5G-09

Efficient User Space Scheduling Library for FreeRTOS

Robin Kase † Thiem Van Chu † Kenji Kise †

† Graduate School of Information Science and Engineering, Tokyo Institute of Technology

1 Introduction

At present, there is a gap between practical implementations of task scheduling on popular Real-Time Operating Systems (RTOS) and theoretical real-time scheduling. This is due to the overhead required when the RTOS offers advanced features. Moreover, implementing advanced scheduling features consumes vast amount of development time. With a real-time scheduler library implemented in user space, the user can choose whether to skip the overhead, or use more advanced theories. At the moment, there are already several scheduling frameworks for FreeRTOS. However, they either do not provide advanced scheduling policies, or require high scheduling overhead.

This paper proposes a low overhead task scheduling library for FreeRTOS (LOTFree) implemented in user space that supports periodic tasks, dependable Timing-Error-Detection, Rate-Monotonic Scheduling (RMS) policy, and Deadline-Monotonic Scheduling (DMS) policy. LOTFree supplies theoretical real-time scheduling features to speed up development of complex projects, and make FreeRTOS friendlier to students who have newly studied real-time scheduling.

2 FreeRTOS

FreeRTOS is an open source RTOS that has been ported to many microcontrollers. It is a microkernel mainly written in C which requires low memory footprint and low overhead. Since the kernel is designed to be simple and small, it does not contain advanced OS features such as file system and IO support. Instead, these features are supported by library extensions.

The default scheduling mechanism in FreeRTOS is implemented inside the tick interrupt. The tick interrupt checks if the currently running task has highest priority. If a task with higher priority is ready, context switch is issued if preemption is enabled. Tasks with equal priorities are scheduled in round-robin manner if time slicing is enabled.

3 Related Work

Heterogeneous Scheduler Task (HST)[1] is an existing open source scheduler for FreeRTOS that implements advanced scheduling policies such as Earliest Deadline First (EDF) in the user space, and supports non-periodic tasks as sporadic tasks and aperiodic tasks, which are scheduled with background execution. However, its implementation requires slightly more than twice as much overhead as FreeRTOS native scheduler for context switches[1]. The overhead is mainly caused by the scheduler task.

Fig. 1(a) illustrates the context switches between

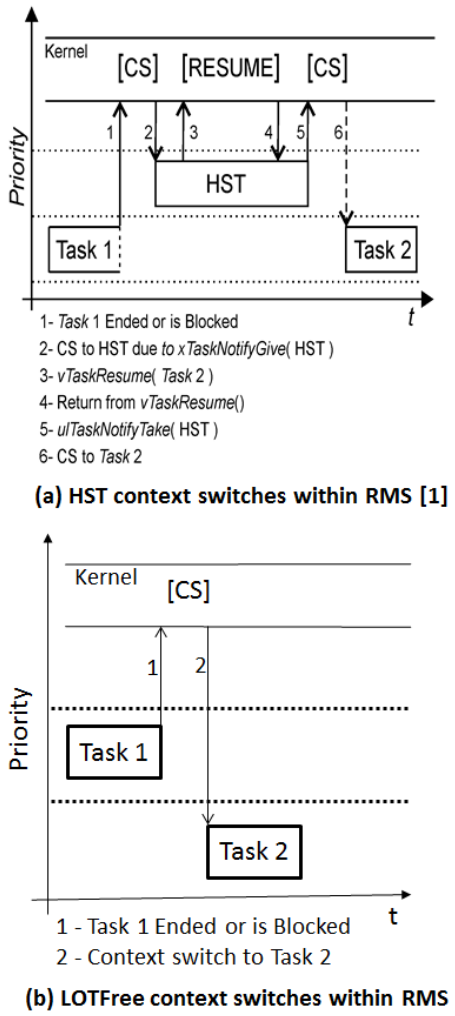

Figure 1: HST and LOTFree context switch examples

tasks in HST for any kind of scheduling policy, including RMS. When Task 1 has finished its execution or is blocked, a context switch to the scheduler task occurs. The scheduler task then unsuspends Task 2 and blocks itself. All other tasks remain suspended, and the kernel will therefore make context switch to Task 2. The scheduler task is also switched in during each tick interrupt to check whether the current user task should continue its execution, or another user task should get CPU burst, based on the scheduling policy. In this way, the scheduler task controls which task will get CPU burst based on the scheduling policy.

As the scheduler task is switched in each tick interrupt, task release, task completion, task block, and task suspension, flexibility for implementing different scheduling policies, especially dynamic priority

scheduling policies as EDF, is provided. However, simple scheduling policies that should not require large overhead is forced to require overhead nearly as much as complex scheduling policies. Moreover the context switches to the scheduler task are too frequent.

