Resource Reservation in Real-Time Operating Systems - a joint industrial and academic position

Liesbeth Steffens
Philips Research
liesbeth.steffen@philips.com

Gerhard Fohler
Mälardalen University
gerhard.fohler@mdh.se

Giuseppe Lipari
Scuola Sup. S.Anna
lipari@sssup.it

Giorgio Buttazzo
Università di Pavia
buttazzo@unipv.it

Abstract

Resource Reservation (RR) is a class of real-time schedulers, proposed by the research community for hard and soft real-time systems, which are receiving considerable attention from industry. With this white paper, we would like to trigger a broad discussion between the academic and the industrial world on the use of RR techniques in standard operating systems. After presenting the major reasons for using RR ("why"), we present a set of propositions on "what" and "how" should be provided, which pinpoint the demands on operating systems and the research issues that remain to be addressed.

1. Introduction

Reservation-based resource partitioning for real-time systems, *resource reservation* (RR) for short, is an emerging paradigm for resource management in real-time systems. We strongly believe that, sooner or later, this paradigm will become a standard for mainstream Real-Time Operating Systems. Instead of sitting by and watching it happen, we want to trigger a broad discussion, involving scientists, practitioners, and operating system designers, with the aim of identifying demands on operating systems and remaining research issues for the deployment of resource reservation methods.

Over the last 15 years or so, many proposals have been made that could be categorized as steps towards reservation-based resource partitioning. These proposals provide a wealth of information on what can be done, what should be done, and also what should not be done.

In this paper, we present a number of propositions, with their arguments, which we obtained by looking at this information from different angles. These propositions are intended to help to make RR mature, and to take away obstacles that prevent RR from becoming a de-facto standard for mainstream real-time kernels. Some of the propositions are almost obvious, and will not raise much debate; others are more provocative. In these propositions, we mainly focus on the processing resource. This is not a matter of principle, but a practical restriction. The propositions address the following questions: what resource reservation, and how resource reservation? However, they do not provide particular solutions. There is no proposition to use a particular scheduling algorithm. As a warming up, we present some answers to the inevitable questions from the unconvinced: why resource reservation, and where resource reservation?

2. Reasons for resource reservation

Why resource reservation should be implemented in standard real-time systems? The most important reasons are listed below. Note that for most of these requirements, RR should be viewed as an approach to resource management in general, not just to processor scheduling.

Temporal protection. Classical real-time schedulability analysis is based on the knowledge of the worst-case execution time (WCET) of all periodic tasks. In modern processors it is very difficult to tightly estimate the WCETs. If a task instance executes more than assumed by the schedulability analysis, this is a *temporal fault*. Temporal faults may occur because the WCET has not been computed correctly, because of a deliberate decision not to allocate for the worst case, or because of a temporary or permanent fault in the application (the Boolean exit condition from a cycle never assumes the value true). The resource reservation mechanisms provide *temporal protection*: they prevent temporal faults from affecting the temporal behaviour of other tasks in the system, by enforcing execution time bounds at run time [6]. Temporal protection is the main reason for adopting some form of RR as a de-facto standard for the aerospace industry [2].

Independent design, analysis and validation of real-time subsystems. Real-time systems are becoming increasingly complex, and the cost (in time and money) of design, analysis and validation is growing out of bounds. The virtual resource abstraction, combined with hierarchical scheduling, allows system designers to tackle this problem in a very straightforward manner. The complex real-time system can be designed as a set of independent (or loosely coupled) subsystems, each consisting of a real-time application running on a set of virtual resources. Each subsystem can be independently designed, analysed, and validated. The final integration phase, which is currently the bottleneck in the design of many real-time systems, reduces to integrating the different virtual resources into the same system. [5][14].

Re-use of legacy applications. Sometimes we want to re-use an already developed and functioning application (or component) in a new real-time system without going back to the design and analysis phase. The reasons are not only cost reduction, but also reduced development time. The virtual resource abstraction allows porting the old application to the new system with relatively little effort, since it is now only necessary to analyse the application on the virtual resource that has the same characteristics of the old processor. [5][14].

Quality of Service (QoS). In some domains, such as real-time control and multimedia, applications can trade quality for resources. The quality reduction can be in the relaxation of timing requirements [8][9], and/or in the acceptance of imprecise or approximated results [13][31]. The general approach to QoS management is QoS (re)negotiation. Resource reservations can be viewed as negotiable service contracts at the level of resource delivery, which form a foundation for negotiable service contracts at the application level.

