2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)

TZDKS: A New TrustZone-based Dual-Criticality System with Balanced Performance

Pan Dong[12] Alan Burns[1] Zhe Jiang[1] Xiangke Liao[2]
[1] Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5GH, UK
[2]School of Computer, National University of Defense Technology, Changsha, Hunan Province, P.R.China

***Abstract*—Many mixed-criticality systems are composed of a RTOS (Real-Time Operating System) and a GPOS (General Purpose Operating System), and we define them as mixed-time-sensitive systems. Complexity, isolation, real-time latency, and overhead are the main metrics to evaluate such a mixed-time-sensitive system (MTSS). These metrics may conflict with each other, so it is difficult for them to be consistently optimized. Most existing implementations only optimize part of the above metrics but not all.**

As the first contribution, this paper provides a detailed analysis of performance influencing factors which are exerted by various runtime mechanisms of existing MTSSs. We figure out the difference in performance across system designs, including task switch, memory management, interrupt handling, and resource isolation. We propose the philosophy of utilizing TrustZone characteristics to optimize various mechanisms in MTSS.

The second contribution is to propose a TrustZone-based solution - termed TZDKS - for MTSS. Appropriate utilization of TrustZone extensions helps TZDKS to implement (i) virtualization environment for GPOS and RTOS, (ii) high efficient task switch, memory access, interrupt handling and device access which are verified by experiments. Therefore, TZDKS can achieve a full-scale balance amongst aforementioned metrics.

I. Introduction

Recently, many applications require integrating components with different levels of criticality on one physical platform, in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation and power consumption. This kind of system is defined as a mixed-criticality system [5]. The most common case is that a real-time system and a non-real-time interactive system are mixed and integrated on one platform, which is defined as a mixed-time-sensitive system (MTSS) in this paper, also deemed as a special Dual-Criticalitiy System [4].

The performance of MTSS is determined by many metrics, such as complexity, isolation [9], real-time latency, and overheads (of merging different OSs). These metrics may conflict with each other, so can hardly be consistently optimized. For example, isolation and complexity collide with performance or overhead. There are many approaches to design and implement MTSS, which can be classified as two sorts. The traditional way is to extend popular GPOS, e.g, Linux. This method usually deploys a small real-time kernel at the underlying of GPOS, and takes GPOS as a pseudo real-time task. We call it a dual-kernel system [11]. A dual-kernel system does not require extra hardware support, and only introduces a low overhead [11]. However, it needs to modify the GPOS kernel heavily, which results in high cost in complexity and flexibility. Additionally, insufficient isolation between OSs leads to many security and reliability problems [15]. As an instance, Linux often goes down because of a bug in a device driver, and the same bug may also lead to the whole system's failure in Xenomai [11]. In contrast, virtualization-based method becomes a more popular and rapid method to design a MTSS through integrating RTOS and GPOS in two virtual machines. This method can provide better security isolation and lower complexity, so it has the advantages of simple development and ideal isolation. However, both OSs suffer from high overhead and remarkable decrease of executing performance. The hypervisor must be redesigned to meet the real-time requirement. Moreover, it heavily relies on hardware supports, which increases the cost of the whole system [12].

The TrustZone technology, which is developed to provide a trusted executing environment, has attracted our attention. With the hardware isolation support, a GPOS may run on the TrustZone-enabled CPU without modification, which leads to a low development cost. Furthermore, as a light-weight isolation scheme, TrustZone introduces a small overhead in software. Therefore, its characteristics do help to develop a MTSS with all-round balance amongst the metrics we focus.

As the first contribution, this paper provides a detailed analysis of performance influencing factors which are exerted by various runtime mechanisms of existing MTSSs. We figured out the difference in efficiency across system designs such as task switch, memory management, interrupt handling, and resource isolation. We propose the philosophy of utilizing TrustZone characteristics to optimize various mechanisms in MTSS. The second contribution of the paper is to propose a Trustzone-based solution for MTSS, termed TZDKS (TrustZone-based Dual-Kernel System). Appropriate utilization of TrustZone extension helps TZDKS implement (i) virtualization environment for GPOS and RTOS, (ii) high efficient task switch, memory access, interrupt handling and device access which are verified by experiments. Therefore, TZDKS achieves a full-scale balance among aforementioned metrics. We believe that our TZDKS is a safe and low-cost solution as the TrustZone-build-in ARM platforms have been used in almost all engineering fields.

