Achieving Performance Balance for Dual-Criticality System Based on ARM TrustZone

Pan Dong[1][2] Alan Burns[1] Zhe Jiang[1] Yan Ding[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 this as a mixed-time-sensitive system. Complexity, isolation, real-time latency, and overhead are the main metrics to design such a mixed-time-sensitive system. These metrics may conflict with each other, so it is difficult for them to be consistently optimized. Most existing implementations only optimize with 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 mixed-time-sensitive systems. We figure out the difference in performance across system designs such as task switching, memory management, interrupt handling, and resource isolation. We propose the philosophy of utilizing TrustZone characteristics to optimize various mechanisms in mixed-time-sensitive systems.

The second contribution of the paper is to propose a Trustzone-based solution - termed TZDKS - for mixed-time-sensitive system. Appropriate utilization of TrustZone extensions helps TZDKS to implements (i) virtualization environment for GPOS and RTOS, (ii) high efficiency task switching, memory accessing, interrupt handling and device accessing 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 in this paper, also deemed as a special Dual-Criticalitiy System [4].

The performance of a mixed-time-sensitive system 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 examples, isolation and complexity collide with performance or overhead, and real-time latency collides with performance which is reflected by overhead. Most existing implementations of a mixed-time-sensitive system (e.g. GPOS dual-kernel extending, and virtualization-based [11] system) have only optimized part of the above metrics but not all. From observation, we get some conclusions that, (i) the dual-kernel system has less software levels and more resources sharing, so it can achieve lower overhead, and (ii) the virtualization system relies on resource partition (especially some hardware supports) to achive better isolation and lower complexity.

As the first contribution, this paper provide a detailed analysis of performance influencing factors which are exerted by various runtime mechanisms of existing mixed-time-sensitive systems. We figured out the difference in efficiency across system designs such as task switching, memory management, interrupt handling, and resource isolation. We propose the philosophy of utilizing TrustZone characteristics to optimize various mechanisms in a mixed-time-sensitive system. The second contribution of the paper is to propose a Trustzone-based solution for mixed-time-sensitive systems, termed TZDKS. Appropriate utilization of TrustZone extension helps TZDKS implement (i) virtualization environment for GPOS and RTOS, (ii) high efficient task switching, memory accessing, interrupt handling and device accessing 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 presents our motivation; Section III introduces related work; Section IV gives the designing philosophy; Section V describes the TZDKS implementation; Section VI evaluates the performance of TZDKS, with conclusions offered in Section VII.

II. MOTIVATION

There are many approaches to design and implement a mixed-time-sensitive system, which can be classified as two sorts. The traditional way is to extend popular GPOS, such as 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 [10]. Dual-kernel systems do not require extra hardware support, and only introduces low overhead [10]. However, it needs to modify the GPOS kernel heavily, which result in much more cost in complexity and flexibility. Additionally, insufficient isolation between OSs leads to many security and reliability problems [15]. In contrast, virtualization-based method becomes a more popular and rapid method to design a mixed-time-sensitive system 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, it heavily relies on the hardware support, which increases the cost of the whole system [12]. And extra software levels also increase the system's overhead. Moreover, the hypervisor must be redesigned to meet the real-time requirement.

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 few overhead in software. Therefore, its characteristics do help to develop a mixed-time-sensitive system with all-round balance amongst complexity, isolation, real-time latency, and overhead.

A new idea is proposed that combine strong points of dual-core and virtualization by the utilizing TrustZone. We will design TZDKS based on this idea.

III. RELATED WORK

A. Two Common Solutions for Integrating Embedded System

A dual-kernel mixed-time-sensitive system 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) interrupt GPOS, and will do rescheduling for the real-time tasks. So this is called as idle-scheduling strategy. RTLinux, RTAI, Xenomai and RTThread [6] [10] are products of dual-kernel system, and are widely applied in industrial systems.

