Implementing Hierarchical Scheduling to Support Multi-Mode System

Rafia Inam*, Mikael Sjödin*, and Reinder J. Bril†
* Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
† Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands
rafia.inam@mdh.se, mikael.sjodin@mdh.se, r.j.bril@tue.nl

***Abstract*—Multi-mode embedded real-time systems exhibit a specific behavior for each mode and upon a mode-change request, the task-set and timing interfaces of the system need to be changed. Hierarchical Scheduling Framework (HSF) is a known technique to partition the CPU time into a number of hierarchically divided subsystems each consists of its own task set. We propose to implement a multi-mode system using a two-level HSF and provide a skeleton (framework) for an adaptive HSFs supporting multi-modes. Upon a mode-change request, the timing interface of each subsystem is changed, thus making the hierarchical scheduling adaptive in nature. We address the main goals for the implementation and describe the initial design details of the Multi-Mode Adaptive Hierarchical Scheduling Framework (MMAHSF) with the emphasis of doing minimal changes to the underlying kernel.**

***Keywords*-real-time systems; hierarchical scheduling**

I. Introduction

A wide spectrum of real-time embedded systems often have to support very different and changing application scenarios. Such systems is said to operate in multiple *modes* where each mode corresponds to a specific application scenario, are called *multi-mode systems (MMS)*. At a specific time, the system can be in one of the predefined modes and is switched from one mode to another upon some condition. A *mode switch mechanism* (or *mode change protocol*) is used to transform the system from one mode to another at runtime.

Normally, a different piece of software is executed for each mode, this means that a different task set, implementing different behaviour, is executed. The system can be viewed as a set of hierarchically organized subsystems, each subsystem consisting of its own set of tasks. For example in Figure 1, the system consists of two subsystems S_1 and S_2; which intern consists of task sets $\mathcal{T}_1 = \{\tau_{1,1}, \tau_{2,1}, \tau_{3,1}\}$, and $\mathcal{T}_2 = \{\tau_{1,2}, \tau_{2,2}\}$ respectively. The system supports two different modes M_1 and M_2, both having different timing properties. Both subsystems are active in both modes while the tasks can be active or deactive in a particular mode. *Active tasks* are those tasks that belong to the currently executing mode. *Deactive tasks* are those tasks that do not belong to the currently executing mode. The task $\tau_{1,1}$ is deactive in mode M_1, tasks $\tau_{2,1}$ and $\tau_{1,2}$ are deactive in M_2, while tasks $\tau_{3,1}$ and $\tau_{2,2}$ are active in both modes.

Hierarchical scheduling framework (HSF) [1] is a known technique used to partition a system into a set of subsystems, each consisting of its own set of tasks. Hence it is well-suited to implement a multi-mode system using the HSF. Both

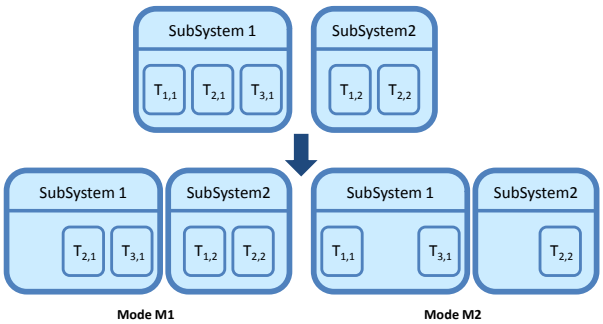

Fig. 1. An example of a Multi-Mode System (MMS)

MMS and HSF are known individually from both a scientific and an engineering perspective. Adaptive interfaces for the compositional analysis of hierarchical multi-mode systems are provided in [2] but no work has been found from an implementation perspective. We focus on implementing an MMS using the HSF technique and call it *multi-mode adaptive hierarchical scheduling framework (MMAHSF)*. In this paper we propose to use HSF to implement MMS and describe the system's main goals and challenges. We start with the design and implementation of a framework for MMAHSF with managing the mode-change request into an HSF implementation [3] to incorporate and to change system modes at runtime. A mode change protocol is to be implemented and its implementation issues/details are not part of the scope of this paper. Our goal is to provide a generic framework that can be used as a basis to instantiate a particular mode-change protocol.