Another disadvantage of HST is that the Timing-Error-Detection lacks handling of cases when the deadline counter or/and tick counter overflows. This can lead to misdetection of Timing-Error when deadline counter overflows, and Timing-Error may not be detected when tick counter overflows.

4 Proposal of LOTFree

LOTFree has a scheduler task which is a periodic task with highest priority. Thus, the scheduler task will not be switched in for each tick interrupt, or before context switches between user tasks. It will only be switched in once during each period that can be set by the user, and will therefore reduce the scheduler overhead drastically compared to HST. At the moment, the scheduler task is only used for Timing-Error-Detection. However, it will be more active in features that are planned to be implemented in future work.

Since RMS and DMS are fixed priority scheduling policies, LOTFree calculates and assigns priorities to each task with kernel Application Programming Interface (API), before FreeRTOS native scheduler starts when using these policies. The native scheduler will then handle all context switches to tasks according to their priorities. Thus, FreeRTOS native scheduler alone is enough for these scheduling policies, and the scheduler task is not needed for RMS and DMS if Timing-Error-Detection is disabled in LOTFree. Fig. 1(b) illustrates the absence of additional overhead during context switches between tasks for fixed priority scheduling policies as RMS or DMS in LOTFree.

Dynamic priority scheduling policies can be implemented by letting the scheduler task calculate new priorities according to the policy, and assign them to user tasks each time the scheduler task is released. The kernel will then schedule all tasks according to their priorities until next time scheduler task period where all user task priorities are recalculated again. As the scheduler task period can be set by the user, context switches to the scheduler task can be adjusted to any desired frequency.

There are some configurations of FreeRTOS that must be considered by the user when using LOTFree. Preemption, time slicing and tick hook must be enabled, and the number of priorities available to tasks should be configured to number of tasks + 1 (for the scheduler task), so that each task can have its own priority.

Two types of Timing-Error-Detection are provided in LOTFree, one for detecting tasks that have exceeded their maximal execution time, and the other for detecting tasks that have missed their deadline. An example scenario of maximal execution time excess handling is described in Fig. 2. If maximal execution time excess of a task is detected, it will be suspended by the scheduler task until the next period of the suspended task. An example scenario of deadline miss handling is described in Fig. 3. If deadline miss of a task is detected, it will be deleted and recreated with phase set to start of its next period by the scheduler task. These two Timing-Error-Detections can also be programmed to take care of thinkable cases when the tick counter or/and deadline counter overflows.

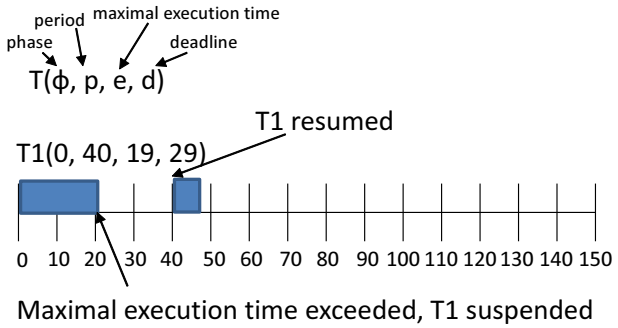

Figure 2: Timing-Error-Detection handling excess of maximal execution time.

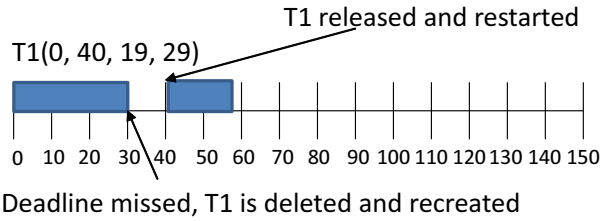

Figure 3: Timing-Error-Detection handling deadline miss.

5 Conclusion & Future Work

A new scheduling library for FreeRTOS LOTFree is proposed. LOTFree requires less overhead than HST, since LOTFree does not need to switch in the scheduler task as frequently as HST. The user can program periodic tasks with simpler interface with LOTFree. Finally, the Timing-Error-Detection in LOTFree is also designed to take care of cases when the tick counter or/and deadline counter overflows.

LOTFree is planned to be extended with features as EDF scheduling policy, Priority-Inheritance protocol, and Periodic Server for scheduling non-periodic tasks, and will be distributed with open source license. The overhead of LOTFree will be evaluated, and the Timing-Error-Detection functionalities will be tested for extreme situations when the tick counter or/and deadline counter overflows.

References

[1] Francisco E Paez, Jose M Urriza, Ricardo Cayssials, and Javier D Orozco. Freertos user mode scheduler for mixed critical systems. In *Sixth Argentine Conference on Embedded Systems (CASE)*, pp. 37–42. IEEE, 2015.