max} - L_{min}) + L_{min}$. We generalize the supply bound function of Eq.

(6.2) with the blackout duration ratio $r$ as follows:

$$\mathtt{sbf}_{\Gamma}(t) = \begin{cases} t-(k+1)(\Pi-\Theta) & \text{if } t \in [k\Pi-\Theta \\ & +r(\Pi-\Theta), \\ & k\Pi+r(\Pi-\Theta)], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{6.12}$$

where $k = \max\Big(\big\lceil\big(t-(\Pi-\Theta)\big)/\Pi\big\rceil, 1\Big)$.

We here explain the notion of *task-subsystem period ratio*, which is the x-axis of the figure. Suppose a periodic resource model $\Gamma_1(\Pi_1, \Theta_1, r_1)$ is an abstraction that guarantees the schedulability of a subsystem $S$. According to Eq. (6.3), there then exists a time instant $t_i^*$, where $0 < t_i^* \leq T_i$, for each task $\tau_i$ within the subsystem $S$ such that

$$\forall \tau_i, \;\; \mathtt{dbf}_{\mathtt{FP}}(i, t_i^*) \leq \mathtt{sbf}_{\Gamma_1}(t_i^*). \tag{6.13}$$

In fact, given the values of subsystem period $\Pi$ and blackout duration ratio $r$, we can find a smallest value of $\Theta$, denoted as $\Theta_i^*$, that can satisfy Eq. (6.13) at $t_i^*$ for each task $\tau_i$. The value of budget $\Theta$ is then finally determined as the maximum value among all $\Theta_i^*$. This way makes sure that $\Theta$ is large enough to guarantee the timing requirements of all tasks. Let $T^*$ denote a time instant $t_k^*$ such that $\Theta_k^*$ is the maximum among the ones. We can see that $T^* \in [T_{min}, T_{max}]$, where $T_{min}$ and $T_{max}$ denote the minimum and maximum task periods within subsystem, respectively. We define the *task-subsystem period ratio* as $T^*/\Pi$.

Given a periodic abstraction $\Gamma_1$ of the subsystem $S$, another periodic resource model $\Gamma_2(\Pi_2, \Theta_2, r_2)$ can be also an abstraction of $S$, if

$$\forall \tau_i, \;\; \mathtt{sbf}_{\Gamma_1}(t_i^*) \leq \mathtt{sbf}_{\Gamma_2}(t_i^*), \tag{6.14}$$

since Eq. (6.3) can be satisfied with $S$ and $\Gamma_2$ as well. More specifically, $\Gamma_2(\Pi_2, \Theta_2, r_2)$ can be an abstraction of $S$, if

$$\mathtt{sbf}_{\Gamma_1}(T^*) \leq \mathtt{sbf}_{\Gamma_2}(T^*). \tag{6.15}$$

That is, given $\Gamma_1$ and the values of $\Pi_2$ and $r_2$, we can find the minimum value of $\Theta_2$ that satisfies Eq. (6.15).

Figure 6.7 shows subsystem utilizations of periodic abstractions under different values of blackout duration ratio $r$, when they have the same subsystem period in abstracting the same subsystem. In general, it shows that dependent abstraction, which can exactly identify the value of $r$, would pro-

duce more resource-efficient subsystem abstractions. Specifically, for example, when $r = 0$, i.e., when the subsystem has the highest priority under fixed-priority global scheduling, a subsystem can be abstracted with 15% less subsystem utilization than in the case of independent abstraction ($r = 1$). The figure also shows that differences in subsystem utilization generally decrease when the task-subsystem period ratio increases and/or the blackout duration ratio increases. For example, when $r = 0.5$, i.e., when the system has a moderately high utilization and subsystems have medium or low priorities under fixed-priority global scheduling or subsystems are scheduled under global EDF scheduling, differences are shown to be smaller than 8%.

## 6.7 Conclusion

In this paper we have presented the novel Subsystem Integration and Resource Allocation Policy (SIRAP), which provides temporal isolation between subsystems that share logical resources. Each subsystem can be developed, tested and analyzed without knowledge of the temporal behaviour of other subsystems. Hence, integration of subsystems, in later phases of product development, will be smooth and seamless.

We have formally proven key features of SIRAP such as bounds on delays for accessing shared resources. Further, we have provided schedulability analysis for tasks executing in the subsystems; allowing for use of hard real-time application 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. From the study we can see that the subsystem period should be chosen as much smaller than the smallest task period in a subsystem and take into account the maximum value of $h_i$ in the subsystem to prevent having high subsystem utilization. Future work includes investigating the effect of context switch overhead on subsystem utilization together with the subsystem period and the maximum value of $h_i$.

# Bibliography


[1] D. Andrews, I. Bate, T. Nolte, C. M. Otero Pérez, and S. M. Petters. Impact of embedded systems evolution on RTOS use and design. In *Proceedings of the* $1^{st}$ *International Workshop Operating System Platforms for Embedded Real-Time Applications (OSPERT'05)*, pages 13–19, July 2005.

[2] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22, Technische Universität at Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040, 2004.

[3] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[4] 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.

[5] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the* $4^{th}$ *ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[6] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the* $26^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[7] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the 23$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[8] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[9] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the 6$^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[10] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[11] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the 26$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 99–110, December 2005.

[12] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarhical fixed-priority scheduling. In *Proceedings of the 14$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, June 2002.

[13] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[14] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the 25$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 57–67, December 2004.

[15] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the* 21$^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[16] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the 7$^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[17] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON'87)*, pages 909–916, November 1987.

[18] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the $9^{th}$ IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, December 1988.

[19] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[20] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the $15^{th}$ International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[21] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in EDF-scheduled systems. In *Proceedings of the $13^{th}$ IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, 2007.

[22] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the $27^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[23] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *Proceedings of the $22^{nd}$ IEEE International Real-Time Systems Symposium (RTSS'01)*, pages 161 – 170, December 2001.

[24] P. Holman and J. Anderson. Locking under pfair scheduling. *ACM Transaction on Computer Systems*, 24(2):140–174, 2006.

[25] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal (British Computer Society)*, 29(5):390–395, October 1986.

[26] R. Bril, W. Verhaegh, and C. Wust. A cognac-glass algorithm for conditionally guaranteed budgets. In *Proceedings of the $27th$ IEEE International Real-Time Systems Symposium(RTSS'06)*, pages 388–400, December 2006.

# Chapter 7

# Paper B: Overrun Methods and Resource Holding Times for Hierarchical Scheduling of Semi-Independent Real-Time Systems

Moris Behnam, Thomas Nolte, Mikael Sjödin, Insik Shin

IEEE Transactions on Industrial Informatics, vol 6, nr 1, pages 93-104, February, 2010.

**Abstract**

The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework to enable compositional schedulability analysis of embedded software systems with real-time properties. In this paper a software 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, in the paper, the proposed methods allow non-intrusive configuration and tuning of subsystem timing-behaviour via subsystem interfaces for selecting scheduling parameters.

This paper considers three methods to handle overruns due to resource sharing between subsystems in the HSF. For each one of these three overrun methods corresponding scheduling algorithms and associated schedulability analysis are presented together with analysis that shows under what circumstances one or the other is preferred. The analysis is generalized to allow for both Fixed Priority Scheduling (FPS) and Earliest Deadline First (EDF) scheduling. Also, a further contribution of the paper is the technique of calculating resource-holding times within the framework under different scheduling algorithms; the resource holding times being an important parameter in the global schedulability analysis.

## 7.1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical resource sharing among applications under different scheduling services. The hierarchical scheduling framework can be generally represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., threads), and resources are allocated from a parent node to its children nodes.

The HSF provides means for decomposing a complex system into well-defined parts. In essence, the HSF provides a mechanism for timing-predictable *composition* of course-grained components or *subsystems*. In the HSF a subsystem provides an *interface* that specifies the timing properties of the subsystem precisely [1]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal behaviour. Also, the HSF facilitates *reusability* of subsystems in timing-critical and resource constrained environments, since the well defined interfaces characterize their computational requirements.

Earlier efforts have been made in supporting compositional subsystem integration in the HSFs, preserving the independently analyzed schedulability of individual subsystems. One common assumption shared by earlier studies is that subsystems are independent. This paper relaxes this assumption by addressing the challenge of enabling efficient compositional integration for independently developed *semi-independent* subsystems interacting through sharing of mutual exclusion access logical resources. Here, semi-independence means that subsystems are allowed to synchronize by the sharing of logical resources.

To enable sharing of logical resources in HSFs, Davis and Burns proposed a synchronization protocol implementing the *overrun* mechanism, allowing the subsystem to overrun (its budget) to complete the execution of a critical section [2]. Two versions of overrun mechanisms were presented in [2], called overrun without payback and overrun with payback, and in the remainder of this paper these overrun mechanisms are called Basic Overrun (BO), and Basic Overrun with Payback (PO), respectively. The study presented by Davis and Burns provides schedulability analysis for both overrun mechanisms; however, the schedulability analysis does not allow independent analysis of individual subsystems. Hence, the presented schedulability analysis does not naturally support composability of subsystems.

The schedulability analysis of Davis and Burns' has been extended assessing composability in [3] for systems running the Earlier Deadline First (EDF) scheduling algorithm. In addition, in the same paper a new overrun mecha-

nism has been presented, called Enhanced Overrun (EO), that potentially increases schedulability within a subsystem by providing CPU allocations more efficiently. Also, in the paper this new mechanism has been evaluated against PO.

The contributions of this paper are as follows; Firstly, BO, the second version of overrun mechanism presented in [2], is included in the comparison between overrun mechanisms presented in [3] and it is shown under which circumstances a certain overrun mechanism is the preferred one among all three (BO, PO and EO) presented mechanisms. In addition, the sechedulability analysis of local and global schedulers is generalized by including Fixed Priority Scheduling (FPS) in the schedulability analysis, as the results of [3] were limited to the EDF scheduling algorithm. Finally, the simplified equation to calculate resource holding time using the EDF scheduling algorithm (presented in [3]) is proven to be valid also when using the FPS scheduling algorithm. Hence, using the results of this paper it is possible to use either FPS or EDF.

The outline of the paper is as follows: Section 7.2 presents related work, while Section 7.3 presents the system model. In Section 7.4 the schedulability analysis for the system model is presented. Section 7.5 presents the three overrun mechanisms (BO, PO and EO), and Section 7.6 presents their analytical comparison. In Section 7.7 it is shown how to calculate the resource holding times under both FPS and EDF, and finally, Section 7.8 concludes.

## 7.2 Related work

This section presents related work in the areas of HSFs as well as resource sharing protocols.

### 7.2.1 Hierarchical scheduling

The HSF for real-time systems, originating in open systems [4] in the late 1990's, has been receiving an increasing research attention. Since Deng and Liu [4] introduced a two-level HSF, its schedulability has been analyzed under fixed-priority global scheduling [5] and under EDF-based global scheduling [6, 7]. Mok *et al.* [8] proposed the bounded-delay resource model to achieve a clean separation in a multi-level HSF, and schedulability analysis techniques [9, 10] have been introduced for this resource model. In addition, Shin and Lee [1, 11] introduced another periodic resource model (to characterize the periodic resource allocation behaviour), and many studies have

been proposed on schedulability analysis with this resource model under fixed-priority scheduling [12, 13, 14] and under EDF scheduling [1]. More recently, Easwaran *et al.* [15] introduced the Explicit Deadline Periodic (EDP) resource model. However, a common assumption shared by all these studies is that tasks are required to be independent.

### 7.2.2 Resource sharing

In many real systems, tasks are semi-independent, interacting with each other through mutually exclusive resource sharing. Many protocols have been introduced to address the priority inversion problem for semi-independent tasks, including the Priority Inheritance Protocol (PIP) [16], the Priority Ceiling Protocol (PCP) [17], and Stack Resource Policy (SRP) [18]. Recently, Fisher *et al.* addressed the problem of minimizing the resource holding time [19] under SRP. There have been studies on extending SRP for HSFs, for sharing of logical resources within a subsystem [20, 5] and across subsystems [2, 21, 22]. Davis and Burns [2] proposed the Hierarchical Stack Resource Policy (HSRP) supporting sharing of logical resources on the basis of an overrun mechanism. Behnam *et al.* [21] proposed the Subsystem Integration and Resource Allocation Policy (SIRAP) protocol that supports subsystem integration in the presence of shared logical resources, on the basis of skipping. Fisher *et al.* [22] proposed the BROE server that extends the Constant Bandwidth Server (CBS) [23] in order to handle sharing of logical resources in a HSF. Behnam *et al.* [24] compared between SIRAP, HSRP and BROE and showed that there is no one silver bullet solution available today, providing an optimal HSF and synchronization protocol for use in open environments. Lipari *et al.* proposed the BandWidth Inheritance protocol (BWI) [25] which extends the resource reservation framework to systems where tasks can share resources. The BWI approach is based on using the CBS algorithm together with a technique that is derived from the Priority Inheritance Protocol (PIP). Particularly, BWI is suitable for systems where the execution time of a task inside a critical section can not be evaluated.

## 7.3 System model and background

### 7.3.1 Resource sharing in the HSF

The Hierarchical Scheduling Framework (HSF) has been introduced to support CPU time sharing among applications (subsystems) under different scheduling

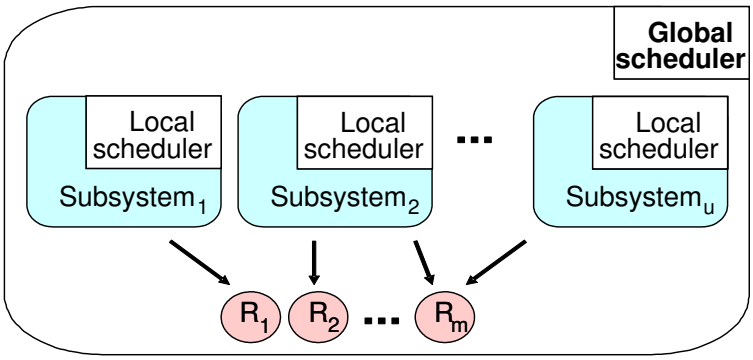


Figure 7.1: Two-level HSF with resource sharing.

policies. In this paper, a two level-hierarchical scheduling framework is considered, which works as follows: a global (system-level) scheduler allocates CPU time to subsystems, and a local (subsystem-level) scheduler subsequently allocates CPU time to its internal tasks.

Having such a HSF also allows for the sharing of logical resources among tasks in a mutually exclusive manner (see Figure 7.1). Specifically, tasks can share *local* logical resources within a subsystem as well as *global* logical resources across (in-between) subsystems. However, note that this paper focuses around mechanisms for sharing of global logical resources in a HSF while local logical resources can be supported by traditional synchronization protocols such as SRP (see, e.g., [20, 2, 5]).

### 7.3.2 Virtual processor models

The notion of real-time virtual processor (resource) model was first introduced by Mok *et al.* [8] to characterize the CPU allocations that a parent node provides to a child node in a HSF. The *CPU supply* of a virtual processor model refers to the amounts of CPU allocations that the virtual processor model can provide. The *supply bound function* of a virtual processor model calculates its minimum possible CPU supply for any given time interval of length $t$.

The periodic virtual processor model $\Gamma(P, Q)$ was proposed by Shin and Lee [1] to characterize periodic resource allocations, where $P$ is a period ($P > 0$) and $Q$ is a periodic allocation time ($0 < Q \leq P$). The capacity $U_\Gamma$ of a

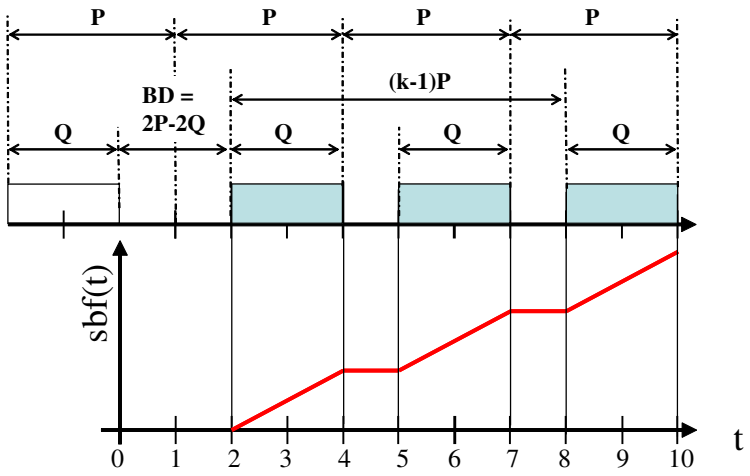


Figure 7.2: The supply bound function of a periodic virtual processor model $\Gamma(3, 2)$.

periodic virtual processor model $\Gamma(P, Q)$ is defined as $Q/P$.

The *supply bound function* $\texttt{sbf}_\Gamma(t)$ of the periodic virtual processor model $\Gamma(P, Q)$ was given in [1] to compute the minimum resource supply during an interval of length $t$.

$$\texttt{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(P-Q) & \text{if } t \in W^{(k)} \\ (k-1)Q & \text{otherwise,} \end{cases} \tag{7.1}$$

where $k = \max\left(\left\lceil (t-(P-Q))/P \right\rceil, 1\right)$ and $W^{(k)}$ denotes an interval $[(k+1)P - 2Q, (k+1)P - Q]$. Note that an interval of length $t$ may not begin synchronously with the beginning of period $P$; as shown in Figure 7.2, the interval of length $t$ can start in the middle of the period of a periodic virtual processor model $\Gamma(P, Q)$. Figure 7.2 illustrates the supply bound function $\texttt{sbf}_\Gamma(t)$ of the periodic virtual processor model. BD represents the longest possible *blackout duration* during which the periodic virtual processor model may provide no resource allocation at all.

### 7.3.3 Stack resource policy (SRP)

To be able to use SRP [18] in the HSF[1], its associated terms are extended as follows:

- *Preemption level*. Each task $\tau_i$ has a preemption level equal to $\pi_i = 1/D_i$, where $D_i$ is the relative deadline of the task. Similarly, each subsystem $S_s$ has an associated preemption level equal to $\Pi_s = 1/P_s$, where $P_s$ is the subsystem's per-period deadline.
- *Resource ceiling*. Each globally shared resource $R_j$ is associated with two types of resource ceilings; one *internal* resource ceiling for local scheduling $rc_j = \max\{\pi_i | \tau_i \text{ accesses } R_j\}$ and one *external* resource ceiling for global scheduling.
- *System/subsystem ceilings*. System/subsystem ceilings are dynamic parameters that change during runtime. The system/subsystem ceiling is equal to the currently locked highest external/internal resource ceiling in the system/subsystem.

Following the rules of SRP, a job $J_i$ that is generated by a task $\tau_i$ can preempt the currently executing job $J_k$ within a subsystem only if $J_i$ has a priority higher than that of job $J_k$ and, at the same time, the preemption level of $\tau_i$ is greater than the current subsystem ceiling. A similar reasoning is made for subsystems from a global scheduling point of view.

### 7.3.4 System model

In this paper a periodic task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$ is considered, where $T_i$, $C_i$ and $D_i$ represent the task's period, worst-case execution time (WCET) and relative deadline, respectively, where $D_i \leq T_i$, and $\{c_{i,j}\}$ is the set of WCETs within critical sections associated with task $\tau_i$. Each element $c_{i,j}$ in $\{c_{i,j}\}$ represents the WCET of the task $\tau_i$ inside a critical section of the global shared resource $R_j$.

Looking at a shared resource $R_j$, the *resource holding time* $h_{j,i}$ of a task $\tau_i$ is defined as the time given by the task's maximum execution time inside

[1] One of the reasons of using SRP is that it can be used with both the EDF and the FPS scheduling algorithms while the other synchronization protocols such as PCP and PIP can only be used with the FPS scheduling algorithm. In addition, Bertogna *et al.* [26] showed that the resource holding time will be higher when using PCP since it should include the execution times of the higher priority tasks that have priorities lower than the ceiling of the shared resource.

a critical section plus the interference (inside the critical section) of higher priority tasks having preemption level greater than the internal ceiling of the locked resource.

A subsystem $S_s \in \mathcal{S}$, where $\mathcal{S}$ is the whole system of subsystems, is characterized by a task set $\mathcal{T}_s$ that contains $n_s$ tasks and a set of internal resource ceilings $\mathcal{RC}_s$ inherent from internal tasks using the globally shared resources. Each subsystem $S_s$ is assumed to have an EDF or FPS local scheduler, and the subsystems are scheduled according to EDF or FPS on a global level. The collective CPU resource requirements by each subsystem $S_s$ is characterized by its *interface* (the subsystem interface) defined as $(P_s, Q_s, H_s)$, where $P_s$ is the subsystem's period, $Q_s$ is it's execution requirement budget, and $H_s$ is the subsystem's maximum resource holding time, i.e., $H_s = \max\{h_{j,i} | \tau_i \in \mathcal{T}_s \text{ accesses } R_j\}$.

## 7.4 Schedulability analysis

This section presents the schedulability analysis of the HSF, starting with local schedulability analysis needed to calculate subsystem interfaces, followed by global schedulability analysis. The analysis presented assumes that SRP is used for synchronization on the local (within subsystems) level.

### 7.4.1 Local schedulability analysis

Let $\texttt{dbf}_{\texttt{EDF}}(i,t)$ denote the demand bound function of a task $\tau_i$ under EDF scheduling [27], i.e.,

$$\texttt{dbf}_{\texttt{EDF}}(i,t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i. \tag{7.2}$$

The local schedulability condition under EDF scheduling is then (by combining the results of [28] and [1])

$$\forall t > 0 \quad \sum_{\tau_i \in \mathcal{T}_s} \texttt{dbf}_{\texttt{EDF}}(i,t) + b(t) \leq \texttt{sbf}(t), \tag{7.3}$$

where $b(t)$ is the blocking function [28] (according to SRP) that represents the longest blocking time during which a job $J_i$ with $D_i \leq t$ may be blocked by a job $J_k$ with $D_k > t$ when both jobs access the same resource. Note that $t$ can

be selected within a finite set of scheduling points using the same techniques from [27][2].

For Fixed Priority Scheduling (FPS) [30], let $\texttt{rbf}_{\texttt{FP}}(i,t)$ denote the request bound function of a task $\tau_i$, i.e.,

$$\texttt{rbf}_{\texttt{FP}}(i,t) = C_i + \sum_{\tau_k \in \texttt{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{7.4}$$

where $\texttt{HP}(i)$ is the set of tasks with priorities higher than that of $\tau_i$. The local schedulability analysis under FPS can then be extended from the results of [18, 1] as follows:

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; \texttt{rbf}_{\texttt{FP}}(i,t) + b_i \leq \texttt{sbf}(t), \tag{7.5}$$

where $b_i$ is the maximum *blocking* (i.e., extra CPU demand) imposed to a task $\tau_i$ when $\tau_i$ is blocked by lower priority tasks that are accessing resources with ceiling greater than or equal to the priority of $\tau_i$. Note that $t$ can be selected within a finite set of scheduling points [31].

### 7.4.2 Subsystem interface calculation

Given a subsystem $S_s$, $\mathcal{R}C_s$, and $P_s$, let $\mathsf{calculateBudget}(S_s, \mathcal{R}C_s, P_s)$ denote a function that calculates the smallest subsystem budget $Q_s$ that satisfies Eq. (7.3) for EDF and Eq. (7.5) for FPS scheduling. Hence, $Q_s = \mathsf{calculateBudget}(S_s, P_s, \mathcal{R}C_s)$. Recently Fisher *et al.* [29] presented an algorithm that finds the exact smallest subsystem budget $Q_s$ using EDF as a local scheduler, called MINIMUMCAPACITY. This work is extended for the FPS local scheduler with a corresponding new algorithm called FPMINIMUMCAPACITY [32]. Both algorithms do not consider resource sharing, however, they can support resource sharing by using Eq. (7.3) for EDF and Eq. (7.5) for FPS scheduling. The complexity of both algorithms is pseudo-polynomial with respect to time.

### 7.4.3 Global schedulability analysis

Following Theorem 1 of [28], global schedulability analysis under EDF scheduling is given using the system load bound function $\texttt{LBF}(t)$ as follows:

[2]Note that although the work in [27] does not consider hierarchical scheduling, the same technique can be used in the context of hierarchical scheduling [22], [29].

$$\forall t > 0, \quad \texttt{LBF}(t) = B(t) + \sum_{S_s \in \mathcal{S}} \texttt{DBF}_s(t) \leq t, \tag{7.6}$$

where

$$\texttt{DBF}_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot Q_s, \tag{7.7}$$

and the system-level blocking function $B(t)$ represents the maximum blocking time (according to SRP) during which a subsystem $S_s$ may be blocked by another subsystem $S_k$, where $P_s \leq t$ and $P_k > t$. $B(t)$ is defined as

$$B(t) = \max\{H_k \mid P_k > t\}. \tag{7.8}$$

Under global FPS scheduling, the subsystem load bound function is as follows (on the basis of a similar reasoning of Eq. (7.4)):

$$\texttt{LBF}_s(t) = \texttt{RBF}_s(t) + B_s \text{ , where} \tag{7.9}$$

$$\texttt{RBF}_s(t) = Q_s + \sum_{S_k \in \texttt{HPS}(s)} \left\lceil \frac{t}{P_k} \right\rceil \cdot Q_k, \tag{7.10}$$

where $\texttt{HPS}(s)$ is the set of subsystems with priority higher than that of $S_s$. Let $B_s$ denote the maximum blocking (i.e., extra CPU demand) imposed to a subsystem $S_s$, when it is blocked by lower-priority subsystems,

$$B_s = \max\{H_j | \ S_j \in \texttt{LPS}(S_s)\}, \tag{7.11}$$

where $\texttt{LPS}(S_s)$ is the set of subsystems with priority lower than that of $S_s$.

A global schedulability condition under FPS is then

$$\forall S_s \ \exists t : 0 < t \leq P_s, \ \ \texttt{LBF}_s(t) \leq \texttt{t}. \tag{7.12}$$

## 7.5 Overrun mechanisms

This section explains three overrun mechanisms that can be used to handle budget expiry during a critical section in the HSF. Consider a global scheduler that schedules subsystems according to their periodic interfaces $(P_s, Q_s, H_s)$. The subsystem budget $Q_s$ is said to *expire* at the point when one or more internal (to the subsystem) tasks have executed a total of $Q_s$ time units within

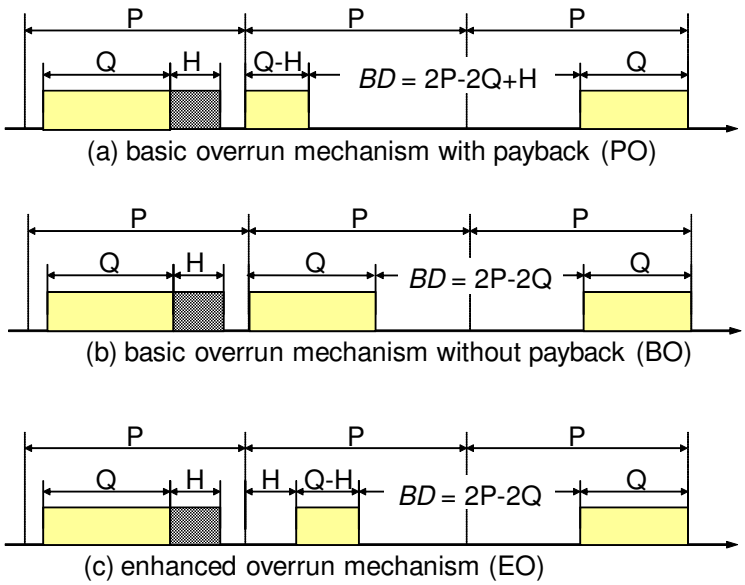


Figure 7.3: Basic and enhanced overrun mechanisms.

the subsystem period $P_s$. Once the budget is expired, no new task within the same subsystem can initiate its 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 $Q_s$.

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

In this paper, an overrun mechanism is considered as follows; when the budget of subsystem $S_s$ expires and $S_s$ has a job $J_i$ that is still locking a globally shared resource, job $J_i$ continues its execution until it releases the locked resource. The extra time that $J_i$ needs to execute after the budget of $S_s$ expires is denoted as *overrun time* $\theta$. The maximum $\theta$ occurs when $J_i$ locks a resource that gives the longest resource holding time just before the budget of

$S_s$ expires.

Here, two versions of overrun mechanisms [2] are considered;

1. The overrun mechanism with payback, introduced as PO and later EO: whenever overrun happens, the subsystem $S_s$ pays back $\theta$ in its next execution instant, i.e., the subsystem budget $Q_s$ will be decreased by $\theta$ for the subsystem's execution instant following the overrun (note that only the instant following the overrun is affected).

2. The overrun mechanism without payback, introduced as BO: in this version of the overrun mechanism, no further actions will be taken after the event of an overrun.

Hereinafter, the overrun mechanism with payback is called PO, and the overrun mechanism without payback is called BO. Both are versions of the basic overrun mechanism. Also, an extended mechanism with payback is introduced as EO.

### 7.5.1 Basic overrun – overrun mechanism 1 and 2

Davis *et al.* [2] presented schedulability analysis for both (BO and PO) versions of basic overrun, however, the presented analysis is not suitable for open environments [4] as it requires detailed information of all tasks in the system in order to calculate global schedulability. This section discusses how to extend the existing schedulability analysis for the basic overrun mechanisms, making them suitable for open environments.

#### Global analysis with basic overrun

**PO – basic overrun with payback** Firstly, the demand bound function (and the request bound function) of a subsystem with the basic overrun mechanism with payback is extended. Looking at the PO mechanism in a subsystem $S_s$, the maximum contribution on $\texttt{DBF}_s(t)$ for EDF scheduling and $\texttt{RBF}_s(t)$ for FPS scheduling is $H_s$. When $S_s$ overruns with its maximum, which is $H_s$, the subsystem's resource demand within the subsystem period $P_s$ will be increased to $Q_s + H_s$. Following this, the budget of the next period will be decreased to $Q_s - H_s$ due to the payback mechanism. Then, suppose that the subsystem overruns again. Now, during the next subsystem period, the subsystem's resource demand will be $Q_s - H_s + H_s = Q_s$. Here, it is easy to observe that the subsystem's resource demand will be at most $kQ_s + H_s$ during $k$ subsystem

periods. Hence, the demand bound function $\texttt{DBF}_s^\circ(t)$ of a subsystem $S_s$ with the basic overrun mechanism using EDF scheduling globally is

$$\texttt{DBF}_s^\circ(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot Q_s^\circ + O_s(t), \tag{7.13}$$

where $Q_s^\circ$ is the subsystem budget when using the PO mechanism and,

$$O_s(t) = \begin{cases} H_s & \text{if } t \geq P_s, \\ 0 & \text{otherwise.} \end{cases} \tag{7.14}$$

The schedulability condition of Eq. (7.6) can then be extended by substituting $\texttt{DBF}_s(t)$ with $\texttt{DBF}_s^\circ(t)$.

When using a global FPS scheduler, the request bound function $\texttt{RBF}_s^\circ(t)$ is

$$\texttt{RBF}_s^\circ(t) = (Q_s^\circ + H_s) + \sum_{S_k \in \texttt{HPS}(s)} \left( \left\lceil \frac{t}{P_k} \right\rceil (Q_k^\circ) + H_k \right). \tag{7.15}$$

**BO – basic overrun without payback** This version of overrun does not payback the budget after overrun happens. This means that the system resource demands within the period of $P_s$ can be up to $Q_s + H_s$ for all periods considering that the maximum overrun will happen every period, which is the worst case scenario. Then for EDF global scheduling, the maximum demand bound function $\texttt{DBF}_s^\#(t)$ using the BO mechanism is

$$\texttt{DBF}_s^\#(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot (Q_s^\# + H_s), \tag{7.16}$$

where $Q_s^\#$ is the subsystem budget when using the BO mechanism.

For a global FPS scheduler, the request bound function $\texttt{RBF}_s^\#(t)$ is

$$\texttt{RBF}_s^\#(t) = (Q_s^\# + H_s) + \sum_{S_k \in \texttt{HPS}(s)} \left\lceil \frac{t}{P_k} \right\rceil \cdot (Q_k^\# + H_k). \tag{7.17}$$

### Independent analysis with basic overrun

Using PO, there is no single worst-case resource supply scenario. In fact, there are two scenarios that constitute the worst-case scenario; the worst-case scenario is either of those two scenarios, depending on an interval length $t$. In

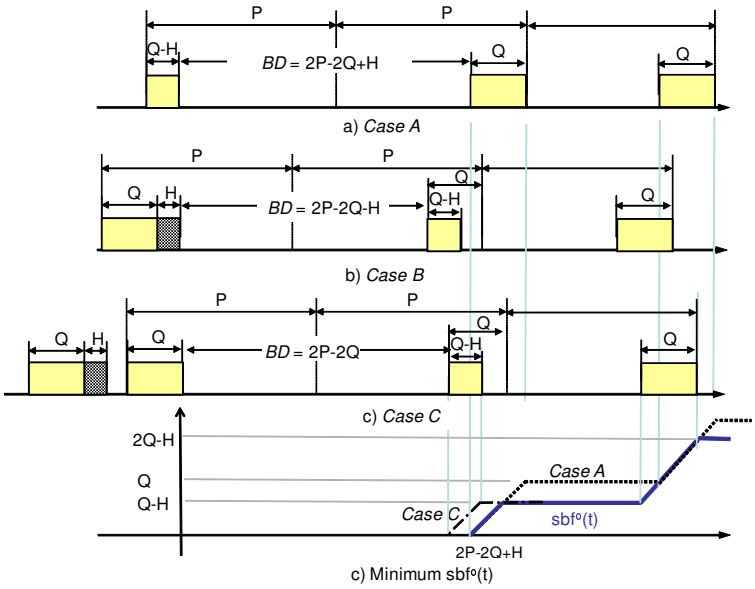


Figure 7.4: $\mathtt{sbf}^{\circ}(\mathtt{t})$ for the PO mechanism.

the first case ("Case A" shown in Figure 7.4a), the greatest Blackout Duration (BD) is $2(P-Q)+H$ assuming that a task is activated upon the subsystem budget expiration and the given budget was $Q_s - H_s$. While in the second case ("Case C" shown in Figure 7.4c), the subsystem can not be supplied by more than $Q_s - H_s$ if a task is released upon the subsystem budget expiration and the subsystem has experienced overrun by $H_s$ in the previous instant. Note that when a system is feasible from a global scheduling perspective, the latest CPU resource availability for a subsystem $S_s$ will be $P_s - Q_s$ even during the pay-back period, see Figure 7.4c within the payback period the latest finalization time of $Q_s - H_s$ is guaranteed to be at least $H_s$ before the next activation of $S_s$.

Let's define two functions, function $f_1(t)$ for "Case A" and $f_2(t)$ for "Case C", to represent the minimum resource supply for each case. Then, the supply bound function for PO ($\mathtt{sbf}^{\circ}_{\Gamma}(t)$) is defined accordingly as the minimum of these two functions (see Figure 7.4). It is defined as follows:

$$\mathtt{sbf}^{\circ}_{\Gamma}(t) = \max\Big(\min\Big(f_1(t), f_2(t)\Big), 0\Big), \tag{7.18}$$

where $f_1(t)$ is

$$f_1(t) = \begin{cases} t-(k+1)(P-Q)-H & \text{if } t \in W^{(k)} \\ (k-1)Q & \text{otherwise,} \end{cases} \tag{7.19}$$

where $k = \max\Big(\big\lceil\big(t-(P-Q)-H\big)/P\big\rceil, 1\Big)$ and $W^{(k)}$ denotes an interval $[(k+1)P-2Q+H, (k+1)P-Q+H]$, and $f_2(t)$ is

$$f_2(t) = \begin{cases} t-(2)(P-Q) & \text{if } t \in V^{(k)} \\ t-(k+1)(P-Q)-H & \text{if } t \in Z^{(k)} \\ (k-1)Q-H & \text{otherwise,} \end{cases} \tag{7.20}$$

where $k = \max\Big(\big\lceil\big(t-(P-Q)\big)/P\big\rceil, 1\Big)$, $V^{(k)}$ denotes an interval $[2P-2Q, 2P-Q-H]$, and $Z^{(k)}$ denotes an interval $[(k+2)P-2Q, (k+2)P-Q]$.

The existing schedulability conditions of Eq. (7.3) can then be extended by substituting $\mathtt{sbf}_{\Gamma}(t)$ with $\mathtt{sbf}^{\circ}_{\Gamma}(t)$.

For BO – basic overrun without payback – Eq. (7.1) can still be used without modification to evaluate the supply bound function since the Blackout Duration is $2(P-Q)$ (as shown in Figure 7.3b), i.e., the supply bound function using BO $\mathtt{sbf}^{\#}_{\Gamma}(t)$ will equal to $\mathtt{sbf}^{\#}_{\Gamma}(t) = \mathtt{sbf}_{\Gamma}(t)$.

### 7.5.2 Enhanced overrun – overrun mechanism 3

As seen in Section 7.5.1, the PO mechanism works with a modified supply bound function $\mathtt{sbf}^{\circ}(t)$ that is less efficient in terms of CPU resource usage compared with the original $\mathtt{sbf(t)}$, as illustrated in Figure 7.4. While for the BO mechanism, the request/demand bound function (DBF/RBF) will be increased by $Q_s + H_s$ in all periods which may require more resources as well. In the following an enhanced overrun mechanism (EO) is proposed. This new overrun mechanism makes it possible to improve $\mathtt{sbf}^{\circ}(t)$ and at the same time the request/demand bound function (DBF/RBF) will be $Q_s + H_s$ for the first instance and then only $Q_s$ for the following periods when applying global schedulability analysis.

The EO mechanism is based on imposing an offset (delaying the budget replenishment of subsystem) equal to the amount of an overrun ($\theta_s = H_s$) to the execution instant that follows a subsystem overrun (at this instant, the

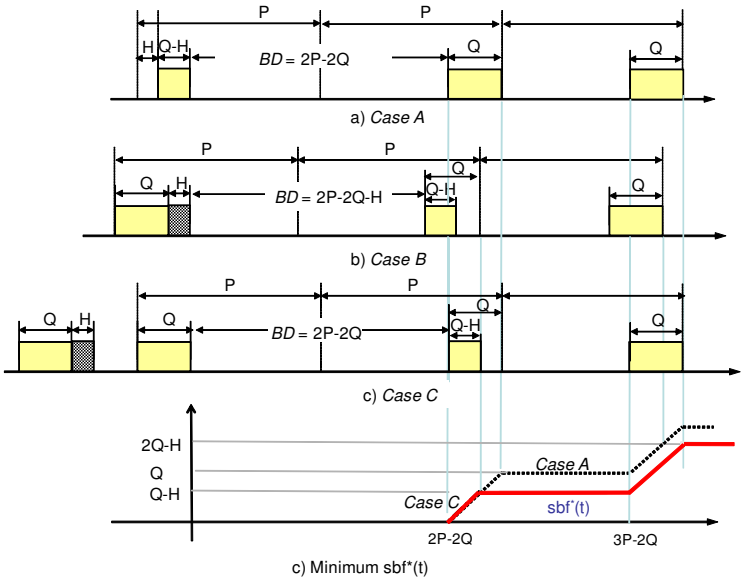


Figure 7.5: $\mathtt{sbf}^*(\mathtt{t})$ for the EO mechanism.

subsystem budget is $Q_s - \theta_s$). As shown in Figure 7.3c, the execution of the subsystem will be delayed by $\theta_s$ after a new period followed by overrun even if that subsystem has the highest priority at that time. By this the maximum BD will be decreased to $2(P-Q)$ compared with PO (basic overrun with payback) shown in Figure 7.3a. By this, the effect of $f_1(t)$ is removed from the supply bound function which makes the supply bound function for EO, better than when using PO. The supply bound function for EO $\mathtt{sbf}^*_\Gamma(t)$ is

$$\mathtt{sbf}^*_\Gamma(t) = \max\Big(f_2(t), 0\Big). \tag{7.21}$$

Note that the schedulability analysis of both the PO and the EO mechanisms presented in [3] are not accurate since the CPU supply "Case C" shown in Figures 7.4 and 7.5 has not been taken into account.

**Global analysis with enhanced overrun**

In the following, a demand bound function $\texttt{DBF}^*_s(t)$ is presented for EDF global scheduling of a subsystem $S_s$ that upper-bounds the demand requested by $S_s$ under the EO mechanism. Now, $\texttt{DBF}^*_s(t)$ includes the offset $\theta_s = H_s$ as follows:

$$\texttt{DBF}^*_s(t) = \left\lfloor \frac{t + H_s}{P_s} \right\rfloor \cdot Q^*_s + O^*_s(t), \tag{7.22}$$

where $Q^*_s$ is the subsystem budget when using the EO mechanism and

$$O^*_s(t) = \begin{cases} H_s & \text{if } t \geq P_s - H_s, \\ 0 & \text{otherwise.} \end{cases} \tag{7.23}$$

The schedulability condition of Eq. (7.6) can then be extended by substituting $\texttt{DBF}_s(t)$ with $\texttt{DBF}^*_s(t)$.

Using an FPS global scheduler, the offset imposed by the EO mechanism for each subsystem $S_s$ can be modeled as a release jitter $J_s$ with the range of $[0, H_s]$ so $J_s = H_s$. The upper bound of the request bound function $\texttt{RBF}^*_s(t)$ calculation is as follows;

$$\texttt{RBF}^*_s(t) = (Q^*_s + H_s) + \sum_{S_k \in \texttt{HPS}(s)} \left( \left\lceil \frac{t + J_k}{P_k} \right\rceil (Q^*_k) + H_k \right). \tag{7.24}$$

The schedulability analysis then

$$\forall S_s, 0 < \exists t \leq P_s - H_s, \;\; \texttt{LBF}^*_s(t) \leq \texttt{t}, \tag{7.25}$$

where

$$\texttt{LBF}^*_s(t) = \texttt{RBF}^*_s(t) + B_s. \tag{7.26}$$

## 7.6 Comparison between the three overrun mechanisms

In this section, the efficiency of the three overrun mechanisms (BO, PO and EO) are compared. First, the effect of using each one of them locally is shown, i.e., on a subsystem level. Then, their effect globally is shown, i.e., on a system level.

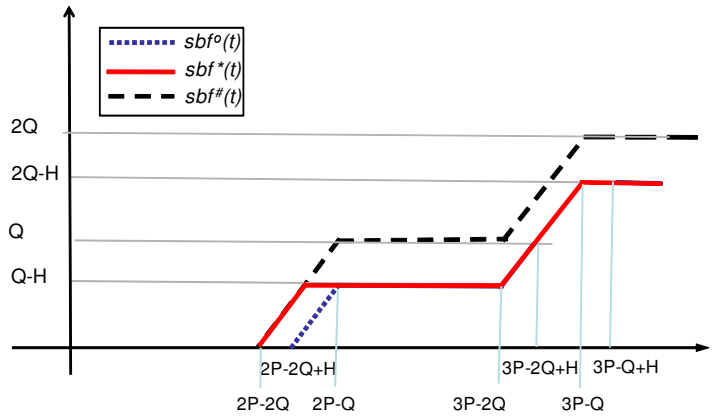


Figure 7.6: Comparing $\texttt{sbf}(\texttt{t})$ for the the three mechanisms using the same $P_s$ and $Q_s$.