A lot of work has been done on scheduling theory for multi-mode systems and on component-based multi-mode systems. Sha *et al.* [4] provided a simple mode switch protocol for a prioritized preemptive scheduling environment. A survey of mode change protocols for fixed-priority preemptive scheduling (FPPS) on a single processor is presented in [5] and along with proposed several new protocols. Multi-mode schedulability analysis is presented in [6], [7], [8], and added to the compositional system using Real-Time Calculus (RTC) in [2]. Some frameworks and programming languages support multi-mode systems, including [9], [10] and [11], [12], [13] respectively. Hang *et al.* [14] provides the details of mode switch logic algorithms to handle mode mapping for component-based systems. A mode-change protocol is implemented for reallocating the memory among tasks in [15] but no work has been done to reallocate the CPU time. To the best of our knowledge, no work has been found with respect to the MMS

implementation using hierarchical scheduling.

Paper Outline: Section II provides an overview of the HSF and its implementation on FreeRTOS that our work uses. Section III describes the support for MMS inclusion into the HSF implementation, and section IV concludes the paper with a description of ongoing and future work.

II. HSF and its Implementation

A two-level HSF is used to divide the system among a set of subsystems. In hierarchical scheduling, the CPU time is partitioned among many subsystems (or servers), that are scheduled by a global (system-level) scheduler. Each server contains its own internal set of tasks that are scheduled by a local (subsystem-level) scheduler. Hence a two-level HSF can be viewed as a tree with one parent node (global scheduler) and many leaf nodes (local schedulers). The global scheduler schedules subsystems. Each subsystem has its own local scheduler, that schedules the tasks within the subsystem.

Our implementation of MMS is based on a two-level HSF implementation for the FreeRTOS operating system [3]. FreeRTOS is a portable, open source (licensed under a modified GPL), mini real-time operating system developed by Real Time Engineers Ltd [16]. It is ported to 23 hardware architectures ranging from 8-bit to 32-bit micro-controllers, and supports many development tools. Its main advantages are portability, scalability and simplicity. The core kernel is simple and small, with a binary image between 4 to 9KB. Since most of the source code is in C language, it is readable, portable, and easily expandable and maintainable.

The official release of FreeRTOS only supports single level fixed-priority preemptive scheduling. However, a recent work has been presented that implements a two-level HSF for FreeRTOS [3] with associated primitives for hard real-time sharing of resources both within and between servers [17]. The HSF implementation supports two kinds of servers, idling periodic [18] and deferrable servers [19]. The implementation uses fixed-priority preemptive scheduling for both global and local-level scheduling. The HSF supports reservations by associating a tuple $\langle Q, P\rangle$ to each server where P is the server period and Q ($0 < Q \leq P$) is the allocated portion of P. Given Q, P, and information on resource holding times, the schedulability of a server and/or a whole system can be calculated with the methods presented in [17]. The implementation has been tested and experimental evaluations have been performed on a 32-bit AVR-based micro-controller board EVK1100 [20].

III. Multi-Mode Adaptive Hierarchical Scheduling Framework

HSF will become adaptive in nature by incorporating different modes of the system and using a mode change protocol to change the system-mode at run-time. An application may not only need to change a mode but also a different mode change protocol semantic. For example, consider an application that can be in startup, normal execution, emergency, and shutdown modes. The mode change from *normal to shutdown* can allow all the tasks to be completed before the mode is changed. While changing a mode from *normal to emergency* may require to abort all the tasks instantly. To address this issue, we are implementing a generic framework that can support multiple mode change protocols. The system will be instantiated in a particular mode and a particular mode change protocol. At run-time, both the system mode and the mode change protocol can be changed.

The mode change protocol is performed within a *Mode Change Request Controller (MCRC)*. To change the mode of the whole system in hierarchical scheduling, the mode-change has to be done at both global and local levels. Therefore, we use a *global MCRC* and a *local MCRC* as shown in Figure 2. The global MCRC is responsible for changing the mode of the whole system upon a mode-change request, and calls the local MCRCs to change the mode of each subsystem (i.e. change the task set). The local MCRC is responsible for handling the mode-change locally, hence change the mode of the subsystem. An event-triggered mode-change request initiates the process of mode-change. The request could be triggered either internally (e.g. deadline misses of tasks or too long execution of the idle task) or by external user request. Global MCRC is called upon a mode-change request to change the system's mode.