In a virtualization-based mixed-time-sensitive system, 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). Such architectures can be found in industrial control systems where the RTOS takes over the time-critical control of a machine while the GPOS runs, for example, the visualization software. Futher examples are single-processor smartphones where an RTOS is used to manage critical tasks of the radio communication while a GPOS hosts the typical set of mobile phone applications. The up-to-date avionics systems specification - ARINC 653 [14] - is another example of the virtualization-based mixed-time-sensitive system. This specification requires integrating many subsystems (such as flight control system, environment control system, and amusement system) into a virtualized platform on modern aircraft.

These two mixed-time-sensitive systems always behave oppositely in many aspects, and detailed analysis will be presented in section IV.

B. Introduction of 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 instructs in one of two possible operating worlds: the normal world and the secure world. The isolation mechanisms of TrustZone are well defined. Access permissions are strictly under the control of the secure world, which forbids access from the normal world. As the processor only runs in one world at a time, to run in the other world requires context switch. This is achieved via a special instruction called the Secure Monitor Call (SMC). In order to facilitate an application in the normal world to connect to and invoke a secure service in the secure world, 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]. ARMs TrustZone security extensions can be utilized to virtualize a system in two ways:

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

(2) Use the efficient switching mechanism of the Secure zone Monitor to host a dual-OS system (Secure zone OS and Normal zone 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 ARM's TrustZone security extensions to efficiently partition the system into Trusted and Non-Trusted states, which provides full system access to trusted software, and limits the capabilities of software running in Non-Trusted state.

IV. BALANCING DESIGNATION PHILOSOPHY OF TZDKS

A. Dual-kernel vs Virtualization

Here we analyse four metrics - complexity, isolation, real-time latency, and overhead - between two types of dual-criticality systems: tasks management, memory management, event management, and runtime environment. Afterwards, we will compare the performance in two systems based these four metrics.

A.1 Tasks management. Both systems adopt a two-level model for task management, and the main difference exists in OS switching. Dual-kernel system's RTOS kernel do scheduling not only for its RT tasks, but also for GPOS. As a contrast, a virtualization system adds an extra hypervisor to manage the switching operation of GPOS/RTOS VMs. Figure 1 (a) gives

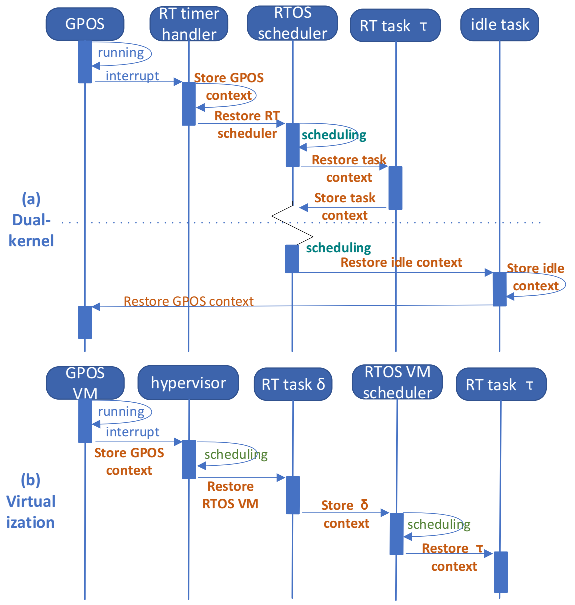


Fig. 1. Main Task Switch Process in Different Systems

a typical process 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 restore the context of the RTOS scheduler. If τ is ready, the RTOS scheduler will restore the context of τ. When 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 switching VMs.

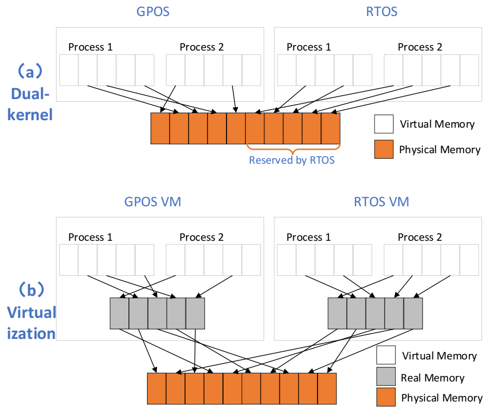


Fig. 2. Address Translation in Different Systems

A.2 Memory model. In 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. 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 the RTOS. In virtualization system, all the interrupts will be firstly treated by the hypervisor (or domain 0 OS), then be forwarded to VMs.

