14th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 24-26, 2018

Performance Analysis of Tasks Synchronization for Real Time Operating Systems

Ioan Ungurean[1,2], Nicoleta Cristina GAITAN[1,2]
[1]Faculty of Electrical Engineering and Computer Science, Stefan cel Mare University of Suceava
[2]Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD)
Suceava, Romania
ioanu@eed.usv.ro, cristinag@eed.usv.ro

***Abstract*—Real time operating systems (RTOS) have a growing importance in the development of embedded projects based on microcontrollers (MCUs). They allow the development of multitasking applications providing deterministic behavior and basic services for inter-task communication. When it is selected an RTOS, it must take into account several criteria such as: license costs, memory footprint, predictability, and latency in handling of the critical operations triggered by internal or external events. This latency can also be influenced by the time required for task context switching operation. In this paper, we compare the time for task context switching in the case of four RTOSs used on ARM Cortex™-M based microcontrollers: FreeRTOS, uC-OS/II, Keil RTX, and RT-Thread. For these RTOSs, the time for task context switching is measured if synchronization is performed through events, semaphores and mailboxes. The tests are performed on ARM Cortex™-M4 and ARM Cortex™-M0+ based MCUs.**

Keywords—real time; operating systems; task synchronization; microcontrollers

I. Introduction

The real time operating systems (RTOS) are used to design and develop hard and soft real time applications [1]. This system must respond as quickly as possible to the external/internal events. Furthermore, these systems must be predictable, deterministic, reliable, fault tolerant and comply with the imposed deadlines [1] [2].

The software design and development for simple embedded systems can be based on the super-loop concept [3] in which, functions are executed in a predefined order within an infinite loop. This concept cannot be used to develop applications that are more complex because they are very difficult to extend and debug. The disadvantages of the super-loop concept are solved by RTOSs [1] [3] that allow modularization of the applications by splinting into tasks that are executed only when are scheduled, allowing an efficient use of resources provided by microcontrollers.

RTOS systems include a kernel that provides services for task management, communication mechanisms, and task synchronization mechanisms. Examples of RTOS include uC-OS/II, KEIL RTX, FreeRTOS, RT-Thread, eCOS, LynxOS, QNX, VxWorks, OSEK, etc. These RTOS are executed on the microcontrollers and are used for the development of multitasking applications with hard real time requirements. These RTOS do not use the virtual memory concept. Linux, Linux variants, or Windows operating systems can not run on microcontrollers due to resource requirements. These operating systems can run on microprocessors that support virtual memory. Furthermore, the RTOS systems are used for the development of devices that are integrated into IoT (Internet of Things) applications [4].

In order to compare the RTOSs, multiple features must be considered. The most important parameter is worst case execution time (WCET) for the execution of a task and of an interrupt service routine. In [5] it is specified that the most important parameter for an RTOS is the maximum amount of time during which the interrupts are disabled. Others important parameters of an RTOS are latency for handle the external/internal events, jitter for event handling, and time for task context switching [1]. Another important feature is modularity and scalability by which unnecessary services can be disabled to obtain a lower memory footprint. Unlike regular operating systems, RTOSs are designed for small memory systems.

Examples of applications that are developed with RTOSs are [1]: Internet of Things, Industrial Internet of Things [4], automotive applications [6], medical systems [7], robotics, military systems [8], avionics [8], telecommunication systems, industrial automation [9], and flight control systems. The most of real time systems applications comply with a combination of hard real time requirements for critical operations and soft real time requirements for noncritical operation.

In this paper, we measure and compare the time for task context switching in the case of four RTOSs if synchronization between two tasks is performed through events, semaphores and mailboxes. The tests are performed on ARM Cortex™-M4 and ARM Cortex™-M0+ based MCUs using the following RTOSs: FreeRTOS, uC-OS/II, Keil RTX, and RT-Thread.

This paper is structured as follows: Section II presents the RTOSs used for the tests and a study related the embedded systems, Section III describes the test applications, and in

Section IV are present the experimental results. The conclusions are drawn in Section V.