### 7.6.1 Subsystem-level comparison

We will explain the effect of using each of the three overrun mechanisms, on the minimum subsystem budget $Q_s$. As described in Section 7.4.2, $Q_s$ should be the smallest value that satisfy Eq. (7.3) for EDF and Eq. (7.5) for FPS scheduling. Note that the demand bound function $\texttt{dbf}_{\texttt{EDF}}(i,t)$ (request bound function $\texttt{rbf}_{\texttt{FP}}(i,t)$) will not be changed when using the BO, PO and EO mechanisms. The only thing that changes when using one of the mechanisms in the subsystem level is the supply bound function as described in the previous section. Figure 7.6 shows the supply bound function for the three mechanisms using the same $P_s$ and $Q_s$. We will show in the following lemmas that the mechanisms that have higher supply bound function will require smaller $Q_s$.

The following lemma shows that the minimum required subsystem budget when using the EO mechanism will be lower than or equal to the minimum required budget when using the PO mechanism for both FPS and EDF local schedulers.

**Lemma 4.** *Assuming that the minimum required budget to schedule all tasks in a subsystem $S_s$ using the PO mechanism is $Q_s^\circ$, and that the corresponding budget when using the EO mechanism is $Q_s^*$, then $Q_s^* \leq Q_s^\circ$.*

*Proof.* The proof is split into two parts, proving the case of having an EDF local scheduler and an FPS local scheduler, respectively.

**EDF local scheduler** A subsystem $S_s$ is exactly schedulable iff in addition to Eq. (7.3), $\sum_i^n \texttt{dbf}_{\mathsf{EDF}}(i,t) + b(t) = \texttt{sbf}(t)$ for $\exists t$ s.t. $min_i^n D_i \leq t \leq LCM_{S_s} + max_i^n D_i$ (see Theorem 2.2 in [15]). This means that if the budget $Q_s$ is the minimum required budget to guarantee the schedulability of tasks in $S_s$, then there is a set of times $t^e$ at which $\sum_i^n \texttt{dbf}_{\mathsf{EDF}}(i,t) + b(t) = \texttt{sbf}(t)$. Without loss of generality, assume that $t^e$ includes one element. If the same subsystem budget $Q_s$ is used when running the PO mechanism and the EO mechanism, respectively, then from Eq. (7.18) and Eq. (7.21), there are two cases:

case 1: $\texttt{sbf}^\circ(t) = \texttt{sbf}^*(t)$ for $t \geq 2P_s - Q_s$ see Figure (7.6). Note that we have excluded the range when $\texttt{sbf}^*(t) = \texttt{sbf}^\circ(t) = 0$ since it is not interesting.

case 2: $\texttt{sbf}^\circ(t) < \texttt{sbf}^*(t)$ for $t$ out of the range specified in case 1.

If $t^e$ in the range given in the case 1 then $\texttt{sbf}^\circ(t^e) = \texttt{sbf}(t^e)$. In turn, $\sum_i^n \texttt{dbf}_{\mathsf{EDF}}(i,t^e) + b(t^e) = \texttt{sbf}^\circ(t^e)$, which means that $Q_s^*$ may be enough to schedule all tasks in a subsystem $S_s$ using the PO mechanism, so $Q_s^* = Q_s^\circ$ at time $t = t^e$. However, Eq. (7.3) must be checked if it holds for all other times $t$, to be sure that the subsystem $S_s$ is still schedulable.

If $t^e$ is not in the range given for case 1, then $\texttt{sbf}^\circ(t^e) < \texttt{sbf}(t^e)$. In turn, $\texttt{sbf}^\circ(t^e) < \sum_i^n \texttt{dbf}_{\mathsf{EDF}}(i,t^e) + b(t^e)$ which means that the budget $Q_s^*$ will not satisfy the condition in Eq. (7.3) using the PO mechanism, hence a higher budget should be provided. In this case $Q_s^* < Q_s^\circ$.

**FPS local scheduler** A subsystem $S_s$ is exactly schedulable iff in addition to Eq. (7.5), $\forall \tau_i, 0 < \forall t \leq D_i,\ \texttt{rbf}_{\mathsf{FP}}(i,t) + b_i \geq \texttt{sbf}(t)$ (see Theorem 2.3 in [15]). This means that if the budget $Q_s^*$ is the minimum required budget to guarantee the schedulability of tasks in $S_s$, then there is a set of times $t^f$ at which $\texttt{rbf}_{\mathsf{FP}}(i,t) + b_i = \texttt{sbf}(t)$.

If all elements in $t^f$ are not in the range given for case 1, then $\forall \tau_i, 0 < \forall t \leq D_i,\ \texttt{rbf}_{\mathsf{FP}}(i,t) + b_i > \texttt{sbf}(t)$ which makes the local scheduler unschedulable. To solve this problem, the budget when using the PO mechanism should be increased. In this case $Q_s^* < Q_s^\circ$.

□

**Lemma 5.** *Assuming that the minimum required budget to schedule all tasks in a subsystem $S_s$ using the BO mechanism is $Q_s^{\#}$, and that the corresponding budget when using the EO mechanism is $Q_s^*$, then $Q_s^{\#} \leq Q_s^*$.*

*Proof.* Comparing Eq. (7.1) and Eq. (7.21), $\texttt{sbf}^{\#}(t) = \texttt{sbf}^{*}(t)$ for $t \in [2P_s - 2Q_s, 2P_s - Q_s - H_s]$ and $\texttt{sbf}^{\#}(t) > \texttt{sbf}^{*}(t)$ otherwise (see Figure (7.6)). Using the same reasoning as in Lemma 1, we can prove that $Q_s^{\#} \geq Q_s^*$. □

### 7.6.2 System-level comparison

As shown in the previous section, the minimum required budget when using the EO mechanism is grater than or equal to the minimum required budget when using the BO mechanism, and lower than or equal to the minimum required budget when using the PO mechanism. However, at system level, it is not easy to see which one of these three approaches that will require minimum overall system CPU resources in the general case.

In doing a comparison among the three approaches, *system load* is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the schedulability of the system $S$. Then, the impact of each overrun mechanism on the system load can be investigated, respectively.

When using EDF as a global scheduler, the system load is computed as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{t} \frac{\texttt{LBF}(t)}{t} \,. \tag{7.27}$$

Note that $\alpha = \mathsf{load}_{\mathsf{sys}}$ is the smallest fraction of the CPU that is required to schedule all the subsystems in the system $S$ (satisfying Eq. (7.6)) assuming that the resource supply function (at system level) is $\alpha t$. This means that $\alpha t$ is greater than or equal to $\texttt{LBF}(t)$ for all $t > 0$ and according to Eq. (7.6), the system is schedulable. As long as $\alpha t = \texttt{LBF}(t)$ at a certain time $t$, then $\alpha$ is the minimum value that guarantee the schedulability, otherwise if we select a lower value than $\alpha$ for the resource supply function, then the system will be unschedulable.

When using FPS as a global scheduler, the system load is computed as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\} \tag{7.28}$$

where

$$\alpha_s = \min_{0<t\leq P_s} \{\frac{\texttt{LBF}_s(t)}{t} \mid \texttt{LBF}_s(t) \leq t\}. \tag{7.29}$$

Looking at Eq. (7.27) and Eq. (7.28), $\mathsf{load}_{\mathsf{sys}}$ can be decreased by lowering $\texttt{LBF}(t)$.

**EDF global scheduler** Comparing between the three overrun mechanisms, the mechanism that requires the lowest $\texttt{DBF}(t)$ at the time $t$ when $\texttt{LBF}(t)/t$ is at its maximum will require less system load. Three cases can be distinguished based on the type of overrun mechanism used and its associated demand bound function:

1. **PO vs BO**. Comparing Eq. (7.13) and Eq. (7.16), it can be concluded that $\texttt{DBF}^{\circ}_s(t) > \texttt{DBF}^{\#}_s(t)$ for $0 \leq t < 2 \cdot P_s$. The reason for this is that according to Lemma 5 and Lemma 4, $Q^{\circ}_s > Q^{\#}_s$. When $t$ is in the range of $0 \leq t < 2 \cdot P_s$, the floor in Eq. (7.13) and Eq. (7.16) will equal to 0 or 1, which makes Eq. (7.13) and Eq. (7.16) identical, and the only difference is the value of the budget. If $t$ is not in this range then it is not possible to decide which mechanism that can give a lower demand bound function without knowing the full interface parameters using both mechanisms. At a certain time instance $t^s$ when $t^s \gg 2P_s$, $\texttt{DBF}^{\circ}_s(t) < \texttt{DBF}^{\#}_s(t)$ for $t \geq t^s$.

2. **BO vs EO**. Comparing Eq. (7.16) and Eq. (7.22), it can be concluded that $\texttt{DBF}^{*}_s(t) \geq \texttt{DBF}^{\#}_s(t)$ for $0 \leq t < 2 \cdot P_s$. The reason for this is that according to Lemma 5 $Q^{*}_s \geq Q^{\#}_s$, while the floor part in Eq. (7.16) and Eq. (7.22) is different. Looking at the EO mechanism, the demand bound function is increased when $t = P_s - H_s$. For the BO mechanism, the demand bound function is increased at $t = P_s$, which means that $\texttt{DBF}^{*}_s(t) > \texttt{DBF}^{\#}_s(t)$ at $P_s - H_s \leq t < P_s$. However, if $t > 2P_s$, then it is not possible to decide which one of the two mechanisms that will be better than the other, as in the first case above.

3. **PO vs EO**. Comparing Eq. (7.13) and Eq. (7.22), it can be concluded that $\texttt{DBF}^{\circ}_s(t) < \texttt{DBF}^{*}_s(t)$ when $t$ is in the range $kP_s - H_s \geq t < kP_s$ and $\texttt{DBF}^{\circ}_s(t) \geq \texttt{DBF}^{*}_s(t)$ when $t$ is in $kP_s \geq t < (k+1)P_s - H_s$, where $k$ is an integer value greater and $k > 0$. Note that, at a certain time instance $t^s$ when $t^s \gg 2P_s$, $\texttt{DBF}^{\circ}_s(t) \geq \texttt{DBF}^{*}_s(t)$ for $t \geq t^s$ if $Q^{*}_s < Q^{\circ}_s$.

The example shown in Figure 7.7 explains the three cases described above.

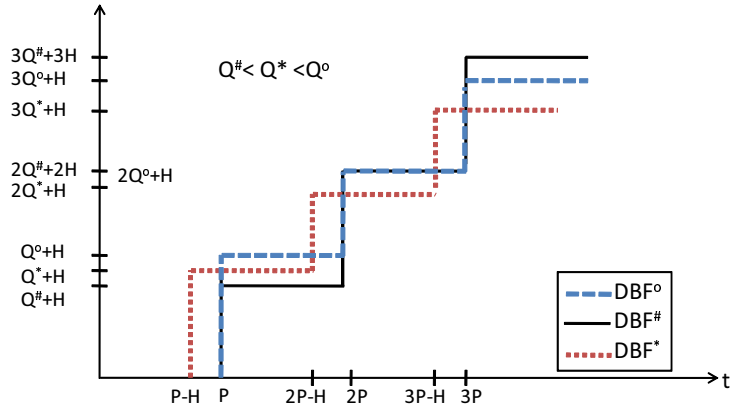


Figure 7.7: Comparing between $\mathtt{DBF}_s^{\circ}(t)$, $\mathtt{DBF}_s^{\#}(t)$ and $\mathtt{DBF}_s^{*}(t)$.

Defining the time $t^l$ as the time at which the system load is evaluated from Eq. (7.27), then, depending on the value of $t^l$ and the type of overrun mechanism used, it would be possible to estimate which one of the three overrun mechanisms that will require the lowest system load. For example, if $t^l \in 2 \cdot P_k$ and $P_k$ is the shortest subsystem period, then the subsystem load when using the BO mechanism is less than or equal to the subsystem load when using any of the other two overrun mechanisms. However, if $t^l \gg P_k$ then the possibility of having good results when using the BO mechanism is very low. Another aspect that can be considered is when $Q_s^{\circ} = Q_s^{*}$ for all subsystems, then the system load using the PO mechanism will always be less than or equal to the system load when using EO. Otherwise, all subsystem parameters should be given in order to evaluate which one of the three mechanisms that can give better results in terms of lowest system load.

**FPS global scheduler** Looking at Eq. (7.29), in order to minimize the system load $\mathtt{LBF}_k(t)$ of the subsystem $S_k$ that generates the maximum $\alpha$ should be minimized. The overrun mechanism that generates the lowest request bound function $\mathtt{RBF}_s(t)$ for the subsystem $S_k$, will require the lowest system load. However, $S_k$ may not be the same subsystem when using different overrun mechanisms, and also, at a certain time instance $t$, the value of $\mathtt{RBF}_s(t)$ when using one of the overrun mechanisms will be less than when using another

overrun mechanism, and for another time instance $t$ the value of $\texttt{RBF}_s(t)$ might be less when using the second mechanism. It can be concluded that none of the three overrun mechanisms can perform better than the other two in the general case, as it depends directly of the system parameters.

The comparison between the three overrun mechanisms in terms of request bound function is shown below;

1. **PO vs BO**. Comparing Eq. (7.15) and Eq. (7.17), it easy to show that $\texttt{RBF}_s^{\circ}(t) > \texttt{RBF}_s^{\#}(t)$ for $0 \leq t < P_s$. The reason is that the interference from other higher priority tasks is always $Q_k + H_k$ for both cases and $Q_s^{\circ} > Q_s^{\#}$. If $t > P_s$ then the mechanism that requires a lower request bound function is different depending on the system parameters. It can be concluded that if the subsystem periods of all subsystems are equal, then the BO mechanism will require less system load than using the PO mechanism. Another interesting observation is that if the subsystem that generates maximum $\alpha$ in Eq. (7.29) has the highest priority, then the BO mechanism will require less system load compared to using the PO mechanism. The reason for this is inherent in the subsystem priority; as the subsystem has higher priority, then there will be no interference from other lower priority subsystems.

2. **BO vs EO**. Comparing Eq. (7.17) and Eq. (7.24), it is easy to show that $\texttt{RBF}_s^{*}(t) \geq \texttt{RBF}_s^{\#}(t)$ for $0 \leq t < P_s$. If $t > P_s$, then finding the best mechanism that requires the least system load depends on the system parameters.

3. **PO vs EO**. Comparing Eq. (7.15) and Eq. (7.24), it can be concluded that $\texttt{RBF}_s^{\circ}(t) < \texttt{RBF}_s^{*}(t)$ when $t$ is in the range $kP_s - H_s \geq t < kP_s$ and $\texttt{RBF}_s^{\circ}(t) \geq \texttt{RBF}_s^{*}(t)$ when $t$ is in $(k-1)P_s \geq t < (k)P_s - H_s$ where $k$ is an integer value greater and $k > 0$.

The following examples show some of the cases discussed above:

**Example 1** Suppose that a system $S$ consists of three subsystems with parameters as shown below;

| Subsystem | $P_s$ | $Q^{\circ}$ | $Q^{\#}$ | $Q_s^{*}$ | $H_s$ |
|---|---|---|---|---|---|
| $S_1$ | 20 | 4.7 | 4 | 4 | 2 |
| $S_2$ | 50 | 15 | 12 | 13 | 4 |
| $S_3$ | 200 | 38 | 34 | 36 | 4 |

The global scheduler is EDF. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.79$ and maximum $\alpha$ is at $t = 200$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.81$ and maximum $\alpha$ is at $t = 200$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.69$ and maximum $\alpha$ is at $t = 198$.

**Example 2** Suppose that a system $S$ consists of three subsystems with parameters as shown below;

| Subsystem | $P_s$ | $Q^\circ$ | $Q^\#$ | $Q_s^*$ | $H_s$ |
|---|---|---|---|---|---|
| $S_1$ | 12 | 2 | 1.5 | 2 | 1 |
| $S_2$ | 15 | 3.1 | 2.1 | 3 | 2 |
| $S_3$ | 120 | 15.2 | 14.6 | 15 | 3 |

The global scheduler is EDF. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.57$ and maximum $\alpha$ is at $t = 15$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.60$ and maximum $\alpha$ is at $t = 120$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.65$ and maximum $\alpha$ is at $t = 13$.

**Example 3** Suppose that a system $S$ consists of three subsystems with parameters as shown below;

| Subsystem | $P_s$ | $Q^\circ$ | $Q^\#$ | $Q_s^*$ | $H_s$ | Priority |
|---|---|---|---|---|---|---|
| $S_1$ | 40 | 4.8 | 4 | 4.5 | 1 | High |
| $S_2$ | 40 | 1.9 | 1.5 | 1.75 | 1 | Middle |
| $S_3$ | 40 | 3.1 | 2.6 | 3 | 2 | Low |

The global scheduler is FPS. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.34$ and maximum $\alpha$ is at $t = 40$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.30$ and maximum $\alpha$ is at $t = 40$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.35$ and maximum $\alpha$ is at $t = 38$.

## 7.7 Computing resource holding time

This section explains how to compute the resource holding time $h_{j,i}$, a very important parameter in the global analysis. The resource holding time is the time given by the tasks maximum execution time inside a critical section plus the interference (inside the critical section) of higher priority tasks having preemption level greater than the internal ceiling of the locked resource. That means the internal resource ceiling $rc_j$ is one of the parameters that can have

a great effect on resource holding times of the global shared resources (see Eq. (7.30) and Eq. (7.31)). Setting the value of $rc_j$ according to SRP may make the resource holding times values, very high. One way to handle this problem is by preventing the preemption inside the subsystem when a task is accessing a shared resource. However, Fisher *et al.* [19] showed that preventing preemption while accessing a global shared resource may violate the local schedulability of the subsystem and proposed an algorithm based on increasing the ceiling of all resources in steps as much as possible without violating the local schedulability. Finally, Shin *et al.* [33] showed that there is a tradeoff between decreasing the value of $H_s$ and the minimum subsystem budget required to guarantee the schedulability of the subsystem. The result of this paper does not depend on any of the discussed methods to set the internal resource ceiling.

For non-hierarchical scheduling, Bertogna *et al.* [26] and Fisher *et al.* [26, 19] presented a method to evaluate the resource holding time assuming FPS and EDF scheduling algorithms respectively. The resource holding time $h_{j,i}$ of a shared resource $R_j$ accessed by $\tau_i$ is the smallest positive time $t^*$ such that $W_j(t^*) = t^*$, with $W_j$ computed as follows;

$$W_j(t) = cx_j + \sum_{\tau_k \in U} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{7.30}$$

where $cx_{j,i} = \max\{c_{i,j}\}$ is the maximum execution time of task $\tau_i$ inside the critical section of the resource $R_j$ and $U$ is the set of tasks such that $U = \{\tau_k | \pi_k > rc_j\}$.

Finally, the resource holding time of a resource $R_j$ is $h_j = \max\{h_{j,i}\}$ for all $\tau_i$ access resource $T_j$.

Now for a hierarchical scheduling framework that uses the overrun mechanism, the following equation shows how to evaluate the resource holding time for a task $\tau_i$ that accesses a resource $R_j$.

$$h_{j,i} = cx_{j,i} + \sum_{\tau_k \in U} C_k \tag{7.31}$$

The difference between Eq. (7.31) and Eq. (7.30) is that all tasks that can preempt inside the critical section are assumed to be executed only once. The reason for why it is safe to assume only one execution of each preempting task inside the critical section is given in the following lemma, showing that if a task executes more than one time inside the critical section, the subsystem will become unschedulable.

**Lemma 6.** *For a subsystem that uses an overrun mechanism to arbitrate access to a global shared resource under the periodic virtual processor model, each task that is allowed to preempt the execution of another task currently inside the critical section of a globally shared resource can, in the worst case, only execute (cause interference) once independent if the local scheduler is EDF or FPS.*

*Proof.* This lemma will be proved by contradiction to show that if more than one job of a task preempts a critical section then the system utilization will be greater than one. The following two cases are considered in the proof:

(1) $P_s < T_m$ (where $T_m = min(T_i)$ for all $i = 1, ..n$), if the task having period $T_m$ executes 2 or more times inside the critical section, this means that the resource will be locked during this period, i.e., $h_{j,i} > T_m$ then $h_{j,i} > P_S$, which in turn means that the CPU utilization required by the subsystem $S_s$ will be $U_s = (Q_s + h_{j,i})/P_s > 1$.

(2) If $P_s \geq T_m$ , $\mathtt{sbf}_\Gamma(t)$ should provide at least $C_m$ at time $t = T_m$ to ensure the schedulability test in Eq. (7.3) for the EDF scheduler and in Eq. (7.4) for the FPS scheduler. Note that $\mathtt{sbf}_\Gamma(t) = 0$ during $t \in [0, 2P_s - 2Q_s]$ so, $2P_s - 2Q_s + C_m \leq T_m$ which means $Q_s \geq P_s - T_m/2 + C_m/2$. Then the minimum subsystem budget is,

$$Q_s^{min} = P_s - T_m/2 + C_m/2. \tag{7.32}$$

Let's define $G_s$ as the maximum time in which a subsystem may not get any budget within the subsystem period $P_s$ because of preemptions from other higher priority subsystems, then $G_s = P_s - Q_s$ (see Figure 7.8) and substituting $Q_s$ by the minimum subsystem budget in Eq. (7.32),

$$G_s = (T_m - C_m)/2. \tag{7.33}$$

The maximum number of activations (release) of the $\tau_m$ within $P_s$ while a lower priority task accessing a global shared resource, will happen when $\tau_m$ is release at the beginning of the subsystem period just after the lower priority task has locked a global shared resource (see Figure 7.8). Now lets assume that $\tau_m$ will execute two times while the global shared resource is locked, then the subsystem budget given to the subsystem within the first period of $T_m$ should be low enough such that the shared resource will not be released before the second activation of $\tau_m$. Let's define $L_m$ as the minimum subsystem budget that will be supplied to the subsystem within the first period of $T_m$, $L_m = T_m - G_s$ and from Eq. (7.33) we get,

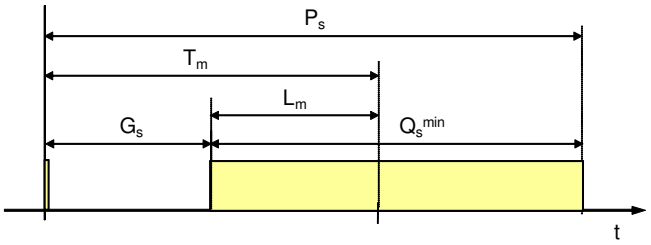


Figure 7.8: Resource holding times for the case $P_s \geq T_m$.

$$L_m = (T_m + C_m)/2. \tag{7.34}$$

The task $\tau_m$ can execute two times while a global shared resource is locked only if $h_{j,i}$ evaluated from Eq. (7.31) is greater than $L_m$ i.e., $h_{j,i} > (T_m + C_m)/2$, otherwise, the shared resource will be released before the new activation of task $\tau_m$. Hence, $Q_s^{min} + h_{j,i} > P_s$ which means $U_s = (Q_s^{min} + h_{j,i})/P_s > 1$.

□

From Lemma 6, it can be concluded that all tasks that can preempt the execution of a critical section should do so maximum one time in order to keep the utilization of a subsystem less than one. If a task preempts the execution of a critical section more than one time then it will be seen from Eq. (7.31) such that $Q_s^{min} + h_{j,i} > P_s$. This proves the correctness of Eq. (7.31) which is based on the assumption that all tasks can interfere only once as a worst case while a task is in the critical section of the resource $R_j$. If the value of $h_{j,i}$ becomes greater than $min(T_m, P_s)$ then it can be concluded that the subsystem will not be schedulable and no further calculation towards finding an exact value of $h_{j,i}$ is needed.

## 7.8 Summary

This paper presents three different overrun mechanisms that all can handle the problem of sharing of logical resources in a hierarchical scheduling framework while at the same time supporting independent subsystem development (open environments). Compared to previous work [3], results have been generalized

by also allowing for the FPS scheduling algorithm for both local and global schedulers, which is suitable for usage in open environments. In addition, a third overrun mechanism, basic overrun without payback (BO), is included in the comparison between the overrun mechanisms. Also, this comparison is performed considering both FPS and EDF scheduling algorithms. The results from this comparison show that it is not trivial to evaluate, in the general case, which overrun method that is better than the other, as their impact on the CPU utilization is highly dependent on global system parameters such as subsystem periods and budgets. Finally, the calculation of resource holding times when using the periodic virtual processor model with both the EDF and FPS scheduling algorithms is presented, as the resource holding time is a very important parameter in the global schedulability analysis.

Future work includes comparing the enhanced overrun mechanism (EO) with other synchronization mechanisms such as BWI [25], the BROE server [22] and SIRAP [21]. In addition, implementing the three overrun mechanisms and comparing the implementation overhead of each mechanism is important. Finally, as the global schedulability analysis gives an upper bound for EO, it will be interesting to find an exact or less pessimistic schedulability analysis.

# Bibliography


[1] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[2] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[3] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Scheduling of semi-independent real-time components: Overrun methods and resource holding times. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, pages 575–582, September 2008.

[4] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[5] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[6] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the 6$^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[7] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments.

In *Proceedings of the* $21^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[8] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the* $7^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[9] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the* $25^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 57–67, December 2004.

[10] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* $23^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[11] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):(30)1–39, April 2008.

[12] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarhical fixed-priority scheduling. In *Proceedings of the* $14^{th}$ *Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, June 2002.

[13] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the* $15^{th}$ *Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[14] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the* $26^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[15] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Proceedings of the* $28^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 129–138, December 2007.

[16] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON'87)*, pages 909–916, November 1987.

[17] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the $9^{th}$ IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, December 1988.

[18] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[19] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in EDF-scheduled systems. In *Proceedings of the* $13^{th}$ *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, 2007.

[20] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the $4^{th}$ ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[21] 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 $7^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[22] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the $28^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[23] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the* $19^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'98)*, pages 4–13, December 1998.

[24] M. Behnam, T. Nolte, M. Åsberg, and I. Shin. Synchronization protocols for hierarchical real-time scheduling frameworks. In *Proceedings of the $1^{st}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08)*, pages 53–60, November 2008.

[25] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization' in reservation-based real-time systems. *Transactions on Computers*, 53(12):1591–1601, December 2004.

[26] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the* $15^{th}$ *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[27] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the* $11^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'90)*, pages 182–190, December 1990.

[28] S. Baruah. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of the* $27^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 379–387, December 2006.

[29] N. Fisher and F. Dewan. Approximate bandwidth allocation for compositional real-time systems. In *Proceedings of* $21^{st}$ *Euromicro Conference on Real-Time Systems (ECRTS'09)*, pages 87–96, 2009.

[30] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the* $10^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'89)*, pages 166–171, December 1989.

[31] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.

[32] F. Dewan and N. Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. Technical report:, Department of Computer Science, Wayne State University, Detroit, USA, 2009.

[33] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Proceedings of the* $29^{th}$ *IEEE Real-Time Systems Symposium (RTSS'08)*, pages 209–22, December 2008.

# Chapter 8

# Paper C: Overrun and Skipping in Hierarchically Scheduled Real-Time Systems

Moris Behnam, Thomas Nolte, Mikael Åsberg, Reinder J. Bril

In Proceedings of the $15^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), pages 519-526, August, 2009.

**Abstract**

Recently, two SRP-based synchronization protocols for hierarchically scheduled real-time systems based on Fixed-Priority Preemptive Scheduling (FPPS) have been presented, i.e., HSRP [1] and SIRAP [2]. Preventing depletion of budget during global resource access, the former implements an overrun mechanism, while the later exploits a skipping mechanism. A theoretical comparison of the performance of these mechanisms revealed that none of them was superior to the other, as their performance is heavily dependent on the system's parameters. To better understand the relative strengths and weaknesses of these mechanisms, this paper presents a comparative evaluation of the depletion prevention mechanisms overrun (with or without payback) and skipping. These mechanisms are investigated in detail and the corresponding system load imposed by these mechanisms is explored in a simulation study. The mechanisms are evaluated assuming FPPS and a periodic resource model [3]. The periodic resource model is selected as it supports locality of schedulability analysis, allowing for a truthful comparison of the mechanisms. Given system characteristics, guiding the design of hierarchically scheduled real-time systems, the results of this paper indicate when one mechanism is better than the other and how a system should be configured in order to operate efficiently.

## 8.1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced as a scheduling approach to support hierarchical CPU sharing among applications under different scheduling services [3]. The HSF can be generally represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., tasks), and resources are allocated from a parent node to its children nodes.

The HSF provides means for decomposing a complex system into well-defined parts called *subsystems*. In essence, the HSF provides a mechanism for timing-predictable *composition* of coarse-grained subsystems. In the HSF a subsystem provides an introspective *interface* that specifies the timing properties of the subsystem precisely [3]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal interference. Temporal isolation between subsystems is provided through budgets which are allocated to subsystems.

**Motivation:** Research on HSFs started with the assumption that subsystems are independent, i.e., inter-subsystem resource sharing other than the CPU fell outside their scope. In some cases [4, 5], intra-subsystem resource sharing is addressed using existing synchronization protocols for resource sharing between tasks, e.g., the Stack Resource Policy (SRP) [6]. Recently, two SRP-based synchronization protocols for inter-subsystem resource sharing in FPPS systems have been presented, i.e., HSRP [1] and SIRAP [2]. An initial comparative assessment of HSF synchronization protocols, based on five criteria, revealed that none of them was superior to the others [7], however. In particular, the performance of the protocol turned out to be heavily dependent on the system parameters.

One of the main differences between these two synchronization protocols is the way they deal with inter-subsystem resource sharing and depletion of budgets. HSRP is based on an *overrun* mechanism (with or without payback), i.e., upon depletion of the budget during global resource access, the budget is temporally increased with a statically determined amount for the duration of that access, whereas SIRAP is based on a *skipping* mechanism, preventing depletion of the budget during global resource access. To better understand the relative strengths and weaknesses of synchronization protocols for HSFs, a comparative evaluation of these underlying mechanisms is presented in this paper.

Although HSRP and SIRAP assume a two-level HSF, it is hard to truthfully compare these protocols in their original settings, because they assume differ-

ent virtual processor models and different scheduling mechanisms. Therefore, the comparison in this paper is performed using a common virtual processor model, the periodic resource model from [3], and a common local and global scheduling mechanism; Fixed-Priority Preemptive Scheduling (FPPS). Schedulability analysis for the overrun mechanism (of HSRP) and the skipping mechanism (of SIRAP) assuming the periodic resource model and FPPS can be found in [8] and [2], respectively.

**Contributions:** In this paper, the efficiency of the mechanisms is shown by exploring the *system load* [7] in a simulation study. The theoretical study of system load shows under which system configuration one mechanism is better than the other.

**Outline:** Section 8.2 presents related work, and Section 8.3 presents the system model and background. Section 8.4 presents a simulation study comparing the system load imposed by overrun and skipping. Finally, Section 8.5 concludes the paper.

## 8.2 Related work

This section presents related work in the areas of hierarchical scheduling and synchronization protocols.

**Hierarchical scheduling** Over the years, there has been a growing attention to hierarchical scheduling of real-time systems [4, 9, 10, 11, 5, 12, 13, 14, 15, 3]. Deng and Liu [10] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [5] presented schedulability analysis techniques for such a two-level framework with the Fixed-Priority Preemptive Scheduling (FPPS) global scheduler. Lipari and Baruah [12, 16] presented schedulability analysis techniques for Earliest Deadline First (EDF) global schedulers. Mok *et al.* [17, 11] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [3] 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 FPPS [4, 9, 13] and under EDF scheduling [3, 18]. However, a common assumption shared by all above studies is that tasks are independent.

**Synchronization** In order to allow for dependencies among tasks, many synchronization protocols have been introduced for arbitrating accesses to shared logical resources, addressing the priority inversion problem, e.g., the Stack Resource Policy (SRP) [6]. For usage in a HSF, additional protocols have been proposed, e.g., the Hierarchical Stack Resource Policy (HSRP) [1], the Subsystem Integration and Resource Allocation Policy (SIRAP) [2] and the Bounded-delay Resource Open Environment (BROE) [19] protocols. The work in this paper concerns the former two, targeting systems implementing FPPS schedulers. To bound the waiting time of tasks from different subsystems that want to access the same shared resource, subsystem budget expiration should be prevented while locking a global shared resource. The following two mechanisms can be used to solve this problem:

**(1) the overrun mechanism** The problem of subsystem budget depletion inside a critical section is handled by adding extra resources to the budget of each subsystem to prevent the budget expiration inside a critical section. HSRP is based on an overrun mechanism. HSRP stops task preemption within the subsystem whenever a task is accessing a global shared resource. SRP is used at the global level to synchronize the execution of subsystems that have tasks accessing global shared resources. Two versions of overrun mechanisms have been presented; 1) *with payback*; whenever overrun happens in a subsystem $S_s$, the budget of the subsystem will, in its next execution instant, be decreased by the amount of the overrun time. 2) *without payback*; no further actions will be taken after the event of an overrun.

**(2) the skipping mechanism** Skipping is another mechanism that prevents a task from locking a shared resource by skipping its execution if its subsystem does not have enough remaining budget at the time when the task tries to lock the resource[1]. SIRAP is based on the skipping mechanism. SIRAP uses the SRP protocol to synchronize the access to global shared resources in both local and global scheduling. SIRAP checks the remaining budget before granting the access to the globally shared resources; if there is sufficient remaining budget then the task enters the critical section, and if there is insufficient remaining budget, the local scheduler delays the critical section entering of the job until the next subsystem budget replenishment (assuming that the subsystem budget

[1]The idea of skipping has been firstly considered in the zone based protocol ZB [20] used in a pfair-scheduling environment, while we use it for hard real-time tasks under hierarchical scheduling.

in the next subsystem budget replenishment is enough to access the global shared resource by the task).

## 8.3 System model and background

This paper focuses on scheduling of a single node, where each node is modeled as a system $\mathcal{S}$ consisting of one or more subsystems $S_s \in \mathcal{S}$. The system is scheduled by a two-level Hierarchical Scheduling Framework (HSF) as shown in Figure 8.1. During runtime, the system level scheduler (global scheduler) selects, at all times, which subsystem will access the common (shared) CPU resource. The synchronization protocols, SRP and HSRP+SIRAP, will mediate access to local and global shared logical resources, respectively.

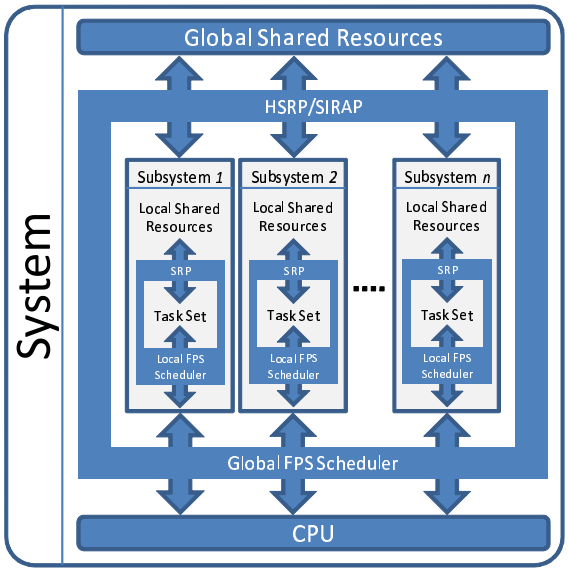


Figure 8.1: HSF with resource sharing.

**Subsystem model** A subsystem $S_s$ consists of a task set $\mathcal{T}_s$ of $n_s$ tasks and a scheduler. Once a subsystem is assigned the processor (CPU), its local scheduler will select which subsystem-internal task will be executed. Each subsys-

tem $S_s$ is associated with a subsystem timing interface $S_s(P_s, Q_s, \{X_{s,j}\})$, where $Q_s$ is the subsystem budget that the subsystem $S_s$ will receive every subsystem period $P_s$, and $X_{s,j}$ is the maximum time that a subsystem internal task may lock a shared resource $R_j$. The subsystem interface is used to specify the collective temporal requirements of a subsystem, and it is used as an interface between the subsystem and the global scheduler. Finally, both the local scheduler of a subsystem $S_s$ as well as the global scheduler of the system $\mathcal{S}$ are assumed to implement the FPPS scheduling policy. Let $\texttt{HP}(s)$ return the set of subsystems with priorities higher than that of $S_s$ and let $\mathcal{R}_s$ be a set of global shared resources accessed by $S_s$. In this paper and for simplicity, $X_s$ is used instead of $\{X_{s,j}\}$ in the subsystem interface. $X_s$ equals to the maximum element in $\{X_{s,j}\}$. Hence, the impact of this simplification will make the results more pessimistic.

**Task model** The task model considered in this paper is the 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 arrival of successive jobs of $\tau_i$, $C_i$ is their worst-case execution-time, and $D_i$ is an arrival-relative deadline ($0 < C_i \leq D_i \leq T_i$) before which the execution of a job must be completed. Each task is allowed to access one or more shared logical resources, and $c_{i,j}$ is a *critical section execution time* that represents a worst-case execution-time requirement of task $\tau_i$ within a critical section of a global shared resource $R_j$ (for simplicity of presentation, we assume that each task accesses a shared resource at most one time). It is assumed that all tasks belonging to the same subsystem are assigned unique static priorities and are sorted according to their priorities in the order of increasing priority. Without loss of generality, it is assumed that the priority of a task is equal to the task ID number after sorting, and the greater a task ID number is, the higher its priority is. The set of shared resources accessed by $\tau_i$ is denoted $\{R^i\}$. Let $\texttt{hp}(i)$ return the set of tasks with priorities higher than the priority of $\tau_i$ and $\texttt{lp}(i)$ return the set of tasks with priorities lower than the priority of task $\tau_i$. For each subsystem, we assume that the subsystem period is selected such that $2P_s \leq T_m$, where $\tau_m$ is the task with the shortest period. This restriction is made for reasons of resource efficiency [21]. Moreover, this assumption simplifies the presentation of the paper (evaluating $X_s$).

**Periodic resource model** The *CPU supply* refers to the amount of CPU allocation that a virtual processor can provide. Shin and Lee [3] proposed the pe-

riodic processor resource model $\Gamma(P, Q)$ to specify periodic CPU allocations, where $P$ is a period ($P > 0$) and $Q$ is a periodic allocation time ($0 < Q \leq P$).

The supply bound function $\texttt{sbf}_\Gamma(t)$ of $\Gamma(P, Q)$ was given in [3] that computes the minimum possible CPU supply for every interval length $t$ as follows:

$$\texttt{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(P-Q) & \text{if } t \in V^{(k)} \\ (k-1)Q & \text{otherwise,} \end{cases} \tag{8.1}$$

where $k = \max\Big(\big\lceil \big(t - (P-Q)\big)/P \big\rceil, 1\Big)$ and $V^{(k)}$ denotes an interval $[(k+1)P - 2Q, (k+1)P - Q]$.

### 8.3.1 Shared resources

The presented HSF allows for sharing of logical resources between arbitrary tasks, located in arbitrary subsystems, 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. At any time, only a single task may hold its lock. 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 denoted a *local shared resource*. The work in this paper targets managing global shared resources, and throughout the remainder of the paper these are simply denoted as shared resources. Management of local shared logical resources can be done by using one of several existing synchronization protocols. In this paper, local shared resources are managed by SRP.

To be able to use SRP in a HSF for synchronizing global shared resources, its associated terms resource, system and subsystem ceilings are extended as follows:

**Resource ceiling** Each global shared resource $R_j$ is associated with two types of resource ceilings; an *internal* resource ceiling ($rc_j$) for local scheduling and an *external* resource ceiling ($RX_j$) for global scheduling. They are defined as $rc_j = \max\{i | \tau_i \in \mathcal{T}_s \text{ accesses } R_j\}$ and $RX_j = \max\{s | S_s \text{ accesses } R_j\}$. However, assigning an internal resource ceiling according to SRP makes the value of $X_s$ very high which makes the subsystem require more CPU resources. Note that HSRP prevents any task preemption locally (within the subsystem) while accessing shared resources, which can be implemented using SRP with internal resource ceiling equal to the maximum task priority $rc_j = n_s$. [22] showed that preventing preemption while accessing a global

shared resource may violate the local schedulability of the subsystem, and proposed an algorithm based on increasing the ceiling of all resources[2] in steps as much as possible without violating the local schedulability.

**System/subsystem ceiling** The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling of a currently locked resource in the system/subsystem.

Under SRP, a task $\tau_k$ can preempt the currently executing task $\tau_i$ (even inside a critical section) within the same subsystem, only if the priority of $\tau_k$ is greater than its corresponding subsystem ceiling. The same reasoning can be made for subsystems from a global scheduling point of view.

### 8.3.2 Schedulability analysis

In order to be able to compare the overrun and skipping mechanisms when used by synchronization protocols in a HSF, HSRP and SIRAP have been implemented in the same HSF allowing for a fair comparison of their performance. HSRP comes in two flavors, with and without payback, denoted as *Overrun With Payback* (OWP) and *Overrun with No Payback* (ONP), respectively. SIRAP implements the *Skipping* (SKP) mechanism. Central to the discussions later in the paper, the general local and a global schedulability analysis, independent on a particular mechanism, is first presented followed by its modifications for each one of the three mechanisms.

**General local schedulability analysis** The general local schedulability analysis under FPPS is as follows [6, 3]:

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; \texttt{rbf}_i(t) \leq \texttt{sbf}(t), \tag{8.2}$$

where $\texttt{rbf}_i(t)$ denotes the request bound function of a task $\tau_i$ (later it is shown how to calculate $\texttt{rbf}_i(t)$, which is dependent on the synchronization protocol in use). Note that $t$ can be selected within a finite set of scheduling points [23].

[2]Lowering the value of $X_s$ may not always decrease the minimum required CPU resources, it may increase it as shown in [8], however this issue is out of the scope of this paper and left for future work.

**General global schedulability analysis** The general condition for global schedulability is

$$\forall S_s \ \exists t : 0 < t \leq P_s, \ \ \texttt{RBF}_s(t) + B_s \leq t, \tag{8.3}$$

where $\texttt{RBF}_s(t)$ denotes the request bound function of a subsystem $S_s$, and $B_s$ is the maximum blocking imposed to a subsystem $S_s$, when it is blocked by lower-priority subsystems (suppose that $S_j$ imposes the maximum blocking on $S_s$ then $B_s = X_j$) . Note that the way of calculating $\texttt{RBF}_s(t)$ depends on the synchronization protocol.