A.4 Runtime environment. 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 and RTOS, and no effective defence to harmful interference from each other. As known, virtualization systems have well-defined and isolated virtualization environments for each OS.

Now we have the following observations and results by comparison.

- C.1 Dual-kernel system achieves better performance in terms of real-time latency for the following reasons.
  - it has less context store/restore operations (3 times vs 5 times in figure 1).
  - it has shorter interrupt response latency, because the interrupt goes directly to the RTOS.
  - it has shorter memory access latency, because the address translation has less layers.
- C.2 Dual-kernel system achieves better performance in overhead, because
  - it has less context store/restore times.
  - it has less times of scheduling in task switching (1 time vs 2 times in figure 1).
  - it has less wast CPU time, because it saves all the idle time of RTOS to run GPOS.
  - it has less memory access overhead.
- C.3 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, template-based deployment, check-pointing, and live migration.
  - virtualization provides not only software but also hardware isolation, which brings it a high level of reliability and security.

B. Designation Philosophy of TZDKS

TZDKS is derived from the following fundamental principles.

- at least two kernels are required to handle different time-sensitive tasks management.
- simple and reduced 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 employment. Both isolation and high performance require that.

- replacement or simplification of the hypervisor. This software level 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.

- With hardware support, 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 between two worlds. Therefore, TrustZone-enabled system only has/needs MMU support for single-level address translation.
- TrustZone Protection Controller (TZPC) allows devices to be (statically or dynamically) 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 [1] have provided sound support for two domains and virtual-machine-like interfaces to Linux and general RTOS, and also give us ideal platform fundamentals.

So it seems that it is a greater 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.

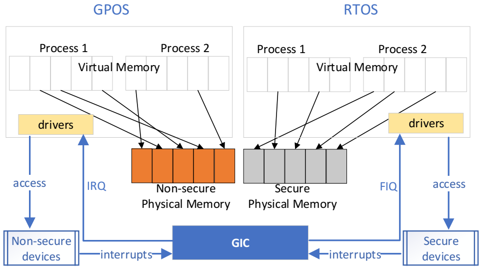


Fig. 3. Address Translation and Device Access in TZDKS

- With the assistance of hardware memory isolation, single level address translation can be implemented in the TZDKS virtual memory subsystem, and makes it have the same efficiency as the memory mapping in a bare-metal OS (figure 3).
- Through the well configuration of GIC, interrupts can be routed to the owner kernel by hardware, that avoids any software interrupt forwarding. Both kernels benefit from the simplification of interrupts management and timer mechanism (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 an 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 switching shown in Figure 4, apparently it has the same efficiency as the traditional dual-kernel system.

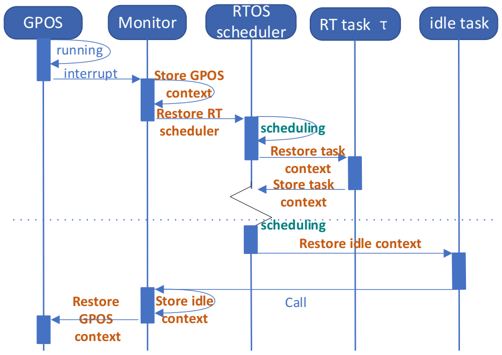


Fig. 4. Tasks Switch Process in TZDKS

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

V. Implementation of TZDKS

A. Architecture of TZDKS

As shown in figure 5, there are two software stacks located in the two worlds of TrustZone-enabled environment on TZDKS. The secure world 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 normal world stack is composed by GPOS and applications, and provides a rich environment for running user-machine interfaces as well as internet-based applications and services.