Hybrid open systems. For practical reasons, large computational platforms are increasingly shared between real-time and non-real time applications. Such platforms are traditionally conceived as open systems. An obvious example of these systems is the personal computer, which is more and more evolving into a multimedia platform, but this trend can also be observed in other domains. [6][13][34][23][15].

3. Application Domains

There is a need for resource reservation in many application domains, from critical hard real-time domains such as aerospace applications, to more soft real-time domains like multimedia.

Aerospace is a traditional domain for hard real-time systems. The aerospace community has identified the need for temporal protection and has introduced it as part of the Avionics Application Software Standard Interface (APEX) [2], one of the standards in Integrated Modular Avionics (IMA) [1]. APEX introduces the concept of *partition*. Partitions are (subsystems that occupy) distinct parts of physical memory. Partitions contain processes. Inter-process communication is allowed within and between partitions. Each partition has a number of properties, such as the criticality level, period, duration, and lock or pre-emption level. Partitions are scheduled by a cyclic schedule, which has been created offline, and takes the abovementioned properties into account. Non-critical partitions are not allowed to consume processing resources outside their slots in the cyclic schedule.

Multimedia is becoming a major player in many application domains, ranging from consumer electronics to military equipment. The underlying media and signal processing is very data intensive, and thus imposes very high requirements on the system. Nevertheless, media and signal processing is gradually moving from dedicated hardware to programmable hardware (in various degrees of programmability) in combination with software. Multimedia processing often shows high load fluctuations and a large gap between average- and worst-case processing requirements, but this is compensated by the fact that multimedia allows a trade off between output quality (image quality, output latency, output frequency, output jitter, motion fidelity, etc.) and resource usage. Consumer devices have the additional requirements of low cost and short time to market on one hand, and robustness and predictability on the other. Guaranteed reservations are very well suited to address robustness, predictability and short time to market (reuse and independent development), and provide a good basis for achieving low cost by taking advantage of the quality-for-resource trade-off [8][19].

Real-time control systems (such as robotics) have always been the domain of hard real-time scheduling. Recent study [24] has shown that the use of resource reservation techniques can bring advantages in terms of control performance. Designing the system for the worst-case guarantees absence of deadline misses, but can impose a low control loop rate. On the other hand, designing the system for the average case allows increasing the control loop rate, and thus improving the control performance, in the average case, but can lead to deadline misses of critical activities. Resource reservation techniques permit to calibrate the resource usage for the different activities. For example, in a complex control system with multi-rate sensor acquisition that uses a video camera for pattern recognition, the most critical low-level control loop can be considered as a hard activity, which should be assigned an amount of resource equal to its worst-case requirement. The less critical activities, like image acquisition and recognition, can be assigned a fraction of the resource equal to their average case conditions, thus increasing their rate [23][24].

4. Propositions

On the one hand, RR provides the abstraction of a virtual dedicated resource to the applications. With this abstraction, an application behaves approximately as if it were using a slower dedicated resource. On the other hand, RR provides the abstraction of a temporal representation of an application to the scheduler. With this abstraction, the scheduler is freed from the responsibility for guaranteeing the intricate timing requirements of the applications; its only responsibility is to guarantee the requirement imposed by the temporal representation. RR relies on combined mechanisms for admission control (at design time or at run time), allocation or scheduling (at design time or at run time), accounting (at run time), and enforcement (at run time) [6][28].

Reservations for applications rather than threads

Recent developments show that reservation is required for clusters of threads as well as for individual threads; see for example the partition in ARINC and Integrity [2][3], the protection domain in VxWorks AE [4], and the Resource Consuming Entity (RCE) in QoS RM and Hola-QoS [29]. In this paper we will use the term application for such a cluster of threads. The applications have the impression to run on a private processor, with private memory. The latter is particularly true for the ARINC partitions: even the name is derived from the partition in physical memory that they occupy. The reservation framework provides firewalls against intrusion by other applications, in both the temporal and spatial domain. Any communication with other partitions follows the distributed system paradigm. In these examples, the threads inside a cluster are scheduled according to a standard RTOS API (fixed priority with local priorities). This allows re-use of existing applications.

Proposition 1 Cluster of threads (what)
Aim reservations at applications (clusters of threads) rather than threads.

Proposition 2 Processor and memory (what)
The application is the recipient of both processor and memory reservations.

Proposition 3 Protection (what)
Make protection an integral aspect of reservation for applications.