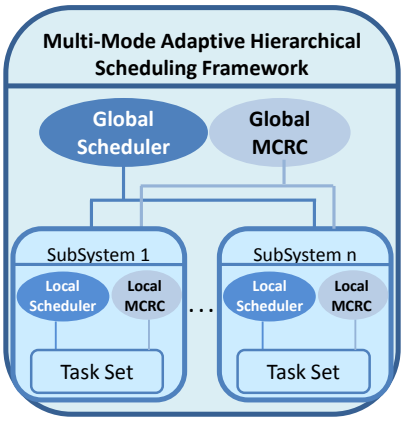

Fig. 2. Multi-mode adaptive hierarchical scheduling framework

We start our design and implementation with managing the mode-change request into our simple HSF implementation. Currently the implementation work of MMAHSF is on-going. We plan to execute, test, and evaluate our implementation on the EVK1100 board.

A. System model

We consider a two-level MMAHSF, in which a global scheduler schedules a system that consists of a set of subsystems $\mathcal{S}$ and a set of modes $\mathcal{M} = \{M_0, \ldots, M_{n-1}\}$ where n is the total number of modes in the system, and a global MCRC to change the system's mode. Each subsystem S_s consists of a local scheduler along with a set of tasks $\mathcal{T}_s$ and a local MCRC. Fixed-priority preemtive scheduling is used at both levels of schedulers. For each mode M_m, each subsystem S_s is specified by a different *timing interface* $I_{s,m} = \langle P_{s,m}, Q_{s,m}, p_{s,m}\rangle$ and a unique set of tasks $\mathcal{T}_{s,m} = \{\tau_{1_s,s,m}, \ldots, \tau_{n_s,s,m}\}$,

where $P_{s,m}$ is the period for that subsystem ($P_{s,m} > 0$), $Q_{s,m}$ is the capacity allocated periodically to the subsystem ($0 < Q_{s,m} \leq P_{s,m}$), and a unique priority $p_{s,m}$ in mode M_m.

B. Assumptions

The following assumptions are made for the initial design:

- All subsystems are active in all modes.
- The number of modes is fixed in the system and new modes are not allowed to be added at run-time. The period, budget, and priority of each subsystems for each mode are defined statically.
- A mode-change request will be served instantly. For scheduling we might need to delay the instantly serving of mode-change request, but we are not considering it in our current implementation.
- We will start by implementing *suspend-resume semantics* in which upon a mode-change request, all old-mode tasks are deactivated (suspended), and all new-mode tasks are activated (resumed). Further, all the events that occurred during the deactive state of the tasks, are delayed to be handled until their corresponding mode become active and starts execution. Tasks that belong to both (old and new) modes will remain active in the new mode.
- Resources that are shared among the tasks of the same subsystem are mode-specific; e.g. resources shared in mode M_1 are not shared with the tasks of mode M_2.

C. Design considerations

Here we present goals of adapting HSF with MMS that our implementation should satisfy:

1) **The use of HSF and the original FreeRTOS operating system:** User should be able to make a choice for using the MMAHSF or the original FreeRTOS scheduler (without HSF and MMAHSF).
2) **Consistency with the FreeRTOS kernel and keeping its API intact:** To get minimal changes and better utilization of the system, it will be good to match the design of the MMAHSF implementation with the underlying FreeRTOS operating system. This includes consistency from the naming conventions to API, data structures and the coding style. It will also increase the usability and understandability of MMAHSF implementation for FreeRTOS users.
3) The deactived tasks of a particular mode should not interfere with the execution of active tasks of that mode for better system performance.

D. Implementation details

Here we describe the initial design, implementation, and functionality details in adapting the two-level HSF with the MMS. The server type is idling periodic here.