II. Real Time Operating System for Small Microcontrollers

In every year, EETimes.com and Embedded.com publish a market study related to the embedded systems [10]. According with the last market study [10], 68% from the ongoing embedded projects are designed around of an embedded operating system (RTOS, kernel, etc.). This percentage is maintained over the past five years, with minor differences, and those that do not use an operating system confirmed that they are not needed. From the ongoing projects based on OS/RTOS, 41% use an open source solution without commercial support (the trend is growing over past five years), 30% use a commercial solution (the trend is decreasing over past five years), 17% use an in-house solution, and 12% use an open source solution with commercial support [10].

The most used operating systems [10] are Embedded Linux (22%), followed by FreeRTOS (20%) and in-house solution (19%). Other RTOS (excepting Windows, Android, Linux solutions) for small microcontroller are: Texas Instruments RTOS (5%), Texas Instruments DSP/BIOS (5%), Micrium uC-OS/III (5%), Keil RTX (4%), Micrium uC-OS/II (4%), Wind River VxWorks (4%). We can observe that the most used RTOS for small microcontrollers is FreeRTOS (it is considered that Embedded Linux is not used in small microcontrollers).

A. FreeRTOS

FreeRTOS [11] is an open source RTOS. According with [10], it is the most used RTOS on project for small microcontrollers ongoing in 2017. It supports multitasking and provides services for inter task communication and synchronization mechanisms such as semaphores, mutexes, event group, mailbox, and software times. FreeRTOS uses a preemptive scheduling and was designed for small microcontrollers with small data and code memory. In FreeRTOS, more tasks can have the same priority. In this case, the scheduler will use a round robin scheduling scheme for the levels of priority with more tasks. OpenRTOS is the version of the FreeRTOS with commercial support and license provided by WITTENSTEIN Group [12].

B. Micrium uC-OS/II and Micrium uC-OS/III

Micro-Controller Operating Systems (uC-OS) [5][13] is a commercial RTOS provided by Micrium (subsidiary of Silicon Labs). The license is provided per product or per product line and was designed for time critical real time applications. It supports multitasking and services for mutexes, semaphores, events, mailboxes, and message queue. The difference between uC-OS/II and Micrium uC-OS/III is that the last support more task with the same priority level using a round robin scheduling policy. It support a large number of CPU architectures such as: ARM7-9-11/Cortex™-M1-3-4-A8/9, MSP430, PowerPC, etc.

C. Keil RTX

RTX [14] is a RTOS provided with source code by KEIL under royalty-free license. It supports multitasking and services for mutexes, semaphores, events, mailboxes. It provides round-robin, preemptive and non-preemptive (collaborative) scheduling policies. The support for debug RTX is integrated in the MDK-ARM tools. Furthermore, it implements the CMSIS-RTOS API standardized by the ARM. Because the source code is provided with MDK-ARM tools, it is exclusively used in the development of the ARM Cortex™-Mx processor-based devices.

D. RT-Thread

RT-Thread [15] is an open source RTOS provided by the RT-Thread Development Team that is designed for MCU with small memory. It supports a large number of CPU architectures such as [16]: ARM Cortex™-Mx, AVR32, MIPS, x86, and ARM Cortex™-A8/A9. It supports preemptive scheduling and round-robin policy for the threads with the same priority level. It provides services for inter-thread communication and synchronization mechanism such as semaphores, mutexes, mailbox, message queue, and events.