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 a gate for GPOS. (iii) optional support for standard TEE (Trusted Execution Environment).

2) Monitor: The monitor component executes in a slave mode though it has the highest executing level, because it has a lower priority than that of real-time tasks. In fact, the monitor is only activated through two ways. One is through a SMC call, the other is through FIQ when GPOS is on. Functions of the monitor component includes: (i) SMC service ports (to

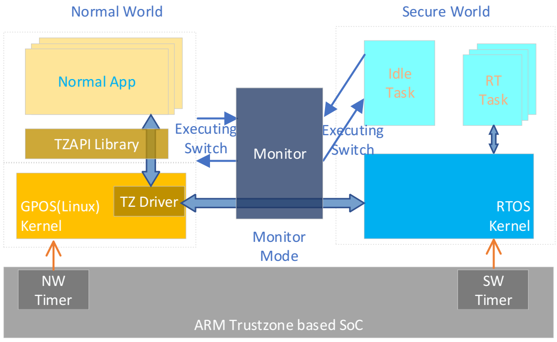


Fig. 5. TZDKS Architecture

support communications between worlds), (ii) timer interrupt handler for RTOS in the period of GPOS running, (iii) world switcher (saves the state of the current world, and restores the state of the ready-to-run world).

3) GPOS: GPOS is a enhanced Linux system. Basically, 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 driver for TrustZone(which encapsulates SMC (Secure Monitor Call) ports as a pseudo-device), application libraries (which provides communication ports and standard TEE service ports defined by the Globle-Platform 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 operations including hardware initialization and configuration, as well as allocating the different resources to the predefined worlds and loading the exception/SMC vectors to the predefined addresses. Then the RTOS kernel is loaded and started. The whole system will run with RTOS as the main body, while the GPOS will be loaded and executed as a special task of RTOS, i.e. the IDLE task. Each OS own its private timer source. 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.

VI. Evaluation

We implemented our TZDKS on an Hikey development board with Trustzone-enabled. Hikey has a octa-core ARM Cortex-A53 CPU up to 1.2 GHz per core, 2GB 800MHz LPDDR3 memory, 8GB eMMC on-board storage, 4 PL011 UARTs, platform peripherals, and secure peripherals. We modified the power management functions in the under level of the software so that only one core is left running in the system.

To evaluate the performance of TZDKS we targeted four metrics discussed in Section IV: Complexity, Overhead, isolation, and RT latency. Because isolation is hardly to be verified by experiments, we conduct a discussion around supporting mechanisms.

A. System Complexity

Benefiting from the light-weight virtualization ability provided by TrustZone technology, 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 using the kernel sourced from kernel.org.

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 [2]. TZDKS obviously has a very low complexity notwithstanding it is only a prototype system.

B. Evaluation on Isolation

Here we discuss the isolation between OSs about three resources: (i) memory, (ii) interrupt, (iii) peripheral. As known, the traditional dual-kernel system has no effective isolation support for above three resources. So we just exploit comparation between TZDKS and the virtualization system. Note that we mainly consider the isolation for RTOS in a dual-criticality system.

In TZDKS, access permission to memory, cache and peripherals are under the control of hardware controllers (TZASC, TZPC), and those resources which belong to RTOS can not be accessed by GPOS. Interrupts are configured (in GIC, which is a hardware interrupt controller) 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 are built in almost all current ARMv8 processors.

In the virtualization system, memory isolation is normally supported by hardware assistance (such as VTx). Hardware isolation for peripherals and interrupts always require extra hardware (such as VT-d), which increases the system cost.

Therefore, TZDKS provides fine isolation for RTOS through low-cost hardware.