1) The design of the scheduling hierarchy: Each subsystem S_s is reflected by its server. A server control block `subSCB` contains lists to store the server's interface $I_{s,m}$ and lists for tasks $T_{s,m}$ of each mode M_m. An idle task is created in each server. An idle server with an idle task is also created to setup the system. The motivations to use idle task and idle server are presented in [3]. The user will provide a total number of modes, an initial executing mode, and an initial mode change protocol (currently suspend-resume). The global scheduler is started by calling `vTaskStartScheduler()` (typically at the end of `main()` function), which is a non-returning function.

The global scheduler maintains a running server pointer and two prioritized lists to schedule servers: a ready-server list and a release-server list for hierarchical scheduling, containing all the active and inactive servers respectively [3]. The additional information to be stored here is the `current mode` of the system, and a total `number of modes` i.e. n in the system as shown in Figure 3. During execution, the system can be in one of these predefined modes $M_0, \ldots, M_{n-1}$. We also store `current mode change protocol`, and a total `number of mode change protocols`.

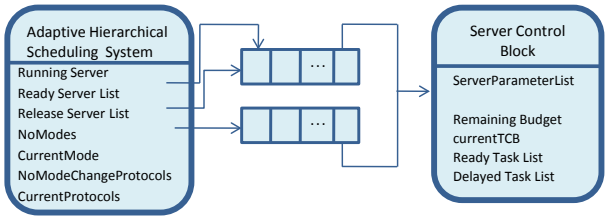

Fig. 3. Data structures for active and inactive servers, and system modes

The local scheduler schedules the tasks that belong to a server in a fixed-priority scheduling manner. For each server a `subSCB` contains all information needed by the server to execute in hierarchical scheduling and the queues for each mode of the system as presented in Figure 4.

Each server maintains a currently running task and two lists to schedule its tasks: a ready-task list, and a delayed-task list for each mode. Since the total number of modes are fixed in the system, the currently executing task and both lists for each mode are stored within an array, where the array index specifies a unique mode. Separate ready-task and delayed-task lists per mode are used to keep track of ready and delayed tasks respectively. The `currentTCB` pointer points to the currently executing task of the current system mode. During system execution, only the lists of the currently executing mode will be active in the system, and the lists of all the other modes are inactive (means not accessed). A `mode list` is added to the Task Control Block (TCB) containing the modes in which the task is active.

Server parameter list is also added to the `subSCB` to support the additional information about interfaces for each mode of each server. Since the servers may have different timing interfaces (periods, budgets, and priorities) in different modes, these three properties are now stored in an array structure i.e `server parameter list`; again whose index corresponds to a unique mode.

E. System functionality

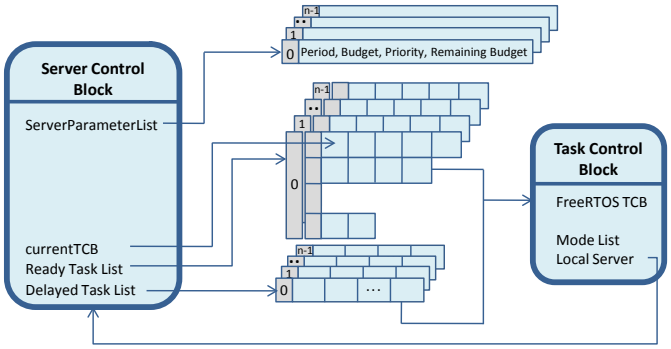

Fig. 4. Data structures for ready and delayed tasks, and server parameters for multi-mode system

1) The functionality of the tick handler: The tick handler is executed at each system tick. At each tick interrupt:

- The system tick is incremented.
- Upon a mode-change request, the global MCRC is called to handle the mode-change event.
- Check for the server activation events.
- The global and local schedulers are called to incorporate server events and task events respectively[1].

2) The functionality of the global and the local MCRC: The global MCRC executes a mode change mechanism upon receiving a mode-change request. It changes the timing interfaces for all the servers for the new-mode. The local MCRC is responsible for activating the lists belonging to the new-mode and for deactivating the lists belonging the old-mode. The active lists of the servers only include all those tasks that belong to the new-mode of the system. Tasks that belong (or active) in both the old and new mode are copied to lists of the new mode. In the end, the system's `current mode` is changed to the new-mode.