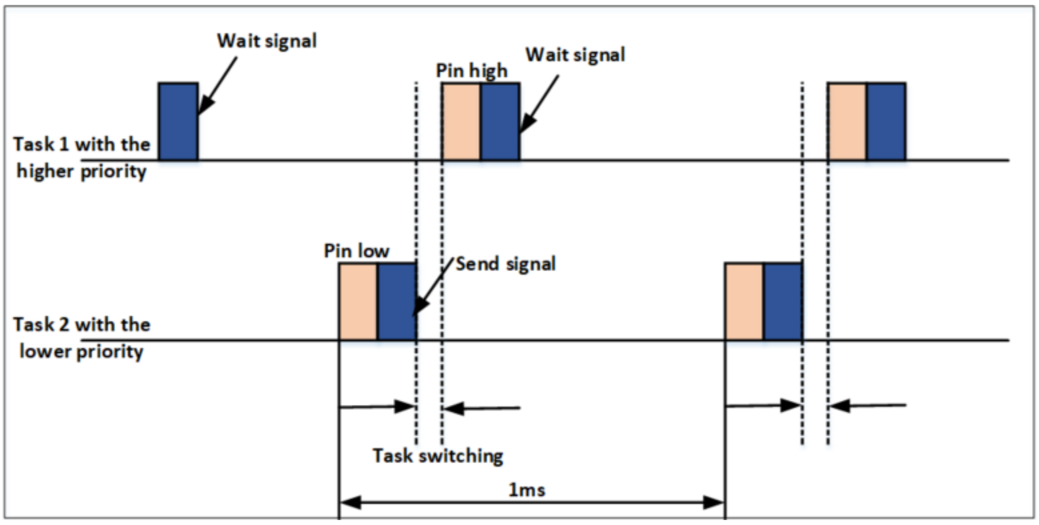

Fig. 1. Time diagram for the tasks of test applications

III. The Test Applications

In this paper, we want to test the performances of task context switching in case of the four RTOSs: FreeRTOS, uC-OS/II, Keil RTX, and RT-Thread. The test applications consist of two tasks/threads with distinctive priorities in order to use only the preemptive scheduling scheme. The highest priority task expects a synchronization mechanism (event / semaphore / mailbox) in an infinite loop (it will be in the waiting state most of the time). When the expected synchronization mechanism occurs, the task sets to high the pin used for the test and the task goes back in the waiting state for the synchronization mechanism. The lower priority task, at every 1ms, sets to low the pin used for the test and performs the post operation for the synchronization mechanism waited by the higher priority task. This operation determines a task context switching operation performed by the RTOS scheduler, and the switching time can be determined by measuring the total time while the test port is on low level. Because the application has only two tasks and there are no active interrupts that can interfere with the task context switching, the jitter will be negligible. The time diagram of the two tasks is presented in Fig. 1.

IV. Experimental Results and Discussions

In order to test the task context switching performance of the RTOSs mentioned in Section III, for each RTOS, there are developed different applications: one for synchronizing through an event, one for synchronizing via a binary semaphore and one for synchronizing through a mailbox. For the tests, two development kits have been used: KEIL MCBSTM32F400 with STM32F407IG ARM Cortex™-M4 MCU and STM32 NUCLEO-L053R8 with STM32L053R8 ARM Cortex™-M0+ MCU. For software development and debug, the MDK-ARM Professional 5.24 toolchain have been used. For the first development kit, the system clock of the MCU was configured for a working frequency at 168MHz and for the second development kit the system clock of the MCU was configured for a working frequency at 32MHz. The internal time tick for each RTOS was configured for a period of 1ms. Furthermore, the RTOSs have been configured to use a fully preemptive scheduling (they do not use multiple tasks on the same priority level).

For FreeRTOS, the xEventGroupWaitBits and xEventGroupSetBits services were used to synchronize the tasks through an event, xTaskNotify and xTaskNotifyWait services to synchronize the tasks via a mailbox, and the xSemaphoreGive and xSemaphoreTake services to synchronize the tasks through a semaphore. For Keil RTX, thee osSignalSet and osSignalWait services were used to synchronize the tasks through an event, osMailPut and osMailGet services to synchronize the tasks via a mailbox, osSemaphoreRelease and osSemaphoreWait services to synchronize the tasks through a semaphore. For uC-OS/II, the OSFlagPost and OSFlagPend services were used to synchronize the tasks through an event, OSMboxPost and OSMboxPend services to synchronize the tasks via a mailbox, and OSSemPost and OSSemPend services to synchronize the tasks through a semaphore. For RT-Thread, the rt_event_send and rt_event_recv services were used to synchronize the tasks through an event, rt_mb_send and rt_mb_recv services to synchronize the tasks via a mailbox, and rt_sem_release and rt_sem_take services to synchronize the tasks through a semaphore. Fig. 2 presents the oscilloscope print screen obtained by measuring the signal from the test pin if the task synchronization is achieved by a FreeRTOS event on the STM32F407IG ARM Cortex ™ -M4 MCU.