**Overrun mechanism without payback** $\texttt{rbf}_i(t)$ using ONP is calculated as follows [8];

$$\texttt{rbf}_i(t) = C_i + b_i + \sum_{\tau_k \in \texttt{hp}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{8.4}$$

where $b_i$ is the maximum *blocking* imposed to a task $\tau_i$ by lower priority tasks that access resources with ceiling greater than or equal to the priority of $\tau_i$ (i.e., $b_i = \max_{\tau_k \in \texttt{lp}(i) \wedge rc_j \geq i}(c_{k,j})$).

$\texttt{RBF}_s(t)$ can be calculated as follows;

$$\texttt{RBF}_s(t) = (Q_s + X_s) + \sum_{S_k \in \texttt{HP}(s)} \left\lceil \frac{t}{P_k} \right\rceil (Q_k + X_k) \tag{8.5}$$

**Overrun mechanism with payback** Eq. (8.4) and Eq. (8.2) can be used for local schedulability analysis for the OWP mechanism, however, calculating the supply bound function $\texttt{sbf}(t)$ will be different. Lets $\texttt{sbf}^*(t)$ be the supply bound function when using the overrun mechanism with payback, then $\texttt{sbf}^*(t)$ can be evaluated as follows [24];

$$\texttt{sbf}^*(t) = \max(\texttt{sbf}(t) - X_s, 0). \tag{8.6}$$

Eq. (8.7) is used to calculate $\texttt{RBF}_s(t)$.

$$\texttt{RBF}_s(t) = (Q_s + X_s) + \sum_{S_k \in \texttt{HP}(s)} \left( \left\lceil \frac{t}{P_k} \right\rceil \cdot Q_k + X_k \right) \tag{8.7}$$

**Skipping mechanism** $\texttt{rbf}_i(t)$ can be computed for SKP as follows [2];

$$\texttt{rbf}_i(t) = C_i + I_i^S + I_i^H(t) + b_i, \tag{8.8}$$

where $I_i^S$ is the task self blocking, $I_i^H(t)$ is the interference from higher priority tasks and $b_i$ is the interference from the lower priority tasks,

$$I_i^S = \sum_{R_j \in \{R^i\}} X_{i,j}, \tag{8.9}$$

$$I_i^H(t) = \sum_{\tau_k \in \texttt{hp}(i)} \left\lceil \frac{t}{T_k} \right\rceil (C_k + \sum_{R_j \in \{R^k\}} X_{k,j}), \tag{8.10}$$

$$b_i = \max_{\tau_k \in \texttt{lp}(i) \wedge rc_j \geq i} (2 \cdot (X_{k,j})). \tag{8.11}$$

Eq. (8.12) is used to evaluate $\texttt{RBF}_s(t)$ when using the SKP mechanism:

$$\texttt{RBF}_s(t) = Q_s + \sum_{S_k \in \texttt{HP}(s)} \left\lceil \frac{t}{P_k} \right\rceil \cdot Q_k. \tag{8.12}$$

**Calculating $X_s$**

Given a subsystem $S_s$, its critical section execution time $X_s$ represents a worst-case CPU demand that internal tasks of $S_s$ may collectively request while executing inside any critical section. Note that any task $\tau_i$ accessing a resource $R_j$ can be preempted by tasks with priority higher than the internal resource ceiling $rc_j$ of $R_j$. Lets denote the maximum time that an internal task $\tau_i$ locks $R_j$ within the subsystem by $X_{i,j}$. Note that both HSRP and SIRAP prevent subsystem budget expiration inside a critical section of a global shared resource. To ensure the global schedulability analysis, $X_{i,j} < P_s$ and since we assume that $2P_s \leq T_m$ then all tasks that are allowed to preempt while $\tau_i$ accesses $R_j$ will be activated at most one time. Otherwise, only if $X_{i,j} > P_s$, tasks will be able to execute more than one time while locking a global shared resource and that means the system is not schedulable globally. Then $X_{i,j}$ can be computed as

$$X_{i,j} = c_{i,j} + \sum_{k=rc_j+1}^{n_s} C_k. \tag{8.13}$$

Let $X_j = \max(X_{i,j} |$ for all $\tau_i$ accessing resource $R_j)$, then $X_s = \max(X_j |$ for all $R_j \in \mathcal{R}_s)$.

**Subsystem budget** In this paper, it is assumed that the subsystem period is given while the minimum subsystem budget should be computed so that the system will require as little CPU resources as possible. Given a subsystem $S_s$, and $P_s$, let $\mathsf{calculateBudget}(S_s, P_s)$ denote a function that calculates the smallest subsystem budget $Q_s$ that satisfies Eq. (8.2). Hence, $Q_s = \mathsf{calculateBudget}(S_s, P_s)$ (the function is similar to the one presented in [3]). Note that for both SKP and OWP, the following condition should be satisfied; $Q_s \geq X_s$. For SKP, this condition is sufficient to make sure that if $\tau_i$ blocks itself then it will execute during the next subsystem period since $2P_s \leq T_m$.

## 8.4 Comparing overrun and skipping

As described in [7], none of the synchronization mechanisms is superior to the others, because their performance heavily depends on the system parameters. The performance of the mechanisms is measured by the amount of CPU resource required at system level (globally) in order to guarantee the schedulability of the subsystems and their internal tasks. The mechanism that requires the lowest CPU resource is considered as the best among all mechanisms. One way to compare the performance of the mechanisms is by comparing the interfaces of the subsystems using each mechanism. Although this comparison can be useful at subsystem level, it is not useful at the system level, i.e., when subsystems are integrated, each of the mechanisms schedule subsystems globally in a different way. In this section, we therefore compare the mechanisms using the notion of system load [25] since it provides an indication of the system CPU requirement in the presence of shared resources. The comparison is done by means of simulation experiments. We start this section by briefly recapitulating the notion of system load. Next, we describe the setup of our experiments, which is followed by the results.

### 8.4.1 System load

For comparison purposes, *system load* is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the global schedulability of the system $\mathcal{S}$. System load $\mathsf{load}_{\mathsf{sys}}$ is calculated as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\} \tag{8.14}$$

where

$$\alpha_s = \min_{0<t\leq P_s} \{\frac{\texttt{RBF}_s(t) + B_s}{t} \mid \texttt{RBF}_s(t) + B_s \leq t\}. \quad (8.15)$$

Note that $\alpha_s$ is the smallest fraction of the CPU resources that is required to schedule a subsystem $S_s$ (satisfying Eq. (8.3)) assuming that the global resource supply function is $\alpha_s t$.

Given a system consisting of more than one subsystem, each subsystem containing a set of tasks, an efficient synchronization mechanism is the one that produces the lowest system load $\mathsf{load}_{\mathsf{sys}}$ for this system. IF $\mathsf{load}_{\mathsf{sys}} > 1$ then the system is not schedulable.

### 8.4.2 Experiment definition

The simulation study is performed by applying the synchronization mechanisms ONP, OWP, and SKP on 1000 different randomly generated systems using the schedulability analysis presented in section 8.3.2, where each system consists of 5 subsystems[3] and, in turn, each subsystem contains 8 tasks. Once a system is generated, it is analyzed by applying the three synchronization mechanisms. The internal resource ceiling of the globally shared resources is assumed to be equal to the highest task priority in each subsystem and we assume that $T_i = D_i$ for all tasks. 2-6 tasks access globally shared resources in each subsystem. The worst-case critical section execution time of a task $\tau_i$ is set to a value between $0.3C_i$ and $0.8C_i$[4]. A task is assumed to access at most one globally shared resource. For each system, the number of the global shared resources is two and all subsystems have at least two internal tasks that access both global shared resources. For each simulation study the following settings are changed and a new 1000 systems is generated:

1. System utilization – the system utilization, i.e., the summation of the utilization of all tasks in the system, is specified to a desired value.
2. Subsystem period – the subsystem period is specified as a range with a lower and upper bound. The simulation program generates a subsystem period randomly within the specified range, following a uniform distribution.
3. Task period – the task period is specified and generated in the same way as the period of a subsystem.

[3] We have tested several scenarios and the number of subsystems does not appear to have a significant impact on the system load.

[4] Having a lower range of the critical section execution times does not yield a significant difference between the mechanisms, which makes it hard to highlight their individual differences.

The system utilization is divided randomly among the subsystems and the assigned utilization to each subsystem is in turn divided randomly to the tasks that belong to that subsystem. Since the task period is generated to a value within the interval as specified, the execution time is derived from the desired task utilization. All randomized system parameters are generated following uniform distributions.

### 8.4.3 Simulation results

Tables 8.1-8.4 show the results of the 4 different simulation studies performed.

- **Study 1** is specified having a task utilization of 15%, task periods between 400 and 1000, and subsystem periods between 50 and 200.
- **Study 2** decrease task periods (compared to Study 1) to the interval of 400-450, and therefore also $X_s$ (a task will have the same utilization compare to Study 1 and since its period is lower then its execution time becomes lower).
- **Study 3** decrease subsystem periods (compared to Study 1) to the interval of 50-60.
- **Study 4** increase task utilization (compared to Study 1) to 30%, and therefore also $X_s$ (a task utilization will be higher and that increases its execution time).

In each of these studies, 1000 systems are randomly generated, and for each generated system all three synchronization mechanisms are applied. For each case, the $\mathsf{load}_{\mathsf{sys}}$ is calculated hence making it possible to determine which of the three mechanisms requires the lowest $\mathsf{load}_{\mathsf{sys}}$ for that particular system. Keeping track of all calculated $\mathsf{load}_{\mathsf{sys}}$ values allow for determining min, max and average of the set of 1000 systems. Also, looking at the 1000 systems, it is possible to determine how often a particular mechanism requires less CPU resource than the other (shown under "Best" row in Tables 8.1-8.3). For reference, the same calculation is performed without sharing any global resources as well, to indicate the cost of running the synchronization protocols, and it is shown in the tables under HSF.

#### Skipping vs. overrun

Looking at Table 8.1, the results from using SKP is relatively better than both other mechanisms. The reason for this is that the range of task periods is much

| | ONP | OWP | SKP | HSF |
|---|---|---|---|---|
| Average | $0,48$ | $0,48$ | $0,42$ | $0,3$ |
| Min | $0,36$ | $0,33$ | $0,31$ | $0,24$ |
| Max | $0,70$ | $0,85$ | $0,61$ | $0,46$ |
| Best | $7\%$ | $3\%$ | $90\%$ | - |

Table 8.1: Measured results of Study 1

higher than the range of subsystem periods. Decreasing the range of task periods and keeping the subsystem period range forces SKP to require a higher $\mathsf{load}_{\mathsf{sys}}$, as show in Table 8.2.

| | ONP | OWP | SKP | HSF |
|---|---|---|---|---|
| Average | $0,47$ | $0,48$ | $0,46$ | $0,33$ |
| Min | $0,36$ | $0,34$ | $0,32$ | $0,24$ |
| Max | $0,62$ | $0,64$ | $0,63$ | $0,43$ |
| Best | $40\%$ | $10\%$ | $50\%$ | - |

Table 8.2: Measured results of Study 2

On the other hand, decreasing the range of subsystem periods and keeping the range of task periods the same as in Study 1 makes the performance of SKP much better than the other two mechanisms, as shown in Table 8.3.

| | ONP | OWP | SKP | HSF |
|---|---|---|---|---|
| Average | $0,58$ | $0,74$ | $0,39$ | $0,37$ |
| Min | $0,41$ | $0,45$ | $0,28$ | $0,23$ |
| Max | $0,80$ | $> 1$ | $0,58$ | $0,57$ |
| Best | $0\%$ | $0\%$ | $100\%$ | - |
| Schedulable | $100\%$ | $99\%$ | $100\%$ | $100\%$ |

Table 8.3: Measured results of Study 3

To explain the reason, comparing Eq. (8.4) and Eq. (8.8) (local schedulability analysis), $\mathtt{rbf}_i(t)$ is higher when using SKP compared to when using ONP and maybe OWP (inherent in the effect of self blocking that is added in Eq. (8.8)) which makes the minimum required budget higher when using SKP. However, the difference between the subsystem budget when using SKP and when using ONP and OWP depends on the following parameters; subsystem period $P_s$, task period $T_i$, critical section execution times of the shared

resources $c_{i,j}$ and $X_s$ for OWP (see Eq. (8.6)). For SKP, if $T_i \gg P_s$ then a small increment in the subsystem budget may be enough to cover the effect of self blocking on the $\texttt{rbf}_i(t)$ (to satisfy the schedulability condition $\texttt{rbf}_i(t) = \texttt{sbf}(t)$ at $t = T_i$).

On the other hand, the effect of using ONP and OWP appears in the calculation of $\texttt{RBF(t)}$ (global schedulability) as the maximum overrun $X_s$, added to $\texttt{RBF(t)}$ (see Eq. (8.5) and Eq. (8.7)). Hence, if the difference between subsystem budget when using SKP, and subsystem budget when using either ONP or OWP, is much less than $X_s$, then SKP will require a lower $\mathsf{load}_{\mathsf{sys}}$ as the $\mathsf{load}_{\mathsf{sys}}$ depends on $\texttt{RBF(t)}$ (see Eq. (8.14) and Eq. (8.15)).

| | ONP | OWP | SKP | HSF |
|---|---|---|---|---|
| Average | $0,92$ | $0,92$ | $0,80$ | $0,60$ |
| Min | $0,67$ | $0,68$ | $0,60$ | $0,46$ |
| Max | $> 1$ | $> 1$ | $> 1$ | $0,91$ |
| Best | 5% | 3% | 92% | - |
| Schedulable | 71% | 76% | 99% | 100% |

Table 8.4: Measured results of Study 4

The results of Study 4 are outlined in Table 8.4. In Study 4 the system utilization is increased which in turn increases the values of the critical section execution time and that increases the value of $X_s$. In this study it is clear that SKP is performing better than the other two mechanisms. Following the same reasoning as above, if the maximum execution time inside a critical section becomes larger, and the range of task periods is much higher than the range of subsystem periods, then SKP will perform better than ONP and OWP.

**No payback vs. payback**

Comparing the results of ONP and OWP in Table 8.1 and Table 8.3, both OWP and ONP gives approximately the same results in Tables 8.1, 8.2 and 8.4. The reason is that OWP requires larger subsystem budget $Q_s$ since $\texttt{sbf}^*(t) < \texttt{sbf}(t)$ (see section 8.3.2) and in the global schedulability, OWP can perform better as $X_s$ is added only one time when computing $\texttt{RBF}_s(t)$ (see Eq. (8.7)). We can conclude that the performance of the mechanisms depends on the size of $X_s$ and the subsystems periods. In Table 8.3 ONP is better than OWP, the difference between the lowest and the highest subsystem period in Table 8.1 (range is 50-200) is much more than the same difference in Table 8.3 (range is 50-60). Hence, ONP requires a lower $\mathsf{load}_{\mathsf{sys}}$ compared with that of using

OWP if the subsystem periods are equal or close to each other. The reason for this is explained as follows; the subsystem budget when using ONP is always less than the subsystem budget when using OWP because:

1. When using OWP, $X_s \leq Q_s$ should hold and if not then the subsystem budget should be increased to $Q_s = X_s$, while for ONP, $Q_s$ can be less or greater than $X_s$ without any problem, i.e., the subsystem budget is not required to be increased.
2. As mentioned previously, OWP requires more subsystem budget than ONP to guarantee the schedulability of the subsystem.

If the subsystem periods are equal, or close to each other, then Eq. (8.7) and Eq. (8.5) will have their main difference in the budget inherent in the use of OWP and ONP, respectively. On the other hand, if the difference in subsystem period is high then the interference from higher priority subsystems will be higher in Eq. (8.5) than in Eq. (8.7) which makes OWP perform better than ONP[5].

#### Number of shared resources

One of the factors that can decrease the performance of SKP is the number of shared resources accessed by tasks that belong to a subsystem. Increasing the number of shared resources will increase the $\texttt{rbf}_i(t)$ of task $\tau_i$ since it sums the critical section execution times as shown in Eq. (8.9)-(8.11), while for the overrun mechanism, calculating $\texttt{rbf}_i(t)$ includes only the maximum blocking from lower priority task. We can conclude that SKP performs better if a low number of resources are accessed by tasks.

## 8.5 Summary

In this paper, a simulation study has been performed on two different synchronization mechanisms; the overrun mechanism and the skipping mechanism. These mechanisms are used to enable the sharing of global logical resources in a Hierarchical Scheduling Framework (HSF), and the study investigates their efficiency in terms of CPU resource requirements. The results from the simulation study show that skipping can perform better than the overrun mechanism if the task periods are much larger than their subsystem period. Otherwise and for a high difference between subsystems periods, the overrun with payback

[5]The same conclusion was shown in [1] and it is presented in this paper to show that it is also valid with the scheduling framework that is used in this paper.

can give better results, and for equal or close to equal subsystem periods, the overrun without payback performs better. Future work include implementing and testing the mechanisms on real industrial systems. Also, it would be interesting to develop an approach that selects different protocols, on subsystem level, in order to minimize the overall amount of required CPU resources.

# Bibliography


[1] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[2] 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 7$^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[3] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[4] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4$^{th}$ ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[5] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[6] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[7] M. Behnam, T. Nolte, M. Åsberg, and I. Shin. Synchronization protocols for hierarchical real-time scheduling frameworks. In *Proceedings of the 1$^{st}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08)*, pages 53–60, November 2008.

[8] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Proceedings of the* $29^{th}$ *IEEE Real-Time Systems Symposium (RTSS'08)*, pages 209–22, December 2008.

[9] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the* $26^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[10] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[11] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* $23^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[12] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the* $6^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[13] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the* $15^{th}$ *Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[14] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the* $26^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 99–110, December 2005.

[15] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarhical fixed-priority scheduling. In *Proceedings of the* $14^{th}$ *Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, June 2002.

[16] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the* $21^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[17] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the* $7^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[18] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the $28^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, December 2007.

[19] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the $28^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[20] Holman P and J. Anderson. Locking in pfair-scheduled multiprocessor systems. In *Proceedings of the $23^{rd}$ International IEEE Real-Time Systems Symposium (RTSS'02)*, pages 149–158, December 2002.

[21] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *IEEE Transaction on Embedded Computing Systems*, 7(3):1–39, 2008.

[22] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the $15^{th}$ International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[23] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.

[24] M. Behnam, M. Sjödin T. Nolte, and I. Shin. Overrun methods for semi-independent real-time hierarchical scheduling. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-237/2009-1-SE, Mälardalen University, June 2009.

[25] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Scheduling of semi-independent real-time components: Overrun methods and resource holding times. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, pages 575–582, September 2008.

# Chapter 9

# Paper D: Bounding the Number of Self-Blocking Occurrences of SIRAP

Moris Behnam, Thomas Nolte, Reinder J. Bril

In Proceedings of the $31^{th}$ IEEE International Real-Time Systems Symposium (RTSS'10), December, 2010.

**Abstract**

In this paper we have developed a new schedulability analysis for hierarchically scheduled real-time systems executing on a single processor using SIRAP; a synchronization protocol for inter subsystem task synchronization. We have shown that it is possible to bound the number of selfblocking occurrences that should be taken into consideration in the schedulability analysis of subsystems, and correspondingly developed and proved correctness of two novel schedulability analysis approaches for SIRAP. An evaluation suggests that this new schedulability analysis can decrease the analytical subsystem utilization significantly.

## 9.1 Introduction

The amount of functionality realized by software in modern embedded systems has steadily increased over the years. More and more software functions have to be developed, implemented and integrated on a common shared hardware architecture. This often results in very complex software systems, where the functions both are dependent on each other for proper operation, and are interfering with each other in terms of, e.g., resource usage and temporal performance.

To remedy this problem inherent in hosting a large number of software functions on the same hardware, research on *platform virtualization* has received an increased interest. Looking at real-time systems, research has focused on partitioned scheduling techniques for single processor architectures, which includes *hierarchical scheduling* where the CPU is hierarchically shared and scheduled among software partitions that can be allocated to the system functions. Hierarchical scheduling can be represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., tasks), and CPU resources are allocated from a parent node to its children nodes. Hence, using hierarchical scheduling techniques, a system can be decomposed into well-defined parts called *subsystems*, each of which receives a dedicated CPU-budget for execution. These subsystems may contain tasks and/or other subsystems that are scheduled by a so-called *subsystem internal scheduler*. Tasks within a subsystem can be allowed to synchronize on logical resources (for example a data structure, a memory map of a peripheral device, etc.) requiring mutually exclusive access by the usage of traditional synchronization protocols such as, e.g., the stack resource policy (SRP) [1]. More recent research has been conducted towards allowing tasks to synchronize on logical resources requiring mutual exclusion across subsystem boundaries, i.e., a task resident in one subsystem shall be allowed to get exclusive access to a logical resource shared with tasks from other subsystems (*global shared resource*). To prevent excessive blocking of subsystems due to budget depletion during global shared resource access, advanced protocols are needed.

One such synchronization protocol for hierarchically scheduled real-time systems executing on a single processor is the subsystem integration and resource allocation policy (SIRAP) [2], which prevents budget depletion during global resource access. SIRAP has been developed with a particular focus of simplifying parallel development of subsystems that require mutually exclusive access to global shared resources. However, a challenge with hierarchical

scheduling is the complexity of performing (or formulating) a tight (preferably exact) analysis of the system behavior. Schedulability analysis typically relies on some simplified assumptions and when the system under analysis is complex, the negative effect of these simplifying assumptions can be significant.

In this paper we look carefully at SIRAP's exact behavior and we identify sources of pessimism in its original *local* schedulability analysis, i.e. the analysis of the schedulability of tasks of a subsystem. By bounding the number of self-blocking occurrences[1] that are taken into consideration in the analysis, we develop two new and tighter schedulability analysis approaches for SIRAP assuming fixed-priority pre-emptive scheduling (FPPS). We present proofs of correctness for the two approaches, and an evaluation shows that they can decrease the analytical subsystem utilization. In addition, the evaluation shows that neither approach is always better than the other. The efficiency of these new approaches is shown to be correlated with the nature of the system and in particular the number of accesses made to logical shared resources.

The outline of this paper is as follows: Section 9.2 outlines related work. In Section 9.3 we present our system model and background. Section 9.4 outlines the SIRAP protocol followed by an example motivating the development of a new schedulability analysis in Section 9.5. Section 9.6 presents our new analysis, which is evaluated in Section 9.7. Finally, Section 9.8 concludes the paper.

## 9.2 Related work

Over the years, there has been a growing attention to hierarchical scheduling of real-time systems. Deng and Liu [4] proposed a two-level Hierarchical Scheduling Framework (HSF) for open systems, where subsystems may be developed and validated independently. Kuo and Li [5] presented schedulability analysis techniques for such an HSF assuming a FPPS system level scheduler. Mok *et al.* [6, 7] proposed the bounded-delay virtual processor model to achieve a clean separation between applications in a multi-level HSF. In addition, Shin and Lee [8] introduced the periodic resource model (to characterize the periodic CPU allocation behavior), and many studies have been proposed on schedulability analysis with this model under FPPS [9, 10] and under Earliest Deadline First (EDF) scheduling [8, 11]. However, a common assumption

[1] A simpler version of bounding self-blocking was presented in [3]. That paper assumes the same maximum self-blocking at every budget supply, which in our case may make the results more pessimistic than the original analysis of SIRAP. In this paper, we consider the maximum possible self-blocking that may occur at each budget supply.

shared by all above studies is that tasks are independent.

Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [12], BROE [13], and SIRAP [2]. Unlike SIRAP, HSRP does not support subsystem level (local) schedulability analysis of subsystems, and the system level schedulability analysis presented for BROE is limited to EDF and can not be generalized to include other scheduling policies.

## 9.3 System model and background

We consider a two-level HSF using FPPS at both the system as well as the subsystem level[2], and the system is executed on a single processor.

**System model** A system contains a set $\mathcal{R}$ of $M$ global logical resources $R_1, R_2, \ldots, R_M$, a set $\mathcal{S}$ of $N$ subsystems $S_1, S_2, \ldots, S_N$, and a set $\mathcal{B}$ of $N$ budgets for which we assume a periodic resource model [8]. Each subsystem $S_s$ has a dedicated budget associated to it. In the remainder of the paper, we leave budgets implicit, i.e. the timing characteristics of budgets are taken care of in the description of subsystems. Subsystems are scheduled by means of FPPS and have fixed, unique priorities. For notational convenience, we assume that subsystems are indexed in priority order, i.e. $S_1$ has highest and $S_N$ has lowest priority.

**Subsystem model** A subsystem $S_s$ contains a set $\mathcal{T}_s$ of $n_s$ tasks $\tau_1, \tau_2, \ldots, \tau_{n_s}$ with fixed, unique priorities that are scheduled by means of FPPS. For notational convenience, we assume that tasks are indexed in priority order, i.e. $\tau_1$ has highest and $\tau_{n_s}$ has lowest priority. The set $\mathcal{R}_s$ denotes the subset of global logical resources accessed by $S_s$. The maximum time that a task of $S_s$ may lock a resource $R_k \in \mathcal{R}_s$ is denoted by $X_{sk}$. This maximum resource locking time $X_{sk}$ includes the critical section execution time of the task that is accessing the global shared resource $R_k$ and the maximum interference from higher priority tasks, within the same subsystem, that will not be blocked by the global shared resource $R_k$. The timing characteristics of $S_s$ are specified by means of a subsystem timing interface $S_s(P_s, Q_s, \mathcal{X}_s)$, where $P_s$ denotes the

[2]Because the improvements only concern schedulability of subsystems, system level scheduling is not important for this paper. We also assume FPPS at the system level scheduler for ease of presentation of the model.

(budget) period, $Q_s$ the budget that $S_s$ will receive every subsystem period $P_s$, and $\mathcal{X}_s$ the set of maximum resource locking times $\mathcal{X}_s = \{X_{sk}\} | \forall R_k \in \mathcal{R}_s$.

**Task model** We consider the deadline-constrained sporadic hard real-time task model $\tau_i(T_i, C_i, D_i, \{c_{ika}\})$[3], where $T_i$ is a minimum inter-arrival time of successive jobs of $\tau_i$, $C_i$ is the worst-case execution-time of a job, and $D_i$ is an arrival-relative deadline ($0 < C_i \leq D_i \leq T_i$) before which the execution of a job must be completed. Each task is allowed to access an arbitrary number of global shared resources (also nested) and the same resource multiple times. The set of global shared resources accessed by $\tau_i$ is denoted by $\{R^i\}$. The number of times that $\tau_i$ accesses $R_k$ is denoted by $rn_{ik}$. The worst-case execution-time of $\tau_i$ during the $a^{th}$ access to $R_k$ is denoted by $c_{ika}$. For each subsystem $S_s$, and without loss of generality, we assume that the subsystem period is selected such that $2P_s \leq T_s^{\min}$, where $T_s^{\min}$ is the shortest period of all tasks in $S_s$. The motivation for this assumption is that it simplifies the evaluation of resource locking time and in addition, allowing a higher $P_s$ would require more CPU resources [14].

**Shared resources** To access a shared resource $R_k$, a task must first lock the shared resource, and the task unlock the shared resource when the task no longer needs it. The time during which a task holds a lock is called a *critical section*. For each logical resource, at any time, only a single task may hold its lock.

SRP is a synchronization protocol proposed to bound the blocking time of higher priority tasks sharing logical resources with other lower priority tasks. SRP can limit the blocking time that a high priority task can face, to the maximum critical section execution time of a lower priority task that shares the same resource with $\tau_i$. SRP associates a resource priority for each shared resource called *resource ceiling* which equals to the priority of the highest priority task (i.e. lowest task index) that accesses the shared resource. In addition, and during runtime, SRP uses *system ceiling* to track the highest resource ceiling (i.e. lowest task index) of all resources that are currently locked. Under SRP, a task $\tau_i$ can preempt the currently executing task $\tau_j$ only if $i < j$ and the priority of $\tau_i$ is greater than the current value of the system ceiling.

To synchronize access to global shared resources in the context of hierarchical scheduling, SRP is used in both system and subsystem level scheduling

[3]Because we only consider local schedulability analysis, we omit the subscript "s" from the task notation representing the subsystem to which tasks belong.

and to enable this, SRP's associated terms *resource*, *system* ceiling should be extended as follows:

*Resource ceiling*: With each global shared resource $R_k$, two types of resource ceilings are associated; an *internal* resource ceiling ($rc_{sk}$) for local scheduling and an *external* resource ceiling ($RX_k$) for system level scheduling. They are defined as $rc_{sk} = \min\{i|\tau_i \in T_s \wedge R_k \in \{R^i\}\}$ and $RX_k = \min\{s|S_s \in \mathcal{S} \wedge R_k \in \mathcal{R}_s\}$.

*System/subsystem ceiling*: The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling (i.e. highest priority) of a currently locked resource in the system/subsystem.

## 9.4 SIRAP

SIRAP prevents depletion of CPU capacity during global resource access through *self-blocking* of tasks. When a job wants to enter a critical section, it first checks the remaining budget $Q^r$ during the current period. If $Q^r$ is sufficient to complete the critical section, then the job is granted entrance, and otherwise entrance is delayed until the next subsystem budget replenishment, i.e. the job blocks itself. Conforming to SRP, the subsystem ceiling is immediately set to the internal resource ceiling $rc$ of the resource $R$ that the job wanted to access, to prevent the execution of tasks with a priority lower than or equal to $rc$ until the job releases $R$. The system ceiling is only set to the external resource ceiling $RX$ of $R$ when the job is granted entrance.

Figure 9.1 illustrates an example of a self-blocking occurrence during the execution of subsystem $S_s$. A job of a task $\tau_i \in \mathcal{T}_s$ tries to lock a global shared resource $R_k$ at time $t_2$. It first determines the remaining subsystem budget $Q^r$ (which is equal to $Q^r = Q_s - (Q^1 + Q^2)$, i.e., the subsystem budget left after consuming $Q^1+Q^2$). Next, it checks if the remaining budget $Q^r$ is greater than or equal to the maximum resource locking time ($X_{ika}$)[4] of the $a^{th}$ access of the job to $R_k$, i.e., if ($Q^r \geq X_{ika}$). In Figure 9.1, this condition is not satisfied, so $\tau_i$ blocks itself and is not allowed to execute before the next replenishment period ($t_3$ in Figure 9.1) and at the same time, the subsystem ceiling is set to $rc_{sk}$.

Self-blocking of tasks is exclusively taken into account in the local schedulability analysis. To consider the worst-case scenario during self-blocking, we assume that the $a^{th}$ request of $\tau_i$ to access a global shared resource $R_k$ always

[4]How to determine $X_{ika}$ will be explained in the next subsection.

happens when the remaining budget is less than $X_{ika}$ by a very small value. Hence, $X_{ika}$ is the maximum amount of budget that $\tau_i$ can not use during self-blocking (also called the *self-blocking of* $\tau_i$). The effect of the interference from higher priority subsystems is exclusively taken into account in system level schedulability analysis; see [2] for more details.

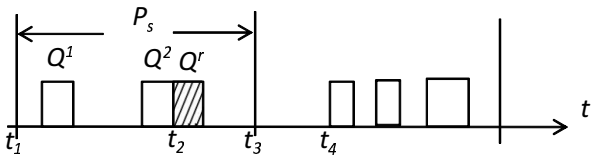


Figure 9.1: An example illustrating self-blocking.

**Local schedulability analysis** The local schedulability analysis under FPPS is given by [8]:

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; \mathtt{rbf_{FP}}(i,t) \leq \mathtt{sbf_s(t)}, \tag{9.1}$$

where $\mathtt{sbf_s(t)}$ is the *supply bound function* that computes the minimum possible CPU supply to $S_s$ for every time interval length $t$, and $\mathtt{rbf_{FP}}(i,t)$ denotes the *request bound function* of a task $\tau_i$ which computes the maximum cumulative execution requests that could be generated from the time that $\tau_i$ is released up to time $t$. $\mathtt{sbf_s(t)}$ is based on the periodic resource model presented in [8] and is calculated as follows:

$$\mathtt{sbf_s(t)} = \begin{cases} t - (g(t)+1)(P_s - Q_s) & \text{if } t \in V^{g(t)} \\ (j-1)Q_s & \text{otherwise,} \end{cases} \tag{9.2}$$

where $g(t) = \max\Big(\big\lceil \big(t - (P_s - Q_s)\big)/P_s \big\rceil, 1\Big)$ and $V^{g(t)}$ denotes an interval $[(g(t)+1)P_s - 2Q_s, (g(t)+1)P_s - Q_s]$ in which the subsystem $S_s$ receives budget. Figure 9.2 shows $\mathtt{sbf}_s(t)$. To guarantee a minimum CPU supply, the worst-case budget provision is considered in Eq. (9.2) assuming that tasks are released at the same time when the subsystem budget depletes (at time $t = 0$ in Figure 9.2) and the budget was supplied as early as possible and all following

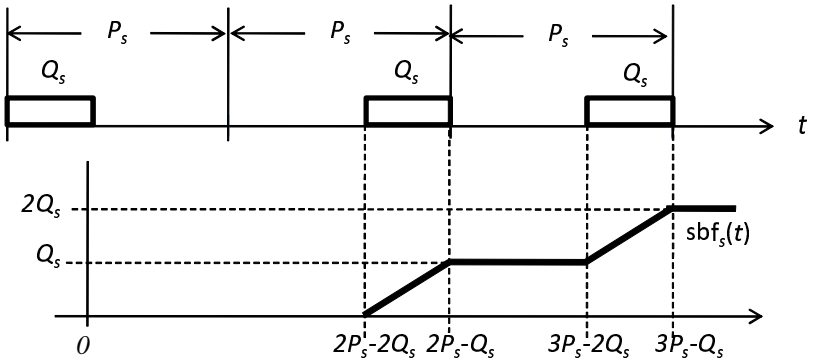


Figure 9.2: Supply bound function $\texttt{sbf}_\texttt{s}(\texttt{t})$.

budgets will be supplied as late as possible due to interference from other, higher priority subsystems.

For the request bound function $\texttt{rbf}_{\texttt{FP}}(i,t)$ of a task $\tau_i$ and to compute the maximum execution request up to time $t$, it is assumed that (*i*) $\tau_i$ and all its higher priority tasks are simultaneously released, (*ii*) each access to a global shared resource by these tasks will generate a self-blocking, (*iii*) a task with priority lower than $\tau_i$ that can cause a maximum blocking has locked a global shared resource just before the release of $\tau_i$, and (*iv*) will also cause a self-blocking. $\texttt{rbf}_{\texttt{FP}}(i,t)$ is given by [2]:

$$\texttt{rbf}_{\texttt{FP}}(i,t) = C_i + I_S(i) + I_H(i,t) + I_L(i), \tag{9.3}$$

where $I_S(i)$ is the self-blocking of task $\tau_i$, $I_H(i,t)$ is the interference from tasks with a priority higher than that of $\tau_i$, and $I_L(i)$ is the interference from tasks with priority lower than that of $\tau_i$, that access shared resources, i.e.,

$$I_S(i) = \sum_{R_k \in \{R^i\}} \sum_{a=1}^{rn_{ik}} X_{ika}, \tag{9.4}$$

$$I_H(i,t) = \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil (C_h + \sum_{R_k \in \{R^h\}} \sum_{a=1}^{rn_{hk}} X_{hka}), \tag{9.5}$$

$$I_L(i) = \max\{0, \max_{l=i+1}^{n_s} \max_{R_k \in \{R^l\} \wedge rc_{sk} \leq i} \max_{a=1}^{rn_{lk}} (c_{lka} + X_{lka})\}. \tag{9.6}$$

Note that we use the outermost $\max$ in (9.6) to also define $I_L(i)$ in those situations where $\tau_i$ can not be blocked by lower priority tasks. Looking at Eqs. (9.4)-(9.6), it is clear that $\texttt{rbf}_{\texttt{FP}}(i,t)$ is a discrete step function that changes its value at certain time points ($t = a \times T_h$ where $a$ is an integer number). Then for Eq. (9.1), $t$ can be selected from a finite set of scheduling points [15].

The term $X_{jka}$ in these equations represents the self-blocking (resource locking time) of task $\tau_j$ due to the $a^{th}$ access to resource $R_k$. Eq. (9.7) can be used to determine $X_{ika}$, where the sum in the equation represents the interference from higher priority tasks that can preempt the execution of $\tau_i$ while accessing $R_j$. Since $2P_s \leq T_s^{\min}$, tasks with a priority higher than $rc_{sk}$ can interfere at most once (the proof of Eq. (9.7) is presented in [16]).

$$X_{ika} = c_{ika} + \sum_{h=1}^{rc_{sk}-1} C_h. \tag{9.7}$$

The self-blocking of $\tau_i$, the higher priority tasks and the maximum self-blocking of the lower priority tasks are given in Eqs. (9.4)-(9.6). We can rearrange these equations by moving all self-blocking terms into one equation $I'_S(i,t)$, resulting in corresponding equations $I'_H(i,t)$ and $I'_L(i)$:

$$\begin{aligned} I'_S(i,t) = &\sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil \Big( \sum_{R_k \in \{R^h\}} \sum_{a=1}^{rn_{hk}} X_{hka} \Big) \\ &+ \sum_{R_k \in \{R^i\}} \sum_{a=1}^{rn_{ik}} X_{ika} \\ &+ \max\{0, \max_{l=i+1}^{n_s} \max_{R_k \in \{R^l\} \wedge rc_{sk} \leq i} \max_{a=1}^{rn_{lk}} (X_{lka})\}, \end{aligned} \tag{9.8}$$

$$I'_H(i,t) = \sum_{1 \leq h < i} \left\lceil \frac{t}{T_h} \right\rceil C_h, \tag{9.9}$$

$$I'_L(i) = \max\{0, \max_{l=i+1}^{n_s} \max_{R_k \in \{R^l\} \wedge rc_{sk} \leq i} \max_{a=1}^{rn_{lk}} (c_{lka})\}. \tag{9.10}$$

Eqs. (9.8)-(9.10) can be used to evaluate $\texttt{rbf}_{\texttt{FP}}(i,t)$ in Eq. (9.3).

**Subsystem timing interface** In this paper, it is assumed that the period $P_s$ of a subsystem $S_s$ is given while the minimum subsystem budget $Q_s$ should be computed. We use $\texttt{calculateBudget}(S_s, P_s)$ to denote a function that calculates

this minimum budget $Q_s$ satisfying Eq. (9.1). This function is similar to the one presented in [8]. We can determine $X_{sk}$ for all $R_k \in \mathcal{R}_s$ by

$$X_{sk} = \max_{\tau_i \in \mathcal{T}_s \wedge R_k \in \{R^i\}} \max_{a=1}^{rn_{ik}}(X_{ika}). \tag{9.11}$$

We define $X_s$ as the maximum resource locking time among all resources accessed by $S_s$, i.e.

$$X_s = \max_{R_k \in \mathcal{R}_s} (X_{sk}). \tag{9.12}$$

Finally, when a task experiences self-blocking during a subsystem period it is guaranteed access to the resource during the next period. To provide this guarantee, the subsystem budget $Q_s$ should satisfy

$$Q_s \geq X_s. \tag{9.13}$$

**System level scheduling** At the system level, each subsystem $S_s$ can be modeled as a simple periodic task. The parameters of such a task are provided by the subsystem timing interface $S_s(P_s, Q_s, \mathcal{X}_s)$, i.e. the task period is $P_s$, the execution time is $Q_s$, and the set of critical section execution times when accessing logical shared resources is $\mathcal{X}_s$. To validate the composability of the system under FPPS and SRP, classical schedulability analysis for periodic tasks can be applied; please refer to [2] for more details.

## 9.5 Motivating example

In this section we will show that the schedulability analysis associated with SIRAP is very pessimistic if multiple resources are accessed by tasks and/or the same resource is accessed multiple times by tasks. We will show this by means of the following example.

| $\mathcal{T}$ | $C_i$ | $T_i$ | $R_k$ | $c_{ika}$ |
|---|---|---|---|---|
| $\tau_1$ | 6 | 100 | $R_1, R_1, R_2$ | 1, 2, 2 |
| $\tau_2$ | 20 | 150 | $R_1, R_2$ | 2, 1 |
| $\tau_3$ | 3 | 500 | $R_2$ | 1 |

Table 9.1: Example task set parameters

**Example:** Consider a subsystem $S_s$ that has three tasks as shown in Table 9.1. Note that task $\tau_1$ accesses $R_1$ twice, i.e. $rn_{1,1} = 2$. Let the subsystem period

be equal to $P_s = 50$. Using the original SIRAP analysis, we find a subsystem budget $Q_s = 23.5$. Task $\tau_2$ requires this budget in order to guarantee its schedulability, i.e. the set of points of time $t$ used to determine schedulability of $\tau_2$ is $\{100, 150\}$ and at time $t = 150$, $\texttt{rbf}_{\texttt{FP}}(2, 150) = \texttt{sbf}_{\texttt{s}}(\texttt{150}) = \texttt{47}$.

To evaluate $\texttt{rbf}_{\texttt{FP}}(i, t)$ for $\tau_i$, the SIRAP analysis assumes that the maximum number of self-blocking instances will occur for $\tau_i$ and all its lower and higher priority tasks. Considering our example, $I'_S(2, 150)$ contains a total of 9 self-blocking instances; 6 self-blocking instances for task $\tau_1$ ($X_{1,1,1} = 1, X_{1,1,1} = 1, X_{1,1,2} = 2, X_{1,1,2} = 2, X_{1,2,1} = 2, X_{1,2,1} = 2$), 2 for task $\tau_2$ ($X_{2,1,1} = 2, X_{2,2,1} = 1$), and 1 for task $\tau_3$ ($X_{3,2,1} = 1$) (see Eq. (9.8)), resulting in $I'_S(2, 150) = 14$. Because $P_s = 50$ and $Q_s = 23.5$, we know that $\tau_2$ needs at least two and at most three activations of the subsystem for its completion. As no self-blocking instance can occur during a subsystem period in which a task completes its execution, the analysis should incorporate at most 2 self-blocking instances for $\tau_2$. This means that the SIRAP analysis adds 7 unnecessary self-blocking instances when calculating $\texttt{rbf}_{\texttt{FP}}(i, t)$ which makes the analysis pessimistic. If 2 self-blocking instances are considered and the two largest self-blocking values that may happen are selected (e.g. $X_{1,1,2} = 2, X_{1,2,1} = 2$), then $I'_S(2, 150) = 4$ and a subsystem budget of $Q_s = 18.5$ suffices. For this subsystem budget, we once again find at most 2 self-blocking instances. In other words, the required subsystem utilization ($Q_s/P_s$) can be decreased by $27\%$ compared with the original SIRAP analysis. This improvement can be achieved assuming that at most one self-blocking instance needs to be considered every budget period (the budget period is a time interval from the time when the budget replenished up to the next following budget replenishment time instant, for example in Figure 9.1, it starts at $t_1$ and ends at $t_1 + P_s = t_3$).