The paper is organized as follows: Section II introduces related work; Section III gives the design philosophy; Section IV describes the TZDKS implementation; Section V evaluates the performance of TZDKS, with conclusions offered in Section VI.

2325-1301/18/$31.00 ©2018 IEEE
DOI 10.1109/RTCSA.2018.00016

II. RELATED WORK

A. Two Common Solutions for Integrating Embedded System

A dual-kernel MTSS introduces a small real-time kernel into the underlying of GPOS, and takes GPOS as a pseudo real-time task. RTOS has a higher priority than GPOS, and consequently GPOS only runs during the idle periods of RTOS. That is to say, when the IDLE task is switched on, a switcher module will be invoked to save the state of RTOS, and restores the state of GPOS, then RTOS will be activated as the timer (belongs to RTOS) interrupting GPOS, and will do rescheduling for the real-time tasks. So this is called as idle-scheduling strategy. RTLinux, RTAI, Xenomai and RTThread [6] [11] are products of dual-kernel system, and are widely applied in industrial systems.

In a virtualization-based MTSS, a hypervisor may be used to manage shared resources and isolate the OSs, and a GPOS can execute aside a RTOS in two virtual machines (VM). The up-to-date avionics systems specification - ARINC 653 [14] - is a typical example. This specification requires integrating many subsystems (such as flight control system, environment control system, and amusement system) into a virtualized platform on modern aircraft. Virtualization has also been developed in the IO system of a MTSS. I/O virtualization [18] [17] [10] enables time and space multiplexing of I/O devices, by mapping multiple logical I/O devices upon a smaller number of physical devices. More than just provide more device ports, this technology can also reduce the software overhead and enhance the I/O performance and timing predictability.

These two MTSSs always behave oppositely in many aspects, and detailed analysis will be presented in section III.

B. TrustZone and TrustZone-based virtualization

ARM TrustZone [16] is a hardware-based security extension technology incorporated into ARM processors. It enables a single physical processor to execute instructions in one of two operating worlds: the normal world (NW) and the secure world (SW). The isolation mechanisms are well defined. Access permissions are strictly under the control of SW, which forbids access of secure resources from NW. As the processor only runs in one world at a time, to enter the other world requires context switch via a special instruction called the Secure Monitor Call (SMC). In order to facilitate the application development, the GlobalPlatform consortium develops the TEE client API specification [8].

The idea of using TrustZone as a virtualization technique in embedded systems was first introduced by Frenzel et al [7]. TrustZone extensions help to virtualize a system in two ways:

(1) Use system access capabilities of the secure world to build a hypervisor that can control VMs running in NW. SierraVisor is an example of such way.The SierraVisor Hypervisor [3] leverages hardware security extensions of TrustZone to run multiple, high-level operating systems concurrently. The guest operating systems (OS) are aware of the fact that they are running on top of a hypervisor, so minor modifications must be made to the guest OS. Guest kernel and applications run in their usual privilege modes respectively. Furthermore, each guest executes in an isolated environment with low overhead.

(2) Use the efficient switch mechanism of the Secure Monitor to host a dual-OS system (Secure OS and Normal OS). Most TrustZone-based virtualization systems [13] [15] are constructed in this way. SafeG [15] is designed to concurrently host a RTOS and a GPOS on TrustZone-enabled ARM SoC devices. SafeG takes advantage of TrustZone security extensions to provide full system access to trusted software, and limit the capabilities of software running in the normal world.

III. BALANCING DESIGN PHILOSOPHY OF TZDKS

A. Dual-kernel vs Virtualization

Here we discuss four metrics mentioned before as key points of system performance. Considering that it is difficult to test these metrics directly, we change to compare another four testable mechanisms - tasks management, memory management, event management, and runtime environment - and believe that they can reflect the performance of former metrics.