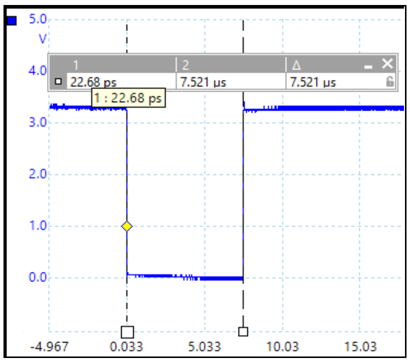

Fig. 2. Time for FreeRTOS task context switching on STM32F407IG ARM Cortex™-M4 for synchronization through an event

Fig. 3 presents all results obtained for STM32F407IG ARM Cortex™ -M4 MCU. It can be observed that the best performances are achieved by Keil RTX RTOS and the lowest performances are achieved by FreeRTOS. In case of FreeRTOS, the lowest latency for task context switching is achieved in the case of synchronization through a mailbox, and in the case of the other three RTOSs, the lowest latency is achieved if the synchronization is through a semaphore. It was expected that the lowest latency would be achieved in case of synchronization through an event. This does not happened because RTOSs provide synchronization services on a group of events and not a single event resulting in greater overhead. In this case, groups of 32 events were used and the synchronization was performed on a single event from the 32. Fig. 4 presents all results obtained for STM32L053R8 ARM Cortex™-M0+ MCU. It can be observed that the same performance differences are maintained as in the previous case (it must be taken into account that in this case the working frequency is lower and that way it is used a MCU based on ARM Cortex™ -M0+). Regarding the measured values, the measurements errors are generated by the oscilloscope (PicoScope 2205MSO – that provides a vertical resolution up to 12 bits and a time base accuracy of ±100 ppm). The jitter for task context switching is not present, because the task that waits a synchronization directive has the highest priority task from the systems.

When an RTOS is selected to develop an embedded application, it should not be considered only the time for task context switching. Other features such as predictability, compliance with the required deadlines, support, available documentation, compatibility with previous projects, licensing costs, and certifications (such as certifications for safety-critical application) should be considered. For these reasons, FreeRTOS is the most used RTOS for embedded systems although it has the longest time for task context switching. When an RTOS is selected, it is very important the field/environment in which the embedded system will be used.