Proposition 4 Inter-application communication (what)
Provide primitives for inter-application communication (comparable to IPC in UNIX) that have predictable temporal characteristics

Proposition 5 RTOS API (what)
Make local RTOS API available to applications.

There is a single level, the application, for which both resources are reserved. This does not exclude local scheduling schemes. For example: an application can have a local reservation-based scheduler for threads, and local memory protection for processes. When memory is local to the application, this implicitly implies that all memory-based resources, such as semaphores and mailboxes, are local to the application, and that the use of shared memory for communication is possible within an application, but not between applications. As a consequence, applications must be able to communicate in some other way, and the RTOS has to provide the appropriate primitives. The propositions above are restricted to processor and memory. For a more detailed discussion of multiple resources, see the paragraph on Multiple resources below. Providing a local RTOS API is a convenience, to allow the use of legacy applications, or to accommodate programmers that are used to a particular paradigm. For an alternative approach see Proposition 8.

Virtual resource abstraction for temporal resource

Given a perfect virtual resource abstraction for a temporal (volatile) resource, such as processors and communication links, an application will receive a fixed share α of the processor bandwidth. The application has the impression of using a uniformly slower virtual resource. Let W be an arbitrary time window of T time units. Let the application be active (ready to execute, not suspended or blocked) throughout W. Then the application will receive αT time units during W. Let W' be an arbitrary time window of T' time units. Let the application be inactive (not runnable, but suspended or blocked) throughout W'. Then the application will receive 0 time units during W'. This ideal model cannot be implemented in a real system, because it requires a division of the time line into infinitesimal intervals, but it is useful as a reference for scheduling algorithms.
A reservation specification requires at least one parameter that represents the acceptable *temporal granularity* of the reservation. Temporal granularity is a measure for the acceptable deviation from the corresponding perfect (=uniformly slower) virtual resource abstraction. Several granularity measures have been proposed [5][17][30]. There is probably not a single solution that is suitable for all applications. Additional research is clearly needed in this area.

Proposition 6 Granularity (what)
Provide means for specifying allocation granularity in the reservation specification.

Temporal constraints and local scheduling

For some application types, the virtual resource abstractions described above are not sufficiently precise. For instance, the adaptive approach of [13] requires that the amount of processor time guaranteed by the reservation up to a future (periodic) deadline can be obtained by the application, whereas the off-line scheduling approach of [16][21] requires that a certain amount of processor time is available during specific intervals. Such application requirements can be viewed as special instances of granularity requirements. An application must have the freedom to specify what it needs. On the other hand, the specified requirements must be feasible.

Proposition 7 Temporal constraints (what)
Reservation contracts should be allowed to specify customised temporal constraints.

Local scheduling

Some applications require a standard RTOS API that includes scheduling-related primitives. Other applications prefer to do their own scheduling. This is for instance the case for application that use off-line scheduling [16], or are self-timed [33]. Allowing applications to do their own scheduling is also practical in situations where a scheduling hierarchy is desired [5][14][34].

Proposition 8 Local scheduling (what)
Allow applications to do their own local scheduling.

Input from QoS practice

Reservations are based on resource estimates. If the application that holds the reservation is a life-critical hard real-time application, these resource estimates will be worst-case, and the corresponding budget will be fixed for the lifetime of the application. This is the case in the Avionics standards [1][2]. For many applications, however, this will not be the case. For such applications, the actual resource usage may deviate from the resource estimates underlying the reservation. Moreover, for some applications, it is more desirable to have a smaller reservation, and thus a reduced quality of service, than have no reservation at all. Thus, the reservation cannot be expected to always correspond to the application's needs.
As a consequence, applications should be robust under overload, and have the ability to adapt to the available resources. Research on adaptive applications was reported in [13][9][31]. In order to be adaptive, an application must be able to monitor its resource consumption and progress. Therefore, the RTOS should provide this monitoring information to the application. However, there are many different models of adaptive applications. There is the need for a deeper understanding of which model is more realistic and more suitable.

Proposition 9 Adaptive applications (how)
Researchers investigate adaptive real-time applications.

Proposition 10 Resource allocation monitoring (what)
The RTOS provides primitives for monitoring resource allocation.

To help overloaded applications, and to use the resources effectively, temporal resources, which are lost when not used, should be re-allocated to applications that temporarily experience overload. This implies that slack time is detected and reclaimed, and that the reclaimed slack time is made available to applications that need it. This re-allocation is best governed by the reservation contract. If the reservation contracts provide ranges instead of fixed parameters (QoS tolerance), automatic re-allocation techniques can be applied.