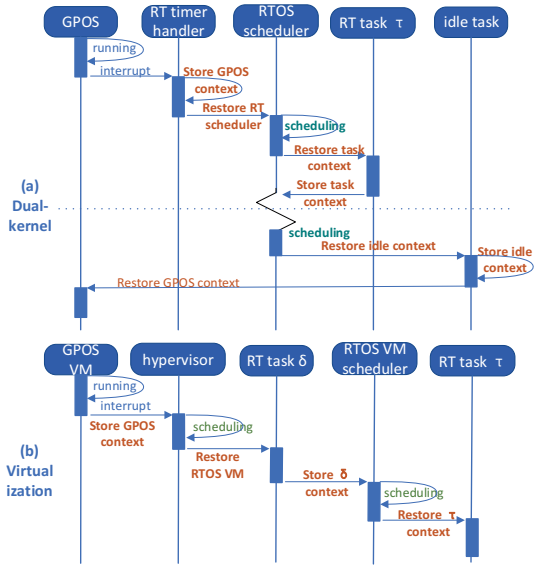

Fig. 1. Main Task Switch Process in Different Systems

A.1 Tasks management. Both systems adopt a two-level model for task management, and the main difference exists in OS switch. Dual-kernel system's RTOS kernel do scheduling not only for its RT tasks, but also for running of GPOS. As a contrast, a virtualization system adds an extra hypervisor to manage the switch operation of GPOS/RTOS VMs. Figure 1 (a) gives a illustration how a given real-time task τ and GPOS are alternately executing, while the process with the same goal in a virtualization system is given in figure 1 (b). As shown, GPOS is interrupted by a real-time timer, and the interrupt handler stores the runtime context of GPOS, then restores the context of the RTOS scheduler. If τ is ready, the

RTOS scheduler will restore the context of τ. When the RTOS scheduler finds no runnable task in the queue, the idle task will be switched on, and it will invoke a system call to store the context of itself, and restore the context of GOPS. In figure 1 (b), a hypervisor runs at the under-layer of two VMs, so extra scheduling and context storing/restoring take place in the process of VM switch.

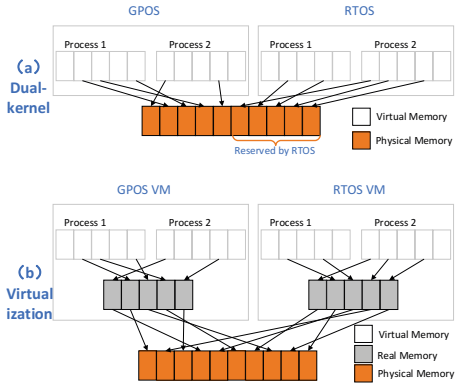

Fig. 2. Address Translation in Different Systems

A.2 Memory model. In the dual-kernel system as shown in figure 2 (a), some physical memory is retained and locked by RTOS, so GPOS can only use other physical memory, though they both adopt two level (virtual/physical) address translation. A virtualization system normally adopts three level (virtual/real/physical) address translation shown in figure 2 (b).

A.3 Interrupt handling. Dual-kernel system ensures that interrupts are first treated by RTOS. Interrupts belonging to GPOS will be put into a pipeline and then be progagated to GPOS when there are no more runnable tasks in RTOS. In virtualization system, all the interrupts will be firstly treated by the hypervisor (or domain 0 OS), then be forwarded to VMs. Though IO virtualization [10] is able to route interrupts to VMs directly, unfortunately there are very few commodity platforms with IO virtualization support.

A.4 Runtime environment. A dual-kernel system integrates two OSs by patching the GPOS kernel and adding many intercoupling function in two kernels, so there is no logical independent environment for GPOS or RTOS, and no effective defence to harmful interference from each other. As known, virtualization systems have well-defined and isolated environments for each OS.

We have following observations through the comparison.

- C.1 Dual-kernel system achieves better real-time latency and suffers from a lower overhead, because
  - less context store/restore (3 vs 4 times in figure 1).
  - less times of scheduling in task switch (1 vs 2 times in figure 1).
  - shorter interrupt latency, because interrupts go to RTOS directly.
  - shorter memory access latency, for less translation layers.
  - less waste CPU time, because it saves all the idle time of RTOS to run GPOS.
- C.2 Virtualization system is much better than dual-kernel system in aspects of complexity and isolation.
  - lower complexity is benefited from the advanced VM capabilities, such as cloning and live migration.
  - virtualization provides software&hardware isolation, which brings it ideal reliability and security.

B. Design Philosophy of TZDKS

TZDKS is derived from the following fundamental principles.