## 9.6 Improved SIRAP analysis

In the previous section, we have shown that the original analysis of SIRAP can be very pessimistic. If we assume that at most one self-blocking instance needs to be considered during every budget period then a significant improvement in the CPU resource usage can be achieved. Although multiple self-blocking instances can occur during one budget period, it is sufficient to consider at most one.

**Lemma 7.** *At most one self-blocking occurrence, i.e. the largest possible,*

*needs to be considered during each subsystem period $P_s$ of $S_s$ for the schedulability of $\tau_i \in \mathcal{T}_s$.*

*Proof.* Upon self-blocking of an arbitrary task $\tau_j$ of $S_s$ due to an attempt to access $R_k$, the subsystem ceiling of $S_s$ becomes at most equal to the internal resource ceiling $rc_{sk}$. Once this situation has been established, the subsystem ceiling may decrease (due to activations of and subsequent attempts to access resources by tasks with a priority higher than $rc_{sk}$, i.e. the task index is lower than $rc_{sk}$), but will not increase during the current subsystem period. A task $\tau_i$ experiences blocking/interference due to self-blocking of an arbitrary task $\tau_j$ trying to access $R_k$ if and only if the internal resource ceiling $rc_{sk}$ of $R_k$ is at most equal to $i$ (i.e. $rc_{sk} \leq i$). Hence, as soon as $\tau_i$ experiences blocking/interference due to self-blocking, that situation will last for the remainder of the budget period, and additional occurrences of self-blocking can at most overlap with earlier occurrences. It is therefore sufficient to consider at most one self-blocking instance, i.e. the largest possible, per budget period. □

### 9.6.1 Problem formulation

Lemma 7 proves that at each subsystem period, one maximum self-blocking can be considered in the schedulability analysis of SIRAP. That means the number of effective self-blocking occurrences at time instant $t$, that should be considered in the schedulability analysis, depends on the maximum number of subsystem periods that have been repeated up to time instant $t$. In other words, the number of self-blocking occurrences is bounded by the number of overlapping budget periods. However, the equations used for the local schedulability analysis Eqs. (9.2) and (9.3) can not express this bound on self-blocking because:

- The $\texttt{sbf}_s(t)$ of Eq. (9.2) is based on the subsystem budget and period, but is agnostic of the behavior of the subsystem internal tasks that cause self-blocking, and therefore also agnostic of self-blocking.
- The $\texttt{rbf}_{\texttt{FP}}(i,t)$ of Eq. (9.3) contains the self-blocking terms, but does not consider the subsystem period.

We propose two different analysis approaches in order to address the bound on self-blocking; the first approach is based on using this knowledge (bound on the self-blocking) in the calculation of $\texttt{rbf}_{\texttt{FP}}(i,t)$ and the second approach is based on using it in the calculation of $\texttt{sbf}_s(t)$.

As long as we are still in the subsystem level development stage, we have all internal information including global shared resources, which task(s) access them and the critical section execution time of each resource access; information that is required to optimize the local schedulability analysis in order to decrease the CPU resources required to be reserved for the subsystem.

Before presenting the two analysis approaches that may decrease the required subsystem utilization compared to the original SIRAP approach, we will describe a self-blocking multi-set that will be used by these new approaches.

### 9.6.2 Self-blocking set

For each task $\tau_i$, we define a multi-set $G_i(t)$ containing the values of all self-blocking instances that a task $\tau_i$ may experience in an interval of length $t$ according to $I'_S(i,t)$; see Eq. (9.8). Similar to Eq. (9.8), the elements in $G_i(t)$ are evaluated based on the assumption that task $\tau_i$ and all its higher priority tasks are simultaneously released.

Note that $G_i(t)$ includes all $X_{jka}$ that may contribute to the self-blocking. Depending on the time $t$, a number of elements will be taken from this list and, to consider the worst-case scenario, the value of these elements should be the highest in the multi-set. To provide this, we define a sequence $G_i^{\text{sort}}(t)$ that contains all elements of $G_i(t)$ sorted in a non-increasing order, i.e. $G_i^{\text{sort}}(t) = \texttt{sort}(G_i(t))$. Considering the example presented in Section 9.5, the sequence $G_2^{\text{sort}}(150)$ for $\tau_2$ and $t = 150$ equals $< X_{1,1,2}, X_{1,1,2}, X_{1,2,1}, X_{1,2,1}, X_{2,1,1}, X_{1,1,1}, X_{1,1,1}, X_{2,2,1}, X_{3,2,1} >$.

### 9.6.3 Analysis based on changing `rbf`

In this section we will present the first approach called IRBF that improves the local schedulability analysis of SIRAP based on changing $\texttt{rbf}_{\texttt{FP}}(i,t)$. Note that as long as we are not changing the supply bound function $\texttt{sbf}_{\texttt{s}}(\texttt{t})$, Eq. (9.2) and the associated assumption concerning worst-case budget provision can still be used. As we explained before, the number of self-blocking occurrences is bounded by the number of overlapping subsystem budget periods. The following lemma presents an upper bound on the number of self-blocking occurrences in an interval of length $t$.

**Lemma 8.** *Given a subsystem $S_s$ and assuming the worst-case budget provision, an upper bound on the number of self-blocking occurrences $z(t)$ in an interval of length $t$ is given by*

$$z(t) = \left\lceil \frac{t}{P_s} \right\rceil. \tag{9.14}$$

*Proof.* Note that $z(t)$ represents an upper bound on the number of subsystem periods that are entirely contained in an interval of length $t$. In addition, the $\texttt{sbf}_s(t)$ calculation in Eq. (9.2) is based on the worst-case budget provision, i.e. task $\tau_i$ under consideration is released at a budget depletion when the budget was supplied as early as possible and all following budget supplies will be at late as possible. From the release time of $\tau_i$, if two self-blocking occurrences happen, at least one $Q_s$ must be fully supplied and another $Q_s$ (at least) partially. Hence, $t > P_s - (Q_s - X^1) + P_s = 2P_s - (Q_s - X^1)$ for $0 < X^1 \leq Q_s < P_s$, where $X^1$ is a (first) self-blocking; see Figure 9.3(a). This assumption is satisfied for $t > P_s$. Similarly, we can prove that for $b$ self-blocking occurrences, $t > b \times P_s$. □

Note that Eq. (9.14) accounts for a first self-blocking occurrence just *after* the release of $\tau_i$, i.e. for $t$ an infinitesimal larger than zero. For SIRAP, this release of $\tau_i$ is assumed at a worst-case budget provision, e.g. at time $t = 0$ in Figure 9.2. At the end of the first budget supply (at time $t = 2P_s - Q_s$ in Figure 9.2), where one complete self-blocking can occur, Eq. (9.14) has accounted for a second self-blocking, as shown in Figure 9.3(b). In general, at any time $t$, the number of self-blocking occurrences evaluated using Eq. (9.14) will be one larger than the number of self-blocking occurrences that can happen in an interval with a worst-case budget provision. This guarantees that we can safely assume that the worst-case situation for the original analysis for SIRAP also applies for IRBF.

After evaluating $z(t)$, it is possible to calculate the self-blocking on task $\tau_i$ from *all* tasks, i.e. lower priority tasks, higher priority tasks and $\tau_i$ itself. Eq. (9.8), that computes the self-blocking on $\tau_i$, can now be replaced by

$$I_S^*(i,t) = \sum_{j=1}^{z(t)} G_i^{\text{sort}}(t)[j]. \tag{9.15}$$

Note that if $z(t)$ is larger than the number of elements in the set $G_i^{\text{sort}}(t)$, then the values of the extra elements are equal to zero, e.g. if $G_i^{\text{sort}}(t^*)$ has $k_i$ elements (i.e. the number of all possible self-blocking occurrences that may block $\tau_i$ in an interval of length $t^*$), then $G_i^{\text{sort}}(t^*)[j] = 0$ for all $j > k_i$.

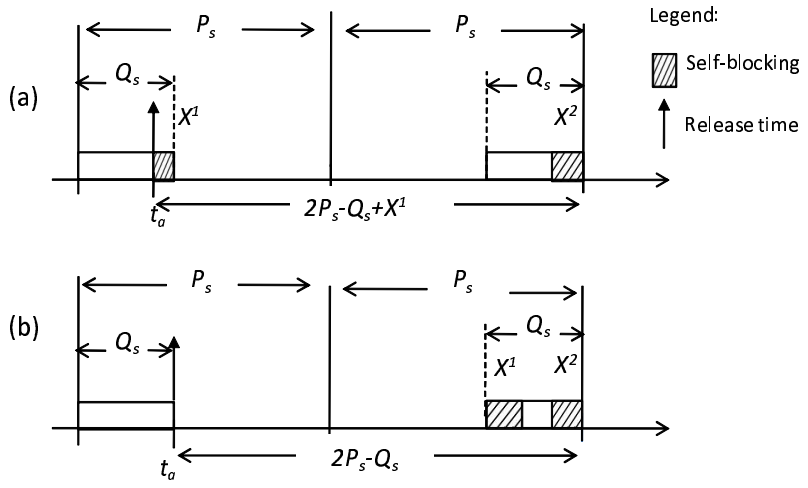


Figure 9.3: A subsystem execution with self-blocking.

**Correctness of the analysis** The following lemma proves the correctness of the IRBF approach.

**Lemma 9.** *Using the IRBF approach,* $\texttt{rbf}_{\texttt{FP}}(i,t)$ *given by*

$$\texttt{rbf}_{\texttt{FP}}(i,t) = C_i + I_S^*(i,t) + I'_H(i,t) + I'_L(i) \tag{9.16}$$

*computes an upper bound on the maximum cumulative execution requests that could be generated from the time that* $\tau_i$ *is released up to time* $t$*.*

*Proof.* We have to prove that Eq. (9.15) computes an upper bound on the maximum resource request generated from self-blocking. As explained earlier, dur-

ing a self-blocking, all tasks with priority less than or equal to the resource ceiling of the resource that caused the self-blocking, are not allowed to execute until the next budget activation. To consume the remaining budget, an idle task is executing if there are no tasks, with priority higher than the subsystem ceiling, released during the self-blocking. To add the effect of self-blocking on the schedulability analysis of $\tau_i$, the execution time of the idle task during the self-blocking can be modeled as an interference from a higher priority task on $\tau_i$. The maximum number of times that the idle task executes up to any time $t$ is equal to the number of self-blocking occurrences during the same time interval and an upper bound is given by $z(t)$. Furthermore, selecting the first $z(t)$ elements from the $G_i^{\text{sort}}(t)$ gives the maximum execution times of the idle task.

We also have to prove that a simultaneous release of $\tau_i$ and all its higher priority tasks at a worst-case budget provision will actually result in an upper bound for $I_S^*(i,t)$. To this end, we show that neither the actual number of self-blocking terms nor their values in an interval of length $t^*$ starting at the release of $\tau_i$ can become larger when a higher priority task $\tau_h$ is either released before or after $\tau_i$. We first observe that the number of self-blocking occurrences $z(t^*)$ in an interval of length $t^*$ is independent of the release of $\tau_h$ relative to $\tau_i$. Next, we consider the values for self-blocking.

A later release of $\tau_h$ will either keep the same (worst-case) value for the self-blocking during $t^*$ or reduce it (and may in addition cause a decrease of the interference in Eq. (9.5)). Releasing $\tau_h$ earlier than $\tau_i$ makes $\tau_h$ receiving some budget and at the same time a self-blocking happens, before the release of $\tau_i$ (remember, $\tau_i$ is released at a worst-case budget provision). Furthermore, and at the end of time interval $t^*$, new self-blocking caused by earlier releasing of $\tau_h$, may be added to the self-blocking set ($G_i(t^*)$). However, since an earlier self-blocking happens (before the release of $\tau_i$) this earlier self-blocking removes the effect of the additional self-blocking on $G_i(t^*)$. For instance, an earlier release of $\tau_h$ may (*i*) keep the self-blocking the same (if the additional self-blocking $X^0$ resulting from the earlier release of $\tau_h$ during the last budget period is less than the one that was considered assuming all tasks are released simultaneously $X^0 \leq X^j$; see Figure 9.4(b)) or (*ii*) add or replace a self-blocking term in the last complete budget period contained in $t^*$. For both cases of (*ii*), the new term for the additional activation of $\tau_h$ will also imply the removal of a similar term for $\tau_h$ at the earlier release of $\tau_h$, effectively rotating the sequence of blocking terms as illustrated in Figure 9.4(c)-(d). Rotating the terms does not change the sum of the blocking terms, however, and the amount of self-blocking in $t^*$ therefore remains the same. □

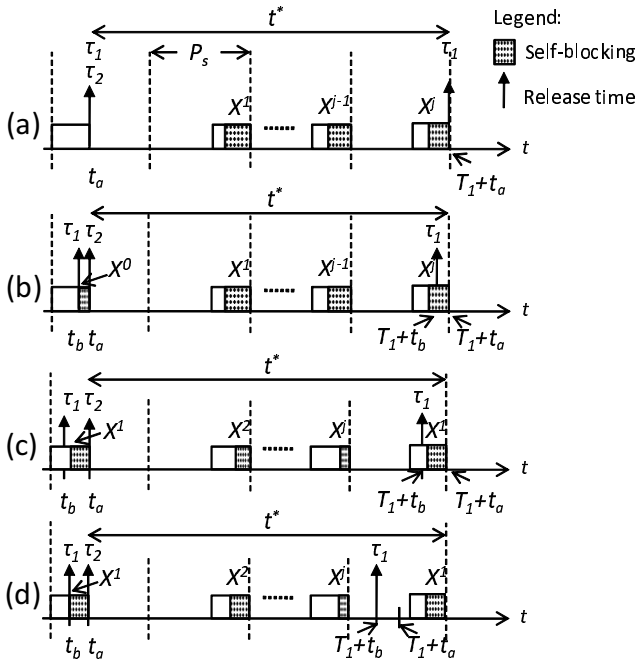


Figure 9.4: Critical instant for two tasks.

**Example** Returning to our example, we find $z(150) = 3$ and that makes $I_S^*(i,t) = 6$ according to Eq. (9.15), we find a minimum subsystem budget $Q_s = 19.5$, which is better than the one obtained using the original SIRAP equations. The analysis is still pessimistic, however, because $z(t)$ is an upper

bound on the number of self-blocking occurrences rather than an exact number and in addition, $t$ is selected from the schedulability test points set of $\tau_2$ rather than the Worst Case Response Time (WCRT) of the task. Note that the WCRT of $\tau_2$ is less than $150$ which indicates remaining pessimism on the results.

**Remark** Based on the new analysis presented in this section, the following lemma proves that the results obtained from the analysis based on IRBF are always better than, or the same as, the original SIRAP approach.

**Lemma 10.** *The minimum subsystem budget obtained using IRBF will be always less than or equal to the subsystem budget obtained using the original SIRAP approach.*

*Proof.* When evaluating $\texttt{rbf}(i,t)$ for a task $\tau_i$, the only difference between the original SIRAP approach and the analysis of IRBF is the calculation of self-blocking $I'_S(i,t)$ in Eq. (9.8) and $I^*_S(i,t)$ in Eq. (9.15). To prove the correctness of this lemma we have to prove that $I^*_S(i,t) \leq I'_S(i,t)$. Because $G_i^{\text{sort}}(t)$ is the sorted multi-set $G_i(t)$ of values contained in $I'_S(i,t)$, the sum of all values contained in $G_i^{\text{sort}}(t)$ is equal to $I'_S(i,t)$, i.e. when $k_i$ is equal to the number of non-zero elements in $G_i^{\text{sort}}(t)$, we have $I'_S(i,t) = \sum_{j=1}^{k_i} G_i^{\text{sort}}(t)[j]$. Since $I^*_S(i,t) = \sum_{j=1}^{z(t)} G_i^{\text{sort}}(t)[j]$, we get $I^*_S(i,t) < I'_S(i,t)$ for $z(t) < k_i$ and $I^*_S(i,t) = I'_S(i,t)$ for $z(t) \geq k_i$, because $G_i^{\text{sort}}(t)[j] = 0$ for all $j > k_i$. □

### 9.6.4 Analysis based on changing `sbf`

The effect of self-blocking in SIRAP has historically been considered in the request bound function (as shown in Sections 9.4 and 9.6.3). Self-blocking is modeled as additional execution time that is added to $\texttt{rbf}_{\texttt{FP}}(i,t)$ when applying the analysis for $\tau_i$. In this section we use a different approach, called ISBF, based on considering the effect of self-blocking in the supply bound function. The main idea is to model self-blocking as *unavailable* budget, which means that the budget that can be delivered to the subsystem will be decreased by the amount of self-blocking. Moving the effect of self-blocking from `rbf` to `sbf` has the potential to improve the results, in terms of requiring less CPU resources, compared to the original SIRAP analysis.

Figure 9.5 shows the supply bound function using the new approach, where $Q_s$ is guaranteed every period $P_s$, however, only a part (denoted $Q^j$) from the $j^{th}$ subsystem budget is provided to the subsystem after the release of $\tau_i$,

while the other part (denoted $X^j$) of the $j^{th}$ subsystem budget is considered as unavailable budget which represents the self-blocking time.

A new supply bound function should be considered taking into account the effect of self-blocking on the worst-case budget provision. In general, the worst-case budget provision happens when $\tau_i$ is released at the same time when the subsystem budget becomes unavailable and the budget was supplied at the beginning of the budget period and all later budget will be supplied as late as possible. Note that self-blocking occurs at the end of a subsystem period, which means that unavailable budget is positioned at the end (last part) of the subsystem budget. The earliest time that the budget becomes unavailable relative to the start of a budget period is therefore $Q_s - X^0$. Conversely, the latest time that the budget will become available after a replenishment (starting time of the next budget period), is $P_s - Q_s$. Hence, the longest time that a subsystem may not get any budget (called *Blackout Duration* BD) is $2P_s - 2Q_s + X^0$. Finally, each task has a specific set of self-blocking occurrences, which means that each task will have its own supply bound function. The new supply bound function $\texttt{sbf}_\texttt{s}(\texttt{i},\texttt{t})$ for $\tau_i$ is given by

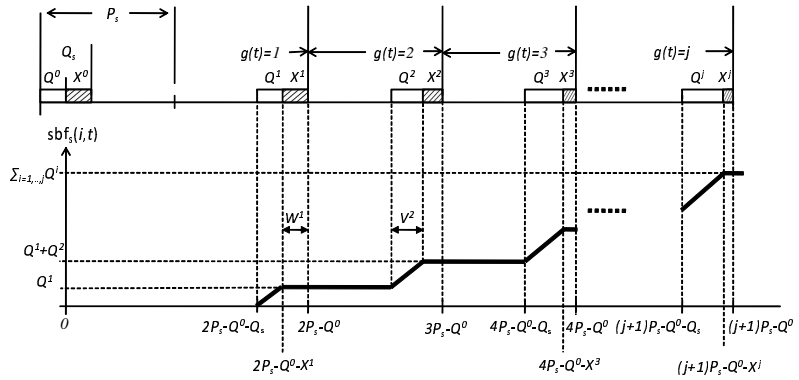


Figure 9.5: New supply bound function $\texttt{sbf}_\texttt{s}(\texttt{i},\texttt{t})$.

$$\texttt{sbf}_s(i,t) = \begin{cases} t - (g(t)+1)P_s + Q^0 + Q_s & \\ \quad + Sum(g(t)-1) & \text{if } t \in V^{g(t)} \\ Sum(g(t)) & \text{if } t \in W^{g(t)} \\ Sum(g(t)-1) & \text{otherwise,} \end{cases} \tag{9.17}$$

where

$$g(t) = \max\Big(\big\lceil\big(t-(P_s-Q^0)\big)/P_s\big\rceil, 1\Big), \tag{9.18}$$

$$Sum(\ell) = \sum_{j=1}^{\ell} Q^j, \tag{9.19}$$

and $Q^j = Q_s - X^j$, $V^{g(t)}$ denotes an interval $[(g(t)+1)P_s - Q^0 - Q_s, (g(t)+1)P_s - Q^0 - X^{g(t)}]$ when the subsystem gets budget, and $W^{g(t)}$ denotes an interval $[(g(t)+1)P_s - Q^0 - X^{g(t)}, (g(t)+1)P_s - Q^0]$ during the $g(t)^{th}$ self-blocking. The intuition for $g(t)$ in Eq. (9.17) is the number of periods of the periodic model that can actually provide budget in an interval of length $t$, as shown in Figure 9.5. To explain Eq. (9.17) let us consider the case for $g(t) = 3$. If $t \in W^3$, i.e. during the $3^{rd}$ self-blocking time interval of length $X^3$, then the amount of budget supplied to the subsystem will by $Q^1 + Q^2 + Q^3$, i.e. $Sum(3)$. If $t \in V^3$, then the resource supply will equal to $Q^1 + Q^2$ plus the value from the linearly increasing region (see Figure 9.5), otherwise, the budget supply is $Q^1 + Q^2$, i.e. $Sum(3-1)$.

Since we consider the effect of self-blocking in the supply bound function, we can now remove all self-blocking from $\texttt{rbf}_{\texttt{FP}}(i,t)$, i.e. $I'_S(i,t) = 0$ in Eq. (9.8) and only Eqs. (9.9) and (9.10) are used to evaluate $\texttt{rbf}_{\texttt{FP}}(i,t)$. Hence, the local schedulability analysis is

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; \texttt{rbf}_{\texttt{FP}}(i,t) \leq \texttt{sbf}_s(i,t). \tag{9.20}$$

The final step on evaluating $\texttt{sbf}_s(i,t^*)$ is to set the values of self-blocking $X^j$ for $0 \leq j \leq g(t)$ such that the supply bound function gives the minimum possible CPU supply for interval length $t^*$. To achieve this, $X^j$ is evaluated as follows

$$X^j = G_i^{\text{sort}}(t^*)[j], \tag{9.21}$$

where $0 < j \leq g$ and $X^0 = X^1$ which is the largest self-blocking.

**Correctness of the analysis** The following lemma proves that setting the self-blocking according to Eq. (9.21) and $X^0 = X^1$ will make the supply bound function giving the the minimum possible CPU supply.

**Lemma 11.** $\texttt{sbf}_\texttt{s}(\texttt{i},\texttt{t})$ *will give the minimum possible CPU supply for every interval length* $t$ *if Eq. (9.21) and* $X^0 = X^1$ *are used to set the values of* $X^j$.

*Proof.* To proof the lemma, we have to prove that the amount of budget supplied to a subsystem using Eq. (9.17) is the minimum and also the budget is supplied as late as possible. Using Eq. (9.21) will set the largest possible values of self-blocking at time $t$ to $X^1, X^2, \ldots, X^j$ and that will make the function $Sum(i)$ in Eq. (9.19) return the minimum possible value ($Q^j = Q_s - X^j$), which in turn will give the minimum $\mathtt{sbf}_s(i,t)$.

On the other hand, the blackout duration $\mathsf{BD}$ should be maximized to guarantee the minimum CPU supply. Since $\mathsf{BD} = 2P_s - 2Q_s + X^0 = 2P_s - Q^0 - Q_s$ (which equals to the starting time of the interval $V^{(1)}$), $\mathsf{BD}$ is maximized if $X^0 = X^1 = G_i^{\text{sort}}(t^*)[1]$. This setting of $X^0$ will also maximize the starting time of the interval $V^j | j = 1, .., g(t^*)$ (time interval when new budget is supplied) which delays the budget supply and decreases $\mathtt{sbf}_s(i,t)$ at any time instant $t$. Considering the two mentioned factors will guarantee that Eq. (9.17) gives the minimum possible CPU resource supply. □

Note that Eq. (9.21) uses the set $G_i^{\text{sort}}(t)$, and the elements of the set are evaluated assuming that $\tau_i$ and all tasks with priority higher than $\tau_i$ are released simultaneously. In the previous section, we have shown that this assumption is correct considering the IRBF approach. For ISBF, setting $X^0 = X^1 = G_i^{\text{sort}}(t)[1]$ makes the analysis more pessimistic than the actual execution since the first element in the set $G_i^{\text{sort}}(t)[1]$ can only happen once before or after the release of $\tau_i$. So the additional self-blocking $X^0$ is considered to maximize the time that tasks will not get any CPU budget, as proven in Lemma 11. If $\tau_i$ or any of its higher priority tasks is released earlier than the beginning of the self-blocking $X^0$ then that task will directly get some budget and since we use $X^1$ self-blocking after the first budget consumption then $X^0$ should be removed (similar scenario is shown in Figure 9.4(c) but $\tau_2$ should be released at the time when self-blocking $X_1$ begins). As a result, and similar to the IRBF approach, same elements taken from $G_i^{\text{sort}}(t^*)$ can at most be rotated if tasks are not released at the same time and that means the supply bound function at time $t^*$ will not be decreased.

The pessimistic assumption $X^0 = X^1 = G_i^{\text{sort}}(t)[1]$ may affect the results of ISBF and the effect depends on the tasks and the subsystem parameters as shown in the following examples.

**Example** Returning to our example, based on the new supply bound function, we find a minimum subsystem budget $Q_s = 18.5$, since two instances of self-blocking can happen at $t = 150$. This is better than IRBF yielding

$Q_s = 19.5$ and the original SIRAP where $Q_s = 23.5$. Note that assigning $X^0 = 2$ did not affect the results of ISBF.

However, it is not always the case that ISBF can give better results than the other approaches, as will be shown in the following example. Suppose a subsystem $S_s$ with $P_s = 100$ and $n$ tasks. The highest priority task $\tau_1$ is the task that requires the highest subsystem budget. $\tau_1$ has the following parameters, $T_1 = 230$, $C_1 = 29.5$, the maximum blocking from lower priority tasks that accesses a global shared resource $R_1$ is $B_1 = 6$ and $\tau_1$ accesses $R_1$ two times with critical section execution time $c_{1,1,1} = 1$ and $c_{1,1,2} = 1$. Using ISBF, the minimum subsystem budget is $Q_s = 39.2$ while using the other two approaches then $Q_s = 37.85$.

The reason that ISBF will require more subsystem budget than the other two approaches in the second example is that using ISBF, the maximum blocking $B_1 = 6$ is considered twice, i.e. $X^0 = X^1 = 6$, whereas the other approaches use the actual possible self-blocking $\{6, 1, 1\}$. Because the difference between the largest and the other self-blocking terms is high, ISBF requires a higher budget.

## 9.7 Evaluation

In this section, we evaluate the performance of the two presented approaches ISBF and IRBF, in terms of the required subsystem utilization, compared to the original SIRAP approach. Looking at the scheduliability analysis of both IRBF and ISBF, the following parameters can directly affect the improvements that both new approaches can achieve:

- The number of global shared resource accesses made by a subsystem (including the number of shared resources and the number of times that each resource is accessed).
- The difference between the subsystem period and its corresponding task periods.
- The length of the critical section execution time, that affects the self-blocking time.

We will explain the effect of the mentioned parameters by means of simulation in the following section.

### 9.7.1 Simulation settings

The simulation is performed by applying the two new analysis approaches in addition to the original SIRAP approach on 1000 different randomly generated subsystems where each subsystem consists of 8 tasks. The internal resource ceilings of the globally shared resources are assumed to be equal to the highest task priority in each subsystem (i.e. $rc_{sk} = 1$) and we assume $T_i = D_i$ for all tasks. The worst-case critical section execution time of a task $\tau_i$ is set to a value between $0.1C_i$ and $0.25C_i$, the subsystem period $P_s = 100$ and the task set utilization is $25\%$. For each simulation study one of the mentioned parameters is changed and a new set of 1000 subsystems is generated (except when changing $P_s$; in that case the same subsystems are used). The task set utilization is divided randomly among the tasks that belong to a subsystem. Task periods are selected within the range of 200 to 1000. The execution time is derived from the desired task utilization. All randomized subsystem parameters are generated following uniform distributions.

### 9.7.2 Simulation results

Tables 9.2-9.4 show the results of 3 different simulation studies performed to measure the performance of the two new analysis approaches.

In these tables, "$U_s^{IRBF} < U_s^{Orig}$" denotes the percentage of subsystems where their subsystem utilization $U_s = Q_s/P_s$ using IRBF is less than the subsystem utilization using the original SIRAP approach, out of 1000 randomly generated subsystems, and "Max I $(U_s^{IRBF}/U_s^{Orig})$" is the maximum improvement that the analysis based on IRBF can achieve compared with the original SIRAP approach, which is computed as $(U_s^{Orig} - U_s^{IRBF})/U_s^{IRBF}$. Finally, "Max D $(U_s^{ISBF}/U_s^{Orig})$" is the maximum degradation in the subsystem utilization as a result of using the analysis based on ISBF compared to the analysis using the original SIRAP approach. As we explained in the previous section, in some cases ISBF may require more CPU resources than the other two approaches.

- **Study 1** is specified having the number of shared resource accesses equal to 2, 4, 8, and 12, critical section execution time $c_{ijk}$ is $(0.1 - 0.25) \times C_i$ and subsystem period $P_s$ is 100. The intention of this study is to show the effect of changing the number of shared resources on the performance of the three approaches.

- **Study 2** changes the subsystem period (compared to Study 1) to 75 and

50 and keeps the number of shared resources to 12. As mentioned previously we use the same 1000 subsystems as in Study 1 and only change the subsystem period. The intention of this study is to show the effect of decreasing the subsystem period on the performance of the three approaches.

- **Study 3** decreases the critical section execution time to $(0.01-0.05)\times C_i$ (compared to Study 1) and keeps the number of shared resources to 12. The intention of this study is to show the effect of decreasing the critical section execution times on the performance of the three approaches.

| Number of shared resources | 2 | 4 | 8 | 12 |
|---|---|---|---|---|
| $(U_s^{IRBF} < U_s^{Orig})$ | 0.2% | 23.1% | 98.7% | 100% |
| $(U_s^{ISBF} < U_s^{Orig})$ | 2.0% | 33.3% | 99.5% | 100% |
| $(U_s^{ISBF} = U_s^{Orig})$ | 50.0% | 29.0% | 0.2% | 0% |
| $(U_s^{ISBF} < U_s^{IRBF})$ | 2.0% | 31.0% | 80.0% | 90.0% |
| $(U_s^{IRBF} < U_s^{ISBF})$ | 50.0% | 40.0% | 18.0% | 8.0% |
| Median $(U_s^{Orig})$ | 35.6 | 37.0 | 40.8 | 43.6 |
| Median $(U_s^{IRBF})$ | 35.6 | 36.9 | 38.8 | 39.3 |
| Median $(U_s^{ISBF})$ | 35.8 | 36.9 | 38.4 | 38.7 |
| Max I $(U_s^{IRBF}/U_s^{Orig})$ | 3.1% | 5.7% | 16.4% | 30.6% |
| Max I $(U_s^{ISBF}/U_s^{Orig})$ | 7.3% | 14.4% | 22.7% | 36.7% |
| Max D $(U_s^{ISBF}/U_s^{Orig})$ | 5.5% | 3.9% | 1.2% | 0% |
| Max I $(U_s^{ISBF}/U_s^{IRBF})$ | 7.3% | 8.8% | 22.1% | 17.2% |
| Max I $(U_s^{IRBF}/U_s^{ISBF})$ | 5.5% | 4.0% | 2.0% | 1.7% |

Table 9.2: Measured results of Study 1

Looking at the results in Table 9.2 (**Study 1**), it is clear that the improvements that both ISBF and IRBF can achieve become more significant when the number of shared resource accesses is increased. This is also clear in Figure 9.6 and Figure 9.7 that show the number of subsystems that have subsystem utilization within the ranges shown in the x-axis (the lines that connect points are only used for illustration) for 8 and 12 shared resource accesses, respectively. The reason is that the self-blocking $I'_S(i,t)$ in Eq. (9.8), used by the original SIRAP approach, will increase significantly which will require more subsystem utilization. Comparing the values in the table, when the number of shared resources is 12 the analysis based on ISBF can decrease the subsystem

| $P_s$ | 50 | 75 | 100 |
|---|---|---|---|
| $(U_s^{IRBF} < U_s^{Orig})$ | 87.0% | 100% | 100% |
| $(U_s^{ISBF} < U_s^{Orig})$ | 83.0% | 99.7% | 100% |
| $(U_s^{ISBF} = U_s^{Orig})$ | 6.0% | 0.1% | 0% |
| $(U_s^{ISBF} < U_s^{IRBF})$ | 55.0% | 82.0% | 90.0% |
| $(U_s^{IRBF} < U_s^{ISBF})$ | 36.0% | 14.0% | 8.0% |
| Median $(U_s^{Orig})$ | 41.0% | 42.3% | 43.6% |
| Median $(U_s^{IRBF})$ | 39.7% | 39.3% | 39.3% |
| Median $(U_s^{ISBF})$ | 39.6% | 38.9% | 38.7% |
| Max I $(U_s^{IRBF}/U_s^{Orig})$ | 16.8% | 30.3% | 30.6% |
| Max I $(U_s^{ISBF}/U_s^{Orig})$ | 17.3% | 36.5% | 36.7% |
| Max D $(U_s^{ISBF}/U_s^{Orig})$ | 2.7% | 0.7% | 0% |
| Max I $(U_s^{ISBF}/U_s^{IRBF})$ | 4.4% | 12.1% | 17.2% |
| Max I $(U_s^{IRBF}/U_s^{ISBF})$ | 2.7% | 1.9% | 1.7% |

Table 9.3: Measured results of Study 2

| $c_{ijk}$ | $(1-5)\% \times C_i$ | $(10-25)\% \times C_i$ |
|---|---|---|
| $(U_s^{IRBF} < U_s^{Orig})$ | 100% | 100% |
| $(U_s^{ISBF} < U_s^{Orig})$ | 100% | 100% |
| $(U_s^{ISBF} < U_s^{IRBF})$ | 78.0% | 90.0% |
| $(U_s^{IRBF} < U_s^{ISBF})$ | 8.0% | 8.0% |
| Median $(U_s^{Orig})$ | 35.0% | 43.6% |
| Median $(U_s^{IRBF})$ | 34.4% | 39.3% |
| Median $(U_s^{ISBF})$ | 34.3% | 38.7% |
| Max I $(U_s^{IRBF}/U_s^{Orig})$ | 5.0% | 30.6% |
| Max I $(U_s^{ISBF}/U_s^{Orig})$ | 7.0% | 36.7% |
| Max I $(U_s^{ISBF}/U_s^{IRBF})$ | 2.1% | 17.2% |
| Max I $(U_s^{IRBF}/U_s^{ISBF})$ | 0.4% | 1.7% |

Table 9.4: Measured results of Study 3

utilization by 36% compared with the original SIRAP approach and the improvement in the median of subsystem utilization is about 12.5%. IRBF can achieve slightly less improvement than ISBF because the number of the considered self-blocking $z(t)$ is an upper bound. However, when the number of

shared resources is low, e.g. 2, ISBF and IRBF can achieve some improvement compared with the original SIRAP, and in many cases ISBF requires higher subsystem utilization compared with the original SIRAP (about $48\%$). It is interesting to see that even if the number of shared resource access is low, ISBF and IRBF can achieve some improvements. Note that IRBF will never require more subsystem utilization than using the original SIRAP approach (see Lemma 10). Now, comparing the results of using ISBF and IRBF, we can see from the table that ISBF gives relatively better results, in terms of the number of subsystems that require less subsystem utilization, median and maximum improvement compared with IRBF if the number of shared resources accesses is high. The reason is that the possibility of having many large self-blocking will be higher which can decrease the effect of $X^0$ on ISBF.

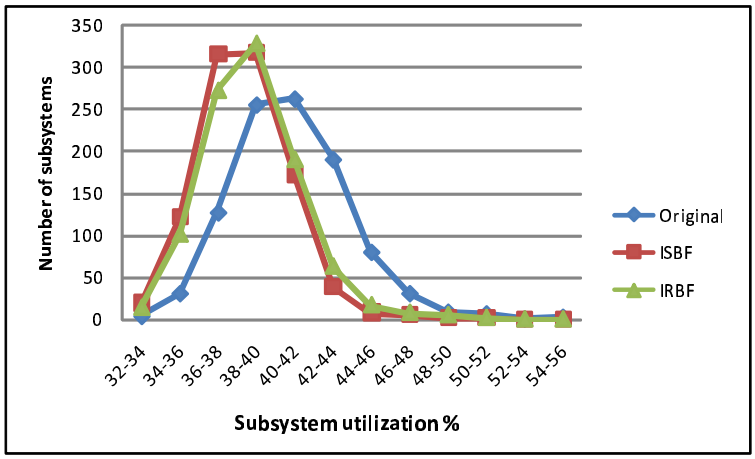


Figure 9.6: Results of Study 1 for 8 global shared resources access.

Looking at Table 9.3 (**Study 2**), it is clear that when the subsystem period is decreased, the improvement that ISBF and IRBF can achieve compared with original SIRAP is also decreased. Comparing the median of the subsystem utilization of the 1000 generated subsystems when changing the subsystem period, we can see that for the original SIRAP analysis the subsystem utilization is decreasing when decreasing the subsystem period. However, using the other two approaches, the subsystem utilization is increasing when decreasing the subsystem period. The reason for this behavior is that the number of self-

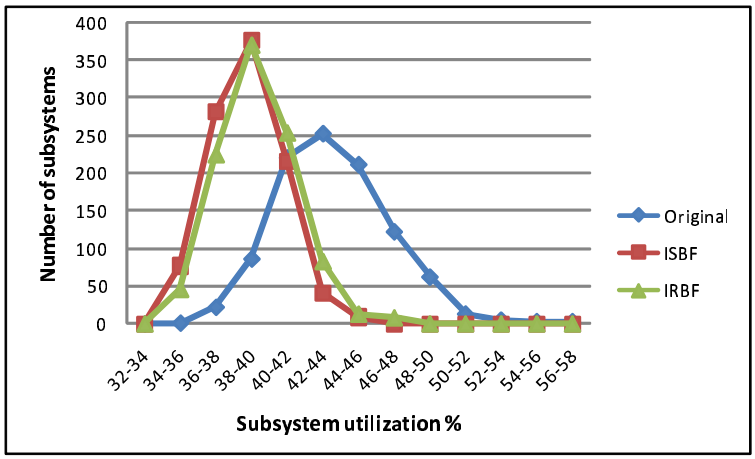


Figure 9.7: Results of Study 1 for 12 global shared resources access.

blocking occurrences will increase when decreasing the subsystem period and in turn it will increase $z(t)$ using IRBF, i.e. the number of $X^j$ for ISBF. This will increase $\texttt{rbf}_{\texttt{FP}}(i,t)$ using IRBF, and decrease $\texttt{sbf}_{\texttt{s}}(\texttt{i},\texttt{t})$ using ISBF, at time $t$ compared with the case when the subsystem period is higher, and that will in turn require more subsystem utilization. Note that this case can happen when the number of shared resource accesses is high. So for a high number of global shared resource accesses, it is recommended to use larger subsystem periods that can decrease the subsystem utilization and at the same time decrease the number of subsystem context switches. Another interesting observation from this table is that the percentage of subsystems that require less subsystem utilization using ISBF compared with IRBF, is decreasing when decreasing the subsystem period. The reason is that more self-blocking occurrences will be considered in both ISBF and IRBF and that will increase the possibility of having a large difference between the considered self-blocking which will increase the effect of $X^0$ for ISBF.

In **Study 3** we have decreased the range of the critical section execution times which will, in turn, decrease the self-blocking execution times. The results in Table 9.4 show that the improvements that ISBF and IRBF can achieve in terms of subsystem utilization compared with the original SIRAP approach, are decreased. The improvement in the subsystem utilization median using

ISBF is decreased from 12.6% to 2% when decreasing the critical section execution time, and using IRBF it is decreased from 10.9% to 1.7%. The reason for this is that the total self-blocking $I'_S(i,t)$ in Eq. (9.8) used by the original SIRAP approach, depends not only on the number of shared resource accesses but also on the size of the self-blocking $X_{ika}$.

## 9.8 Summary

In this paper, we have presented new schedulability analysis for SIRAP; a synchronization protocol for hierarchically scheduled real-time systems. We have shown that the original local schedulability analysis for SIRAP is pessimistic when the tasks of a subsystem make a high number of accesses to global shared resources. This pessimism is inherent in the fact that the original SIRAP schedulability analysis does not take the maximum number of self-blocking instances into account, when in fact this number is bounded by the maximum number of subsystem period intervals in which these resource accessing tasks execute. We have presented two new analysis approaches that take this bounded number of self-blocking instances into account; the first approach based on changing `rbf` and second approach based on changing `sbf`. We have identified the parameters that have effect on the improvement that these new approaches can achieve over the original SIRAP schedulability analysis and we have explored and explained the effect of these parameters by means of simulation analysis. The results of the simulation show that significant improvements can be achieved by the new approaches compared to the original SIRAP approach, if the number of accesses to global shared resources made by the tasks of a subsystem is high. Generalizing the analysis of this paper to include other scheduling algorithms, e.g. EDF, as a subsystem level scheduler, is a topic of future work.

# Bibliography


[1] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[2] 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 $7^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[3] Holman P and J. Anderson. Locking in pfair-scheduled multiprocessor systems. In *Proceedings of the $23^{rd}$ International IEEE Real-Time Systems Symposium (RTSS'02)*, pages 149–158, December 2002.

[4] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the $18^{th}$ IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[5] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the $20^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[6] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the $23^{th}$ IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[7] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the $7^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[8] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[9] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[10] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 26$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[11] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, December 2007.

[12] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[13] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[14] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *IEEE Transaction on Embedded Computing Systems*, 7(3):1–39, 2008.

[15] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.

[16] M. Behnam, T. Nolte, M. Åsberg, and R.J. Bril. Overrun and skipping in hierarchical scheduled real-time systems. In *Proceedings of the 15$^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, pages 519–526, August 2009.

# Chapter 10

# Paper E: Improved Schedulability Analysis of Synchronization Protocols Based on Overrun Without Payback for Hierarchical Scheduling Frameworks

Moris Behnam, Thomas Nolte, Reinder J. Bril

MRTC report ISSN 1404-3041 ISRN MDH-MRTC-237/2010-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, October, 2010.

**Abstract**

In this paper, we show that both global as well as local schedulability analysis of synchronization protocols based on the stack resource protocol (SRP) and overrun without payback for hierarchical scheduling frameworks based on fixed-priority pre-emptive scheduling (FPPS) are pessimistic. We present improved global and local schedulability analysis, illustrate the improvements by means of examples, and show that the improved global analysis is both uniform and sustainable. We evaluate the improved global and local schedulability analysis based on an extensive simulation study and compare the results with the existing analysis.

# 10.1 Introduction

## 10.1.1 Background

Over the years, there has been a growing attention on hierarchical scheduling of real-time systems due to its ability to provide temporal isolation between multiple real-time subsystems executing upon a common processing platform. The Hierarchical Scheduling Framework (HSF) provides means for decomposing a complex system into well-defined parts called subsystems, and a subsystem provides an introspective interface that specifies the timing properties of the subsystem precisely. This implies that subsystems can be independently developed, analyzed and tested, and later assembled without introducing unwanted temporal interference.