C. Overhead

Considering that it is difficult to find a method to test the integral performance of TZDKS, We use UnixBench to measure the comprehensive performance of Linux (GPOS) with zero load in the RTOS. The results will reflect the performance of TZDKS. Then we compare the performance with other two Linux systems. One is a native Linux on a bare-metal, the other is a Linux in a Xen virtual machine. From the results shown in figure 6, we can see that there is almost no performance loss in the GPOS of TZDKS when the load of RTOS is very light. As a contrast, Linux in the Xen virtual machine has obvious performance loss.

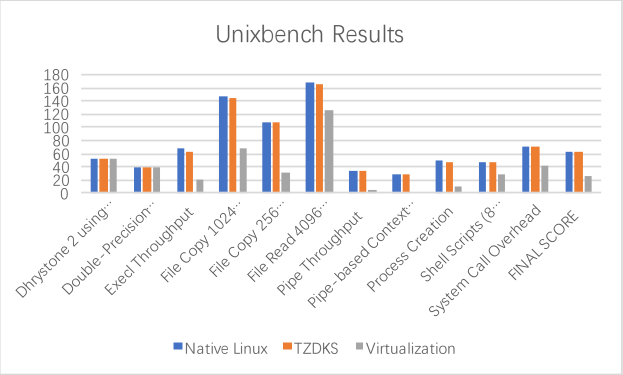


Fig. 6. Unixbench Results

D. Interrupt Latency for RTOS

For the interrupt latency test, we measure the time from when a interrupt is triggered to when the interrupt handler begins to run. Our experiment begins countering when a SGI (Software Generated Interrupt) instruction is executed, and ends countering at the start-point of the SGI handler function. Two thousands times SGIs were repeated in the experiment, and we lists the maximum latency, minimum latency, average latency, and MSE (Mean Squared Error) of latencies in table II. Results show that the interrupt latency in RTOS of TZDKS is slightly influenced by GPOS, but is still deterministic and short enough for most real-time applications.

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

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 that a ready real-time task τ waits 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. Here we find the fact that this latency is heavily influenced by GPOS, which is out of our expectations. A possible cause is hardware resources conflict in the CPU when the system executes switching between two worlds, and more research needs to be done about it in 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

VII. 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, interoperation, and overhead.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 61502510) and Foundation of Science and Technology on Information Assurance Laboratory (No.KJ-15-101). Much of the work reported in this paper took place while the first author was visiting the University of York.

REFERENCES

[1] *SECURITY ON ARM TRUSTZONE.* https://www.arm.com/products/security-on-arm/trustzone.
[2] *Soucecode of Xenomai.* https://xenomai.org.
[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 *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 13–24. IEEE, 2015.
[10] 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.
[11] Y. Li, R. West, and E. Missimer. A virtualized separation kernel for mixed criticality systems. In *ACM SIGPLAN Notices*, volume 49, pages 201–212. ACM, 2014.
[12] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho. Vosysmonitor, 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 *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1–E. IEEE, 2008.
[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. ACM, 2014.

Achieving Performance Balance for Dual-Criticality System Based on ARM TrustZone

Pan Dong | Alan Burns | Zhe Jiang | Yan Ding | Long Gao

Introduction

A mixed-criticality system composed of a RTOS and a GPOS is defined as a mixed-time-sensitive system (MTSS). Most existing MTSSs can optimize parts of main performance *metrics* (Complexity, isolation, real-time latency, and overhead) but not all.

Our contributions:

- detailed analysis and comparison of performance influencing factors across system designs such as task switching, memory management, interrupt handling, and resource isolation.
- propose the philosophy of utilizing TrustZone characteristics to optimize various mechanisms in MTSS.
- propose the design of TZDKS **(TrustZone-based Dual Kernel System)**, achieving a full-scale balance amongst aforementioned performance *metrics*.