- at least two kernels are required to handle different time-sensitive tasks' management.
- simplified structures. Dual-kernel system has less components and management levels, which is the main cause of the less overhead and the lower latency.
- hardware virtualization support. Both isolation and high performance require that.
- replacement or simplification of the hypervisor. This software layer decreases the performance.

Normal virtualization technologies seem more heavy-weight than above principles, while TrustZone - a lightweight isolation extension of ARM - comes into our view. We can easily get two isolated domains (or virtual machines) with the assistance of the following TrustZone hardware mechanisms.

- Each physical CPU is virtualized into two virtual CPUs: one for the secure world and the other for the non-secure world. Cache of each level is also virtualized and isolated.
- TrustZone Address-Space Controller (TZASC) allows partition of memory, which can be exploited to guarantee strong spatial isolation. Therefore, TrustZone-enabled system only has/needs MMU support for two-level address translation.
- TrustZone Protection Controller (TZPC) allows devices to be configured as secure or non-secure, that allows the isolation of devices at the hardware level.
- Generic Interrupt Controller (GIC) supports the coexistence of secure and non-secure interrupt sources. It allows the configuration of secure interrupts with a higher priority, and also allows to assign IRQs and FIQs to secure or non-secure interrupt sources.

Some opensource projects like Trusted Firmware [2] have provided sound support for two domains and virtual-machine-like interfaces to Linux and general RTOS, and also give us ideal platform foundation.

So it seems that it is an obstacle to pursue greater performance in designing this new system. We are fortunate enough to discover that many mechanisms provided by TrustZone are very helpful to improving the performance of TZDKS - our TrustZone-based Dual-Kernel System.

- With the assistance of hardware memory isolation, two-level address translation can be implemented in the TZDKS virtual memory subsystem, and makes it have

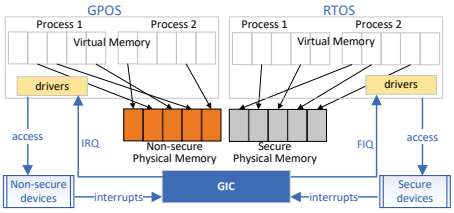

Fig. 3. Address Translation and Device Access in TZDKS

the same efficiency as the memory mapping in a bare-metal OS (figure 3).

- Through the proper configuration of GIC, interrupts can be routed to the owner kernel by hardware, that avoids software interrupt forwarding. Both kernels benefit from the simplification of interrupts management (figure 3).
- Devices can be partitioned according to requirement, so the IO software stacks can be simplified and the IO latency can be kept at the lowest level.
- Some software characters of TrustZone can also be exploited. For example, we can use the monitor mode as a context switcher for two kernels, so as to replace the functions of a hypervisor. We will implement the kernel switch shown in Figure 4, apparently it has the same efficiency as the traditional dual-kernel system.

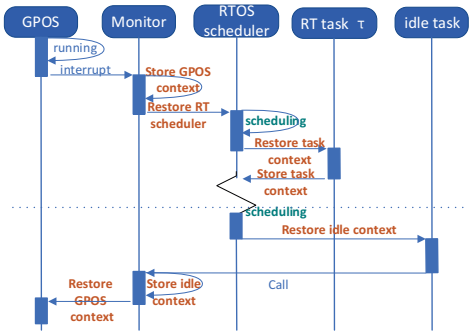

Fig. 4. Tasks Switch Process in TZDKS

In a word, TrustZone extension provides sufficient support to achieve a balance among isolation, virtualization, and performance for dual-kernel structure.

IV. Implementation of TZDKS

A. Architecture of TZDKS

There are more than one possible structures may be adopted to integrate a RTOS and a GPOS in a TrustZone-enabled multi-core platform. For example, RTOS can use CPU cores in sharing (with GPOS) way or in exclusive way.TZDKS chooses the sharing way, named multi-core shared structure (MSS). In MSS, all CPU cores are time-shared by two OSs. MSS can achieve high processor utilization, so an uniprocessor platform can also support it. It is more suitable to develop a dual-criticality system based on MSS because time slices can be neatly deployed to meet the requirements of high critical applications. A complicated OS switch mechanism should be designed in a MSS system. In order to implement a smooth OS switch, we leverage the monitor mode of TrustZone. As shown in figure 5, there are two software stacks located in the two worlds of TrustZone-enabled environment on TZDKS. Considering that SW has higher priority than NW, we build RTOS in SW for high criticailty guarantee. The SW stack is composed by the monitor module, RTOS and real-time tasks/services, and provides a real-time environment for the development of applications which need to guarantee specific deadlines. While the NW stack is composed by GPOS and applications, and provides a rich environment for running human-machine interaction as well as internet-based applications.