Supporting global resource sharing among subsystems is a major challenge, however, since it increases the complexity of the analysis of a system considerably. Due to this complexity, most of the proposed techniques (such as those in [1] and [2]) are based on some simplifying assumptions which make the analysis easier. The consequence of these assumptions is that they add pessimism in the analysis which increases the required CPU resources of systems. For some systems, the pessimism in the analysis is not significant and can be ignored, but for others it may be significant.

As large extents of embedded systems are resource constrained, a tight analysis is instrumental in a successful deployment of HSF techniques in real applications. We therefore aim at reducing potential pessimism in existing schedulability analysis for HSFs that support sharing of global shared resources. Looking further at existing industrial real-time systems, Fixed Priority Preemptive Scheduling (FPPS) is the de facto standard of task scheduling, hence we focus on an HSF with support for FPPS for tasks within a subsystem. Having such support will simplify migration to and integration of existing legacy applications into the HSF, avoiding a too big technology revolution for engineers.

Our current research efforts are directed towards the conception and realization of a two-level HSF that is based on (*i*) FPPS for both *global scheduling* of budgets (allocated to subsystems) and *local scheduling* of tasks (within a subsystem), (*ii*) the periodic resource model [3] for budgets, and (*iii*) the Stack Resource Policy (SRP) [4] for both inter- and intra-subsystem resource sharing. For such an HSF, two mechanisms have been studied that prevent depletion of a budget during global resource access, i.e. *skipping* [1] and *overrun* [2]. The overrun mechanism comes in two flavors, i.e. *with payback* and *without pay-*

*back*.

In this paper, we aim at tighter analysis for the overrun mechanism without payback, assuming the same introspective interface for subsystems as the existing analysis.

### 10.1.2 Contributions

We show that existing global and local schedulability analysis of synchronization protocols based on SRP and overrun without payback for two-level hierarchical scheduling based on FPPS is pessimistic. We present improved global and local analysis assuming that the deadline of a subsystem holds for the sum of its normal budget and its overrun budget, and illustrate the improvements by means of examples. We identify the system parameters that have a great effect on the improvement that the proposed global and local analysis can achieve. In addition we evaluate the improvements that both global and local improved analysis can achieve compared with the traditional analysis, in terms of requiring less CPU resources, by exploring the system load [2] in a simulation study.

### 10.1.3 Overview

This paper has the following structure. In Section 10.2 we present related work. A real-time scheduling model is the topic of Section 10.3. The existing global and local schedulability analysis is recapitulated in Section 10.4, and improved global and local analysis is presented in Sections 10.5 and 10.6, respectively. Section 10.7 presents a simulation study evaluating the improvement that both global and local improved analysis can achieve. The paper is concluded in Section 10.8.

## 10.2 Related work

During the past decade, there has been considerable interest on hierarchical scheduling of real-time systems [5, 6, 7, 3].Deng and Liu [5] proposed a two-level HSF for open systems, where subsystems may be developed and validated independently. Kuo and Li [6] and Lipari and Baruah [7] presented schedulability analysis techniques for such a two-level framework with the FPPS global scheduler and the Earliest Deadline First (EDF) global scheduler, respectively. Shin and Lee [3] proposed the periodic resource model $\Gamma(\Pi, \Theta)$ to specify guaranteed periodic CPU allocations, where $\Pi \in \mathbb{R}^+$ is a period and $\Theta \in \mathbb{R}^+$

is a periodic allocation time ($0 < \Theta \leq \Pi$). Easwaran, Anand, and Lee [8] proposed the explicit deadline periodic (EDP) resource model $\Omega(\Pi, \Theta, \Delta)$ that extends the periodic resource model by explicitly distinguishing a relative deadline $\Delta \in \mathbb{R}^+$ for the allocation time $\Theta$ ($0 < \Theta \leq \Delta \leq \Pi$).

For synchronization protocols in HSFs, two mechanisms have been studied to prevent depletion of a budget during global resource access, i.e. *overrun* (*with payback* and *without payback*) and *skipping*. Overrun with payback was first introduced in the context of aperiodic servers in [9]. The mechanism was later (re-) used for a synchronization protocol in the context of two-level hierarchical scheduling in [10] and extended with overrun without payback. The analysis presented in [10] does not allow analysis of individual subsystems, however an analysis supporting composability was described in [2, 11]. The idea of skipping in the context of HSFs, was used by the SIRAP protocol [1], and its associated analysis supports composability.

## 10.3 Real-time scheduling model

We consider a two-level hierarchical FPPS model using the periodic resource model to specify guaranteed CPU allocations to tasks of subsystems and using a synchronization protocol for mutual exclusive resource access to global logical resources based on SRP[1] and overrun without payback.

### 10.3.1 System model

A system $Sys$ contains a set $\mathcal{R}$ of $M$ global logical resources $R_1$, $R_2$, ..., $R_M$, a set $\mathcal{S}$ of $N$ subsystems $S_1$, $S_2$, ..., $S_N$, a set $\mathcal{B}$ of $N$ budgets for which we assume a periodic resource model [3], and a single processor. Each subsystem $S_s$ has a dedicated budget associated to it. In the remainder of this paper, we leave budgets implicit, i.e. the timing characteristics of budgets are taken care of in the description of subsystems. Subsystems are scheduled by means of FPPS and have fixed, unique priorities. For notational convenience, we assume that subsystems are given in order of decreasing priorities, i.e. $S_1$ has highest priority and $S_N$ has lowest priority.

[1] The focus of this paper is on synchronization protocols for *global* logical resources. We do therefore not consider local logical resources.

### 10.3.2 Subsystem model

Each subsystem $S_s$ contains a set $\mathcal{T}_s$ of $n_s$ periodic tasks $\tau_{s,1}$, $\tau_{s,2}$, ..., $\tau_{s,n_s}$ with fixed, unique priorities, which are scheduled by means of FPPS. For notational convenience, we assume that tasks are given in order of decreasing priorities, i.e. $\tau_1$ has highest priority and $\tau_{n_s}$ has lowest priority. The set $\mathcal{R}_s$ denotes the subset of $M_s$ global resources accessed by subsystem $S_s$. The maximum time that a subsystem $S_s$ executes while accessing resource $R_l \in \mathcal{R}$ is denoted by $X_{sl}$, where $X_{sl} \in \mathbb{R}^+ \cup \{0\}$ and $X_{sl} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$. The timing characteristics of $S_s$ are specified by means of a triple $< P_s, Q_s, \mathcal{X}_s >$, where $P_s \in \mathbb{R}^+$ denotes its (budget) period, $Q_s \in \mathbb{R}^+$ its (normal) budget, and $\mathcal{X}_s$ the set of maximum execution access times of $S_s$ to global resources. The maximum value in $\mathcal{X}_s$ (or zero when $\mathcal{X}_s$ is empty) is denoted by $X_s$.

### 10.3.3 Task model

The timing characteristics of a task $\tau_{si} \in \mathcal{T}_s$ are specified by means of a quartet $< T_{si}, C_{si}, D_{si}, \mathcal{C}_{si} >$, where $T_{si} \in \mathbb{R}^+$ denotes its minimum inter-arrival time, $C_{si} \in \mathbb{R}^+$ its worst-case computation time, $D_{si} \in \mathbb{R}^+$ its (relative) deadline, $\mathcal{C}_{si}$ a set of maximum execution times of $\tau_{si}$ to global resources, where $C_{si} \leq D_{si} \leq T_{si}$. The set $\mathcal{R}_{si}$ denotes the subset of $\mathcal{R}_s$ accessed by task $\tau_{si}$. The maximum time that a task $\tau_{si}$ executes while accessing resource $R_l \in \mathcal{R}$ is denoted by $c_{sil}$, where $c_{sil} \in \mathbb{R}^+ \cup \{0\}$, $C_{si} \geq c_{sil}$, and $c_{sil} > 0 \Leftrightarrow R_l \in \mathcal{R}_{si}$.[2]

### 10.3.4 Resource model

The *CPU supply* refers to the amount of CPU allocation that a virtual processor can provide. The supply bound function $\mathtt{sbf}_\Omega(t)$ of the EDP resource model $\Omega(\Pi, \Theta, \Delta)$ that computes the minimum possible CPU supply for every interval length $t$ is given by

$$\mathtt{sbf}_\Omega(t) = \begin{cases} t - (k+1)(\Pi - \Theta) + (\Pi - \Delta) & \text{if } t \in V^{(k)} \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{10.1}$$

where $k = \max\left(\left\lceil \big(t - (\Delta - \Theta)\big)/\Pi \right\rceil, 1\right)$ and $V^{(k)}$ denotes an interval $[k\Pi + \Delta - 2\Theta, k\Pi + \Delta - \Theta]$.

[2] In [10], it is required that $c_{sil} < C_{si}$ and $c_{sil} < Q_s$. Moreover, it is observed that $c_{sil}$ will typically be much smaller than both $C_{si}$ and $Q_s$.

The supply bound function $\mathtt{sbf}_\Gamma(t)$ of the periodic resource model $\Gamma(\Pi, \Theta)$ is a special case of (10.1), i.e. with $\Delta = \Pi$.

### 10.3.5 Synchronization protocol

Overrun without payback prevents depletion of a budget of a subsystem $S_s$ during access to a global resource $R_l$ by temporarily increasing the budget of $S_s$ with $X_{sl}$, the maximum time that $S_s$ executes while accessing $R_l$. To be able to use SRP in an HSF for synchronizing global resources, its associated ceiling terms needs to be extended.

#### Resource ceiling

With every global resource $R_l$, two types of resource ceilings are associated; an *external* resource ceiling $RC_l$ for global scheduling and an *internal* resource ceiling $rc_{sl}$ for local scheduling. According to SRP, these ceilings are defined as

$$RC_l = \min(N, \min\{s \mid X_{sl} > 0\}), \tag{10.2}$$

$$rc_{sl} = \min(n_s, \min\{i \mid c_{sil} > 0\}). \tag{10.3}$$

Note that we use the outermost $\min$ in (10.2) and (10.3) to define $RC_l$ and $rc_{sl}$ also in those situations where no subsystem uses $R_l$ and no task of $\mathcal{T}_s$ uses $R_l$, respectively.

#### System/subsystem ceiling

The system/subsystem ceilings are dynamic parameters that change during the execution. The system/subsystem ceiling is equal to the lowest external/internal resource ceiling of a currently locked resource in the system/ subsystem.

Under SRP, a task $\tau_{si}$ can only preempt the currently executing task $\tau_{sj}$ (even when accessing a global resource) if the priority of $\tau_{si}$ is greater (i.e. the index $i$ is lower) than $S_s$ its subsystem ceiling. A similar condition for preemption holds for subsystems.

#### Concluding remarks

The maximum time $X_{sl}$ that $S_s$ executes while accessing $R_l$ can be reduced by assigning a value to $rc_{sl}$ that is *smaller* than the value according to SRP. For

HSRP [10], the internal resource ceiling is therefore set to the highest priority, i.e. $rc_{sl}^{\mathrm{HSRP}} = 1$. Decreasing $rc_{sl}$ may cause a subsystem to become unfeasible for a given budget [12], however, because the tasks with a priority higher than the old ceiling and at most equal to the new ceiling may no longer be feasible.

The results in this paper apply for any internal resource ceiling $rc'_{sl}$ where $rc_{sl} \geq rc'_{sl} \geq rc_{sl}^{\mathrm{HSRP}} = 1$.[3]

## 10.4 Recap of existing schedulability analysis

In this section, we briefly recapitulate the global schedulability analysis presented in [10] and the local schedulability analysis described in [2, 11]. Although the global schedulability analysis presented in [2, 11] looks different, it is based on the analysis described in [10] and therefore yields the same result.

For illustration purposes, we will use an example system $Sys_{10.1}$ containing two subsystems $S_1$ and $S_2$ sharing a global resource $R_1$. The characteristics of the subsystems are given in Table 10.1.

| subsystem | $P_s$ | $Q_s + X_s$ |
|---|---|---|
| $S_1$ | 5 | 2 |
| $S_2$ | 7 | $Q_2 + X_2$ |

Table 10.1: Subsystem characteristics of $Sys_{10.1}$.

### 10.4.1 Global analysis

The worst-case response time $WR_s$ of subsystem $S_s$ is given by the smallest $x \in \mathbb{R}^+$ satisfying[4]

$$x = B_s + (Q_s + X_s) + \sum_{t<s} \left\lceil \frac{x}{P_t} \right\rceil (Q_t + X_t), \tag{10.4}$$

[3] Because $rc_{sl}^{\mathrm{HSRP}} = 1$ for $R_l \in \mathcal{R}_s$, $X_{sl} = \max_i c_{sil}$. Hence, from $c_{sil} < Q_s$ we derive $X_s < Q_s$. Without the constraint on the internal resource ceiling, $X_s$ may be larger than $Q_s$. For illustration purposes, we also allow $X_s > Q_s$ in this paper.

[4] Strictly spoken, [10] uses (10.4) *excluding* $X_s$ for $WR_s$. The smallest positive solution of (10.4) is required to be at most equal to $P_s$ to prevent additional interference of the next activation of (the budget of) $S_s$.

where $B_s$ is the maximum blocking time of $S_s$ by lower priority subsystems, i.e.

$$B_s = \max(0, \max\{X_{tl} \mid t > s \wedge X_{tl} > 0 \wedge RC_l \leq s\}). \tag{10.5}$$

Note that we use the outermost $\max$ in (10.5) to define $B_s$ also in those situations where the set of values of the innermost $\max$ is empty. To calculate $WR_s$, we can use an iterative procedure based on recurrence relationships, starting with a lower bound, e.g. $B_s + \sum_{t \leq s}(Q_t + X_t)$. The condition for global schedulability is given by

$$\underset{1 \leq s \leq N}{\forall} WR_s \leq P_s. \tag{10.6}$$

We merely observe that the global analysis is similar to basic analysis for FPPS with resource sharing, where the period $P_s$ of a subsystem $S_s$ serves as deadline for the sum of the normal budget $Q_s$ and the overrun budget $X_s$, and the interference of higher priority subsystems $S_t$ is based on the sum $Q_t + X_t$. We will therefore use a superscript P to refer to this basic analysis for subsystems, e.g. $WR_s^{\mathrm{P}}$.

In the sequel, we are not only interested in the worst-case response time of a subsystem $S_s$ for particular values of $B_s$, $Q_s$, and $X_s$, but in the value as a function of the sum of these three values. We will therefore use a functional notation when needed, e.g. $WR_s(B_s + Q_s + X_s)$.

The global feasibility area of the existing analysis is illustrated for our example system $Sys_{10.1}$ in Figure 10.1(a). Note that the $y$-axis is excluded, because we assume the capacity of subsystems to be positive, i.e. $Q_2 > 0$.

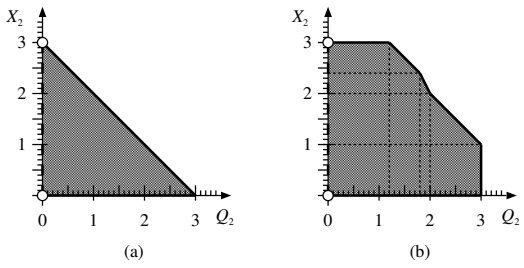


Figure 10.1: Global feasibility area assuming (a) FPPS and (b) improved global analysis.

### 10.4.2 Local analysis

The existing condition for local schedulability of a subsystem $S_s$ [2] is given by

$$\underset{1\leq i\leq n_s}{\forall}\ \underset{0<x\leq D_{si}}{\exists} b_{si} + C_{si} + \sum_{j<i} \left\lceil \frac{x}{T_{sj}} \right\rceil \cdot C_{sj} \leq \mathtt{sbf}_{\Gamma_s}(x), \tag{10.7}$$

where $b_{si}$ is the maximum blocking time of $\tau_{si}$ by lower priority tasks, i.e.

$$b_{si} = \max(0, \max\{c_{sjl} \mid j > i \wedge c_{sjl} > 0 \wedge rc_{sl} \leq i\}), \tag{10.8}$$

and $\mathtt{sbf}_{\Gamma_s}(x)$ is the supply bound function of the periodic resource model $\Gamma_s(P_s, Q_s)$ for the subsystem $S_s$ under consideration. Note that we use the outermost $\max$ in (10.8) to define $b_{si}$ also in those situations where the set of values of the innermost $\max$ is empty.

The value for $X_{sl}$ depends on the local scheduler and the synchronization protocol. The maximum time that subsystem $S_s$ executes while task $\tau_{si}$ accesses resource $R_l \in \mathcal{R}$ is denoted by $X_{sil}$, where $X_{sil} \in \mathbb{R}^+ \cup \{0\}$ and $X_{sil} > 0 \Leftrightarrow c_{sil} > 0$. For $c_{sil} > 0$, $X_{sil}$ is given by [2]

$$X_{sil} = c_{sil} + \sum_{j<rc_{sl}} C_{sj}. \tag{10.9}$$

The value for $X_{sl}$ is given by

$$X_{sl} = \max_{1\leq i\leq n_s} X_{sil}. \tag{10.10}$$

## 10.5 Improved global analysis

As described in Section 10.4.1, the existing global schedulability analysis is based on FPPS, where the period $P_s$ serves as deadline for the sum of the normal budget $Q_s$ and overrun budget $X_s$.

### 10.5.1 Illustrating the improvement

The improvement of the global analysis is based on two observations:

1. *Limited pre-emption of overrun budget $X_s$*: while $S_s$ is accessing $R_l$ using $X_s$, it can only be pre-empted by subsystems with a priority higher than $RC_l$.

2. *Blocking starts* before *the execution based on the overrun budget $X_s$ starts*: to use its overrun budget $X_s$, $S_s$ needs to first lock a global resource.

From the first observation, we conclude that subsystem $S_1$ can not preempt $S_2$ during those intervals of time when $S_2$ is accessing resource $R_1$ in general, and when $S_2$ is executing based on its overrun budget $X_2$ in particular.

From the second observation, we conclude that whenever $S_2$ uses its overrun budget $X_2$, it must have locked $R_1$ already during the consumption of its normal budget $Q_2$, i.e. *before* it starts consuming its overrun budget $X_2$. Hence, the system ceiling is already set to the priority of $S_1$ before $S_2$ starts consuming $X_2$, preventing $S_1$ to preempt. The resulting improvement is illustrated in Figure 10.1(b).

### 10.5.2 Improving the global analysis

The improved global analysis is similar to the analysis for FPDS [13, 14] and FPPS with preemption thresholds [15] in the sense that we have to consider all jobs in a so-called level-$s$ active period to determine the worst-case response time $WR_s$ of subsystem $S_s$. Unlike the analysis described in [13, 14, 15], subsystems $S_{s-1}$ till $S_{RC_l}$ cannot preempt $S_s$ at the finalization time of $Q_s$ when $S_s$ is accessing $R_l$.

In the remainder of this section, we first recapitulate the notion of a level-$s$ active period. Next, we derive analysis for the worst-case finalization time $WF_{sk}^{\mathrm{Q}}$ of the normal budget $Q_s$ of job $\iota_{sk}$ of subsystem $S_s$ relative to start of the constituting level-$s$ active period. Finally, we derive analysis for the worst-case response time $WR_s$ of $S_s$.

#### Level-$s$ active period

The worst-case length $WL_s$ of a level-$s$ active period with $s \leq N$ is given by the smallest $x \in \mathbb{R}^+$ that satisfies

$$x = B_s + \sum_{t \leq s} \left\lceil \frac{x}{P_t} \right\rceil (Q_t + X_t). \tag{10.11}$$

To calculate $WL_s$, we can use an iterative procedure based on recurrence relationships, starting with a lower bound, e.g. $B_s + \sum_{t \leq s}(Q_t + X_t)$. The maximum number $wl_s$ of jobs of $S_s$ in a level-$s$ active period is given by

$$wl_s = \left\lceil \frac{WL_s}{P_s} \right\rceil. \tag{10.12}$$

### Worst-case finalization time

For a job $\iota_{sk}$ of $S_s$ with $0 \leq k < wl_s$, we split the interval from the start of the level-$s$ active period to the finalization of job $\iota_{sk}$ in two sub-intervals: a first sub-interval including the execution of the normal budget $Q_s$ by job $\iota_{sk}$ and a second sub-interval from the finalization of $Q_s$ by $\iota_{sk}$ till the finalization of $\iota_{sk}$, i.e. constituting the execution of the overrun budget $X_s$.

Let $WF^{\mathrm{Q}}_{sk}$ denote the worst-case finalization time of the normal budget $Q_s$ of job $\iota_{sk}$ with $0 \leq k < wl_s$ relative to the start of the constituting level-$s$ active period. To determine $WF^{\mathrm{Q}}_{sk}$, we have to consider up to three suprema. First, the sequence of jobs $\iota_{s0}$ till $\iota_{sk}$ experience a blocking $B_s \geq 0$ by lower priority subsystems in the worst-case situation. Similar to FPDS [13, 14], the worst-case blocking is a supremum for $B_s > 0$ rather than a maximum. Second, the jobs $\iota_{s0}$ till $\iota_{s,k-1}$ need their overrun budget $X_s$ to access global resources. Because the access to a global resource starts during the execution of the normal budget, the actual amount $X$ of overrun budget used is a supremum rather than a maximum. Finally, the access to the global resource also starts "as late as possible" during the execution of job $\iota_{sk}$ in a worst-case situation, to maximize the interference of higher priority subsystems. This "as late as possible" also gives rise to a supremum rather than a maximum. The worst-case finalization time $WF^{\mathrm{Q}}_{sk}$ can therefore be described as

$$WF^{\mathrm{Q}}_{sk} = \lim_{Q \uparrow Q_s} \lim_{X \uparrow X_s} \lim_{B \uparrow B_s} WR^{\mathrm{P}}_{s}(B + k(Q_s + X) + Q),$$

where $WR^{\mathrm{P}}_{s}$ is the worst-case response time of a fictive subsystem $S'_s$ with a period $P'_s = (k+1)T_s$, a normal budget $Q'_s = k(Q_s+X)+Q$, and a maximum blocking time $B$. Using the following equation from [14]

$$\lim_{x \uparrow C} WR^{\mathrm{P}}_{i}(x) = WR^{\mathrm{P}}_{i}(C) \tag{10.13}$$

we derive

$$WF^{\mathrm{Q}}_{sk} = WR^{\mathrm{P}}_{s}(B_s + (k+1)Q_s + kX_s). \tag{10.14}$$

### Worst-case response time

Let job $\iota_{sk}$ of $S_s$ access $R_l \in \mathcal{R}$. When $\iota_{sk}$ starts to consume its overrun budget, the subsystems $S_{s-1}$ till $S_{RC_l}$ are already blocked, and only subsystems with a priority higher than $RC_l$ can therefore still pre-empt $X_s$. To determine the worst-case response time $WR_{skl}$ of job $\iota_{sk}$ of $S_s$, we now introduce a fictive subsystem $S'_{RC_l}$, i.e. a subsystem that can only be pre-empted by tasks

with a priority higher than $RC_l$. The preemptions during $WF_{sk}^{\mathrm{Q}}$ by subsystems $S_{s-1}$ till $S_{RC_l}$ are treated as *additional* blocking of $S'_{RC_l}$. The worst-case interference of the subsystems $S_{s-1}$ till $S_{RC_l}$ in the interval of length $WF_{sk}^{\mathrm{Q}}$ is denoted by $WI_{RC_l,k}^{s-1}$ and given by

$$WI_{RC_l,k}^{s-1} = \sum_{s-1 \le t \le RC_l} \left\lceil \frac{WF_{sk}^{\mathrm{Q}}}{P_t} \right\rceil (Q_t + X_t). \tag{10.15}$$

The worst-case response time $WR_{skl}$ of job $\iota_{sk}$ of subsystem $S_s$ when it accesses $R_l$ is now given by

$$\begin{aligned} WR_{skl} &= \lim_{X \uparrow X_{sl}} WR_{RC_l}^{\mathrm{P}}(B'_{RC_l} + (k+1)Q_s + kX_s + X)) - kP_s \\ &= WR_{RC_l}^{\mathrm{P}}(B'_{RC_l} + (k+1)Q_s + kX_s + X_{sl})) - kP_s, \end{aligned} \tag{10.16}$$

where $WR_{RC_l}^{\mathrm{P}}$ represents the worst-case response time of a fictive subsystem $S'_{RC_l}$ with a (budget) period $P'_{RC_l}$ and a deadline equal to $(k+1)P_s$, a normal budget $Q'_s$ equal to $(k+1)Q_s + kX_s$, an overrun budget $X'_s$ equal to $X_{sl}$, and a maximum blocking time $B'_{RC_l}$ given by

$$B'_{RC_l} = B_s + WI_{RC_l,k}^{s-1}. \tag{10.17}$$

When a subsystem uses multiple global resources, we have to be very careful. In particular, when the resource ceiling $RC_{sl}$ of resource $R_l \in \mathcal{R}_s$ is *larger* than $RC_{sl'}$ of resource $R_{l'} \in \mathcal{R}_s$, i.e. *more* subsystems can pre-empt $S_s$ during its access to $R_l$ than to $R_{l'}$, and the maximum execution access time $X_{sl}$ of $S_s$ to $R_l$ is *smaller* than $X_{sl'}$, the system may be schedulable for $R_{l'}$ but not for $R_l$. As an example consider a system containing 2 global resources $R_1$ and $R_2$ and 3 subsystems $S_1$, $S_2$, and $S_3$, where the subsystems have timing characteristics as given in Table 10.2.

| subsystem | $P_s$ | $Q_s$ | $X_{s,1}$ | $X_{s,2}$ |
|---|---|---|---|---|
| $S_1$ | 5 | 1 | 0.6 | 0 |
| $S_2$ | 5 | 0.2 | 0 | 0.2 |
| $S_3$ | 7 | 3 | 1 | 0.4 |

Table 10.2: Subsystem characteristics of $Sys_{10.2}$.

The schedulability of $S_3$ for $X_{3,1}$ follows immediately from the similarity of systems $Sys_{10.1}$ and $Sys_{10.2}$, and the feasibility area shown in Figure 10.1(b). Subsystem $S_3$ just meets its deadline at $t = 7$ for its overrun

budget $X_{3,2} = 0.4$ under worst-case conditions, i.e. a simultaneous release of all three subsystems at time $t = 0$ and resources accessed by both $S_1$ and $S_2$ requiring the usage of their overrun budgets at every activation; see Figure 10.2. Note that subsystem $S_3$ will miss its deadline at time $t = 7$ for an infinitesimal increase $\epsilon > 0$ of $X_{3,2}$. The worst-case response time for job $\iota_{sk}$ is therefore

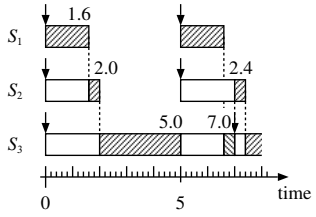


Figure 10.2: Subsystem $S_3$ just meets it deadline at $t = 7$ for $X_{3,2} = 0.4$.

the maximum for all global resources accessed by $S_s$, i.e.

$$WR_{sk} = \max_{l} WR_{skl}. \tag{10.18}$$

Finally, the worst-case response time $WR_s$ of subsystem $S_s$ is given by [5]

$$WR_s = \max_{0 \le k < wl_s} WR_{sk}. \tag{10.19}$$

### 10.5.3 Concluding remarks

In this section, we briefly discuss three aspects of the global analysis, i.e. the global analysis is *uniform* and *sustainable* and it will never give worst result than the original analysis.

The analysis for FPDS [13, 14] is not uniform for all tasks, i.e. the analysis for the lowest priority task differs from the analysis of the other tasks. This anomaly is caused by the fact that the lowest priority task cannot be blocked, i.e. its blocking time is zero, and the blocking time of all other tasks is a supremum rather than a maximum. Unlike the analysis for FPDS [13, 14], the global analysis presented in this section is uniform. This is an immediate consequence of the fact that blocking of a global resource $R_l$ by a subsystem $S_s$ is already done during the execution of the normal budget, i.e. *before* the execution based

[5] The interested reader is referred to [16], which explains the improvement in detail by means of a variety of timelines.

on the overrun budget starts. As a result, subsystems $S_{s-1}$ till $S_{RC_l}$ cannot preempt $S_s$ at the finalization time of $Q_s$, irrespective of $s$.

As described in [17], a schedulability test is *sustainable* if any task system deemed schedulable by the test remains so if it behaves 'better' than mandated by its system specifications, i.e. sustainability requires that schedulability be preserved in situations in which it should be 'easier' to ensure schedulability. Given our scheduling model, we use the following definition for sustainability of our improved global schedulability test.

**Definition 1.** *A schedulability test for our real-time scheduling model for subsystems is* sustainable *if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual job[s] are changed in any, some, or all of the following ways: (*i*) decreased normal budgets; (*ii*) decreased overrun budgets, (*iii*) later arrival times; and (*iv*) larger relative deadlines.*

With this definition, sustainability of our global schedulability test immediately follows from (10.6), i.e. $WR_s \leq P_s = D_s$ and the fact that

- the maximum number $wl_s$ of jobs of subsystem $S_s$ in a level-$s$ active period, and
- the worst-case finalization time $WF_{sk}^{\mathrm{Q}}$ in (10.14), the worst-case interference $WI_{RC_l,k}^{s-1}$ in (10.15), and the worst-case response time $WR_{skl}$ in (10.16)

are strictly non-increasing for decreasing normal budgets, decreasing overrun budgets, and increasing budget periods of subsystems.

Finally, we will prove that the improved global analysis will never give worse results than the original analysis i.e., it will give better results or in the worst case the same results as the original analysis. Looking at (10.12), if $wl_s > 1$, then the system will be unschedulable using the original analysis because the first job will miss its deadline according to the original analysis. While using the modified analysis, the same systems can be schedulable. If $wl_s = 1$ then $k = 0$ and $X_s$ will not have any effect in (10.14) since $k = 0$. For modified analysis, only the subsystems with a higher priority than the resource ceiling of the resource being locked are able to preempt. Taking this into account can reduce the amount of interference considered due to higher priority subsystems in general and for $k = 0$ in particular. Which in turn can improve the results in terms of response time, schedulability and the CPU-resource requirement.

## 10.6 Improved local analysis

Both the existing global schedulability analysis and the improved global schedulability analysis assume a deadline for a subsystem $S_s$ equal to its period $P_s$ for the sum of the normal budget $Q_s$ and the overrun budget $X_s$. The existing local schedulability analysis for the tasks of $S_s$ is exclusively based on $Q_s$, however. Hence, when a system is feasible from a global scheduling perspective, the latest finalization time of $Q_s$ is guaranteed to be at least $X_s$ before the next activation of $S_s$. Hence, we can use the supply bound function $\mathtt{sbf}_{\Omega}(t)$ of the EDP resource model $\Omega_s(P_s, Q_s, \Delta_s)$ for overrun without payback rather than $\mathtt{sbf}_{\Gamma_s}(t)$ of $\Gamma_s(P_s, Q_s)$ in (10.7), where $\Delta_s = P_s - X_s$. Because $X_s \geq 0$ for all subsystems (by definition), $\mathtt{sbf}_{\Gamma_s}(t) \leq \mathtt{sbf}_{\Omega_s}(t)$ for all subsystems. As a result, a subsystem may be schedulable according to the local analysis based on $\mathtt{sbf}_{\Omega_s}(t)$, but not be schedulable based on $\mathtt{sbf}_{\Gamma_s}(t)$.

Figure 10.3 shows an example of the supply bound functions $\mathtt{sbf}_{\Omega}(t)$ and $\mathtt{sbf}_{\Gamma}(t)$ for subsystem $S_2$ of system $Sys_{10.1}$ with $Q_2 = 1.8$ and $X_2 = 2.4$.

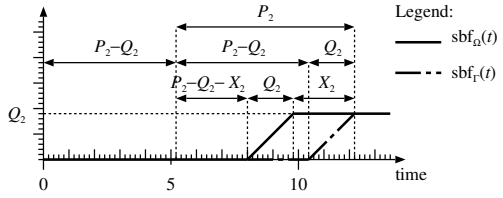


Figure 10.3: Supply bound functions $\mathtt{sbf}_{\Omega}(t)$ and $\mathtt{sbf}_{\Gamma}(t)$ for $S_2$ with $Q_2 = 1.8$ and $X_2 = 2.4$.

## 10.7 Evaluation

In this section, we evaluate the modified overrun without payback analysis (MONP), including both local and global improved analysis, with respect to CPU resource. We compare MONP with the traditional overrun without payback mechanism (ONP) using the notion of system load [2], as system load provides an indication of the system CPU requirement in the presence of shared resources. The comparison is carried out by means of simulation experiments. To show the performance of MONP relative to alternative approaches.

We start this section by briefly explaining the notion of system load and how it should be adapted for MONP.

### 10.7.1 System load

System load is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the global schedulability of the system $\mathcal{S}$.

For ONP, system load $\mathsf{load}_{\mathsf{sys}}$ is calculated as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\} \tag{10.20}$$

where

$$\alpha_s = \min_{0<x\leq P_s} \{\frac{\mathtt{RBF}_s(x)}{x} \mid \mathtt{RBF}_s(x) \leq x\} \tag{10.21}$$

and

$$\mathtt{RBF}_s(x) = B_s + (Q_s + X_s) + \sum_{t<s} \left\lceil \frac{x}{P_t} \right\rceil (Q_t + X_t). \tag{10.22}$$

Note that $x$ can be selected within a finite set of scheduling points [18], and that $\alpha_s$ is the smallest fraction of the CPU resource, required to schedule a subsystem $S_s$ (satisfying the schedulability condition presented in Section 10.4.1), assuming that the global resource supply function is $\alpha_s x$.

One can think of system load as decreasing the speed of the processor by the factor $\mathsf{load}_{\mathsf{sys}}$, which will increase the subsystems' normal budgets, the overrun budgets, and blocking times by a factor $1/\mathsf{load}_{\mathsf{sys}}$.

For MONP, evaluating system load is more complex than e.g., for ONP, because it has more than one response time equation for global schedulability analysis (see Section 10.5), unlike the case for ONP which has only one equation. To perform the schedulability analysis for MONP, firstly, the value of $wl_s$ should be evaluated in order to evaluate the range of $k$ that is used by the other equations. However, we can not evaluate the value of $wl_s$ in (10.12) without having the value of system load known. Without having the range of $k$, we can not use the equations (10.14) - (10.19) that are required in the calculation of system load. We solve this problem by using a binary search algorithm, such that the system load is selected by the search algorithm and corresponding system schedulability is checked. To do this we mutiply all subsystems normal budgets, maximum overrun budgets, and blocking times in equations (10.12) - (10.19) by a factor $1/\mathsf{load}_{\mathsf{sys}}$. If the system is schedulable then the algorithm

will select a lower system load and try again. If the system is unschedulable then the algorithm will select a higher system load. The algorithm terminates if the selected system load $\mathsf{load}_{\mathsf{sys}} > 1$ and the system is unschedulable, or when the difference between the previous and the current system load is less than a given acceptance limit. Since we have used a binary search algorithm for MONP, the complexity of evaluating the system load is higher compared with ONP. However, note that we use the system load for comparison purposes only, hence it does not have any relationship with the complexity of the schedulability analysis.

The efficiency of MONP is measured by the amount of system load required for schedulability, relative to ONP.

Both the local and global improved analysis in MONP can decrease the system load. For the improved local analysis, it has the potential to decrease the subsystem normal budget for certain subsystems, which in turn, can decrease the system load since it decreases the effect of the interference from higher priority subsystems and the required normal budget of the subsystem itself, in equations (10.12) - (10.19). However, there is no guarantee that the improved local analysis can decrease the subsystem normal budget. Note that $\mathtt{sbf}_{\Gamma_s}(x) < \mathtt{sbf}_{\Omega_s}(x)$ for $AP_s - 2Q_s - X_s < x < AP_s - Q_s$ where $A \in \mathbb{N} | A \geq 2$, and $\mathtt{sbf}_{\Gamma_s}(x) = \mathtt{sbf}_{\Omega_s}(x)$ otherwise. So looking at (10.7), the improved local analysis depends on where the value of $x$ (that makes the left hand side of the equation, which represent the resource demand, equal to the right hand side, which is the supply bound function) is located in the above mentioned ranges. If that value of $x$ is in the range that makes $\mathtt{sbf}_{\Gamma_s}(x) < \mathtt{sbf}_{\Omega_s}(x)$ then it will decrease the subsystem normal budget, otherwise, it will not. The amount of improvement when using the improved local analysis compared with the original analysis, on the system load, depends on many factors such as the size of $X_s$, the subsystem period, and the difference between the subsystem period and tasks' deadlines. The higher the value of $X_s$, the more improvement can be achieved. Also, if the difference between the subsystem period and tasks' deadlines is low then the improvement in system load becomes higher. If the difference between the subsystem period and tasks' deadlines is high then the $x$ that makes the left and right hand side of (10.7) to be equal, becomes very far from the subsystem period and in this case a small increment in the subsystem normal budget will be enough to cover the difference between $\mathtt{sbf}_{\Gamma_s}(x)$ and $\mathtt{sbf}_{\Omega_s}(x)$ which also affect the improvement in the system load.

Now we will explain the impact of the improved global analysis on the system load, and we will use Figure 10.4 for illustration. When the $X_s/\mathsf{load}_{\mathsf{sys}}$

part in Figure 10.4 is as large as possible, the improved global analysis contributes with a greater improvement. The reason for this is that during this part there will be no (or limited) preemptions from higher priority subsystems. Hence, the difference between this part and the other part $(I + Q_s)/\mathsf{load}_{\mathsf{sys}}$ should be low to achieve a greater improvement, where $I$ is the interference from higher priority subsystems (including the sum of $Q_t + X_t$ of the higher priority subsystems) and also the blocking from lower priority subsystems. We can distinguish some cases that the improved global analysis can not reduce the system load. First, if the subsystem period of all subsystems are equal then there will be no preemptions from higher priority subsystems during the overrun time. Since the improved global analysis is based on removing the interference from higher priority subsystems during the overrun from the global analysis, then using the improved global analysis can not decrease the system load. Another case that the improved global analysis can not decrease the system load, is when the subsystem that requires the maximum CPU resource (i.e., the system load was computed based on its CPU requirement), is the highest priority subsystem or it does not access a global shared resource.

Finally, combining both the local improvement and the global improvement can require lower system load. As mentioned previously, the improved local analysis has a potential to decrease the subsystem budget which will decrease the interference from higher priority subsystems and the budget of the subsystem itself $(I + Q_s)$.

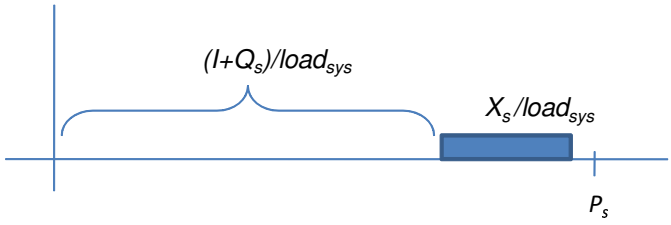


Figure 10.4: Considering MONP analysis for $S_s$.

### 10.7.2 Simulation setting

The simulation is performed by applying the modified overrun without payback analysis (MONP), including the improved local and global analysis, on 1000 different randomly generated systems. Initially, we assumed that each

system consists of 5 subsystems and each subsystem contains 4 tasks. A task is assumed to access at most one globally shared resource and 2 tasks in each subsystem access globally shared resources and we assume that there is only one global shared resource.

For simplicity, we assume that the internal resource ceilings of the globally shared resources are equal to the highest task priority in each subsystem (i.e., $rc_{sl} = 1$), and $T_i$ = $D_i$ for all tasks. For each simulation study the following settings are changed and a new 1000 systems is generated:

1. Critical section execution time $CS_s$. It specifies the maximum absolute time that a task may access a global shared resource. Changing this parameter does not require to generate new 1000 system, since changing only this parameter will not have effect on the other task parameters as we will show later.

2. Subsystem period $P_s$ and task period $T_{si}$. The subsystem/task period is specified as a range with a lower and upper bound. The simulation program generates a subsystem/task period randomly within the specified range, following a uniform distribution.

3. Number of subsystems $N$.

4. System utilization $U^{\mathcal{S}}$. The sum of the utilization of all tasks in the system, is specified to a desired value.

The given system utilization is divided randomly among the subsystems, and the assigned utilization to each subsystem is in turn divided randomly to the tasks that belong to that subsystem. Since the task period is generated to a value within the interval as specified, the execution time is derived from the desired task utilization. The critical section execution time is given as an input parameter, however, its value can not be greater than the execution time of its task. Therefore the critical section is set to be equal to the minimum value of the task execution time and the given critical section execution time, i.e., $c_{sil} = \min(CS_s, C_{si})$. All randomized system parameters are generated following uniform distributions.

### 10.7.3 Simulation results

We have performed 4 different simulation studies as described below;

- **Study 1** is specified having critical section execution time $CS_s \in \{2, 4, 6, 8\}$, task periods $T_{si} \in [140, 1000]$, subsystem periods $P_s \in [40, 70]$, $U^{\mathcal{S}} = 20\%$ and $N = 5$.
- **Study 2** increase the range of the subsystem periods $P_s$ and task periods $T_{si}$ (compared to Study 1) to
  A.) $P_s \in [50, 200]$ and $T_{si} \in [400, 1000]$,
  B.) $P_s \in [100, 200]$ and $T_{si} \in [400, 1000]$.
- **Study 3** change the number of subsystems (compared to Study 1) to $N \in \{4, 6, 8\}$.
- **Study 4** change the system utilization (compared to Study 1) to $U^{\mathcal{S}} \in \{10\%, 30\%\}$ with $CS_s = 2$.

| $CS_s$ | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Q1 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.505 | 0.664 | 0.763 | 0.836 |
| Median $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.532 | 0.717 | 0.849 | 0.940 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.560 | 0.771 | 0.929 | $> 1$ |
| schedulable ONP | 100% | 100% | 89.1% | 67% |
| Q1 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.475 | 0.613 | 0.702 | 0.760 |
| Median $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.495 | 0.655 | 0.770 | 0.845 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.516 | 0.696 | 0.837 | 0.940 |
| schedulable MONP | 100% | 100% | 98.3% | 84.7% |
| MONP/ONP med. improv. | 7.4% | 9.4% | 10.2% | 11.1% |
| MONP/ONP max. improv. | 13.2% | 19.6% | 25.7% | 30.0% |

Table 10.3: Measured results of Study 1

Table 10.3 shows results of **Study 1**. The results of each method (ONP and MONP) are shown using the median, lower quartile ($Q1$) and the higher quartile ($Q3$) of the system load values of the 1000 generated systems. The percentage of schedulable systems, out of the 1000 generated systems, is shown. In addition, it shows the percentage of improvement in the system load based on the evaluated median (explained above) and the maximum improvement when using MONP compared with ONP. It is calculated as $100 * (\mathsf{load}_{\mathsf{sys}}^{\mathrm{ONP}} - \mathsf{load}_{\mathsf{sys}}^{\mathrm{MONP}})/\mathsf{load}_{\mathsf{sys}}^{\mathrm{MONP}}$, where $\mathsf{load}_{\mathsf{sys}}^{\mathrm{ONP}}$ is the median or maximum system load, depending on what is required to be evaluated, using ONP analysis. Figure 10.5 shows the number of systems that have system load within the ranges shown in the x-axis (the lines that connect points are only used for illustration) for $CS_s = 2$, while Figure 10.6 shows the results for the case when $CS_s = 8$. Note that for the case of $CS_s = 8$, some of the systems are unschedulable (i.e., having $\mathsf{load}_{\mathsf{sys}} > 1$) using both ONP and MONP, and they are shown in

Figure 10.6 as the systems that have $\mathsf{load}_{\mathsf{sys}} > 100\%$ as it is not important to find the actual system load for unschedulable systems.