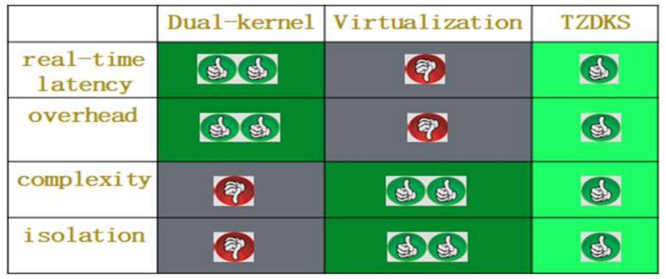
Observation

Existing MTSSs can be classified as two sorts:

- Dual-kernel systems. Extending popular GPOS by deploying a small RT kernel at the underlying of GPOS i.e. Xenomai
- virtualization-based systems. Integrating RTOS and GPOS through two virtual machines on one physical platform. i.e. ARINC 653

The Following observations from comparison give us inspiration to design TZDKS.

C.1 Dual-kernel system achieves better latency
– less context store/restore operations
– shorter interrupt response latency
– shorter memory access latency
C.2 Dual-kernel system suffers from less overhead
– less context store/restore times.
– less times of scheduling in task switching
– less waste of CPU time
– less memory access overhead
C.3 Virtualization system has better complexity and isolation.



	Dual-kernel	Virtualization	TZDKS
real-time latency	👍👍	👎	👍
overhead	👍👍	👎	👍
complexity	👎	👍👍	👍
isolation	👎	👍👍	👍

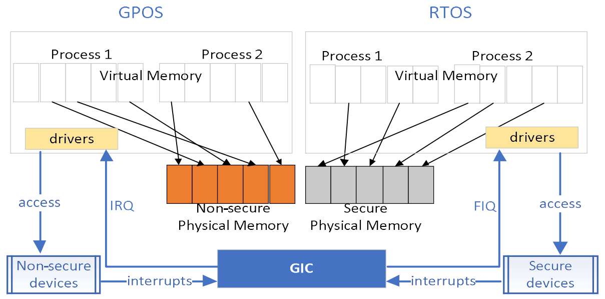
TZDKS Design Philosophy

TZDKS is derived from the following fundamental principles.

- at least two kernels are required to handle different time-sensitive tasks management: RTOS + GPOS
- simplified structures. Dual-kernel system has less components and management levels, which is the main reason of the less overhead and the lower latency in MTSS.
- hardware virtualization support. Required by both isolation and high performance.
- replacement or simplification of the hypervisor. hypervisor decreases the performance.

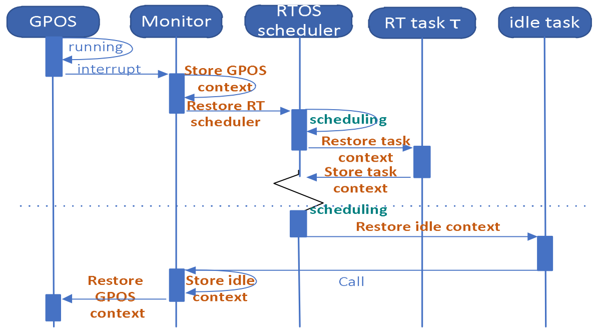
TZDKS leverages TrustZone mechanisms to get two isolated domains.

- CPU worlds switch capability
- TZASC (TrustZone Address-Space Controller) for memory isolation
- TZPC (TrustZone Protection Controller) for device isolation
- GIC (Generic Interrupt Controller) for interrupt isolation



TZDKS also leverages TrustZone mechanisms to improve its performance.

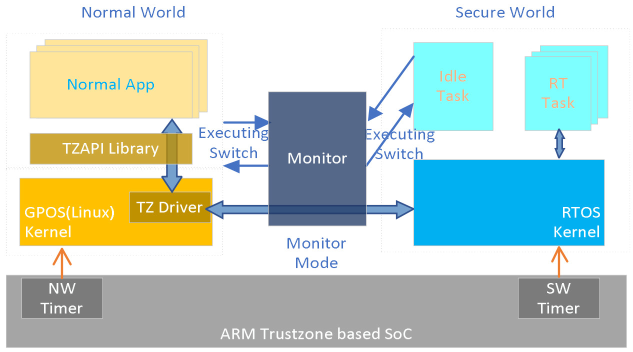
- single level address translation
- hardware routed interrupts
- simple I/O software stacks and short I/O latency
- monitor-mode-based switcher for dual-kernel