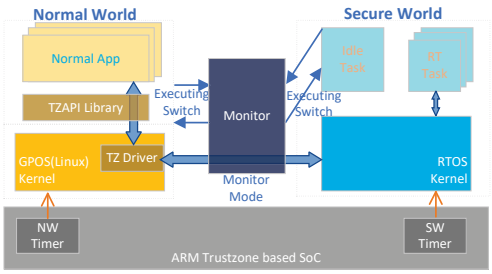

Fig. 5. TZDKS Architecture

B. Components of TZDKS

1) RTOS: RTOS is the partly modified version of a typical real-time system - μcOSII. The main modifications on the μcOSII kernel side includes: (i) a new port to enter-into/exit-from GPOS, (ii) implementation of idle-scheduling, that is to modify the idle task as an entrance for GPOS. (iii) optional support for standard TEE (Trusted Execution Environment).

2) Monitor: The monitor component executs as a slave module, though it runs in EL3 mode of ARM CPU. In fact, the monitor is only activated through two ways. One is through a SMC call, the other is through FIQ when GPOS is executing. Functions of the monitor component includes: (i) SMC service ports, (ii) timer interrupt handler for RTOS in the period of GPOS running, (iii) world switcher.

3) GPOS: GPOS is an enhanced Linux system. Actually, Linux can run in the normal world without modification. Some new modules have been added to Linux for the communication with RTOS, including the kernel drivers for TrustZone(which encapsulates SMC ports as a pseudo-device), application libraries (provide communication ports and standard TEE service ports defined by the GloblePlatform consortium), some daemon services for the RTOS requirement, and a configuration module for RTOS.

C. Working Process of TZDKS

The system starts booting on the secure world side by performing a series of initialization operations, such as allocating

different resources to the predefined worlds and loading the exception/SMC vectors to the predefined addresses. Afterwards, the RTOS kernel is loaded and started. The whole system will run with RTOS as the main body, while GPOS will be loaded and executed as a special task of RTOS, i.e. the IDLE task. Each OS owns its private timer source. Meanwhile, different interrupt types are configured to each OS (IRQ for GPOS, and FIQ for RTOS). IRQs are masked during the secure world execution for the priority of real-time tasks.

There are two kinds of event can trigger OS switch, SMC instruction and interrupt. When CPU is executing in the GPOS mode, a SMC call or a FIQ interrupt will trap CPU into the monitor mode, and the monitor will store environment information of GPOS, then redirect the control to the scheduler or the ISR (Interrupt Service Routing) in RTOS, thus the system will switch to RTOS. Switch to GPOS from RTOS is under the control of the idle-scheduling policy of TZDKS. To do this, RTOS stores the current environment and triggers a SMC call to the monitor, so the monitor will restore the GPOS context directly.

D. Mixed-Criticality Design

In TZDKS, idle-scheduling policy enables that RTOS has a higher scheduling priority than GPOS, and consequently GPOS is only scheduled during the idle periods of RTOS. We find that pure idle-scheduling makes some troubles for GPOS even when the CPU is not fully occupied by RTOS. One problem is timer loss. That is because GPOS (with the lowest priority) may be blocked by other real-time tasks for a long time, and which will make GPOS lose some timer interrupts. The other problem is the priority reverse brought by communications between two OSs. Some tasks in RTOS possibly require communicate with GPOS, and maybe wait GPOS for a long time. We hence add another real-time task τ_G also serving as a container of GPOS (when τ_G get CPU, it will switch GPOS on), but with a variable priority. Due to the limitation of the pages, we will give the detailed description of τ_G in the future paper.

V. Evaluation

We implemented TZDKS on a Hikey development board with Trustzone-enabled. Hikey has an octa-core Cortex-A53 CPU (1.2 GHz), 2GB memory, 8GB eMMC storage. Because μcOSII only supports uniprocessor, we modified the power management functions in the under level of the software so that only one core is left running. Nevertheless, the design of TZDKS can also support a multi-core RTOS, and the following experiments also reflect the performances in the multi-core environment.