Looking at the results in Table 10.3, it is clear that MONP can give better results compared with traditional ONP, in terms of a lower system load and more schedulable systems when increasing $CS_s$ (same results are shown in Tables 10.4 and 10.5). In this study, the ratio $CS_s/P_s$ is relatively high and that is the reason why MONP performs significantly better than ONP, which is the characteristics that we are looking for.

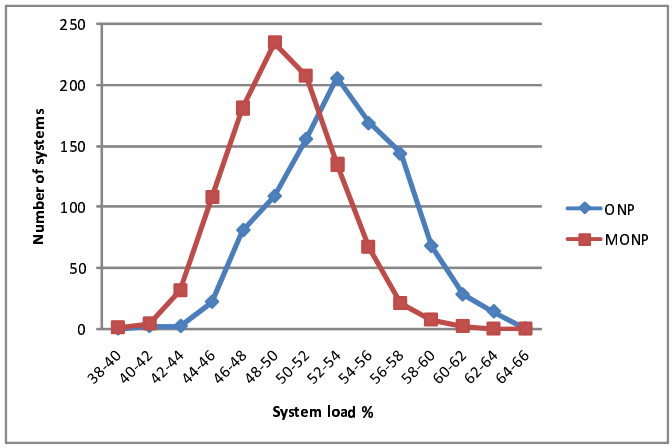


Figure 10.5: Results of Study 1 for $CS_s = 2$.

In **Study 2**, we decrease the ratio $CS_s/P_s$ by increasing the range of subsystem period. In Table 10.4, the subsystem period is selected as $P_s = [50, 200]$. In this case, the improvement that MONP can achieve is less than the case in **Study 1** because $X_s/\mathsf{load}_{\mathsf{sys}}$ becomes less significant within the subsystem period $P_s$ (see Figure 10.4). In Table 10.5, we change the range of subsystem period to $P_s = [100, 200]$, and that cause MONP to give better results compared with the results in Table 10.4. The reason for this improvement is that when the difference between the minimum and maximum subsystem period is decreased, then also the maximum number of interference (preemptions) from higher priority subsystems is decreased. This will decrease the contribution of the higher priority subsystems in equations (10.12) - (10.19) which in turn decreases the required subsystem load when using MONP.

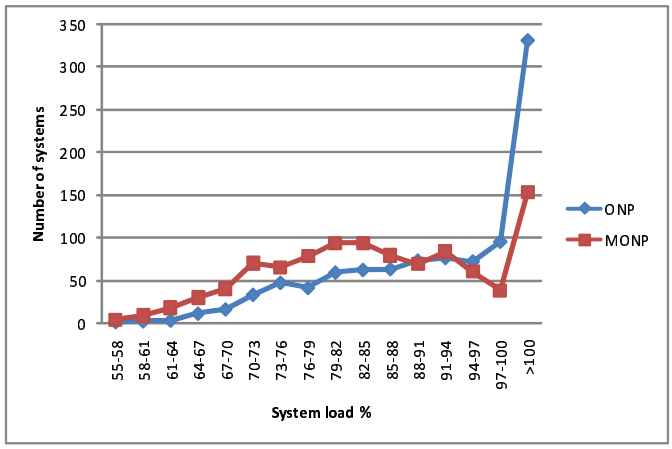


Figure 10.6: Results of Study 1 for $CS_s = 8$.

Looking at the MONP/ONP med. improv. line in Table 10.4 and Table 10.5, the rate of the improvement when increasing $CS_s$ from 2 to 4 is lower when increasing $CS_s$ from 4 to 6 and when increasing $CS_s$ from 6 to 8 (for example in Table 10.4, the difference between $4.9\% - 3.1\%$ is higher than $5.6\% - 4.9\%$ ). The reason for this is that increasing $CS_s$ of tasks will increase the required subsystems normal budgets for their subsystems (see (10.7)) which, in turn, will increase the interference from higher priority subsystems in (10.12) - (10.19) and that limits the improvement that MONP can achieve.

| $CS_s$ | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Q1 $\text{load}_{\text{sys}}$ ONP | 0.444 | 0.538 | 0.607 | 0.659 |
| Median $\text{load}_{\text{sys}}$ ONP | 0.462 | 0.562 | 0.644 | 0.704 |
| Q3 $\text{load}_{\text{sys}}$ ONP | 0.481 | 0.589 | 0.683 | 0.755 |
| schedulable ONP | 100% | 100% | 100% | 99.8% |
| Q1 $\text{load}_{\text{sys}}$ MONP | 0.430 | 0.512 | 0.573 | 0.620 |
| Median $\text{load}_{\text{sys}}$ MONP | 0.448 | 0.536 | 0.609 | 0.661 |
| Q3 $\text{load}_{\text{sys}}$ MONP | 0.467 | 0.563 | 0.643 | 0.708 |
| schedulable MONP | 100% | 100% | 100% | 100% |
| MONP/ONP med. improv. | 3.1% | 4.9% | 5.6% | 6.2% |
| MONP/ONP max. improv. | 8.1% | 12.3% | 16.2% | 16.8% |

Table 10.4: Measured results of Study 2A, i.e. $P_s \in [50, 200]$

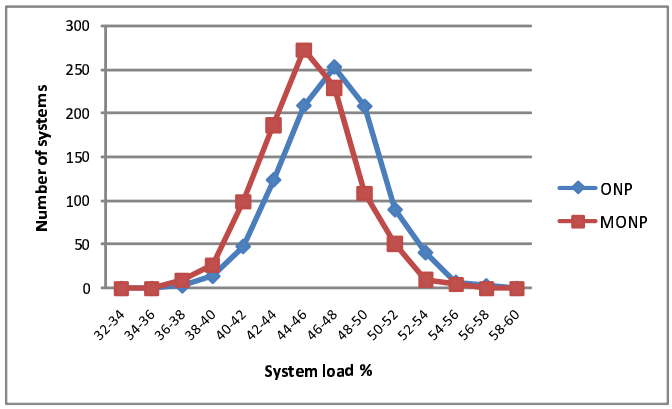


Figure 10.7: Results of Study 2 for $P_s \in [100, 200]$ and $CS_s = 2$.

| $CS_s$ | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Q1 $load_{sys}$ ONP | 0.446 | 0.522 | 0.579 | 0.623 |
| Median $load_{sys}$ ONP | 0.468 | 0.548 | 0.611 | 0.662 |
| Q3 $load_{sys}$ ONP | 0.488 | 0.572 | 0.643 | 0.699 |
| schedulable ONP | 100% | 100% | 100% | 100% |
| Q1 $load_{sys}$ MONP | 0.432 | 0.497 | 0.545 | 0.583 |
| Median $load_{sys}$ MONP | 0.454 | 0.518 | 0.573 | 0.616 |
| Q3 $load_{sys}$ MONP | 0.473 | 0.543 | 0.604 | 0.651 |
| schedulable MONP | 100% | 100% | 100% | 100% |
| MONP/ONP med. improv. | 3, 1% | 5.8% | 6, 6% | 7.5% |
| MONP/ONP max. improv. | 6.4% | 11.9% | 16.5% | 17.2% |

Table 10.5: Measured results of Study 2B, i.e. $P_s \in [100, 200]$

In **Study 3**, we investigate the effect of changing the number of subsystems, and the results are shown in Table 10.6. We can see that increasing $N$ will decrease the improvement that MONP can achieve over ONP. The reason for this is that increasing the number of subsystems will increase the interference $I$ of the higher priority subsystems which, in turn, will decrease the improvement as explained in the previous section.

Finally, in **Study 4** we investigate the effect of changing the system utilization on the performance of MONP. The results in Table 10.7 show that increasing the value of $U^S$ will decrease the improvement that MONP can achieve over ONP. The reason for this is that increasing the value of $U^S$ will increase the subsystem normal budget for all subsystems which increases the contribution of the higher priority subsystems in (10.12) - (10.19) and will limit the

| $N$ | 4 | 5 | 6 | 8 |
|---|---|---|---|---|
| Q1 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.459 | 0.505 | 0.560 | 0.674 |
| Median $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.483 | 0.531 | 0.590 | 0.708 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.506 | 0.560 | 0.617 | 0.743 |
| schedulable ONP | 100% | 100% | 100% | 100% |
| Q1 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.430 | 0.475 | 0.526 | 0.637 |
| Median $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.448 | 0.495 | 0.549 | 0.669 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.467 | 0.516 | 0.575 | 0.702 |
| schedulable MONP | 100% | 100% | 100% | 100% |
| MONP/ONP med. improv. | 7.8% | 7.4% | 7.3% | 5.9% |

Table 10.6: Measured results of Study 3 for $CS_s = 2$

potential improvement of MONP as explained previously.

| $U^{\mathcal{S}}$ | 10% | 20% | 30% |
|---|---|---|---|
| Q1 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.348 | 0.505 | 0.661 |
| Median $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.376 | 0.531 | 0.690 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ ONP | 0.402 | 0.560 | 0.718 |
| schedulable ONP | 100% | 100% | 100% |
| Q1 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.323 | 0.475 | 0.625 |
| Median $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.344 | 0.495 | 0.649 |
| Q3 $\mathsf{load}_{\mathsf{sys}}$ MONP | 0.368 | 0.516 | 0.674 |
| schedulable MONP | 100% | 100% | 100% |
| MONP/ONP med. improv. | 9.3% | 7.4% | 6.2% |

Table 10.7: Measured results of Study 4 for $CS_s = 2$

## 10.8 Conclusion

In this paper we have showed that existing global and local schedulability analysis of synchronization protocols based on SRP and overrun without payback for two-level hierarchical scheduling based on FPPS is pessimistic. We have presented an improved global and local analysis assuming that the deadline of a subsystem holds for the sum of its normal budget and its overrun budget, and showed that the global analysis is both uniform and sustainable. We have illustrated the improvements by means of examples, and have evaluated the improvement through an extensive simulation study. The evaluation results show that MONP can improve the CPU requirement significantly for certain cases especially when the ration between $X_s/P_s$ is high which makes the performance of the traditional ONP, very low.

# Bibliography


[1] 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 $7^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[2] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Scheduling of semi-independent real-time components: Overrun methods and resource holding times. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, pages 575–582, September 2008.

[3] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the $24^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[4] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[5] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[6] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the $20^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[7] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the* $21^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[8] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 129–138, December 2007.

[9] T.M. Ghazalie and T.P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1):31–67, July 1995.

[10] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[11] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Proceedings of the* 29$^{th}$ *IEEE Real-Time Systems Symposium (RTSS'08)*, pages 209–22, December 2008.

[12] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the* 15$^{th}$ *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[13] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *Proceedings of the 19$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'19)*, pages 269–279, July 2007.

[14] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems journal*, 42(1-3):63–119, August 2009.

[15] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *Proceedings of the* 23$^{rd}$ *IEEE Real-Time Systems Symposium (RTSS'02)*, pages 315–326, December 2002.

[16] R. Bril, U. Keskin, M. Behnam, and T. Nolte. Schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks revisited. In *Proceedings of the* 2$^{nd}$ *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09)*, pages 24–32, December 2009.

[17] S. Baruah and A. Burns. Sustainable schedulability analysis. In *Proceedings of the 27$^{th}$ IEEE Real-Time Systems Symposium (RTSS'06)*, pages 159–168, December 2006.

[18] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.

# Chapter 11

# Paper F: Refining SIRAP with a Dedicated Resource Ceiling for Self-Blocking

Moris Behnam, Thomas Nolte, Reinder J. Bril

In Proceedings of the $9^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'09), pages 157-166, October, 2009.

**Abstract**

In recent years, several synchronization protocols for re- source sharing have been presented for use in a Hierarchical Scheduling Framework (HSF). An initial comparative assess- ment of existing protocols revealed that none of the proto- cols is superior to the others and that the performance of a protocol heavily depends on system parameters. In this paper, we aim at efficiency improvements of the synchro- nization protocol SIRAP and its associated schedulability analysis, where efficiency refers to calculated CPU resource needs. The contribution of the paper is threefold. Firstly, we present an improvement of the schedulability analysis for SIRAP, which makes SIRAP more efficient. Secondly, we generalize SIRAP by distinguishing separate resource ceil- ings for self-blocking and resource access. Using a separate resource ceiling for self-blocking enables a reduction of the interference from lower priority tasks, which can result in ef- ficiency improvements. The efficiency improvement depends on both subsystem characteristics and the value selected for the resource ceiling for self-blocking, however. The third contribution of this paper is therefore an algorithm that given a subsystem selects for each globally shared resource an optimal value in terms of efficiency for its resource ceil- ing for self-blocking. The efficiency improvement gained by the algorithm compared to the original SIRAP approach is evaluated by means of simulation.

## 11.1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical CPU sharing among applications under different scheduling services [1]. The HSF can be represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., tasks), and resources are allocated from a parent node to its children nodes.

The HSF provides means for decomposing a complex system into well-defined parts called *subsystems*. In essence, the HSF provides a mechanism for timing-predictable *composition* of course-grained subsystems. In the HSF a subsystem provides an introspective *interface* that specifies the timing properties of the subsystem precisely [1]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal interference. Temporal isolation between subsystems is provided through budgets which are allocated to subsystems.

**Motivation:** Research on HSFs started with the assumption that subsystems are independent, i.e., inter-subsystem resource sharing other than the CPU fell outside their scope. In some cases [2, 3], intra-subsystem resource sharing is addressed using existing synchronization protocols for resource sharing between tasks, e.g., the Stack Resource Policy (SRP) [4]. Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [5], BROE [6], and SIRAP [7]. Although all three protocols are SRP-based, they rely on different mechanisms to deal with inter-subsystem resource sharing and depletion of budgets. In particular, HSRP is based on a so-called *overrun* mechanism, whereas both BROE and SIRAP are based on a so-called *skipping approach*. Moreover, their constituting frameworks are based on different assumptions. As an example, scheduling (of subsystems as well as of tasks) in the frameworks of HSRP and SIRAP is based on FPPS, whereas it is based on EDF for BROE. Finally, unlike BROE and SIRAP, HSRP does not support local schedulability analysis, and the local schedulability analysis in BROE as described in [6] is incomplete. An initial comparative assessment of these three synchronization protocols [8] revealed that none of them was superior to the others and that the performance of a protocol heavily depends on the system parameters. A comparative evaluation of the mechanisms overrun and skipping using a single framework can be found in [9]. In this paper, we focus on SIRAP and aim at improving the efficiency of the protocol and its associated schedulability analysis, where efficiency refers to calculated CPU resource needs of a subsystem.

SIRAP is based on a skipping mechanism to prevent depletion of a subsystem budget during global shared resource access. That is, whenever a task tries to lock a global shared resource and the remaining budget is insufficient to complete the access, the task experiences *self-blocking* during the remainder of the current budget period and is guaranteed to access the resource during the next budget period. The contribution of this paper is threefold. Firstly, we remove some pessimism from SIRAP by improving its associated schedulability analysis. Secondly, we generalize SIRAP by distinguishing separate resource ceilings for self-blocking and resource access. Using a dedicated resource ceiling for self-blocking enables a reduction of the interference from lower priority tasks which may reduce the calculated resource needs of the subsystem whilst guaranteeing the schedulability of all its internal tasks. Thirdly, we propose an algorithm to select the optimal value per global shared resource for this novel resource ceiling for a subsystem with given characteristics, resulting in the lowest calculated resource needs for that subsystem. The efficiency of the algorithm is evaluated by comparing its calculated resource needs with those of the original SIRAP protocol in a simulation.

## 11.2 Related work

Over the years, there has been a growing attention to hierarchical scheduling of real-time systems [2, 10, 11, 12, 3, 13, 14, 15, 16, 1]. Deng and Liu [11] proposed a two-level HSF for open systems, where subsystems may be developed and validated independently. Kuo and Li [3] presented schedulability analysis techniques for such a two-level framework with the Fixed-Priority Scheduling (FPS) global scheduler. Lipari and Baruah [13, 17] presented schedulability analysis techniques for Earliest Deadline First (EDF) global schedulers. Mok *et al.* [18, 12] 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 FPS [2, 14, 10] and under EDF scheduling [1, 19]. However, a common assumption shared by all above studies is that tasks are independent.

Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [5], BROE [6], and SIRAP [7]. A comparative assessment of these three synchronization protocols [8] revealed that none of them was superior to the others and that the performance of a pro-

tocol heavily depends on system parameters.

## 11.3 System model and background

This paper focuses on scheduling of a single node or a single network link, where each node (or link) is modeled as a system $\mathcal{S}$ consisting of one or more subsystems $S_s \in \mathcal{S}$. The system is scheduled by a two-level HSF as shown in Figure 11.1. During runtime, the system level scheduler (global scheduler) selects, at all times, which subsystem will access the common (shared) CPU resource.

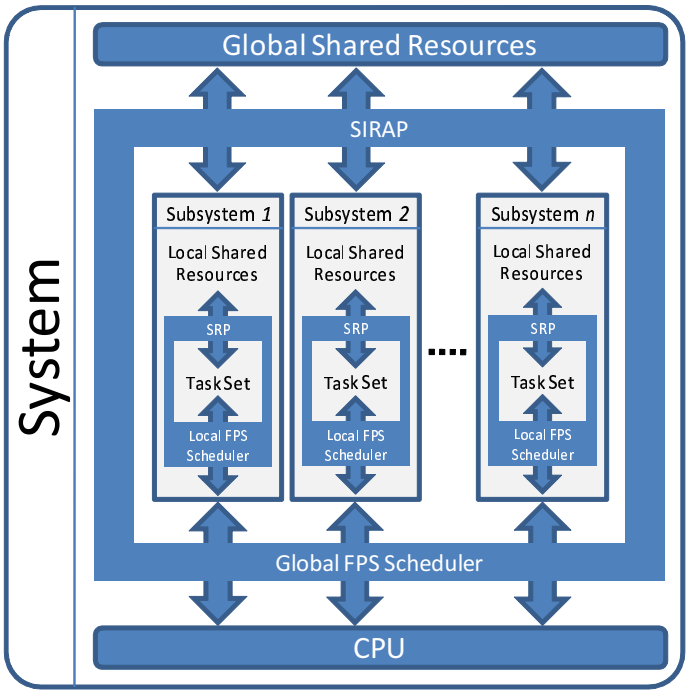


Figure 11.1: HSF with resource sharing.

**Subsystem model** A subsystem $S_s$ consists of a set $\mathcal{T}_s$ of $n_s$ tasks and a local scheduler. Once a subsystem is assigned the processor (CPU), its scheduler will select which of its tasks will be executed. With each subsystem $S_s$, a subsystem timing interface $S_s(P_s, Q_s, X_s)$ is associated, where $Q_s$ is the subsystem budget that the subsystem $S_s$ will receive every subsystem period $P_s$, and $X_s$ is the maximum time that a subsystem internal task may lock a shared resource. Finally, both the local scheduler of a subsystem $S_s$ as well as the global scheduler of the system $\mathcal{S}$ is assumed to implement the fixed priority preemptive scheduling policy. Let $\mathcal{R}_s$ be the set of $m_s$ global shared resources accessed by $S_s$.

**Task model** The task model considered in this paper is the 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 arrival of successive jobs of $\tau_i$, $C_i$ is their worst-case execution-time, and $D_i$ is an arrival-relative deadline ($0 < C_i \leq D_i \leq T_i$) before which the execution of a job must be completed. Each task is allowed to access one or more shared 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 $R_j$. It is assumed that all tasks belonging to the same subsystem are assigned unique static priorities and are sorted according to their priorities in the order of increasing priority. Without loss of generality, it is assumed that the priority of a task is equal to the task ID number after sorting, and the greater a task ID number is, the higher its priority is. The same assumption is made for the subsystems. The set of shared resources accessed by $\tau_i$ is denoted $\{R^i\}$. Let $\mathtt{hp}(i)$ return the set of local tasks that belong to a subsystem with priorities higher than that of $\tau_i$ and $\mathtt{lp}(i)$ return the set of local tasks with priorities lower than that of task $\tau_i$. Table 11.1 shows the summary of the notations used in this paper. For each subsystem, we assume that the subsystem period is selected such that $2P_s \leq T_{min}$, where $\tau_{min}$ is the task with the shortest period. The motivation for this assumption is that higher $P_s$ will require more CPU resources [20]. In addition, this assumption simplifies the presentation of the paper (evaluating $X_s$).

**Shared resources** The presented HSF allows for sharing of logical resources between arbitrary tasks, located in arbitrary subsystems, 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.

| Notation | Description |
|---|---|
| $S$ | System |
| $S_s$ | Subsystem |
| $\mathcal{T}_s$ | Sporadic task set |
| $n_s$ | Number of local tasks. |
| $P_s$ | Subsystem period |
| $Q_s$ | Subsystem budget |
| $X_s$ | Maximum time that $S_s$ locks a global shared resource. |
| $\mathcal{R}_s$ | Set of global shared resources accessed by $S_s$ |
| $R_j$ | Global shared resource |
| $m_s$ | Cardinality of $\mathcal{R}_s$ |
| $\tau_i$ | Sporadic task |
| $T_i$ | Period of $\tau_i$ |
| $C_i$ | Worst case execution time of $\tau_i$ |
| $D_i$ | Relative deadline of $\tau_i$ |
| $c_{i,j}$ | Critical section execution times of $\tau_i$ accessing $R_j$ |
| $\{c_{i,j}\}$ | Set of critical section execution times of $\tau_i$ accessing $R_j$ |
| $\{R^i\}$ | Set of shared resources accessed by $\tau_i$ |
| $\mathtt{hp}(i)$ | Set of local tasks with priorities higher than that of $\tau_i$ |
| $\mathtt{lp}(i)$ | Set of local tasks with priorities lower than that of $\tau_i$ |
| $rc_j$ | Internal resource ceiling of $R_j$ |
| $RX_j$ | External resource ceiling |
| $SC$ | System ceiling |
| $sc_s$ | Subsystem ceiling |
| $rc_j^{\mathrm{LWB}}$ | Lower bounds for $rc_j$ |
| $RX_j^{\mathrm{LWB}}$ | Lower bound for $RX_j$ |
| $\mathtt{sbf}_s(t)$ | Supply bound function |
| $\mathtt{rbf}_{\mathtt{FP}}(i,t)$ | Request bound function of $\tau_i$ |
| $I_S(i)$ | Self blocking of $\tau_i$ |
| $I_H(i,t)$ | Interference from tasks with priority higher than that of $\tau_i$ |
| $I_L(i)$ | Interference from tasks, with priority lower than that of $\tau_i$ |
| $src_j$ | Self blocking ceiling of $R_j$ |

Table 11.1: Summary of notations.

A resource that is used by tasks in more than one subsystem is denoted a *global shared resource*. The work in this paper targets managing global shared resources, and throughout the remainder of the paper these are simply denoted as shared resources.

To be able to use SRP in a HSF for synchronizing global shared resources, its associated terms resource, system and subsystem ceilings are extended as follows:

*Resource ceiling*: Each global shared resource $R_j$ is associated with two types of resource ceilings; an *internal* resource ceiling ($rc_j$) for local scheduling and an *external* resource ceiling ($RX_j$) for global scheduling. Lower bounds for $rc_j$ and $RX_j$ are defined as $rc_j^{\mathrm{LWB}} = \max\{i|\tau_i \in \mathcal{T}_s \text{ accesses } R_j\}$ and $RX_j^{\mathrm{LWB}} = \max\{s|S_s \text{ accesses } R_j\}$, respectively.

*System/subsystem ceiling*: The system/subsystem ceilings ($SC/sc_s$) are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling of a currently locked resource in the system/subsystem.

Under SRP, a task $\tau_k$ can preempt the currently executing task $\tau_i$ (even inside a critical section) within the same subsystem, only if the priority of $\tau_k$ is greater than its corresponding subsystem ceiling. The same reasoning applies for subsystems from a global scheduling point of view. An attractive property of SRP is that it allows tasks within a subsystem to share a common stack.

## 11.4 SIRAP

The SIRAP [7] protocol can be used for independent development of subsystems and it supports subsystem integration in the presence of globally shared logical resources. It uses a periodic resource model [1] to abstract the timing requirements of each subsystem. SIRAP uses the SRP protocol to synchronize access to global shared resources in both local and global scheduling. SIRAP applies a *skipping approach* to prevent the budget expiration inside critical section problem. The mechanism 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 execution before the subsystem budget expires. This can be achieved by checking the remaining budget before granting the access to globally shared resources; if there is sufficient remaining budget then the job enters the critical section, and if there is insufficient remaining budget, the local scheduler delays the critical section entering of the job (i.e., the job blocks itself and its state becomes *self blocking*) until the next sub-

system budget replenishment and guarantees access to the resource during the next subsystem budget period. In addition, it sets the subsystem ceiling equal to the internal resource ceiling of the resource that the self blocked job wanted to access, to prevent the execution of all tasks that have a priority at most equal to the ceiling of the resource until the job releases the resource.

**Local schedulability analysis** The local schedulability analysis under FPS is as follows [7, 1]:

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; \mathtt{rbf}_{\mathsf{FP}}(i,t) \leq \mathtt{sbf}_s(t), \tag{11.1}$$

where $\mathtt{sbf}_s(t)$ is the *supply bound function* based on the periodic resource model presented in [1] that computes the minimum possible CPU supply to $S_s$ for every interval length $t$, and $\mathtt{rbf}_{\mathsf{FP}}(i,t)$ denotes the *request bound function* of a task $\tau_i$. $\mathtt{sbf}_s(t)$ can be calculated as follows:

$$\mathtt{sbf}_s(t) = \begin{cases} t - (k+1)(P_s - Q_s) & \text{if } t \in V^{(k)} \\ (k-1)Q_s & \text{otherwise,} \end{cases} \tag{11.2}$$

where $k = \max\Big(\big\lceil \big(t - (P_s - Q_s)\big)/P_s \big\rceil, 1\Big)$ and $V^{(k)}$ denotes an interval $[(k+1)P_s - 2Q_s, (k+1)P_s - Q_s]$.

Note that, for Eq. (11.1), $t$ can be selected within a finite set of scheduling points [21]. The request bound function $\mathtt{rbf}_{\mathsf{FP}}(i,t)$ of a task $\tau_i$ is given by:

$$\mathtt{rbf}_{\mathsf{FP}}(i,t) = C_i + I_S(i) + I_H(i,t) + I_L(i), \tag{11.3}$$

$$I_S(i) = \sum_{R_k \in \{R^i\}} X_{i,k}, \tag{11.4}$$

$$I_H(i,t) = \sum_{\tau_j \in \mathtt{hp}(i)} \left\lceil \frac{t}{T_j} \right\rceil (C_j + \sum_{R_k \in \{R^j\}} X_{j,k}), \tag{11.5}$$

$$I_L(i) = \max_{\tau_f \in \mathtt{lp}(i)} (2 \cdot \max_{\forall R_j | rc_j \geq i} (X_{f,j})), \tag{11.6}$$

where $I_S(i)$ is the self blocking of task $\tau_i$, $I_H(i,t)$ is the interference from tasks with priority higher than that of $\tau_i$, and $I_L(i)$ is the interference from tasks, with priority lower than that of $\tau_i$, that access shared resources.

**Subsystem budget** In this paper, it is assumed that the subsystem period is given while the minimum subsystem budget should be computed so that the system will require lower CPU resources. Given a subsystem $S_s$, and $P_s$, let $\mathsf{calculateBudget}(S_s, P_s)$ denote a function that calculates the smallest subsystem budget $Q_s$ that satisfies Eq. (11.1) (the function is similar to the one presented in [1]). Hence,

$$Q_s = \mathsf{calculateBudget}(S_s, P_s). \tag{11.7}$$

**Calculating $X_s$** Given a subsystem $S_s$, its critical section execution time $X_s$ represents the maximum time that a subsystem internal task may lock a shared resource. Note that any task $\tau_i$ accessing a resource $R_j$ can be preempted by tasks with priority higher than $rc_j$. Note that SIRAP prevents subsystem budget expiration inside a critical section of a global shared resource. When a task experiences self-blocking during a subsystem budget period it is guaranteed access to the resource during the next period. A sufficient condition to provide this guarantee is

$$Q_s \geq X_s. \tag{11.8}$$

We now derive $X_s \leq Q_s < P_s$ and since we assume that $2P_s \leq T_{min}$ then all tasks that are allowed to preempt while $\tau_i$ accesses $R_j$ will be activated at most one time from the time that self blocking happens until the end of the next subsystem period. Then $X_{i,j}$ which represents the maximum time that $\tau_i$ locks $R_j$, can be computed as follows,

$$X_{i,j} = c_{i,j} + \sum_{k=rc_j+1}^{n_s} C_k. \tag{11.9}$$

Let $X_j = \max\{X_{i,j}|$ for all $\tau_i \in \mathcal{T}_s$ accessing $R_j\}$, then $X_s = \max\{X_j|$ for all $R_j \in \mathcal{R}_s\}$.

**Internal resource ceiling** Looking at Eq. (11.9), assigning internal resource ceilings according to SRP may make the value of $X_s$ very high which causes the subsystem to require more CPU resources. One way to handle this problem is by preventing the preemption inside the subsystem when a task is accessing a shared resource as proposed in [5] so $X_{i,j} = c_{i,j}$. It can be implemented using SRP by assigning the resource ceiling of all resources equal to the maximum task priority $rc_j = n_s$ where $n_s$ is the task ID number of the highest priority task. However, Bertogna *et al.* [22] showed that preventing preemption while

accessing a global shared resource may violate the local schedulability of the subsystem and proposed an algorithm based on increasing the ceiling of all resources in steps as much as possible without violating the local schedulability. Finally, Shin *et al.* [23] showed that there is a tradeoff between decreasing the value of $X_s$ and the minimum subsystem budget required to guarantee the schedulability of the subsystem.

The result of this paper does not depend on any of the discussed methods to set the internal resource ceiling. So we assume that the internal ceiling of resource $R_j$ can be selected within the following range $n_s \geq rc_j \geq rc_j^{\mathrm{LWB}}$.

## 11.5 Improved SIRAP analysis

In this section we will show that Eq. (11.6) is pessimistic and can be improved such that the subsystem budget may decrease. Each task $\tau_i$ that shares a global resource $R_j$ with a lower priority task $\tau_f$ can be blocked by $\tau_f$ due to (*i*) self blocking of $\tau_f$ and in addition due to (*ii*) access of $R_j$ by $\tau_f$. The maximum blocking times of (*i*) and (*ii*) are given by the self blocking time $X_{f,j}$, and the maximum execution time $c_{f,j}$ of $\tau_f$ inside a critical section of $R_j$, respectively. Note that preemption of tasks with priority higher than $rc_j$ can be excluded from the resource access of $R_j$ by $\tau_f$, because those preemptions are already incorporated in $I_H(i,t)$ (in Eq. (11.5)). The worst-case blocking is the summation of the blocking from these two scenarios, as shown in Eq. (11.10).

$$I_L(i) = \max_{\tau_f \in \mathtt{lp}(i)} \big( \max_{\forall R_j | rc_j \geq i} (X_{f,j} + c_{f,j}) \big). \tag{11.10}$$

Since $c_{f,j} \leq X_{f,j}$, the interference $I_L(i)$ of tasks with a priority lower than that of task $\tau_i$, based on (8), is at most equal to that of (6). As a result, $\mathtt{rbf}_{\mathtt{FP}}(i,t)$ may decrease, and the corresponding subsystem budget $Q_s$ may therefore decrease as well.

## 11.6 Improved SIRAP protocol

In this section, we present a generalization of SIRAP, providing options for efficiency improvements of the protocol. First, we consider a dedicated setting for the subsystem ceiling during self-blocking. Next, we show that the efficiency of the protocol depends on both that setting and the subsystem parameters. Selecting an optimal setting is the topic of the next section.

### 11.6.1 Subsystem ceiling for self-blocking

Looking at Eq. (11.1), one way to reduce the subsystem budget $Q_s$ is by decreasing $\texttt{rbf}_{\texttt{FP}}(i,t)$ for tasks that require highest subsystem budget. In Section 11.5, we have described one way to decrease $\texttt{rbf}_{\texttt{FP}}(i,t)$ for higher priority tasks that share resources by decreasing $I_L(i)$. In this section we propose a method that allows for a further reduction of $I_L(i)$. According to SIRAP, when a task $\tau_i$ wants to enter a critical section of $R_j$ it first checks if the remaining budget is enough to release the shared resource before the budget expiration. If there is not enough budget remaining, then the task $\tau_i$ blocks itself and changes only the subsystem ceiling to be equal to $rc_j$. This prevents the execution of all tasks $\{\tau_k\}$ that have priority higher than that of $\tau_i$ and at most equal to the ceiling of $R_j$ (i.e., $rc_j \geq k > i$) that will be released after the self blocking instance.

The new method called E-SIRAP is based on allowing tasks in $\{\tau_k\}$ to execute during the self blocking time of $\tau_i$. This can be achieved by setting the subsystem ceiling equal to the priority of $\tau_i$ upon self blocking of task $\tau_i$ and raising the subsystem ceiling to the resource ceiling when $\tau_i$ actually accesses the resource. The main difference between SIRAP and E-SIRAP is the setting of subsystem ceiling when a task $\tau_i$ enters self blocking (wants to access a shared resource $R_j$ and there is not enough budget left). In SIRAP, the subsystem ceiling is set to $rc_j$, i.e., the resource ceiling of $R_j$ (the resource that caused the self blocking). While using E-SIRAP the subsystem ceiling is set to $i$, i.e., the priority of $\tau_i$, which is at most equal to $rc_j$. By choosing $i$ during self-blocking, we allow a maximum number of tasks to execute while preserving the attractive property of SRP that we can use a single stack for all tasks of a subsystem.

When using E-SIRAP, the maximum interference from lower priority tasks $I_L(i)$ will be decreased compared to Eq. (11.10), and can be calculated as;

$$I_L(i) = \max_{\tau_f \in \texttt{lp}(i)} \Big( \max_{\forall R_j | rc_j \geq i} (c_{f,j}) \Big). \tag{11.11}$$

According to the original SIRAP approach, if $\tau_i$ blocks itself, it should enter the critical section at the next subsystem budget replenishment. However, using E-SIRAP there is no guarantee that $\tau_i$ will enter the critical section at the next subsystem activation, since tasks with priority higher than that of $\tau_i$ and less than or equal to the ceiling of $R_j$ are also allowed to execute in the next subsystem activation. To guarantee that $\tau_i$ will enter its critical sections at the next subsystem budget replenishment, the subsystem budget should be

big enough to include the execution of those tasks. When using E-SIRAP, the sufficient condition (11.8) has to be revised to:

$$Q_s \geq X_{i,j} + \sum_{k \in \{i+1,\ldots,rc_j\}} C_k. \tag{11.12}$$

Hence, the minimum amount of budget needed for E-SIRAP may increase compared to SIRAP.

Since we assume that $2P_s \leq T_{min}$ then all higher priority tasks will be activated at most one time during the time $t \in [t_{rep}, t_{rep} + P_s]$ where $t_{rep}$ is the subsystem replenishment time after self blocking of task $\tau_i$.

Note that to evaluate $X_{i,j}$, Eq. (11.9) can be used without modification since E-SIRAP changes the behavior of SIRAP only within the self blocking time, and during the self blocking the task that caused self blocking is not allowed to access the shared resource. The only effect of using E-SIRAP is on the subsystem budget, hence efficiency can be defined exclusively in terms of $Q_s$.

Comparing Eq. (11.11) with Eq. (11.10), $I_L(i)$ may decrease significantly and that may decrease the subsystem budget. However, Eq. (11.12) is a stronger condition than Eq. (11.8), which may require a higher subsystem budget. Given these opposite forces, we conclude that E-SIRAP will not always decrease the minimum subsystem budget and therefore will not always give better results than the original SIRAP. We will illustrate this by the following example.

**Example 1:** Consider a subsystem $S_s$ that has three tasks and two of them share resource $R_1$ as shown in Table 11.2.

| $\mathcal{T}$ | $C_i$ | $T_i$ | $R_j$ | $c_{i,j}$ |
|---|---|---|---|---|
| $\tau_3$ | 2 | 30 | - | 0 |
| $\tau_2$ | 1 | 32 | $R_1$ | 1 |
| $\tau_1$ | 4 | 80 | $R_1$ | 4 |

Table 11.2: Task set parameters of Example 1.

Let the subsystem period be equal to $P_s = 15$. Using the original SIRAP, we derive $X_s = X_{1,1} = 6$ and $Q_s = 9.34$. Using E-SIRAP, we derive $X_s = X_{1,1} = 6$ and $Q_s = 7$. This latter value satisfies Eq. (11.12), i.e., $Q_s \geq X_{1,1} + C_2 = 7$. In this case, E-SIRAP decreases the subsystem budget, hence requires less CPU resources. Conversely, for $C_2 = 5$, we derive $Q_s = 10.67$ for the original SIRAP and derive $Q_s \geq X_{1,1} + C_2 = 11$ by applying Eq. (11.12) for E-SIRAP. In this case, the original SIRAP outperforms E-SIRAP.

### 11.6.2 Subsystem ceiling upon self-blocking

As described in the previous section, the subsystem ceiling using E-SIRAP is equal to the priority of the task that enters self blocking state during the self blocking time. However, using this setting for all shared resources during the self blocking of tasks may limit the performance improvement of E-SIRAP in terms of decreasing the subsystem budget as shown in the following example.

**Example 2:** Consider a subsystem $S_s$ that has four tasks as shown in Table 11.3 and the subsystem period $P_s = 50$.

| $\mathcal{T}$ | $C_i$ | $T_i$ | $R_j$ | $c_{i,j}$ |
|---|---|---|---|---|
| $\tau_4$ | 14.7 | 100 | $R_1, R_2, R_3$ | 0.1, 0.1, 0.1 |
| $\tau_3$ | 5 | 250 | $R_3$ | 4 |
| $\tau_2$ | 5 | 300 | $R_2$ | 3 |
| $\tau_1$ | 10 | 500 | $R_1$ | 0.1 |

Table 11.3: Task set parameters of Example 2.

Using original SIRAP, $Q_s = 23$, $X_s = X_3 = 4$ and $rc_1 = rc_2 = rc_3 = 4$. Using E-SIRAP the minimum budget that satisfies Eq. (11.1) is $Q_s = 19$ , however, to satisfy the condition in Eq. (11.12) when $\tau_1$ access $R_1$, the subsystem budget should be $Q_s \geq X_{1,1} + C_2 + C_3 + C_4 = 24.8$, when $\tau_2$ access $R_2$ then $Q_s \geq 22.7$ and for $\tau_3$ access $R_3$ then $Q_s \geq 18.7$. This means that the subsystem budget should be $Q_s = 24.8$.

If we use SIRAP setting for $R_1$ and E-SIRAP setting for the other shared resources then $Q_s = 19$ to satisfy Eq. (11.1), and to satisfy the condition in Eq. (11.12) for $\tau_1$ access $R_1$, then $Q_s \geq= 0.1$, when $\tau_2$ access $R_2$ then $Q_s \geq 22.7$ and for $\tau_3$ access $R_3$ then $Q_s \geq 18.7$. This means that the subsystem budget should be $Q_s = 22.7$.

Finally, if we set the subsystem ceiling equal to 3 when $\tau_2$ block itself after trying to lock $R_2$ then $Q_s = 19$ to satisfy Eq. (11.1), and to satisfy the condition in Eq. (11.12) for for $\tau_3$ access $R_3$ should be $Q_s \geq 18.7$ and $\tau_2$ the subsystem budget should be $Q_s \geq 17.7$. The subsystem budget for this case should be $Q_s = 19$.

It is clear that combining SIRAP and E-SIRAP gives better results than each alone but the last setting gives even better results (lowest subsystem budget) which is the combination of SIRAP (for $\tau_1$ access $R_1$) and E-SIRAP (for $\tau_3$ access $R_3$) and in between (for $\tau_2$ access $R_2$). However, there are two problems associated with this approach. First, each task access a shared resource should have its own setting for subsystem ceiling during the self blocking time

and that means we need $n_s \times m_s$ extra memory space to save these values as a worst case. The second problem is finding the best setting of subsystem ceiling for each task access a global shared resource. In Section 11.7 we present an algorithm that finds the best setting of the subsystem ceiling to decrease the subsystem budget $Q_s$.

To solve the first problem, we introduce *self blocking ceiling* $src_j$ as the ceiling of a global shared resource $R_j$ during the self blocking time of all tasks that access this resource. The value of the self blocking ceiling should be within $src_j \in [k, rc_j]$ where $k$ is the index of the lowest priority task that access $R_j$ i.e., $k = \min\{v|\tau_v \in \mathcal{T}_s \text{ accesses } R_j\}$. The self blocking ceiling will be used in assigning the subsystem ceiling $sc_s$ value during the self blocking, e.g., when $\tau_i$ blocks itself after failing to lock $R_j$, the following assignment takes place:

$$sc_s = \max(src_j, i). \tag{11.13}$$

The max function in Eq. (11.13) is used to prevent all lower priority tasks $\tau_k$ that have $src_j \leq k < i$, from being executed during the self blocking of $\tau_i$. One of the advantages of using self blocking ceiling is that it decreases the memory space required to save the setting during the self blocking of tasks to $m_s$, however, it increases the runtime overhead since it uses the max function.

Note that if it is required to use SIRAP setting for $R_j$ then it is simply achieved by setting $src_j = rc_j$ and if it is required to use E-SIRAP instead then $src_j = k$, so using self blocking ceiling generalizes this version of SIRAP to include original SIRAP, first E-SIRAP and in between.