IV. Conclusions and Ongoing Work

In this paper, we have proposed a generic framework for the multi-mode adaptive hierarchical scheduling (MMAHSF) implementation, supporting multiple modes and multiple mode change protocol semantics, that can be instantiated in a particular mode change protocol. We have proposed to extend an HSF implementation to support MMS by keeping the original FreeRTOS API semantics. We have presented the system model, design consideration with initial design details, and main goals of the implementation to be achieved.

Ongoing work is to implement the MMAHSF in FreeRTOS for our target platform EVK1100. We will start by implementing suspend-resume semantics for the mode change protocol, in which all tasks that are inactive in the new mode are suspended, and all tasks that are active in the new mode are resumed. We also plan to support completion semantics for the mode change protocol in future, in which all tasks that are inactive in the new mode are allowed to complete their execution before the mode is changed.

[1]Their functionality remains the same as in [3]

Once the implementation efforts for the framework and mode-change protocol are complete and basic testing is performed, we measure/assess the performance of MMAHSF, like measuring overheads of MMAHSF w.r.t HSF, overheads of the global and the local MCRC etc. The next step will be to make the assumptions more flexible like adding new modes in the system dynamically at run-time, providing resource sharing among the tasks of different modes etc. We also plan to provide support for resource sharing among tasks of different subsystems in a MMAHSF.

References

[1] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *IEEE Real-Time Systems Symposium(RTSS'97)*, 1997.

[2] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multimode systems. In *Euromicro Conference Real-Time Systems*, 2010.

[3] R. Inam, J. Mäki-Turja, M. Sjödin, S. Ashjaei, and S. Afshar. Hierarchical Scheduling Framework Implementation in FreeRTOS. In *IEEE Conference on Emerging Technologies and Factory Automation*. IEEE Computer Society, 2011.

[4] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1:243–264, 1989.

[5] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161197, 2004.

[6] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Real Time Systems Symposium*, 1992.

[7] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *10th Euromicro Conference on Real-Time Systems*, 1998.

[8] V. Nelis, B. Andersson, J. Marinho, and S. M. Petters. Global-EDF scheduling of multimode real-time systems considering mode independent tasks. In *23rd Euromicro Conference on Real-Time Systems*, 2011.

[9] X. Ke, K. Sierszecki, and C. Angelov. COMDES-II: A component-based framework for generative development of distributed real-time control systems. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, 2007.

[10] E. Borde, G. Haik, and L. Pautet. Mode-based reconfiguration of critical software component architectures. In *Conference on Design, Automation and Test in Europe (DATE 2009)*, pages 1160–1165, 2009.

[11] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis and design language (AADL): An introduction. Technical Report, CMU/SEI-2006-TN-011, 2006.

[12] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A timetriggered language for embedded programming. In *PROCEEDINGS OF THE IEEE*, pages 166–184, 2001.

[13] J. Templ. TDL specification and report. Technical Report, Univ. of Salzburg, 2003.

[14] Hang Yin and Hans Hansson. A mode mapping mechanism for component-based multi-mode systems. In *4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2011.

[15] M. Holenderski, R. J. Bril, and J. J. Lukkien. Swift mode changes in memory constrained real-time systems. In *International Conference on Computational Science and Engineering*, 2009.

[16] FreeRTOS web-site. http://www.freertos.org/.

[17] R. Inam, J. Mäki-Turja, M. Sjödin, and M. Behnam. Hard Real-time Support for Hierarchical Scheduling in FreeRTOS. In *(OSPERT' 11)*, pages 51–60, 2011.

[18] L. Sha, J.P. Lehoczky, and R. Rajkumar. Solutions for some Practical problems in Prioritised Preemptive Scheduling. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 181–191, 1986.

[19] J.K. Strosnider, J.P. Lehoczky, and L. Sha. The deferrable server algorithm for Enhanced Aperiodic Responsiveness in Hard Real-time Environments. *IEEE Transactions on Computers*, 44(1), 1995.

[20] ATMEL EVK1100 product page. http://www.atmel.com.