Proposition 11 Spare time (what)
Provide means for specifying use of spare time in the reservation specification.

Proposition 12 QoS tolerance (what)
Reservation specifications for temporal resources provide ranges instead of fixed parameters.

Finally, reservations should be renegotiated when appropriate. The renegotiation may be done by the application itself, or by a middleware agent. Therefore, the RTOS should provide a (re)negotiation interface that is separate from the standard API. Moreover, renegotiation is based on some knowledge of the application and system load. To keep options open, the RTOS should provide a monitoring interface that is separate from the standard API.

Proposition 13 Renegotiable reservation (what)
Allow renegotiation of service contracts.

Proposition 14 Separate (re)negotiation interface (what)
Separate the (re)negotiation interface from the standard RTOS API.

Proposition 15 Separate monitoring interface (what)
Separate the monitoring interface from the standard RTOS API.

Multiple resources

It is a well-known phenomenon: solutions that work well for a single shared resource break down in the presence of other shared resources. According to the ARINC standard [2] and Proposition 2, processing resources and memory-based resources should be local to the application in the case of a single CPU situation. [28] makes the case for a single unified strategy in the context of rate monotonic scheduling for distributed systems. We propose to do the same with the partitioning strategy. From this perspective, an application reserves not just a single virtual resource, but rather a complete virtual system. This is to a large extent uncharted territory.

Proposition 16 (what)
Provide resource partitioning (in space and/or time) and associated protection for applications as a unified strategy for all resources in a multi-resource environment.

Proposition 17 (how)
Investigate resource partitioning (in space and/or time) and associated protection as a unified strategy for all resources in a multi-resource environment.

Practical considerations

Smaller granularity allows tighter time constraints, but requires larger overhead cost, due to context switching and possibly cache flushing. Unfortunately, cache partitioning, as proposed for instance in [4], is not a feasible solution in many practical situations, since in these cases partitioning defeats the reason-of-being for the cache. In general, system overhead (which includes blocking, small priority inversions, interrupt handling costs, etc.) is a non-negligible factor for using resource reservation in practice. Interrupt handlers and non-preemptive RTOS services are notorious source of interference.

Proposition 18 Granularity (how)
Provide measures for weighing allocation granularity against the cost of context switching and cache flushing.

Proposition 19 Cache issues (how)
Investigate cache issues in the context of sharing the memory access path.

Proposition 20 System overhead (how)
Take system overhead into account when dimensioning a reservation (in the analysis phase).

Proposition 21 Interrupt handlers (how)
Account the cost of interrupt handling and RTOS services to the applications that effectively use them.

1. Concluding remarks

In this white paper, we presented reasons for adopting resource reservation techniques in standard real-time operating systems, a set of propositions on "what" and "how" should be provided at the operating system level and on the research issues that remain to be addressed. The intention is not to put any definitive word on these issues. On the contrary, our goal is to trigger a broad discussion between academic and industrial world on the steps that should be taken toward a new standard in real-time operating systems. Therefore, the proposed issues are probably far from being complete.

We encourage scientist, practitioners and operating system's designers to join the discussion with a critical frame of mind, by reporting experiences, problems and suggestions.

Bibliography

[1] "ARINC 651: Design Guidance for Integrated Modular Avionics". Airlines Electronic Engineering Committee (AEEC), 1991.

[2] "ARINC 653: Avionics Application Software Standard Interface". Airlines Electronic Engineering Committee (AEEC), 1996.

[3] "Integrity, the most advanced RTOS technology". Green Hills Software Inc. 2002.

[4] "VxWorks AE Datasheet." Wind River. MCL-DS-VAE-0111, 2001.

[5] A K. Mok, X. Feng. *Towards Compositionality in Real-Time Resource Partitioning Based on Regularity Bounds.* IEEE Real-Time Systems Symposium, 2001.

[6] C. W. Mercer, S. Savage, H. Tokuda. *Temporal Protection in Real-Time Operating Systems*. Proceedings of the 11th IEEE workshop on Real-Time Operating System and Software, 1994

[7] C.C. Wüst, W.F.J. Verhaegh, *Dynamic Control of scalable media-processing applications*. E.H.L. Aarts, J.M. Korst, W.F.J. Verhaegh, eds, Algorithms in Ambient Intelligence, Kluwer Academic Publishers, 2003

[8] D. Isovic, G. Fohler, L. Steffens. *Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions.* Euromicro Conference on Real-Time Systems, 2003.

[9] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. *Elastic Scheduling for Flexible Workload Management*. IEEE Transactions on Computers, Vol. 51, No. 3, March 2002.