In order to evaluate the performance, four metrics discussed in Section IV are measured: Complexity, Overhead, isolation, and RT latency. Because isolation is hardly to be verified by experiments, we conduct a discussion around supporting mechanisms. Note that, in order to ensure the readability of experimentation results, we have normalized the result data, because different Linux versions are used in target platforms.

A. System Complexity

Benefiting from the TrustZone light-weight virtualization, we can rapidly develop the prototype of TZDKS in a few weeks. At the side of adapted μcOSII, we only modifed two exception handler functions and IDLE task body to make the OS running in the secure world. At the Linux side, it can run directly in the normal world without modefication.

TABLE I
Necessary Code lines added to the TZDKS Components

	Linux	**μcOS**	**Trusted Firmware etc.**
Code Lines	0	< 300	< 100

Besides that, some code lines were added to the Trusted Firmware to enable a timer for μcOS. Applications and their developments can be migrated to the new system easily. Table I lists the code lines needed to develop the TZDKS. We note that Xenomai require a patch to Linux kernel which has more than 15 thousands code lines [1]. TZDKS obviously has a very low complexity notwithstanding it is only a prototype system.

B. Evaluation on Isolation

Note that we mainly consider the isolation for RTOS in a dual-criticality system. In TZDKS, access permissions to memory and peripherals are under the control of hardware controllers, and resources belonging to RTOS can not be accessed by GPOS. Interrupts are configured (in GIC) as two groups: group 0 and group 1. Group 0 interrupts are only hardware routed to RTOS, while group 1 are only to GPOS. These hardware components have been built in almost all current ARMv8 processors. While in a virtualization system, memory isolation is normally supported by hardware assistances (such as VTx). Hardware isolation for peripherals and interrupts also require extra hardware assistances(such as VT-d), which suffers from significant system cost. Therefore, TZDKS provides fine isolation for RTOS with low-cost hardware.

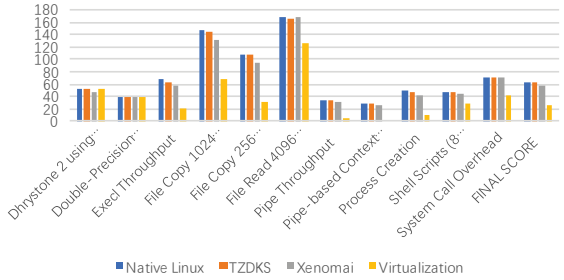

Fig. 6. Unixbench Results

C. Overhead

Due to the lack of method to test the integral performance of TZDKS, we use UnixBench to measure the comprehensive performance of Linux (GPOS) with zero load in RTOS. The results will reflect the performance of TZDKS. Afterwards, we compare the performance with other three Linux systems:

a native Linux, a Xenomai Linux, and a Linux in a Xen VM. As shown in figure 6, we can see that there is almost no performance loss in both GPOSs of TZDKS and Xenomai Linux when the load of RTOS is very light. TZDKS is even better than Xenomai as a whole, and a possible reason is that Xenomai has more cost of context switch than TZDKS (we will explain this later in the task switch experiment). As a contrast, Linux in the Xen virtual machine has obvious performance loss.

D. Interrupt Latency for RTOS

We measure the time from a interrupt triggered to the interrupt handler running. Two thousands times SGI (Software Generated Interrupt) were repeated in the experiment, and we lists the maximum, minimum, average, as well as MSE (Mean Squared Error) of latencies in table II. We compare the latencies with the bare-metal μcOS, and results show that the interrupt latencies in RTOS of TZDKS are slightly influenced by GPOS, but are still deterministic and short enough for most real-time applications. We also run the latency test (of the real-time timer) provided by Xenomai package (the test can not give MSE results), and the results show that TZDKS always has shorter latencies than Xenomai.