Eq. (11.11) and (11.12) should be changed to include the self blocking ceiling which has a great effect on them. The interference from lower priority tasks on $\tau_i$ depends on $src_j$. During self-blocking of a lower priority task $\tau_f$ that tried to access $R_j$, task $\tau_i$ is allowed to execute if $src_j < i$. Hence, $\tau_i$ will not be blocked during the self blocking of $\tau_f$ on $R_j$ when $src_j < i$. The interference from lower priority tasks can be calculated as follows;

$$I_L(i) = \max_{\tau_f \in \mathtt{lp}(i)} (\max_{\forall R_j | rc_j \geq i} (A(i,j) \times X_{f,j} + c_{f,j})), \tag{11.14}$$

where

$$A(i,j) = \begin{cases} 0 & \text{if } src_j < i \\ 1 & \text{otherwise.} \end{cases} \tag{11.15}$$

$src_j$ should also be included in Eq. (11.12) as shown below;

$$Q_s \geq X_{i,j} + \sum_{k \in \{max(i,src_j)+1,\ldots,rc_j\}} C_k. \tag{11.16}$$

## 11.7 Selection algorithm

In this section, we will present an algorithm that finds the best setting of the self blocking ceilings that minimize the subsystem budget $Q_s$. The algorithm searches for the best values for $\{src_j\}, \forall R_j \in \mathcal{R}_s$ through iteration steps (see Figure 11.2). The algorithm is explained as follows;

**Input and output** $S_s$, $\mathcal{R}_s$ and $\{rc_j\}, \forall R_j \in \mathcal{R}_s$ are the inputs to the algorithm, and the outputs from the algorithm are $Q_s$ and $\{src_j\}$.

**Initialization** In the beginning, the algorithm sets the self blocking ceiling of resources equal to the resource ceiling (line 1 in Figure (11.2)) which is equivalent to SIRAP. In this case, the interference from lower priority tasks will be the highest and is counted using Eq (11.10).

**Iteration step** In line 4, the algorithm calculates the subsystem budget $Q_s$ and it checks the condition in Eq. (11.16) to guarantee the correctness of SIRAP/ E-SIRAP (lines $5-6$). In line 14 it finds the task $\tau_h$ that requires the highest CPU resource, that the value of $Q_s$ was selected according to the CPU resource demand of this task. Then the algorithm finds the resource $R_b$ that cause the maximum blocking on task $\tau_h$ (line 15). Finally, it sets the $src_b$ to be less than the priority of $\tau_h$ in line 19 ($src_b = h - 1$). The interference from lower priority tasks that access $R_b$, on task $\tau_h$ will be lower and will be computed according to Eq. (11.14) (with $A(h, b) = 0$) which decreases $\texttt{rbf}_{\texttt{FP}}(h, t)$ and can decrease the subsystem budget $Q_s$. Finally, the algorithm computes the subsystem budget after the changes of the self blocking ceiling and repeat the operation.

**Iteration termination** The algorithm terminates if one of the following conditions becomes true:

1. The self blocking ceiling of the resource $R_b$ is lower than the priority of the task $\tau_h$ (line 16). In this case, lowering the self blocking ceiling

will not decrease $Q_s$ because the maximum blocking on $\tau_h$ can not be decreased, i.e., maximum does not contain the term $X_{h,b}$ in Eq. (11.14).

2. If there is not a resource that block the task $\tau_h$, i.e., $I_L(h) = 0$ in Eq. (11.14) (for example the lowest priority task).

3. If the current budget is greater than the one that is evaluated in the previous iteration (lines $8, 12$). Note that the budget may increase only due to the condition in Eq. (11.16). The reason is that in each iteration the self blocking ceiling of $R_b$ is decreased which can decrease the required $Q_s$ that schedule $\tau_h$. On the other hand, decreasing $src_b$ will increase the right hand side of Eq. (11.16) which may require higher $Q_s$, and continuing to decrease $src_b$ will increase $Q_s$ even more.

**Complexity and runtime overhead** During an i-th iteration, the algorithm only decreases the self blocking ceiling of $R_b$. Then, it can repeat at most O($n_s \times m_s$) iterations for a subsystem that its lowest priority task accesses all shared resources and the algorithm decreased the self blocking ceiling of the shared resources to the priority of that task.

During runtime, the improved E-SIRAP adds some runtime overhead compared with the original SIRAP since it uses a max function in Eq. (11.13) when assigning the value of the subsystem ceiling when entering self blocking state. In addition, it requires more memory to save the self blocking ceiling of shared resources compared with SIRAP as explained in section 11.6.1.

**Improvement compared to SIRAP** The resulting $Q_s$ when using this algorithm is always less than or equal to the subsystem budget when using the original SIRAP. The algorithm initializes the self blocking ceiling according to SIRAP (i.e., $src_j = rc_j$) and it will continue iterating as long as there is a possibility to decrease $Q_s$. It stops if the value of $Q_s$ starts to increase or the algorithm can not decrease it anymore. Then we can conclude that the algorithm will give same or better results compared with SIRAP.

**Algorithm's functions**

- The findTaskMaxQ function returns the index of one task. In case there are more than one task that require at least $Q_s$ then the algorithm will handle one subsystem at each iteration and the order of handling them does not affect the results of the algorithm. The same holds for the function findResourceMaxB where it returns the index of one resource and

it might happen that more than one shared resource cause the maximum blocking.

- We will explain the function "findTaskMaxQ"; for each task $\tau_i$, lets define $\mathsf{slack}_i$ as the maximum positive difference between the supply bound function and the request bound function of $\tau_i$,

  $\mathsf{slack}_i = \max_{t\in[0,D_i]}(\mathtt{sbf}_s(t) - \mathtt{rbf}_{\mathsf{FP}}(i,t))$ where $t$ can be selected within a finite set of points [21].

  Then the function $\mathsf{findTaskMaxQ} = i$ such that $\mathsf{slack}_i = \min_{\tau_k\in\ \mathcal{T}_s}(\mathsf{slack}_k)$.

**Example** We will explain the operation of the algorithm using the example in Table 11.3.

1. First, the algorithm initializes the values of self blocking such that $src_1 = src_2 = src_3 = 4$, then it finds the minimum budget required to guarantee the schedulability $Q_s = 23$. At line 14, it finds the task that requires maximum CPU resources which is $\tau_4$. It tries to decrease the $\mathtt{rbf}_{\mathsf{FP}}(4,t)$ by decreasing the interference from lower priority tasks, looking at Eq. (11.14) at this step, $A(4,1) = A(4,2) = A(4,3) = 1$, i.e., the maximum blocking from each shared resource. At line 15 the algorithm finds $R_3$ as the shared resource that imposes the maximum blocking on $\tau_4$ so it decreases the self blocking ceiling of $R_3$ such that $src_3 = 3$ which makes $A(4,3) = 0$.

2. The algorithm calculates the new budget $Q_s = 21$ and check the condition in Eq. (11.16) which is $Q_s \geq 18.7$ so $\mathsf{finalQ_s} = 21$. It finds the task that requires maximum CPU resources, and it is task $\tau_4$. After that its finds the resource that imposes maximum blocking which is $R_2$. Then it sets $src_2 = 3$ and by this $A(4,2) = 0$ in Eq. (11.14).

3. The new subsystem budget will be $Q_s = 19$ and the condition in Eq. (11.16) then $Q_s \geq 18.7$, $\mathsf{finalQ_s} = 19$. After that the algorithm finds the task that requires maximum CPU, and it is still task $\tau_4$, and it finds that $R_3$ is imposing the maximum blocking. But $src_3 < 4$ (at line 16) the blocking from this resource is the minimum and can not be minimized more (case 1 in the iteration termination). So the algorithm stops and returns the subsystem budget $\mathsf{finalQ_s} = 19$ and the self blocking ceilings of the shared resources $src_1 = 4, src_2 = 3, src_3 = 3$.

- calculateBudget($S_s, P_s, RC_s, SRC_s$) returns the smallest subsystem budget that satisfy (11.1).
- findTaskMaxQ($S_s, P_s, Q_s, RC_s$) returns the task index of the task that requires at least $Q_s$ to be scheduled.
- findResourceMaxB($S_s, P_s, RC_s, h$) returns the resource number that imposes the maximum blocking on $\tau_h$
- SIRAPCondition($S_s, SRC_s$) returns the value of budget that satisfy Eq. (11.16).
- Pri($\tau_h$) returns the priority of the task $\tau_h$.

1: $RC_s$ = finalSRC$_s$ = $SRC_s = \{rc_1, \cdots, rc_{m_s}\}$
2: finalQ$_s$ = calculateBudget($S_s, P_s, RC_s, SRC_s$)
3: `do`
4: $Q_s$ = calculateBudget($S_s, P_s, RC_s, SRC_s$)
5: `if` (SIRAPCondition($S_s, SRC_s$) $> Q_s$)
6: $Q_s$ = SIRAPCondition($S_s, SRC_s$)
7: `end if`
8: `if` ($Q_s \leq$ finalQ$_s$)
9: finalQ$_s$ = $Q_s$
10: finalSRC$_s$ = $SRC_s$
11: `else`
12: `return` finalQ$_s$, finalSRC$_s$
13: `end if`
14: $h$ = findTaskMaxQ($S_s, P_s, Q_s, RC_s$)
15: $b$ = findResourceMaxB($S_s, P_s, RC_s, h$)
16: `if` ($R_b \notin \mathcal{R}_s$ ) OR ($SRC_s[b] <$ Pri($\tau_h$))
17: `return` finalQ$_s$, finalSRC$_s$
18: `else`
19: $SRC_s[b]$ = Pri($\tau_h$) $- 1$
20: `end if`
21: `while` (true)

Figure 11.2: The selection algorithm.

## 11.8 Algorithm evaluation

In this section, we evaluate the performance of the presented algorithm, in terms of requiring less CPU resource than using original SIRAP.

Based on Eqs. (11.1), (11.2), (11.14) and (11.16), we can distinguish two parameters that have great effect on the performance of the algorithm:

- $X_{i,j}$ – since the algorithm decreases the interference of the lower priority tasks by $X_{i,j}$ compared with SIRAP (see Eq. (11.16)) then higher values of $X_{i,j}$ can decrease the subsystem budget more.
- The difference between $P_s$ and $T_{min}$ – the lower the difference is the better results the algorithm will give. The reason behind this is that if $P_s$ is much lower than $T_{min}$, then the subsystem budget using SIRAP will be lower and because of the condition in Eq. (11.16) the algorithm may not be able to decrease the subsystem budget.

We will explain the effect of the mentioned parameters by means of the simulation in the following section.

### 11.8.1 Simulation settings

The simulation is performed by applying the algorithm on 1000 different randomly generated subsystems where each subsystem consists of 5 tasks, and then we have increased the number of tasks to 8 tasks to investigate the effect of changing the number of task on the algorithm performance. The internal resource ceilings of the globally shared resources are assumed to be equal to the highest task priority in each subsystem (i.e., $rc_j = n_s$) and we assume that $T_i = D_i$ for all tasks. For the subsystems that contain 8 tasks, 2-6 tasks access globally shared resources and 1-4 tasks access global shared resources for the subsystems that contains 5 tasks. The worst-case critical section execution time of a task $\tau_i$ is set to a value between $0.3C_i$ and $0.8C_i$. A task is assumed to access at most one globally shared resource. For each simulation study the following settings are changed and a new 1000 subsystems is generated except when changing the subsystem period where the same subsystems are used:

1. Number of tasks – the number of tasks in subsystems.
2. Task set utilization $U^{\mathcal{T}_s}$ – the task set utilization is the summation of the utilization of all tasks in the subsystem, is specified to a desired value.

3. The subsystem period – the subsystem period is specified to a desired value.

The task set utilization is divided randomly among the tasks that belong to that subsystem. Task periods are selected within the range of 200 to 1000. Since the task period is generated to a value within the interval as specified, the execution time is derived from the desired task utilization. All randomized subsystem parameters are generated following uniform distributions.

### 11.8.2 Simulation results

Tables 11.4-11.6 show the results of 3 different simulation studies performed to measure the performance of the algorithm. The tables present four main measures. Firstly, the percentage of subsystems for which the subsystem budget decreased when the algorithm was applied is presented in the row labeled by "Improve". Secondly, the absolute improvement, i.e. decrement, $Q_s^{Dec}$ of the subsystem budget is computed, which is defined as $Q_s^{Dec} = Q_s^{SIRAP} - Q_s^{alg}$, where $Q_s^{SIRAP}$ and $Q_s^{alg}$ are the subsystem budget using SIRAP and using the selection algorithm, respectively. The tables present both the average decrement and the maximum decrement in rows labeled by "Avg. $Q_s^{Dec}$" and "Max. $Q_s^{Dec}$", respectively. Thirdly, the relative improvement of the subsystem utilization $U_s^{Imp}$ is computed, which is defined as

$$U_s^{Imp} = (U_s^{SIRAP} - U_s^{alg})/U_s^{SIRAP}, \tag{11.17}$$

where $U_s^{SIRAP} = Q_s^{SIRAP}/P_s$ and $U_s^{alg} = Q_s^{alg}/P_s$ denote the subsystem utilization using SIRAP and using the selection algorithm, respectively. Similar to the improvement of the subsystem budget, the tables present both the average decrement ("Avg. $U_s^{Imp}$") and maximum decrement ("Max. $U_s^{Imp}$") of the subsystem utilization. Finally, the maximum number of iterations that the algorithm needed to find the lowest subsystem budget is determined and presented in the row labeled by "Max. iterations".

- **Study 1** is specified having task utilizations $U^{\mathcal{T}_s}$ of 5%, 10% and 20%, number of tasks $n_s$ equals to 5, task periods between 200 and 1000, and subsystem period $P_s$ is 100.
- **Study 2** changes the subsystem period $P_s$ (compared to Study 1) to 75, 70 and 65 and keeps $U^{\mathcal{T}_s} = 5\%$. As mentioned previously we use the same 1000 subsystems in Study 1 that have $U^{\mathcal{T}_s} = 5\%$ and only change the subsystem period.

- **Study 3** increase the number of tasks (compared to Study 1) to 8 tasks.

Looking at the results in Table 11.4, it is clear that for some subsystems the algorithm can decrease the required budget significantly (a maximum decrease "Max. $Q_s^{Dec}$ of 17,4 and maximum relative subsystem utilization improvement $Max.U_s^{Imp}$ of $35\%$). It also shows that increasing $U^{\mathcal{T}_s}$ decreases the number of subsystems for which the algorithm can improve their budgets compared with SIRAP. The reason is that increasing $U^{\mathcal{T}_s}$ will increase $C_i$ of the tasks and will increase the required budget that satisfy Eq. (11.16). However, it will also increase $c_{i,j}$ which is clear from observing the "Avg." and "Max." rows in the table.

Looking at Table 11.5, it is clear that when the subsystem period is decreased, the number of the subsystems that the algorithm can improve will be decreased. However, the decrement in the subsystem budget will be more significant for the subsystem utilization when the subsystem period is lower since $U_s = Q_s/P_s$ (see the "Ave." and "Max." rows in Table 11.5 for the case $P_s = 75$). When $P_s = 65 \leq T_{min}/3$ then the algorithm can not improve any subsystem as explained in the beginning of this section. This can be seen as a limitation of the algorithm and it could be better to decrease the subsystem period instead of using the algorithm to decrease the subsystem utilization $U_s = Q_s/P_s$. However, this is not always true, first, from the simulation results we have compared the subsystem utilization $U_s$ when $P_s = 100$ and $P_s = 65$ and we have found that when $P_s = 100$, 97 out of 410 subsystems that the algorithm improved, require less or equal subsystem $U_s$ than when using only SIRAP with $P_s = 65$. The reason is that when the algorithm is able to improve the subsystem budget, then the request bound function of the task that needs maximum CPU resources will be lower than the case when using original SIRAP with lower subsystem period, and this affects Eq. (11.1). The second issue of reducing the subsystem budget is that it increases the context switch overhead because the subsystem budget will be lower. Finally, as showed in [7] decreasing the subsystem period may increase $U_s$ to satisfy the condition $Q_s \geq X_s$ of SIRAP.

In **Study 3** we have increased the number of tasks to 8 tasks in each subsystem, and the results in Table 11.6 shows that increasing the number of tasks does not change the effect of $U^{\mathcal{T}_s}$. However, increasing the number of tasks will increase the number of subsystems that the algorithm can improve (compare the "Improve" row in Table 11.4 and Table 11.6). The reason is that there are more task and more shared resources in each subsystem so the algorithm can improve the subsystem more before it stops. The maximum number of

iterations for 8 tasks is 6 iterations while for 5 tasks it is 4. In addition, increasing the number of tasks and keeping $U^{\mathcal{T}_s}$ the same, decreases the utilization of each task which improves the algorithm performance in terms of the number of the subsystems that the algorithm can improve as explained previously.

| | $U^{\mathcal{T}_s}$=5% | $U^{\mathcal{T}_s}$=10% | $U^{\mathcal{T}_s}$=20% |
|---|---|---|---|
| Improve | $41\%$ | $34,1\%$ | $26\%$ |
| Avg. $Q_s^{Dec}$ | $2$ | $3,70$ | $5,3$ |
| Avg. $U_s^{Imp}$ | $16,4\%$ | $15,3\%$ | $12,4\%$ |
| Max. $Q_s^{Dec}$ | $5$ | $10$ | $17,4$ |
| Max. $U_s^{Imp}$ | $36,6\%$ | $37,2\%$ | $35\%$ |
| Max. iterations | $4$ | $4$ | $4$ |

Table 11.4: Measured results of Study 1.

| | $P_s$=75 | $P_s$=70 | $P_s$=65 |
|---|---|---|---|
| Improve | $19,5\%$ | $5,4\%$ | $0\%$ |
| Avg. $Q_s^{Dec}$ | $1,9$ | $0,55$ | $0$ |
| Avg. $U_s^{Imp}$ | $20\%$ | $8\%$ | $0\%$ |
| Max. $Q_s^{Dec}$ | $5$ | $1,2$ | $0$ |
| Max. $U_s^{Imp}$ | $41,9\%$ | $16,6\%$ | $0\%$ |
| Max. iterations | $4$ | $4$ | $0$ |

Table 11.5: Measured results of Study 2.

| | $U^{\mathcal{T}_s}$=5% | $U^{\mathcal{T}_s}$=10% | $U^{\mathcal{T}_s}$=20% |
|---|---|---|---|
| Improve | $47,6\%$ | $43,4\%$ | $32,3\%$ |
| Avg. $Q_s^{Dec}$ | $2$ | $3,3$ | $5$ |
| Avg. $U_s^{Imp}$ | $14,3\%$ | $13,6\%$ | $11\%$ |
| Max. $Q_s^{Dec}$ | $5$ | $10$ | $14,6$ |
| Max. $U_s^{Imp}$ | $34,5\%$ | $35,4\%$ | $33,5\%$ |
| Max. iterations | $6$ | $6$ | $6$ |

Table 11.6: Measured results of Study 3.

## 11.9 Summary

In this paper, we presented an improved schedulability analysis for the synchronization protocol SIRAP. The improved analysis may decrease the minimum subsystem budget while still guaranteeing the schedulability of all tasks in a subsystem. We also presented a generalization of SIRAP, which distinguishes separate resource ceilings for self blocking and for actual resource access, with the aim to reduce the required CPU resource for each subsystem by reducing the interference from lower priority tasks. Because the efficiency of the protocol depends on both the setting of the resource ceilings and the subsystem parameters, we presented an algorithm that finds the best settings for resource ceilings during the self blocking for each shared resource in order to minimize the required subsystem budget. The simulation results shows that the algorithm can significantly reduce the CPU resource needs of a subsystem, but that the effectiveness of the algorithm heavily depends on the tasks parameters and the subsystem period.

Our future work includes further improvements of SIRAP in two directions: I) Applying runtime mechanisms to decrease the value of $X_{i,j}$ that is used to check if there is enough remaining budget before accessing a shared resource, based on the arrival time of the higher priority tasks, in order to improve the average response time of tasks. II) Investigating the case of allowing lower priority tasks to execute during the self blocking in order to reduce the interference from higher priority tasks. For this improvement, a runtime mechanism may be required to decide the correct execution order of tasks during the next activation subsystem period. III) Finally, showing the advantages of the proposed algorithm on real industrial systems.

# Bibliography


[1] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[2] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4$^{th}$ ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[3] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[4] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[5] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[6] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[7] 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 7$^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[8] M. Behnam, T. Nolte, M. Åsberg, and I. Shin. Synchronization protocols for hierarchical real-time scheduling frameworks. In *Proceedings of the $1^{st}$ Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08)*, pages 53–60, November 2008.

[9] M. Behnam, T. Nolte, M. Åsberg, and R.J. Bril. Overrun and skipping in hierarchical scheduled real-time systems. In *Proceedings of the $15^{th}$ IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, pages 519–526, August 2009.

[10] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the $26^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[11] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[12] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the $23^{th}$ IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[13] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the $6^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[14] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the $15^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[15] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the $26^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 99–110, December 2005.

[16] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarhical fixed-priority scheduling. In *Proceedings of the $14^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, June 2002.

[17] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments.

In *Proceedings of the* $21^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'00)*, pages 217–226, December 2000.

[18] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the* $7^{th}$ *IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[19] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the* $28^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, December 2007.

[20] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *IEEE Transaction on Embedded Computing Systems*, 7(3):1–39, 2008.

[21] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.

[22] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the* $15^{th}$ *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[23] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Proceedings of the* $29^{th}$ *IEEE Real-Time Systems Symposium (RTSS'08)*, pages 209–22, December 2008.

# Chapter 12

# Paper G: Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources

Insik Shin, Moris Behnam, Thomas Nolte, Mikael Nolin

In Proceedings of the $29^{th}$ IEEE International Real-Time Systems Symposium (RTSS'08), pages 209-220, December, 2008.

**Abstract**

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 recourses (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 scheduability.

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 also suitable for adaptable and reconfigurable systems.

## 12.1 Introduction

Hierarchical scheduling has emerged as a promising vehicle for simplifying the development of complex real-time software systems. Hierarchical scheduling frameworks (HSFs) provide an effective mechanism for achieving temporal partitioning, making it easier to enforce the principle of separation of concerns in the design and analysis of real-time systems. HSFs allow hierarchical CPU sharing among subsystems (applications). The whole CPU is available and shared among subsystems. Subsequently, each subsystem's allocated CPU-share is divided among its internal tasks by the usage of an internal scheduler.

Substantial studies [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] have been introduced for the schedulability analysis of HSFs, where subsystems are independent. For dependent subsystems, synchronization protocols [14, 15, 16] have been proposed for arbitrating accesses to logical resources (i.e., semaphore) across subsystems in HSFs. There have been a few studies [11, 5] on the *system load minimization* problem, which finds the minimum collective CPU requirement (i.e., system load) necessary to guarantee the schedulability of an entire HSF. However, this problem has not been addressed taking into account global (logical) resource sharing (across subsystems).

The difficulty of finding the minimum system load substantially grows with the presence of global sharing of logical resources, in comparison to without it. Without it, it is a straightforward bottom-up process; individual subsystems develop their *timing-interfaces* [11, 17], describing their minimum CPU requirements needed to ensure schedulability, and individual subsystem interfaces can easily be combined to determine the minimum system load that guarantees the schedulability of an entire HSF. However, global resource sharing produces interference among subsystems, complicating the process of finding subsystem interfaces that impose the minimum CPU requirements into the system load.

An inherent feature with global resource sharing is that 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 locking time, for example, using the mechanism presented in [18], in order to potentially reduce the blocking of other subsystems towards decrease of the system load. However, in doing so, we present in this paper an unexpected consequence of reducing resource locking time; it can increase the CPU demands of the subsystem itself (locking the resource), subsequently increasing the system load. Hence, this paper introduces a potentially contradicting effect of reducing resource locking time on the system

load, and it entails methods that can effectively explore such a tradeoff.

In this paper, we consider a two-step approach towards the system load minimization problem. In the first step, each subsystem generates its own interface candidates in isolation, investigating the intra-subsystem aspect of the tradeoff. In the second step, putting all subsystems together on system-level, interfaces of all subsystems are selected from their own candidates to find the minimum resulting system load, examining the inter-subsystem aspect of the tradeoff. For the first step, we present an algorithm that derives a bounded number of interface candidates for each subsystem such that it is guaranteed to carry an interface candidate that constitutes the minimum system load no matter which other subsystems it will be later integrated with. The first step allows the interface candidates of subsystems to be developed independently, making it also suitable for open environments [3], requiring no knowledge of other subsystems. For the second step, we present another algorithm that determines optimal interface selection to find the minimum system load. The complexity of both algorithms is very low ($O(n)$), making the approach good for execution during run-time, e.g., suitable for adaptable and reconfigurable systems.

In the remainder of the paper, Section 12.2 presents related work, followed by system model and background in Section 12.3. Section 12.4 presents schedulability analysis in our HSF, followed by problem formulation and solution outline in Section 12.5. Section 12.6 addresses the first step of the two-step approach; efficiently generating interface candidates, and Section 12.7 resolves the second step finding an optimal solution out of the candidates. Finally, Section 12.8 concludes.

## 12.2 Related work

This section presents related work in the areas of HSFs as well as synchronization protocols.

**Hierarchical scheduling.** The HSF for real-time systems, originating in open systems [3] in the late 1990's, has been receiving an increasing research attention. Since Deng and Liu [3] introduced a two-level HSF, its schedulability has been analyzed under fixed-priority global scheduling [7] and under Earliest Deadline First (EDF) based global scheduling [8]. Mok *et al.* [10] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF, and schedulability analysis techniques [6, 12] have been introduced for this resource model. In addition, Shin and Lee [11, 17] intro-

duced 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 [1, 9, 2] and under EDF scheduling [11, 13]. More recently, Easwaran *et al.* [4] introduced Explicit Deadline Periodic (EDP) virtual processor model. However, a common assumption shared by all above studies is that tasks are independent.

**Synchronization.** Many synchronization protocols have been introduced for arbitrating accesses to shared logical resources addressing the priority inversion problem, including Priority Inheritance Protocol (PIP) [19], Priority Ceiling Protocol (PCP) [20], and Stack Resource Policy (SRP) [21]. There have been studies on supporting resource sharing within subsystems [1, 7] in HSFs. For supporting global resource sharing across subsystems, two protocols have been proposed for periodic virtual processor model (or periodic server) based HSFs on the basis of an overrun mechanism [15] and skipping [14], and another protocol [16] for bounded-delay virtual processor model based HSFs. Bertogna *et al.* [18] addressed the problem of minimizing the resource holding time under SRP. In summary, compared to the work in this paper, none of the above approaches have addressed the tradeoff between how long subsystems can lock shared resources and the resulting CPU requirement required in guaranteeing schedulability.

## 12.3 System model and background

A Hierarchical Scheduling Framework (HSF) is introduced to support CPU resource sharing among applications (subsystems) under different scheduling services. In this paper, we are considering a two-level HSF, where the system-level global scheduler allocates CPU resources to subsystems, and the subsystem-level local schedulers subsequently schedule CPU resources to their internal tasks. This framework also allows logical resource sharing between tasks in a mutually exclusive manner.

### 12.3.1 Virtual processor models

The notion of real-time virtual processor model was first introduced by Mok *et al.* [10] to characterize the CPU allocations that a parent node provides to a child node in a HSF. The *CPU supply* refers to the amounts of CPU allocations that a virtual processor can provide. Shin and Lee [11] proposed the periodic processor model $\Gamma(P, Q)$ to specify periodic CPU allocations, where $P$ is a

period ($P > 0$) and $Q$ is a periodic allocation time ($0 < Q \leq P$). The supply bound function $\texttt{sbf}_\Gamma(t)$ of $\Gamma(P, Q)$ was given in [11] that computes the minimum possible CPU supply for every interval length $t$ as follows:

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

where $k = \max\Big(\big\lceil \big(t - (P-Q)\big)/P \big\rceil, 1\Big)$.

### 12.3.2 System model

We consider a deadline-constrained sporadic 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, $D_i$ is a relative deadline ($C_i \leq D_i \leq T_i$), 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 $R_j$. We assume that all tasks, that belong to same subsystem, are assigned unique static priorities and are sorted according to their priorities in the order of increasing priority. Without loss of generality, we assume that the priority of a task is equal to the task ID number after sorting, and the greater a task ID number is, the higher its priority is. Let $\texttt{HP}(i)$ returns the set of tasks with higher priorities than that of $\tau_i$.

A subsystem $S_s \in \mathcal{S}$, where $\mathcal{S}$ is the set representing the whole system of subsystems, is characterized by $\langle \mathcal{T}_s, \mathcal{RC}_s \rangle$, where $\mathcal{T}_s$ is a task set and $\mathcal{RC}_s$ is a set of internal resource ceilings of the global shared logical resources. We will explain the resource ceilings in Section 12.3.3. We assume that each subsystem has a unique static priority and subsystems are sorted in an increasing order of priority, as is the case with tasks. We also assume that each subsystem $S_s$ has a local Fixed-Priority Scheduler (FPS) and the system has a global FPS. Let $\texttt{HPS}(s)$ returns the set of subsystems with higher priority than that of $S_s$.

Let us define a *timing-interface* of a subsystem $S_s$ such that it specifies the collective real-time requirements of $S_s$. The subsystem interface is defined as $(P_s, Q_s, X_s)$, where $P_s$ is a period, $Q_s$ is a *budget* that represents an execution time requirement, and $X_s$ is a *maximum critical section execution time* of all global logical resources accessed by $S_s$. We note that $X_s$ is similar to the concept of *resource holding time (RHT)* in [18], however, developed for a different virtual-processor model. RHT in [18] is developed for a dedicated

processor model[1] (or a fractional processor model [10]), where subsystems do not preempt each other. However, our HSF is based on a time-shared (partitioned) processor model [11], where subsystem-level preemptions can take place. Therefore, $X_s$ does not represent RHT in our HSF[2], but indicates the worst-case execution time requirement that $S_s$ demands inside a critical section. We will explain later how to derive the values of $P_s, Q_s$ and $X_s$ for a given subsystem $S_s$.

### 12.3.3 Stack resource policy (SRP)

In this paper, we consider the SRP protocol [21] for arbitrating accesses to shared logical resources. Considering that the protocol was developed without taking hierarchical scheduling into account, we generalize its terminologies for hierarchical scheduling.

• **Resource ceiling**. Each global shared resource $R_j$ is associated with two types of resource ceilings; an *internal* resource ceiling ($rc_j$) for local scheduling and an *external* resource ceiling ($RX_s$) for global scheduling. They are defined as $rc_j = \max\{i|\tau_i \in \mathcal{T}_s \text{ accesses } R_j\}$ and $RX_s = \max\{s|S_s \text{ accesses } R_j\}$.

• **System/subsystem ceiling**. The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling of a currently locked resource in the system/subsystem.

Under SRP, a task $\tau_k$ can preempt the currently executing task $\tau_i$ (even inside a critical section) within the same subsystem, only if the priority of $\tau_k$ is greater than its corresponding subsystem ceiling. The same reasoning can be made for subsystems from a global scheduling point of view.

Given a subsystem $S_s$, let us consider how to derive the value of its critical section execution time ($X_s$). Basically, $X_s$ represents a worst-case CPU demand that internal tasks of $S_s$ may collectively request inside any critical section. Note that any task $\tau_i$ accessing a resource $R_j$ can be preempted by tasks with priority higher than the internal ceiling of $R_j$. From the viewpoint of $S_s$, let $w_j$ denote the maximum collective CPU demand necessary to complete an access of any internal task to $R_j$. Then, $w_j$ can be computed through iterative process as follows (similarly to [18]):

[1] A processor is said to be *dedicated* to a subsystem, if the subsystem exclusively utilizes the processor with no other subsystems.

[2] As the computation of RHT is not main focus of this paper, we refer to our technical report [22] for its computation in our HSF.

$$w_j^{(m+1)} = cx_j + \sum_{k=rc_j+1}^{n} \lceil \frac{w_j^{(m)}}{T_k} \rceil \cdot C_k, \tag{12.1}$$

where $cx_j = \max\{c_{i,j}\}$ for all tasks $\tau_i$ accessing resource $R_j$ and $n$ is the number of tasks within the subsystem. The recurrence relation given by Eq. (12.1) starts with $w_j^{(0)} = cx_j$ and ends when $w_j^{(m+1)} = w_j^{(m)}$ or when $w_j^{(m+1)} > D_i^*$, where $D_i^*$ is the smallest deadline of tasks $\tau_i$ accessing $R_j$. If $w_j^{(m+1)} > D_i^*$, no task $\tau_i$ is guaranteed to be schedulable, and subsequently neither is its subsystem $S_s$.

Then, $X_s = \max\{w_j |$ for all $R_j \in \mathcal{R}_s\}$, where $\mathcal{R}_s$ is a set of global shared resources accessed by $S_s$.

## 12.4 Resource sharing in the HSF

### 12.4.1 Overrun mechanism

This section explains overrun mechanisms that can be used to handle budget expiry during a critical section in a HSF. Consider a global scheduler that schedules subsystems according to their periodic interfaces $(P_s, Q_s, X_s)$. The subsystem budget $Q_s$ is said to *expire* at the point when one or more internal (to the subsystem) tasks have executed a total of $Q_s$ time units within the subsystem period $P_s$. 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 $Q_s$.

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

In this paper, we consider a mechanism based on overrun [15] that works as follows; when the budget of the subsystem $S_s$ expires and $S_s$ has a task $\tau_i$ that is still locking a global shared resource, the task $\tau_i$ continues its execution until it releases the locked resource. The extra time that $\tau_i$ needs to execute after the budget of $S_s$ expires is denoted as *overrun time* $\theta_s$. The maximum

$\theta_s$ occurs when $\tau_i$ locks a resource such that $S_s$ requests a maximum critical section execution time ($X_s$) just before its budget ($Q_s$) expires.

### 12.4.2 Schedulability analysis

In this paper, we use HSRP [15] for resource synchronization in HSF. Schedulability analysis under global and local FPS with the overrun mechanism is presented in [15]. However, the presented approach is not suitable for open environments because the schedulability analysis of an internal task within a subsystem requires information of all the other subsystems. Hence, this section presents the schedulability analysis of local and global FPS using subsystem interfaces, which is suitable for open environments.

**Local schedulability analysis.** Let $\texttt{rbf}_{\texttt{FP}}(i,t)$ denote the request bound function of a task $\tau_i$ under FPS [23], i.e.,

$$\texttt{rbf}_{\texttt{FP}}(i,t) = C_i + \sum_{\tau_k \in \texttt{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{12.2}$$

The local schedulability analysis under FPS can be then easily extended from the results of [21, 11] as follows:

$$\forall \tau_i, 0 < \exists t \leq D_i \;\; \texttt{rbf}_{\texttt{FP}}(i,t) + b_i \leq \texttt{sbf}(t), \tag{12.3}$$

where $b_i$ is the maximum *blocking* (i.e., extra CPU demand) imposed to a task $\tau_i$ when $\tau_i$ is blocked by lower priority tasks that are accessing resources with ceiling greater than or equal to the priority of $\tau_i$, and $\texttt{sbf}(t)$ is the supply bound function. Note that $t$ can be selected within a finite set of scheduling points [24].

**Subsystem interface.** We now explain how to derive the budget $Q_s$ of the subsystem interface. Given $S_s$, $\mathcal{RC}_s$, and $P_s$, let $\mathsf{calculateBudget}(S_s, P_s, \mathcal{RC}_s)$ denote a function that calculates the smallest subsystem budget that satisfies Eq. (12.3) depending on the local scheduler of $S_s$. Such a function is similar to the one in [11]. Then, $Q_s = \mathsf{calculateBudget}(S_s, P_s, \mathcal{RC}_s)$.

**Global schedulability analysis.** Under global FPS scheduling, we present the subsystem load bound function as follows (on the basis of a similar reasoning of Eq. (12.2)):

$$\texttt{LBF}_s(t) = \texttt{RBF}_s(t) + B_s \text{ , where} \tag{12.4}$$

$$\texttt{RBF}_s(t) = (Q_s + O_s(t)) + \sum_{S_k \in \texttt{HPS}(s)} \left\lceil \frac{t}{P_k} \right\rceil (Q_k + O_k(t)), \tag{12.5}$$

where $O_k(t) = X_k$ and $O_s(t) = X_s$ for $t \geq 0$. Let $B_s$ denote the maximum blocking (i.e., extra CPU demand) imposed to a subsystem $S_s$, when it is blocked by lower-priority subsystems,

$$B_s = \max\{X_j |\ S_j \in \texttt{LPS}(S_s)\}, \tag{12.6}$$

where $\texttt{LPS}(S_s) = \{S_j | j < s\}$.

A global schedulability condition under FPS is then

$$\forall S_s, 0 < \exists t \leq P_s\ \texttt{LBF}_s(t) \leq \texttt{t} \tag{12.7}$$

**System load.** As a quantitative measure to represent the minimum amount of processor allocations necessary to guarantee the schedulability of a subsystem $S_s$, let us define *processor request bound* ($\alpha_s$) as

$$\alpha_s = \min_{0<t\leq P_s} \{\frac{\texttt{LBF}_s(t)}{t} \mid \texttt{LBF}_s(t) \leq t\}. \tag{12.8}$$

In addition, let us define the *system load* $\mathsf{load}_{\mathsf{sys}}$ of the system under global FPS as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\}. \tag{12.9}$$

Note that $\alpha_s$ is the smallest fraction of the CPU resources that is required to schedule a subsystem $S_s$ (satisfying Eq. (12.7)) assuming that the global resource supply function is $\alpha t$. For example, consider a system $\mathcal{S}$ that consists of two subsystems; $S_1$ that has interface $(10, 1, 0.5)$ and $S_2$ $(48, 1, 1)$. To guarantee the schedulability of $S_1$ and $S_2$ then $\alpha_1 = 0.25$ and $\alpha_2 = 0.198$. Then $\mathsf{load}_{\mathsf{sys}} = \alpha_1 = 0.25$, which can schedule both $S_1$ and $S_2$.

## 12.5 Problem formulation and solution outline

In this paper, we aim at maintaining the system load as low as possible while satisfying the real-time requirements of all subsystems in the presence of global resource sharing. To achieve this, we address the problem of developing the interfaces $(P_s, Q_s, X_s)$ of all subsystems $S_s$. In particular, assuming $P_s$ is given,

we focus on determing $Q_s$ and $X_s$ such that a resulting system load ($\mathsf{load}_{\mathsf{sys}}$) is minimized subject to the schedulability of all subsystems. It is suggested from Eqs. (12.4) and (12.9) that $\mathsf{load}_{\mathsf{sys}}$ can be minimized by reducing $Q_s$ and $X_s$ for all subsystem $S_s$.

A recent study [18] introduced a method to reduce $X_i$. According to Eq. (12.1), the value of $X_s$ can decrease, when it has less interference (i.e., the summation part of Eq. (12.1)) from the tasks $\tau_k$ with priorities greater than the ceiling of a resource $R_j$ (i.e., $k > rc_j$). Such interference can be reduced by allowing fewer tasks to preempt inside the critical section of $R_j$. As proposed by [18], the ceiling of $R_j$ can be increased to its greatest possible value in order to allow no preemption inside the critical section. This way, $X_s$ can be minimized.

In this paper, we show that achieving the minimum $X_s$ of all subsystems $S_s$ does not simply produce the minimum system load, since minimizing $X_s$ may end up with a larger $Q_s$. To explain why this happens, let us assume that for a resource $R_j$, its ceiling $rc_j$ is $i-1$. In this case, a task $\tau_i$ can preempt any job that is executing inside the critical section of $R_j$. Now, suppose $rc_j$ is increased to $i$. Then, $\tau_i$ is no longer able to preempt any job that is accessing $R_j$, and it needs to be blocked. Then, the blocking ($b_i$) of $\tau_i$ can potentially increase, and, according to Eq. (12.3), this may require more CPU supply (i.e., $Q_s$). Figure 12.1 illustrates a tradeoff between decreasing $X_s$ and increasing $Q_s$ with an example subsystem $S_s$, where $S_s$ includes 7 internal tasks and accesses 3 global resources. In the figure, each point represents a possible pair of $(X_s, Q_s)$, and the line shows the tradeoff.

In addition to such a tradeoff, there is another factor that complicates the system load minimization problem further. It is not straightforward to determine $Q_s$ and $X_s$ of $S_s$ such that they contribute to $\mathsf{load}_{\mathsf{sys}}$ in a minimal way. According to Eq. (12.6), $X_s$ can serve as the blocking of its higher-priority subsystem $S_k$ depending on the value of $X_j$ of other lower-priority subsystems $S_j$. Hence, it is impossible to determine $X_s$ and $Q_s$ in an optimal way, without knowledge of other subsystems' interfaces.

We consider a two-step approach to the system load minimization problem. In the first step, each subsystem generates a set of interface candidates independently (with no information about other subsystems), which is suitable for subsystems to be developed in open environments. The second step is performed when subsystems are integrated to form a system. During this integration of subsystems, being aware of all interface candidates of all subsystems, only one out of all interface candidates for each subsystem is selected (that will be used by the system-level scheduler later on) such that a resulting

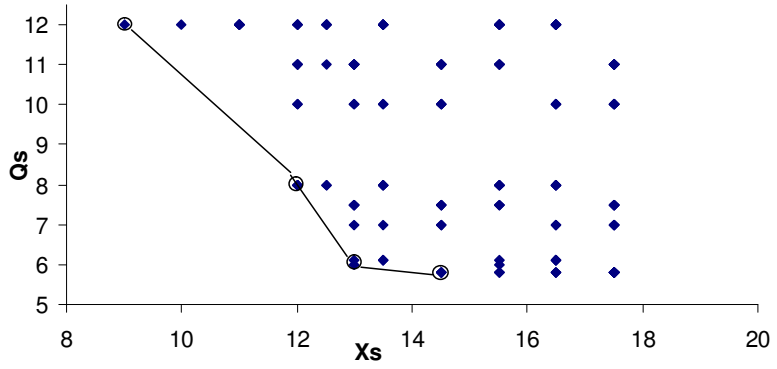


Figure 12.1: Tradeoff between $Q_s$ and $X_s$.

system load can be minimized.

## 12.6 Interface candidate generation

We define the *interface candidate generation* problem as follows. Given a subsystem $S_s$ and a set of global resources, the problem is to generate a set of interface candidates $IC_s$ such that there must exist an element of $IC_s$ that constitutes an optimal solution to the system load problem.

Suppose $S_s$ contains $n$ internal tasks that access $m$ global shared resources. Note that as explained in Section 12.5, each global resource may have up to $n$ different internal resource ceilings, and one interface candidate can be generated from each combination of $m$ resource ceilings. A brute-force solution to the interface generation problem is then to generate all possible $m^n$ interface candidates. However, not all of these $m^n$ candidates have the potential to constitute the optimal solution; those that require more CPU demand and impose greater blocking on other subsystems can be considered as replicate candidates.

Hence, we present the ICG (Interface Candidate Generation) algorithm that is not only computationally efficient, but also produces a bounded number of interface candidates. We first provide some notions and properties on which our algorithm is based. We then explain our algorithm and illustrate it. Hereinafter, we assume that $P_s$ is given by the system designer and is fixed during

the whole process of generating a set of interface candidates. Therefore an interface candidate can be denoted as $(Q_{s,j}, X_{s,j})$ where $j$ indicates interface candidate index.