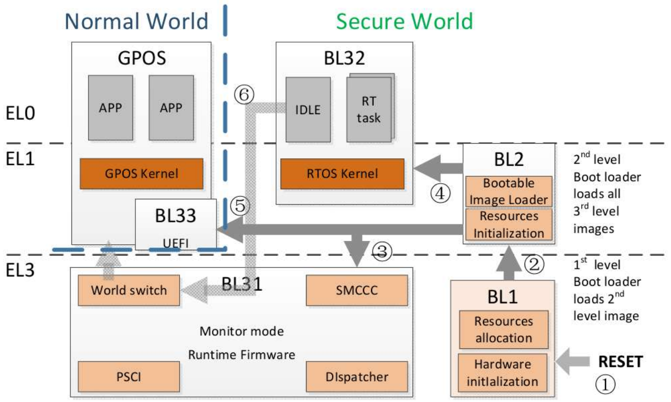
TZDKS Implementation

Architecture

Two software stacks are located on TZDKS. The secure world stack provides a real-time environment for the development of applications which need to guarantee specific deadlines. The normal world stack is composed by GPOS and applications, and provides a rich environment.



Booting & Running of TZDKS



Mixed-Criticality Design

- TZDKS assigns RTOS the highest priority to RTOS through idle-scheduling policy.
- Pure idle-scheduling makes troubles (e.g. timer loss, priority reverse) for GPOS even when CPU is not fully occupied by RTOS.
- Enhanced idle-scheduling policy is induced by adding another real-time task τ_G also serving as a container of GPOS but with a variable priority.

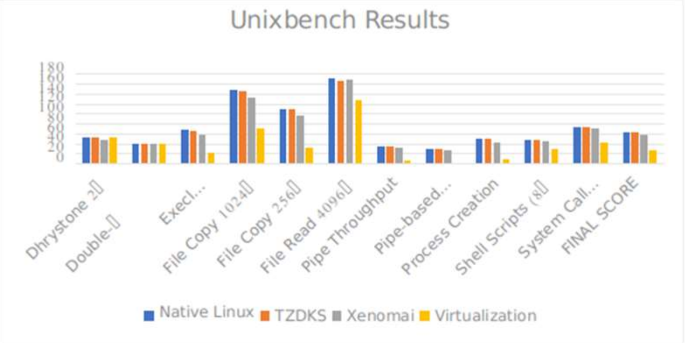
Evaluation

Platform: Hikey development board, a octa-core ARM Cortex-A53 CPU, 1.2 GHz per core, 2GB 800MHz DDR3 Memory

Complexity

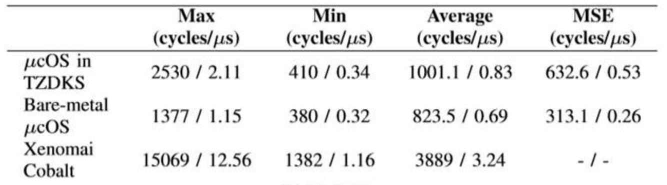
- Very few parts of the original system need to be modefied (two exception handler functions and IDLE task of µcOSII).
- No more than 100 modefied codelines are necessary in porting µcOSII into the secure world.

Overhead



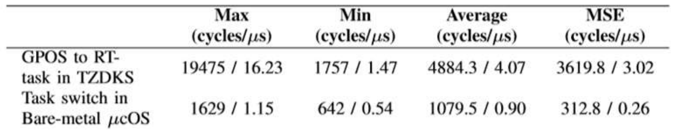
Latency for RTOS

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	- / -

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

Acknowledgements

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

Contact information

- Pan Dong. School of Computer, National University of Defense Technology
- Email: **pandong@nudt.edu.cn**