TABLE II
INTERRUPT LATENCY FOR RTOS

	Max (cycles/μs)	Min (cycles/μs)	Average (cycles/μs)	MSE (cycles/μs)
μcOS in TZDKS	2530 / 2.11	410 / 0.34	1001.1 / 0.83	632.6 / 0.53
Bare-metal μcOS	1377 / 1.15	380 / 0.32	823.5 / 0.69	313.1 / 0.26
Xenomai Cobalt	15069 / 12.56	1382 / 1.16	3889 / 3.24	- / -

E. Context Switch Latency for Real-Time Tasks

In this measurement, we measure the CPU cycles used in the process shown in upper half of Figure 4, e.g. the longest time of a ready real-time task τ waiting to run. Results in table III show that the longest time is less than 20 μs in TZDKS when GPOS has very high load (especially when there are many EXECL calls), so the context switch performance is good enough for most applications. We run the switchtest provided by Xenomai package to test the performance of thread switch in kernel mode of Xenomai, and the maximum number of switches is no more than 1800 per second on our board (the average switch period is more than 550 μs).We think this result probably can not reflect the true performance difference between TZDKS and Xenomai for diverse methods and details in each test, but it still shows that TZDKS has a good task switch performance, and more researches will concern it in the further work.

TABLE III
CONTEXT SWITCH LATENCY

	Max (cycles/μs)	Min (cycles/μs)	Average (cycles/μs)	MSE (cycles/μs)
GPOS to RT-task in TZDKS	19475 / 16.23	1757 / 1.47	4884.3 / 4.07	3619.8 / 3.02
Task switch in Bare-metal μcOS	1629 / 1.15	642 / 0.54	1079.5 / 0.90	312.8 / 0.26

VI. CONCLUSIONS

The mixed-time-sensitive system, which combines different types of OSs on unique hardware platform, has wide requirements and applications in many fields such as robot, aviation etc. Two traditional solutions, dual-kernel and virtualization, provide just reverse merit and demerit in different performances. This paper proposes an idea to realize the dual-kernel system based on the TrustZone isolation, and give the design of TZDKS to verify this idea. TZDKS achieves suitable balance among complexity, isolation, latency, and overhead.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 61502510). Much of the work reported in this paper took place while the first author was visiting the University of York.

REFERENCES

[1] *Soucecode of Xenomai*. https://xenomai.org.
[2] *TRUSTZONE*. https://arm.com/products/security-on-arm/trustzone.
[3] Sierravisor virtualization for arm. Technical report, Sierraware, http://www.sierraware.com, 2017.
[4] S. Baruah and A. Burns. Fixed-priority scheduling of dual-criticality systems. In *Proceedings of the 21st International conference on Real-Time Networks and Systems*, pages 173–181. ACM, 2013.
[5] A. Burns and R. Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.
[6] M. Franke. A quantitative comparison of realtime linux solutions. *Chemnitz University of Technology*, 2007.
[7] T. Frenzel, A. Lackorzynski, A. Warg, and H. Härtig. Arm trustzone as a virtualization technique in embedded systems. In *Proceedings of Twelfth Real-Time Linux Workshop, Nairobi, Kenya*, 2010.
[8] Global Platform, http://www. globalplatform.org. *TEE client API specification*, 1.0 edition, 2010.
[9] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin. Resource efficient isolation mechanisms in mixed-criticality scheduling. In *27th Euromicro Conference on Real-Time Systems (ECRTS2015)*, pages 13–24.
[10] A. N. Jiang Zhe and P. Dong. Bluevisor: A scalable real-time hardware hypervisor for heterogeneous many-core embedded systems.
[11] J. H. Koh and B. W. Choi. Real-time performance of real-time mechanisms for rtai and xenomai in various running conditions. *International Journal of Control and Automation*, 6(1):235–246, 2013.
[12] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho. Vosys-monitor, a low latency monitor layer for mixed-criticality systems on armv8-a. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[13] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares. Towards a lightweight embedded virtualization architecture exploiting arm trustzone. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–4. IEEE, 2014.
[14] P. J. Prisaznuk. Arinc 653 role in integrated modular avionics (ima). In *27th Digital Avionics Systems Conference (DASC 2008)*, pages 1–E.
[15] D. Sangorrín, S. Honda, and H. Takada. Reliable and efficient dual-os communications for real-time embedded virtualization. *Information and Media Technologies*, 8(1):1–17, 2013.
[16] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 67–80, 2014.
[17] J. Zhe and A. Neil. Vcdc: The virtualized complicated device controller. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[18] N. A. Zhe Jiang. Gpiocp: Timing-accurate general purpose i/o controller for many-core real-time systems. In *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2017.