[10] G. Buttazzo, M. Caccamo. *Minimizing Aperiodic Response Times in a Firm Real-Time Environment*. IEEE Transactions on Software Engineering, Vol. 25, No. 1, January/February 1999.

[11] G. Fohler, Tomas Lennvall, Radu Dobrin *A Component Based Real-time Scheduling Architecture.* Architectures for Dependable Systems, 2003. Springer Verlag, Editor(s):Rogerio de Lemos, Cristina Gacek, and Alexander Romanovsky

[12] G. Koren and D. Shasha. Skip-Over: *Algorithms and Complexity for Overloaded Systems that Allow Skips*. IEEE Real-Time System Symposium, 1995.

[13] G. Lipari, E. Bini. *Resource Partitioning among Real-Time Applications*. Euromicro Conference on Real-Time Systems, 2003.

[14] G. Lipari, G. Fohler, E. Bini, *A Framework for Composing Real-Time Schedulers*, Test and Analysis of Component based Systems, 2003.

[15] G. Lipari, J. Carpenter and S.K. Baruah. *A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments*. IEEE Real-Time System Symposium, 2000.

[16] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schütz. *An engineering approach to hard real-time system design*. European Software Engineering Conference, 1991.

[17] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, C. G. Plaxton. *A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems*. IEEE Real Time Systems Symposium, 1996.

[18] J. V. Busquets-Mataix, J. J. Serrano, A. Wellings. *Hybrid Instruction Cache Partitioning for Preemptive Real-Time Systems*. Euromicro Workshop on Real Time Systems, 1988.

[19] J.M. Ruiz Martínez. "A Hierachical and Decentralised QoS and Resource Management Architecture for Embedded Systems". PhD Thesis, Technical Univerisity of Madrid, 2002.

[20] K. Jeffay, F. D. Smith, A. Moorthy and J. Anderson. *Proportional Share Scheduling of Operating Systems Services for Real-Time Applications.* IEEE Real-Time Systems Symposium 1998.

[21] K. Ramamritham. *Allocation and Scheduling of Complex Periodic Tasks.* International Conference on Distributed Computing Systems, 1990.

[22] K.Jeffay, D.L. Stone and F.D. Smith, *Kernel support for live digital audio and video*. Computer Communications (UK) 15 (6) 1992.

[23] L. Abeni, G.Buttazzo. *Integrating Multimedia Applications in Hard Real-Time Systems.* IEEE Real-Time Systems Symposium, 1998

[24] L. Palopoli, L. Abeni, F. Conticelli, Marco Di Natale and Giorgio Buttazzo. *Real-Time control system analysis: An integrated approach*. IEEE Real Time System Symposium, 2000.

[25] L. Sha, R. Rajkumar, S. Sathaye. Generalized rate monotonic scheduling theory: a framework for developing real-time systems. Invited paper, Proceedings of the IEEE, vol. 82, 1 (1994).

[26] M. Caccamo, G. Buttazzo, and L. Sha. *Capacity Sharing for Overrun Control*. IEEE Real-Time Systems Symposium, 2000.

[27] P. Goyal, X. Guo and H. M. Vin. *A Hierarchical CPU Scheduler for Multimedia Operating Systems*. OSDI Symposium, 1996.

[28] R. Rajkumar, K. Juvva, A. Molano and S. Oikawa. *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems*. Real-Time Computing Systems and Application Workshop, 1997.

[29] R.J. Bril, C. Hentschel, E.F.M. Steffens, M. Gabrani, G.C. van Loo, and J.H.A. Gelissen, *Multimedia QoS in consumer terminals.* IEEE Workshop on Signal Processing Systems, 2001.

[30] S. K. Baruah, *Fairness in periodic real-time scheduling*, IEEE Real-Time Systems Symposium, 1995.

[31] S. Natarajan, ed. "Imprecise and Approximate Computation". Kluwer Academic Publishers, 1995.

[32] S. Saewong, R. Rajkumar, J. P. Lehoczky, M. H. Klein. *Analysis of Hierarchical Fixed-Priority Scheduling*. Proceedings of the 14th IEEE Euromicro Conference on Real-Time Systems June 2002.

[33] S. Sriram, S. Bhattacharyya, "Embedded Multiprocessors". Marcel Dekker, Inc. 2000.

[34] Z. Deng and J. W. S. Liu. *Scheduling Real-Time Applications in Open Environment*. IEEE Real-Time Systems Symposium, 1997.