**Definition 2.** *An interface candidate $(Q_{s,k}, X_{s,k})$ is said to be* redundant *if there exists $(Q_{s,i}, X_{s,i})$ such that $X_{s,i} \leq X_{s,k}$ and $Q_{s,i} \leq Q_{s,k}$, where $k < i$ (denoted as $(Q_{s,i}, X_{s,i}) \leq (Q_{s,k}, X_{s,k})$). In addition, $(Q_{s,i}, X_{s,i})$ is said to be* non-redundant *if it is not redundant.*

Suppose $(Q'_s, X'_s) \leq (Q^*_s, X^*_s)$. Then, the former candidate will never yield a larger $\mathtt{RBF}_s(t)$ than the latter does. This immediately follows from Eqs. (12.4) and (12.5). That is, a subsystem $S_s$ will never impose more CPU requirement to the system load with $(Q'_s, X'_s)$ than with $(Q^*_s, X^*_s)$. The following lemma records this property.

**Lemma 12.** *If $(Q'_s, X'_s) \leq (Q^*_s, X^*_s)$, $(Q'_s, X'_s)$ will never contribute more to $\mathsf{load}_{\mathsf{sys}}$ than $(Q^*_s, X^*_s)$ does.*

*Proof.* Suppose an interface candidate $(Q_{s,a}, X_{s,a})$ is redundant. By definition, there exists another candidate $(Q_{s,b}, X_{s,b})$ such that

- $X_{s,b} \leq X_{s,a}$ and $Q_{s,b} \leq Q_{s,a}$. So $(Q_{s,b} + X_{s,b}) <= (Q_{s,a} + X_{s,a})$. Using a redundant interface candidate will never decrease $\mathtt{RBF_s(t)}$ (see Eq. (12.5)) and the blocking $B_s$, respectively, compared to a non redundant candidate. It means that using a redundant candidate can increases $\mathtt{LBF_s(t)}$ and thereby $\mathsf{load}_s$ (see Eq. (12.8)). That is, a redundant candidate only has a potential to increase $\mathsf{load}_{\mathsf{sys}}$ (see Eq. (12.9)).
- both interfaces are equivalent then system load for both is the same.

□

Lemma 12 suggests that redundant candidates be excluded from a solution, and it reduces the number of interface candidates significantly. However, a brute-force approach to reduce redundant candidates is still computationally intractable, since the complexity of an exhaustive search is very high $O(m^n)$. We now present important properties that serve as the basis for the development of a computationally efficient algorithm.

In order to discuss some subtle properties in detail, let us further refine some of our notations with additional parameters. Firstly, the maximum blocking ($b_i$) imposed to a task $\tau_i$ can vary depending on which resource $\tau_i$ accesses. Hence, let $b_{i,j}$ denote the maximum blocking that a task with priority higher

than $i$ can experience in accessing a resource $R_j$, i.e., $b_{i,j} = \max\{c_{k,j}\}$ for all $\tau_k \leq \tau_i$. Secondly, the maximum CPU demand ($w_j$) imposed to any task accessing a resource $R_j$ can also be different depending on the internal ceiling ($rc_j$) of $R_j$. So let $w_{j,k}$ particularly represent $w_j$ when $rc_j = k$.

The following two lemmas show the properties of redundant interfaces, suggesting insights for how to effectively exclude them.

**Lemma 13.** *Let $\mathcal{R}^i$ denote a set of resources whose resource ceilings are $i$. Suppose a resource $R_k \in \mathcal{R}^i$ yields the greatest blocking among all the elements of $\mathcal{R}^i$. Then, it is the resource $R_k$ that requires the greatest CPU demand to complete any task's execution inside a critical section among all elements of $\mathcal{R}^i$, i.e.,*

$$\Big(b_{i,k} = \max_{\forall R_j \in \mathcal{R}^i}\{b_{i,j}\}\Big) \rightarrow \Big(w_{k,i} = \max_{\forall R_j \in \mathcal{R}^i}\{w_{j,i}\}\Big). \tag{12.10}$$

*Proof.* The $w_{j,i}$ depends on two parameters (see Eq. (12.1)); $cx_j$, which is equal to ($b_{i,j}$) since $rc_j = i$, and the interference from tasks with higher priority (the summation part denoted as $I$). Note that $I$ in invariant to difference resources $R_j \in \mathcal{R}^i$, since it considers only the tasks with priority greater than $i$ in the summation. Then, it is clear that $w_{j,i}$ depends only on $b_{i,j}$, and it follows that the resource with the maximum $b_{i,j}$, will be consequently associated with the maximum $w_{i,j}$. □

Using Lemma 13, the following lemma particularly shows how we can effectively exclude redundant candidates.

**Lemma 14.** *Consider a resource $R_y$ of a ceiling $k$ ($rc_y = k$) and another resource $R_z$ of a ceiling $i$ ($rc_z = i$), where $k < i$. Suppose $b_{k,y} < b_{k,z}$ and $rc_y < rc_z$. Then, an interface candidate generated by having the ceiling $rc_y = k + 1, .., i$ is redundant. Hence it is possible to increase the ceiling of $R_y$ to that of $R_z$ directly (i.e., $rc_y = rc_z = i$).*

*Proof.* Let $(Q', X')$ denote an interface candidate generated when $rc_y = k$ and $rc_z = i$, where $k < i$. Let $(Q^*, X^*)$ denote another interface candidate generated when $rc_y = rc_z = i$. We wish to show that $(Q^*, X^*) \leq (Q', X')$, i.e., $Q^* \leq Q'$ and $X^* \leq X'$.

Given $b_{i,y} < b_{i,z}$, it follows from Lemma 13 that $w_{y,i} < w_{z,i}$. This means that even though the ceiling of $R_y$ increases to $i$, it does not change the maximum blocking ($b_i$) of tasks $\tau_i$. Therefore, it does not change the request bound function either. As a result, $Q^* = Q'$.

We wish to show that $X^* \leq X'$. When the ceiling of $R_y$ increases to $i$ from $k$, its resulting $w_{y,i}$ becomes smaller than $w_y^k$ because there will be less interference from higher priority tasks, (i.e., $w_{y,i} < w_{y,k}$). In fact, this is the only change that occurs to the subsystem critical section execution time of all shared resources when $rc_y$ increases. Hence, the maximum subsystem critical section execution time $X$ can remain the same (if $w_{y,k} < X'$) or decrease (if $w_{y,k} = X'$) after $rc_y$ increases. That is, $X^* \leq X'$. □

---

- calculateBudget$(S_s, P_s, \mathcal{RC}_s)$ returns the smallest subsystem budget that satisfies Eq. (12.2).
- increaseCeilingX$^*(\mathcal{RC}_s)$ returns whether or not the ceiling of the resource associated with $X^*$ can be increased by one. If so, it increases the ceiling of the selected resource as well as the ceiling of all resources that have the same ceiling as the selected resource (Lemma 14).
- Interface is an array of interface candidates; each candidate is ($Q$, $X$, $RC$).
- addInterface(Interface, $Q^*, X^*, \mathcal{RC}_s$) adds new interface in the interface list array.
- removeRedundant(Interface) removes all redundant interfaces from the interface list.

1: $\mathcal{RC}_s = \{rc_1, \cdots, rc_m\}$ // $rc_j$=initial ceiling of $R_j$ using SRP
2: num = 0
3: do
4: $\quad Q^* =$ calculateBudget$(S_s, P_s, \mathcal{RC}_s)$
5: $\quad X^* = \max\{w_{1,rc_1}, \cdots, w_{m,rc_m})\}$
6: $\quad$ addInterface(Interface, $Q^*, X^*, \mathcal{RC}_s$)
7: $\quad$ num=removeRedundant(Interface)
8: while (increaseCeilingX$^*(\mathcal{RC}_s)$)
9: return (Interface, num)

---

Figure 12.2: The ICG algorithm.

| $\mathcal{T}$ | $C_i$ | $T_i$ | $R_j$ | $c_{i,j}$ | $\mathcal{T}$ | $C_i$ | $T_i$ | $R_j$ | $c_{i,j}$ |
|---|---|---|---|---|---|---|---|---|---|
| $\tau_1$ | 8 | 750 | $R_2$ | 4 | $\tau_2$ | 50 | 650 | $R_1$ | 5 |
| $\tau_3$ | 10 | 600 | - | 0 | $\tau_4$ | 35 | 500 | $R_1$ | 10 |
| $\tau_5$ | 1 | 160 | - | 0 | $\tau_6$ | 2 | 150 | - | 0 |

Table 12.1: Example task set parameters

### 12.6.1 ICG algorithm

**Description.** Using Lemmas 12, 13, and 14, we can reduce the complexity of a search algorithm. The algorithm shown in Figure 12.2 is based on these lemmas. In the beginning (at line 1), each resource ceiling $rc_j$ is set to its initial ceiling value according to SRP (without applying the technique in [18]). The algorithm then generates an interface candidate $(Q^*, X^*)$ based on the current resource ceilings (line 4 and 5). This new interface candidate is added into a list (line 6). Such addition can make some candidate redundant according to Lemma 1, and those redundant candidates are removed (line 7). Let $R^*$ denote the resource that determines $X^*$ in line 5, and $v^*$ denote the value of the ceiling ($rc^*$) of $R^*$ at that moment. In line 8, the algorithm 1) increases the ceiling $rc^*$ by one 2) checks the conditions given in Lemma 14 to further increase $rc^*$ if possible, and 3) increases the ceiling of all other resources that have the same ceiling as $v^* + 1$, to the current value of $rc^*$. This way, we can further reduce redundant interface candidates.

**Example.** We illustrate the ICG algorithm with the following example. Consider a subsystem $S_s$ that has six tasks as shown in Table 12.1. The local scheduler for the subsystem $S_s$ is Rate-Monotonic (RM) and we choose subsystem period $P_s = 125$. The algorithm works as shown in Table 12.2. The results from step 1 are $(Q_{s,1} = 51, X_{s,1} = 102)$, at step 2 $(Q_{s,1}, X_{s,1}) > (Q_{s,2}, X_{s,2})$. So $(Q_{s,1}, X_{s,1})$ is redundant (see Definition 2). That is, this interface can be removed according to Lemma 12. For the same reason, $(Q_{s,2}$, $X_{s,2})$ can be removed after step 3. At step 3, the $rc_2$ is increased directly to 4 according to Lemma 14 since $rc_1 > rc_2$ and $b_{2,1} > b_{2,2}$. At both steps 4 and 5, the ceiling $rc_1$ is increased by one since $X_{s,i} = w_1$ but we increase the ceiling of $rc_2$ according to Lemma 14. The algorithm selects the interface candidates from steps $3, 4$ and $5$.

**Correctness.** The following lemma proves the correctness of the ICG algorithm.

**Lemma 15.** *Let $\mathcal{IC}$ denote a set of up to $n$ interface candidates that are generated by the ICG algorithm of Figure 12.2. There exists no non-redundant*

| $Step$ | $rc_1$ | $rc_2$ | $w_1$ | $w_2$ | $Q_{s,i}$ | $X_{s,i}$ |
|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 13 | 102 | 51 | 102 |
| 2 | 4 | 2 | 13 | 52 | 51 | 52 |
| 3 | 4 | 4 | 13 | 7 | 51 | 13 |
| 4 | 5 | 5 | 12 | 6 | 52.5 | 12 |
| 5 | 6 | 6 | 10 | 4 | 56 | 10 |

Table 12.2: Example algorithm

*interface candidate* $(Q_{s,y}, X_{s,y})$ *such that* $(Q_{s,y}, X_{s,y}) \notin \mathcal{IC}$.

*Proof.* Assume that $(Q_{s,y}, X_{s,y})$ is a non-redundant interface candidate and that $X_{s,y} = w_{k,i}$, i.e., the subsystem critical section execution time of $R_k$ is the maximum among all global shared resources when $rc_k = i$. Then we shall prove that

1. There is no $R_j$ such that $b_{i,j} > b_{i,k}$ for all $rc_j > i$. Otherwise we could change the ceiling $rc_k = rc_j$ according to Lemma 14, and by this $w_{k,i} \neq X_{s,y}$.

2. There is no $R_j$ such that $b_{t,j} > b_{i,k}$ for all $rc_j < i, t < i$. Otherwise $w_{j,t} > w_{k,i}$ because when we compute the $w_k$ and $w_j$, the interference from higher priority tasks as well as blocking is higher for $R_j$, and then $w_{k,i} \neq X_{s,y}$. If we increase the ceiling $rc_j = i$, it will not give other non-redundant interface candidates (see Lemma 13 and 14).

We can conclude that there is only one resource $R_k$ that may generate a non-redundant interface at resource ceiling $i$, and this is the one that imposes the highest blocking at that level. The initial ceiling of $R_k$ is $v$, where $v \in [1, i]$. From Lemma 13, $b_{f,k}$ (where $f \in [v, i]$) is the maximum blocking at resource ceiling $rc_k \in [v, i]$. Since the presented algorithm increases the ceiling of the global resource that generate the maximum subsystem critical section execution time, it will increase the ceiling of $R_k$ when $rc_k = v$ up to $i$. Hence, we can guarantee that the algorithm will include the interface when $X_{s,y} = w_{k,i}$. □

The proof of the previous property also shows that the complexity of the proposed algorithm is $O(n)$ since we have $n$ tasks (which equals to the number of possible resource ceilings) and there is either 0 or 1 non-redundant interface for each resource ceiling level, and the algorithm will only traverse these non-redundant interfaces. Moreover, the proposed algorithm thereby produce at most $n$ interface candidates.

**Post-processing.** The ICG algorithm generates non-redundant interface candidates on the basis of Lemma 12. The notion of redundant candidate is so general that the ICG algorithm can be applicable to many synchronization protocols. In some cases, however, a set of interface candidates can be further refined, for instance, when the overrun mechanism described in Section 12.4.1 is used. Consider two candidates $(Q'_s, X'_s)$ and $(Q^*_s, X^*_s)$ such that $Q'_s + X'_s <= Q^*_s + X^*_s$ and $X'_s <= X^*_s$. Then, $(Q'_s, X'_s)$ will never produce not only a larger $\texttt{RBF}_s(t)$ for the subsystem $S_s$ itself, but also a larger blocking $B_j$ for other subsystems $S_j$, than $(Q^*_s, X^*_s)$ does. This immediately follows from Eqs. (12.4)-(12.6). Then, the following lemma directly follows:

**Lemma 16.** *Consider two candidates $(Q'_s, X'_s)$ and $(Q^*_s, X^*_s)$ such that $Q'_s + X'_s <= Q^*_s + X^*_s$ and $X'_s <= X^*_s$. Then, $(Q'_s, X'_s)$ will never impose more CPU requirement to $\mathsf{load}_{\mathsf{sys}}$ in any way than $(Q^*_s, X^*_s)$ does.*

*Proof.* Looking at Eq. (12.4), we can decrease $\texttt{LBF}_s(t)$ to decrease the system load by decreasing the blocking $B_s$ and/or $\texttt{RBF}_s(t)$. For the blocking, using the interface $Q_{s,i}, X_{s,i}$ may increase the blocking on the higher priority subsystems because $X_{s,i} > X_{s,j}$. For $\texttt{RBF}_s(t)$, it will be increased if we use $Q_{s,i}, X_{s,i}$ because $(Q_{s,i} + X_{s,i}) > (Q_{s,j} + X_{s,j})$ see Eq. (12.5). For this we can conclude that we can remove the interface $(Q_{s,i}, X_{s,i})$ since it will not reduce the system load compared with the other interfaces. □

According to Lemma 16, a set of interface candidates generated by the ICG algorithm goes through its post-processing for further refinement, and this is very useful for the second step of our approach.

## 12.7 Interface selection

In this section, we consider a problem, called the *optimal interface selection* problem, that selects a *system configuration* consisting of a set of subsystem interfaces, one from each subsystem that together minimize the system load subject to the schedulability of system. We present the ICS (Interface Candidate Selection) algorithm, an algorithm that finds an optimal solution to this problem through a finite number of iterative steps.

### 12.7.1 Description of the ICS algorithm

The ICS algorithm assumes that each set of interface candidates $(Q_s, X_s)$ is sorted in a decreasing order of $X_s$. In other words, each set is sorted in an

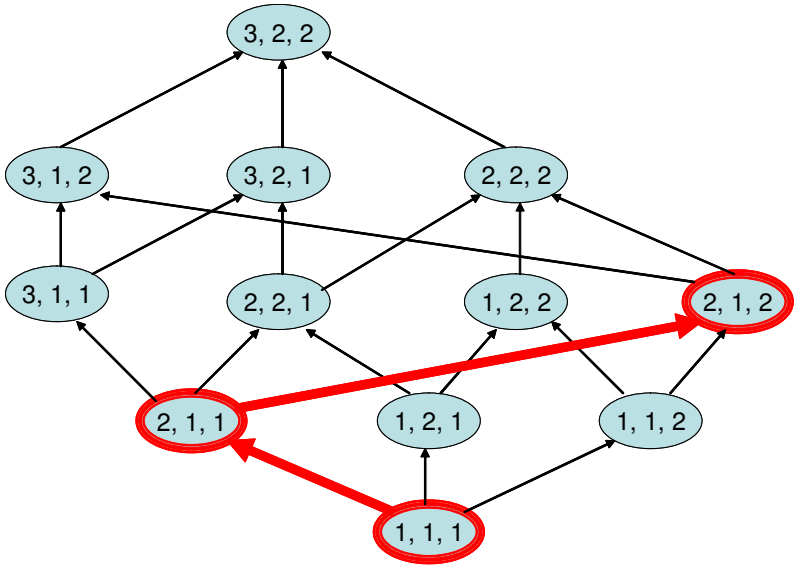


Figure 12.3: Search space for a system consisting of 3 subsystems.

increasing order of collective demands ($Q_s + X_s$) (see Lemma 16). Then, the first candidate $(Q_{s,1}, X_{s,1})$ has the largest critical section execution time but the smallest collective demands.

The ICS algorithm generates a finite number of system configurations through iteration steps. Each configuration is a set of individual interface candidates of all subsystems. Let $\mathsf{CF}_i$ denote a *configuration* that ICS generates at an $i$-th iteration step. For notational convenience, we introduce a variable $f_k^i$ to denote an element of $\mathsf{CF}_i$, i.e., $\mathsf{CF}_i = \{f_1^i, \ldots, f_N^i\}$. The variable $f_k^i$ represents the interface candidate index of a subsystem $S_k$, indicating that the configuration in the $i$-th step includes $(Q_{k,f_k^i}, X_{k,f_k^i})$.

Figure 12.3 shows an example to illustrate the ICS algorithm, where the system contains 3 subsystems such that subsystem $S_1$ has 3 interface candidates, and two other subsystems $S_2$ and $S_3$ have 2 candidates, respectively. Each node in the graph represents a possible configuration, and each number in the node corresponds to an interface candidate index in the order of $S_1$, $S_2$, and $S_3$. The arrows show the possible transitions between nodes at $i$-th iteration step, by increasing $f_k^i$ by 1 for each subsystem $S_k$ one by one. We describe

the ICS algorithm with this example.

**Initialization.** In the beginning, this algorithm generates an initial configuration $\mathsf{CF}_0$ such that it consists of the first interface candidates of all subsystems. In Figure 12.3, $\mathsf{CF}_0 = \{1, 1, 1\}$ (see line 2 of Figure 12.4).

**Iteration step.** The ICS algorithm transits from $(i-1)$-th step to $i$-th step, increasing only one element of $\mathsf{CF}_{i-1}$ in value by one. In Figure 12.3, the arrows with bold lines illustrate the path that ICS can take. For instance, ICS moves from the initialization step ($\mathsf{CF}_0 = \{1, 1, 1\}$) to the first step ($\mathsf{CF}_1 = \{2, 1, 1\}$). Then, the ICS algorithm excludes the two sibling nodes of $\mathsf{CF}_1$ in the figure (i.e., $\{1, 2, 1\}$ and $\{1, 1, 2\}$) from the remaining search space; the algorithm will never visit those nodes from this step on. This way, ICS can efficiently explore the search space. Let us describe how ICS behaves at each iteration step more formally.

Firstly, let $\delta_i$ denote the only single element whose value increases by one between $\mathsf{CF}_{i-1}$ and $\mathsf{CF}_i$, i.e.,

$$f_k^i = \begin{cases} f_k^{i-1} + 1 & \text{if } k = \delta_i, \\ f_k^{i-1} & \text{otherwise.} \end{cases} \tag{12.11}$$

In the example shown in Figure 12.3, $\delta_1 = 1$.

Let us explain how to determine $\delta_i$ at an $i$-th step. We can potentially increase every elements of $\mathsf{CF}_{i-1}$, and thereby we have at most $N$ candidates for the value of $\delta_i$. Here, we choose one out of at most $N$ candidates such that a resulting $\mathsf{CF}_i$ can cause the system load to be minimized.

Let $\mathsf{load}_{\mathsf{sys}}(i)$ denote the value of $\mathsf{load}_{\mathsf{sys}}$ when a configuration $\mathsf{CF}_i$ is used as a *system interface*. We are now interested in reducing the value of $\mathsf{load}_{\mathsf{sys}}(i-1)$. Let $s^*$ denote the subsystem $S_{s^*}$ that has the largest *processor request bound* among all subsystems. That is, $\mathsf{load}_{\mathsf{sys}}(i-1) = \alpha_{s^*}$ (see Eq. (12.9)). We can find such $S_{s^*}$ by evaluating the *processor request bound*'s of all subsystems (in line 5 of Figure 12.4).

By the definition of $s^*$, we can reduce the value of $\mathsf{load}_{\mathsf{sys}}(i-1)$ by reducing the value of $\mathtt{LBF}_{\mathtt{s}^*}(\mathtt{t})$. There are two potential ways to reduce the value of $\mathtt{LBF}_{\mathtt{s}^*}(\mathtt{t})$. From the definition of $\mathtt{LBF}_{\mathtt{s}}(\mathtt{t})$ in Eq. (12.4), one is to reduce its maximum blocking $B_{s^*}$ and the other is to reduce the subsystem CPU demands ($\mathtt{RBF}_{s^*}(t)$). A key aspect of this algorithm is that it always reduces the blocking part, but does not reduce the request bound function part. An intuition behind is as follows: this algorithm starts from the interface candidates that have the smallest demands but the largest subsystem critical section execution times, respectively. Hence, for each interface candidate, there is no room to further

reduce its demand. However, there is a chance to reduce the maximum blocking $B_{s^*}$ of $S_{s^*}$. It can be reduced by decreasing the $X_{k^*}$ of a subsystem $S_{k^*}$ that imposes the largest blocking to the subsystem $S_{s^*}$. We define $k^*$ in a more detail.

Let $k^*$ denote the subsystem $s_{k^*}$ that imposes the largest blocking to the subsystem $S_{s^*}$, i.e., $B_{s^*} = X_{k^*} = \max\{X_j \mid \text{for all} X_s \in LPS(s^*)\}$[3], where $LPS(i)$ is a set of lower-priority subsystems of $S_{s^*}$. We can find such $S_{k^*}$ easily by looking at the subsystem critical section execution times of all lower-priority subsystems of $S_{s^*}$ (in line 6 of Figure 12.4).

When such $S_{k^*}$ is found, it then checks whether the $X_{k^*}$ can be further reduced (in line 7 of Figure 12.4). If so, it is reduced (in line 8), and $\mathsf{CF}_{i-1}$ becomes to $\mathsf{CF}_i$ (in line 9). That is, $\delta_i = k^*$.

**Iteration termination.** The above iteration process terminates when the blocking $B_{s*}$ of subsystem $S_{s^*}$ cannot be reduced further. The algorithm then finds the smallest value of $\mathsf{load}_{\mathsf{sys}}$ out of the values saved during the iteration, and it returns a set of interfaces corresponding to the smallest value.

**Complexity of the algorithm.** During an $i$-th iteration, the algorithm only increases the interface candidate index of a subsystem $S_{\delta_i}$. Then, it can repeat $O(N * m')$ iterations, where $N$ is the number of subsystems and $m'$ is the greatest number of interface candidates of a subsystem among all.

### 12.7.2 Correctness of the ICS algorithm

In this section, we show that the ICS algorithm produces a set of system configurations that contains an optimal solution. We first present notations that are useful to prove the correctness of the algorithm.

• $\mathcal{AS}$ We consider the entire search space of the optimal interface selection problem. It contains all possible subsystem interfaces comprising a system configuration, and let $\mathcal{AS}$ denote it, i.e.,

$$\mathcal{AS} = IC_1 \times \cdots \times IC_n. \tag{12.12}$$

In the example shown in Figure 12.3, the entire solution space ($\mathcal{AS}$) has 12 elements.

We present some notations to denote the properties of the ICS algorithm at an arbitrary $i$-th iteration step.

[3] If more than one lower priority subsystem impose the same maximum blocking on $Ss^*$, then we select the one with lowest priority.

- $IC_s$ is an array of interface candidates of subsystem $S_s$,
  sorted in a decreasing order of $X_s$.
- $\mathsf{ici}_s$ is an index to $IC_s$ of subsystem $S_s$.
- $\mathcal{I}$ is a set of interfaces $\{I_s\}$, each of which indicated by $\mathsf{ici}_s$.
- subsystemWithMaxLoad() returns the subsystem $S_{s^*}$
  that has the greatest *processor request bound* among
  all subsystems, i.e., $\mathsf{load}_{\mathsf{sys}} = \alpha_{s^*}$.
- maxBlockingSubsystemToSysload($\mathsf{s}^*$) returns a subsystem
  $S_{k^*}$ that produces the greatest blocking to a subsystem $S_{s^*}$.
  Note that $S_{s^*}$ determines the system load.

```
1:  for all  S_s ∈ 𝒮
2:       ici_s = 1;  I_s = IC_s[ici_s]
3:  load*_sys = 1.0;  ℐ* = ℐ
4:  do
5:       s* = subsystemWithMaxLoad()
6:       k* = maxBlockingSubsystemToSysload(s*)
7:       if (ici_k* can increase by one)
8:            ici_k* = ici_k* + 1
9:            I_k* = IC_k*[ici_k*]
10:           compute load_sys according to Eq. (12.9)
11:           if (load_sys < load*_sys)
12:                load*_sys = load_sys
13:                ℐ* = ℐ
14:      else
15:           return ℐ* (that determines load*_sys)
16: until (true)
```

Figure 12.4: The ICS algorithm.

• $\widehat{IC}_k^i$ In the beginning, the ICS algorithm has the entire search space ($\mathcal{AS}$) to explore. Basically, this algorithm gradually reduces a remaining search space to explore during iteration. For notation convenience, we introduce a variable ($\widehat{IC}_k^i$) to indicate the remaining interface candidates of a subsystem

$S_k$ to explore. By definition, $f_k^i$ indicates which interface candidate of a subsystem $S_k$ is selected by $\mathsf{CF}_i$. This algorithm continues exploration from the interface candidate indicated by $f_k^i$ from the end of an $i$-th step. Then, $\widehat{IC}_k^i$ is defined as

$$\widehat{IC}_k^i = \{f_k^i, \ldots, max_k\} \text{ for all } k = 1, \ldots, n, \tag{12.13}$$

where $max_k$ is the number of interface. In the example shown in Figure 12.3, $\widehat{IC}_1^1 = \{2, 3\}$.

• $\mathsf{XP}_i$ Let us define $\mathsf{XP}_i$ to denote the search space remaining to explore after the end of an $i$-th iteration step. Note that such a remaining search space does not have to include the solution candidate $\mathsf{CF}_i$ chosen at the $i$-th step. Then, $\mathsf{XP}_i$ is defined as

$$\mathsf{XP}_i = (\widehat{IC}_1^i \times \cdots \times \widehat{IC}_n^i) \setminus \mathsf{CF}_i. \tag{12.14}$$

• $\mathsf{RM}_i$ In essence, the ICS algorithm gradually decreases a remaining search space during iteration. That is, at an $i$-th step, it keeps reducing $\mathsf{XP}_{i-1}$ to $\mathsf{XP}_i$, where $\mathsf{XP}_i \subset \mathsf{XP}_{i-1}$. Let $\mathsf{RM}_i$ denote a set of interface settings that is excluded from $\mathsf{XP}_{i-1}$ at the $i$-th step. Note that at the $i$-th step, the interface candidate of a subsystem $S_{\delta_i}$ changes from $f_{\delta_i}^{i-1}$ to $f_{\delta_i}^i$. Then, a subset of $\mathsf{XP}_i$ that contains the value of $f_{\delta_i}^{i-1}$, is excluded at the $i$-th step. $\mathsf{RM}_i$ is defined as

$$\mathsf{RM}_i = (\widehat{IC}_1^{(i-1)*} \times \cdots \times \widehat{IC}_n^{(i-1)*}) \setminus \{\mathsf{CF}_{i-1}\}, \text{ where} \tag{12.15}$$

$$\widehat{IC}_k^{(i-1)*} = \begin{cases} \{f_k^{i-1}\} & \text{if } k = \delta_i, \\ \widehat{IC}_k^i & \text{otherwise.} \end{cases} \tag{12.16}$$

In the example shown in Figure 12.3, $\mathsf{RM}_1 = \{\{1, 2, 1\}, \{1, 2, 2\}, \{1, 1, 2\}\}$.

• $\mathsf{AH}_i$ Let $\mathsf{AH}_i$ represent a set of system configurations that the ICS algorithm selects from the first step through to an $i$-th step, i.e.,

$$\mathsf{AH}_i = \{\mathsf{CF}_1, \ldots, \mathsf{CF}_i\}. \tag{12.17}$$

• $\mathsf{AR}_i$ Let $\mathsf{AR}_i$ represent a set of interface candidates that the ICS algorithm excludes from the first step through to an $i$-th step, i.e.,

$$\mathsf{AR}_i = \mathsf{RM}_{(i-1)} \cup \mathsf{RM}_i, \text{ where } \mathsf{AR}_0 = \phi. \tag{12.18}$$

We define partial ordering between interface candidates as follows:

**Definition 3.** *A interface candidate* $\mathsf{sc} = \{c_1, \ldots, c_n\}$ *is said to be* strictly precedent *of another interface candidate* $\mathsf{sc}' = \{c'_1, \ldots, c'_n\}$ *(denoted as* $\mathsf{sc} \prec \mathsf{sc}'$*) if* $c_j < c'_j$ *for some* $j$ *and* $c_k \leq c'_k$ *for all* $k$*, where* $1 \leq (j, k) \leq n$.

As an example, $\{1, 1, 1\} \prec \{1, 2, 1\}$.

The following lemma states that when the algorithm excludes a set of interface candidates from further exploration at an arbitrary $i$-th step, a set of such excluded interface candidates does not contain an optimal solution.

**Lemma 17.** *At an arbitrary* $i$*-th iteration step, the ICS algorithm excludes a set of interface candidates (*$\mathsf{RM}_i$*), and any excluded solution candidate* $r \in \mathsf{RM}_i$ *does not yield a smaller system load than that by* $\mathsf{CF}_{i-1}$.

*Proof.* As explained in Section 12.7.1, there are two potential ways to reduce the value of $\mathsf{load}_{\mathsf{sys}}(\mathsf{CF}_{i-1})$ at the $i$-th step. One is to reduce the CPU resource demand of the subsystem $S_{s_i^*}$ (i.e., $\mathtt{RBF}_{\mathtt{s}_\mathtt{i}^*}(\mathtt{t})$), and the other is to reduce its maximum blocking $B_{s_i^*}$.

Firstly, we wish to show that $\mathtt{RBF}_{\mathtt{s}_\mathtt{i}^*}(\mathtt{t})$ does not decrease when we transform $\mathsf{CF}_{i-1}$ to any interface candidate $r \in \mathsf{RM}_i$. Note that each interface candidate set is sorted in an increasing order of resource requirement budget ($Q$). One can easily see that $\mathsf{CF}_{i-1} \prec r$. Then, it follows that $\mathtt{RBF}_{\mathtt{s}_\mathtt{i}^*}(\mathtt{t})$ never decreases when $\mathsf{CF}_{i-1}$ changes to $r$.

Secondly, we wish to show that when we change $\mathsf{CF}_{i-1}$ to any interface candidate $r \in \mathsf{RM}_i$, $B_{s_i^*}$ does not decrease. As shown in line 6 in Figure 12.4, the ICS algorithm finds the subsystem $S_{\delta_i}$ that generates the maximum blocking to for subsystem $S_{s_i^*}$. Then, the algorithm increases $f_{\delta_i}^{i-1}$ by one, if possible, to decrease $B_{s_i^*}$. However, by definition, for all elements $r$ of $\mathsf{RM}_i$, the element for the subsystem $S_{\delta_i}$ has the value of $f_{\delta_i}^{i-1}$, rather than the value of $f_{\delta_i}^{i}$. This means that $B_{s_i^*}$ never decreases when we change $\mathsf{CF}_{i-1}$ to $r$. □

The following lemma states that when the algorithm terminates at an arbitrary $f$-th step, a set of remaining interface candidates does not contain an optimal solution.

**Lemma 18.** *When the ICS algorithm terminates at an arbitrary* $f$*-th step, any remaining interface candidate (*$\mathsf{xp} \in \mathsf{XP}_f$*) does not yield a smaller system load than* $\mathsf{CF}_f$ *does.*

*Proof.* As explained in the proof of Lemma 17, there are two ways to reduce $\mathsf{load}_{\mathsf{sys}}$ (i.e., $\mathtt{LBF}_{\mathtt{s}_\mathtt{f}^*}(\mathtt{t})$).

One is to reduce $\mathtt{RBF}_{\mathtt{s}_f^*}(\mathtt{t})$ in Eq. (12.5) . However, it does not decrease, since $\mathsf{CF}_f \prec \mathsf{xp}$ for all $\mathsf{xp} \in \mathsf{XP}_f$.

The other is to reduce the maximum blocking ($B_{s_f^*}$). In fact, the ICS algorithm terminates at the $f$-th step because there is no way to decrease $B_{s_f^*}$. That is, $B_f$ does not decrease when $\mathsf{CF}_f$ changes to any $\mathsf{xp}$. □

The following lemma states that at $i$-th step, the remaining search space to explore decreases by $(\mathsf{RM}_i \cup \{\mathsf{CF}_i\})$.

**Lemma 19.** *At an arbitrary $i$-th iteration step,*

$$\mathsf{XP}_i = \mathsf{XP}_{i-1} \setminus (\mathsf{RM}_i \cup \{\mathsf{CF}_i\}). \tag{12.19}$$

*Proof.* The ICS algorithm transforms $\mathsf{CF}_{i-1}$ to $\mathsf{CF}_i$ at an $i$-th step by increasing the value of its $\delta_i$-th element. Then, we have

$$\widehat{IC}_k^i = \begin{cases} \widehat{IC}_k^{i-1} \setminus \{f_k^{i-1}\} & \text{if } k = \delta_i, \\ \widehat{IC}_k^{i-1} & \text{otherwise.} \end{cases} \tag{12.20}$$

Without loss of generality, we assume that $\delta_i = 1$. For notational convenience, let $\mathsf{XP}_i^* = \mathsf{XP}_i \cup \{\mathsf{CF}_i\}$, and $\mathsf{RM}_i^* = \mathsf{RM}_i \cup \{\mathsf{CF}_i\}$. Then, we have

$$\begin{aligned} \mathsf{XP}_i^* &= \widehat{IC}_1^i \times \widehat{IC}_2^i \times \cdots \times \widehat{IC}_n^i \\ &= \left(\widehat{IC}_1^{i-1} \setminus \{f_1^{i-1}\}\right) \times \widehat{IC}_2^i \times \cdots \times \widehat{IC}_n^i \\ &= \left(\widehat{IC}_1^{i-1} \times \widehat{IC}_2^{i-1} \times \cdots \times \widehat{IC}_n^{i-1}\right) \setminus \\ &\quad \left(\{f_1^{i-1}\} \times \widehat{IC}_2^i \times \cdots \times \widehat{IC}_n^i\right) \\ &= \mathsf{XP}_{i-1}^* \setminus \mathsf{RM}_i^* \\ &= \left(XP_{i-1} \cup \{\mathsf{CF}_{i-1}\}\right) \setminus \left(\mathsf{RM}_i \cup \{\mathsf{CF}_{i-1}\}\right) \\ &= XP_{i-1} \setminus \mathsf{RM}_i \,. \end{aligned} \tag{12.21}$$

That is, considering $XP_i^* = XP_i \cup \{\mathsf{CF}_i\}$, it follows

$$XP_i = XP_{i-1} \setminus (\mathsf{RM}_i \cup \{\mathsf{CF}_i\}) \,. \tag{12.22}$$

□

The following lemma states that at any $i$-th iteration step, the entire search space can be divided into a set of explored candidates ($\mathsf{AH}_i$), a set of excluded candidates ($\mathsf{AR}_i$), and a set of remaining candidates to explore ($\mathsf{XP}_i$).

**Lemma 20.** *At an arbitrary $i$-th step, the sets of $\mathsf{AR}_i$, $\mathsf{AH}_i$, and $\mathsf{XP}_i$ include all possible interface candidates.*

$$\mathsf{AR}_i \cup \mathsf{AH}_i \cup \mathsf{XP}_i = \mathcal{AS} \tag{12.23}$$

*Proof.* We will prove this lemma by using mathematical induction. As a base step, we wish to show Eq. (12.23) is true, when $i = 1$. Note that $\mathsf{AR}_0 = \phi$ and $\mathsf{AH}_0 = \{\mathsf{CF}_0\}$. In addition, $\mathsf{XP}_0 = \mathcal{AS} \setminus \mathsf{CF}_0$, according to Eq. (12.14). It follows that $\mathsf{AR}_0 \cup \mathsf{AH}_0 \cup \mathsf{XP}_0 = AP$.

We assume that Eq. (12.23) is true at the $i$-th iteration step of the ICS algorithm. We then wish to prove that it also holds at the $(i+1)$-th step, i.e.,

$$\mathsf{AR}_i \cup \mathsf{AH}_i \cup \mathsf{XP}_i = \mathsf{AR}_{i+1} \cup \mathsf{AH}_{i+1} \cup \mathsf{XP}_{i+1}. \tag{12.24}$$

According to the definitions $\mathsf{AH}_{i+1}$, $\mathsf{AR}_{i+1}$, and $\mathsf{XP}_{i+1}$ (see Eq. (12.17), (12.18) and (12.19)), we can rewrite the right-hand side of Eq. (12.24) as follows:

$$\begin{aligned} & \mathsf{AR}_{i+1} \cup \mathsf{AH}_{i+1} \cup \mathsf{XP}_{i+1} \\ = \; & \Big(\mathsf{AR}_i \cup \mathsf{RM}_{i+1}\Big) \cup \Big(\mathsf{AH}_i \cup \{\mathsf{CF}_{i+1}\}\Big) \cup \\ & \Big(\mathsf{XP}_i \setminus (\mathsf{RM}_{i+1} \cup \{\mathsf{CF}_{i+1}\})\Big) \\ = \; & \mathsf{AR}_i \cup \mathsf{AH}_i \cup \mathsf{XP}_i \; . \end{aligned}$$

□

The following theorem states that the ICS algorithm produces a set of system configurations, which must contain an optimal solution.

**Theorem 21.** *When the ICS algorithm terminates at the $f$-th step, a set of system configurations ($\mathsf{AH}_f$) includes an optimal solution.*

*Proof.* Let *opt* denote an optimal solution. We prove this theorem by contradiction, i.e., by showing that $opt \notin \mathsf{AR}_f$ and $opt \notin \mathsf{XP}_f$.

Suppose $opt \in \mathsf{AR}_f$. Then, by definition, there should exist $\mathsf{RM}_i$ such that $opt \in \mathsf{RM}_i$ for an arbitrary $i \leq f$. According to Lemma 17, $\mathsf{load}_{\mathsf{sys}}(\mathsf{CF}_{i-1}) < \mathsf{load}_{\mathsf{sys}}(opt)$, which contradicts the definition of *opt*. Hence, $opt \notin \mathsf{AR}_f$.

Suppose $opt \in \mathsf{XP}_f$. Then, according to Lemma 18, it should be $\mathsf{load}_{\mathsf{sys}}(\mathsf{CF}_f) < \mathsf{load}_{\mathsf{sys}}(opt)$, which contradicts the definition of $opt$ as well. Hence, $opt \notin \mathsf{AR}_f$.

According to Lemma 20, it follows that $opt \in \mathsf{CF}_f$. □

## 12.8 Conclusion

When subsystems share logical resources in a hierarchical scheduling framework, they can block each other. In particular, when a budget expiry problem exists, such blocking can impose extra CPU demands. However, simply reducing the blocking of subsystems does not monotonically decrease the system load, since imposing less blocking to other subsystems can impose more CPU requirements of the subsystems themselves. This paper introduced such a tradeoff and 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.

In this paper, we considered only fixed-priority scheduling, and we plan to extend our framework to EDF scheduling. Furthermore, our future work includes generalizing our framework to other synchronization protocols. For example, this paper considered only the overrun mechanism without payback [15], and we are extending towards another overrun mechanism (with-payback version) [15]. Unlike with the former overrun mechanism, the intra- and inter-subsystem aspects of the tradeoff are not clearly separated with the latter mechanism. The latter mechanism changes the way of a subsystem's own contributing to the system load (i.e., Eq. (12.5)), and this requires appropriate changes to the post-processing part of the ICG algorithm. We are investigating how to make changes to the post-processing part in ways that require less subsequent changes to the ICS algorithm.

# Bibliography


[1] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the $4^{th}$ ACM International Conference on Embedded Software (EMSOFT'04)*, pages 95–103, September 2004.

[2] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the $26^{th}$ IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, December 2005.

[3] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* $18^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, December 1997.

[4] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Proceedings of the $28^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 129–138, December 2007.

[5] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hieararchical real-time components. In *Proceedings of the 6th ACM Conference on Embedded Software*, September 2006.

[6] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the $23^{th}$ IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[7] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the $20^{th}$ IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, December 1999.

[8] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the 6$^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[9] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15$^{th}$ Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, July 2003.

[10] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the 7$^{th}$ IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 75–84, May 2001.

[11] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, December 2003.

[12] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the 25$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 57–67, December 2004.

[13] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 423–434, December 2007.

[14] 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 7$^{th}$ ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, October 2007.

[15] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, December 2006.

[16] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the 28$^{th}$ IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[17] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):(30)1–39, April 2008.

[18] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the* $15^{th}$ *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'07)*, pages 1–8, March 2007.

[19] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON'87)*, pages 909–916, November 1987.

[20] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the* $9^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, December 1988.

[21] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[22] Insik Shin, Moris Behnam, Thomas Nolte, and Mikael Nolin. On optimal hierarchical resource sharing in open environments. Technical report, 2008. Available at http://www.idt.mdh.se/∼tnt/rtss08long.pdf.

[23] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the* $10^{th}$ *IEEE International Real-Time Systems Symposium (RTSS'89)*, pages 166–171, December 1989.

[24] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal Embedded Computing*, 1(2):257–269, 2005.