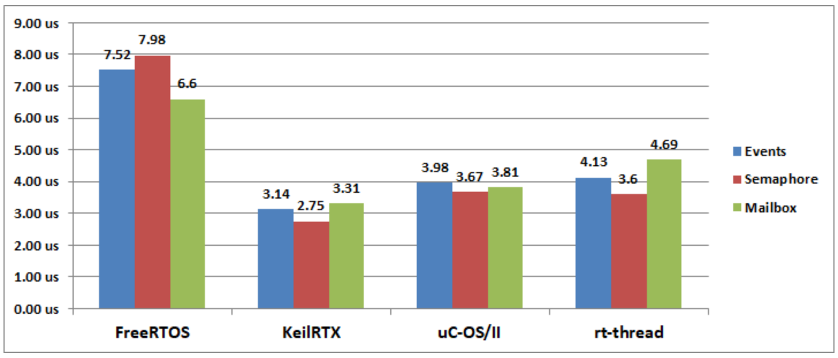

Fig. 3. Time for task context switching on STM32F407IG ARM Cortex™-M4 MCU

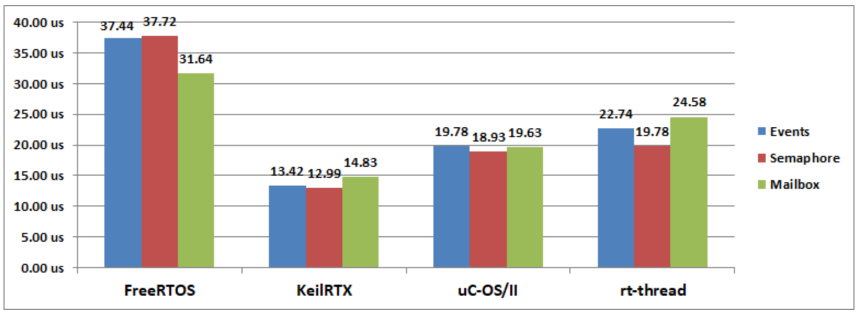

Fig. 4. Time for task context switching on STM32L053R8 ARM Cortex™-M0+ MCU

V. Conclusions

In this paper, it was presented a comparison in terms of task context switching time for four RTOSs (FreeRTOS, uC-OS/II, Keil RTX, and RT-Thread). For each RTOS, we developed an application that triggers the task context switching through an event, a semaphore, and a mailbox. By means of a test pin, it was possible to measure the time for task context switching. The tests were performed on ARM Cortex™-M4 and ARM Cortex™-M0+ based MCUs. It has been noticed that the FreeRTOS is the most used RTOS but it has the highest context switching time. The best performances were obtained for Keil RTX. Although FreeRTOS has the largest latency, it is a good choice due to predictability, deterministic behavior, documentations accessibility, compatibility with previous projects, and open sources license.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI-UEFISCDI, Contract no. 220PED⁄2017, PN-III-P2-2.1-PED-2016-1473, within PNCDI III.

References

[1] G. C. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer Science & Business Media, 2011.

[2] S. Daniel, A. Seuret, and O. Sename, "Real-time control systems: feedback, scheduling and robustness." International Journal of Systems Science 48.11 (2017): 2368-2378.

[3] K. C. Wang, "Models of Embedded Systems, "Embedded and Real-Time Operating Systems. Springer, Cham, 2017. 95-111.

[4] I. Ungurean, N. C. Gaitan, and V. G. Gaitan. "A Middleware Based Architecture for the Industrial Internet of Things." KSII Transactions on Internet & Information Systems 10.7 (2016).

[5] J. J. Labrosse, "MicroC/OS II: The Real Time Kernel", KA, Lawrence:CMP Books, 2002.

[6] C. Dietrich and D. Lohmann, "OSEK-V: application-specific RTOS instantiation in hardware." Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. ACM, 2017.

[7] T. Carpenter, J. Hatcliff, and E. Y. Vasserman. "A Reference Separation Architecture for Mixed-Criticality Medical and IoT Devices." Proceedings of the 1st ACM Workshop on the Internet of Safe Things. ACM, 2017.

[8] E. Fedosov, I. Koverninsky, A. Kan, V. Volkov, and Y. Solodelov, "Use of real-time operating systems in the integrated modular avionics." Procedia Computer Science 103 (2017): 384-387.

[9] I. Ungurean and N. C. Gaitan. "Monitoring and control system for smart buildings based on OPC UA specifications." Development and Application Systems (DAS), 2016 International Conference on. IEEE, 2016.

[10] ***, "2017 Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments", https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

[11] ***, "Why RTOS and What Is RTOS?", https://www.freertos.org/about-RTOS.html

[12] ***, "OPENRTOS, part of embedded FreeRTOS – OpenRTOS – SafeRTOS family", https://www.highintegritysystems.com/openrtos

[13] ***, Real-Time Kernels: µC/OS-II and µC/OS-III, https://www.micrium.com/rtos/kernels/

[14] ***, RTX v5 Implementation, http://arm-software.github.io/CMSIS_5/RTOS2/html/rtx5_impl.html

[15] ***, "RT-Thread, RTOS", https://www.rt-thread.org/

[16] ***, "RT-Thread github homepage", https://github.com/RT-Thread/rt-thread