Distributed RTOS Microkernel for Multicore ARM-based Control Systems

Ivan Evgeniev Ivanov, Alexander Hotmar, Martin Minkov, Vesselin Evgueniev Gueorguiev
Desislava Georgieva

Abstract— **We are seeing an ever-increasing amount of real-time systems. Many of these are embedded in a variety of field devices requiring significant amounts of computation. Others are connected to a large number of sensors and actuators. The characteristic of all of them is that even the available increased performance of modern processors designed for embedded applications does not always meet the requirements. In this case, the switch is to processors with many cores. Real-time control of computational processes in multicore systems is a task of non-trivial complexity. The paper presented here discusses some aspects of the implementation of the RTOS microkernel for ARM-based multicore control systems.**

Index Terms— **RTOS, distributed kernel, multicore ARM.**

INTRODUCTION

[1]Today we see a ubiquitous usage of cyber-physical systems – from washing machine controller to robots, cars, etc. Most of these systems work real time. These means that, according to one very useful explanation of Edward Lee, they have not only to deliver exact result, but in exact time. The real time world includes very small systems operating on interrupt-driven manner to huge systems driven by Real-Time Operating Systems (RTOS). The complexity of cyber-physical systems increases permanently. They are covering from multi-axes motion control to real-time image recognition. These tasks need complex internal software structure and cooperation of multiple programs working in pseudoparallel manner.

Many providers of computers-on-chip started to produce multicore systems to answer the need of high-performance calculations together with more ordinary process input-output and control.

Today there are many versions of RTOS, several of them shown as "industrial standard". Some examples (but not a full list) are FreeRTOS [4], µC/OS-III [5], RTX[6]. They are very flexible, they support both internal and external communications, multitasking, synchronization, deterministic priority scheduling. Several of them now have versions supporting processors with several cores [7], including some massively-parallel solutions based on ARMs [8]. The overview of these systems encounters that they control cores as they are individual computers. The similar control is known from the era of multiprocessor computers. A single instance of the operating system is attached to a single processor and manages it. Based on inter-processor communication synchronization between actions attached to corresponding processor takes place. As opposite – in case of many (more than 4) processors, one of them is occupied only by the operating system and all the others are attached to some task according to OS decisions. Both these solutions are well known, stable and implemented many times. The main backward of these solutions is the fact that they are planned to operate with variable number of tasks. This means that some task can quit the system forever, some other can be included in the execution list and so on. This makes the system, supposed to be real-time similar to general purpose system. The problem is in the fact that nobody can predict the schedulability of such a system in a specific moment of its work.

Working in the area of hard real-time applications several year ago a group from the Advanced Systems Laboratory of the Technical University of Sofia designed and developed a small predictive real-time kernel named HARTEX (HArd Real-Time Executive) [1] [2]. Constant execution time of kernel operations is one of the most important characteristics of this kernel. This kernel has very limited communication support. It was designed before wide use of USB and of TCP for real-time communication. But its main positive characteristic was very small (few microseconds on the processors of its era) switching time and as it was said – constant time.

Most of the cyber-physical systems are embedded in some machine/apparatus and they have known number of tasks, moreover – this number of tasks is constant. They do not need the powerful but cumbersome task management system following the requirements of variable tasks number and the procedure for task load/discard.

Following requirements of the industrial applications developed by our team and need for simple observable kernel for educational purposes a decision to port the HARTEX kernel on a ARM core took place. In the previous years we did some steps on this direction and ported it to a RENESAS 32 bit ARM core. Experiments were very successive but

Received: 06.08.2024
Published: 08.09.2024
https://doi.org/10.47978/TUS.2024.74.02.005

Ivan Evgeniev Ivanov , FA, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: iei@tu-sofia.bg).
Alexander Hotmar, FA, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: hotmar@tu-sofia.bg)
Martin Minkov, FA, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: mminkov@tu-sofia.bg)
Vesselin Evgueniev Gueorguiev, FCST, Technical University of Sofia, 1000 Sofia, Bulgaria (e-mail: veg@tu-sofia.bg).
Desislava Georgieva, NBU, New Bulgarian University, Sofia, Bulgaria (e-mail: author@ie-bas.org dvelcheva@nbu.bg).

changes in scientific environment stopped next development. The most important for that solution was the decision to run all programs (system and users) on the same level (supervisor mode).

The development of computer technology, the increase in processor performance, the widespread use of real-time systems and the features of the well-known RT OS necessitated a return to the capabilities embedded in HARTEX. One of the main challenges that were addressed to some extent and which will be discussed further below is the possibility of running an RT OS process on any available processor if there are no specific hardware constraints.

HARTEX – BASIC CHARACTERISTICS

Before starting with the evolution of HARTEX, here will be presented general characteristics of this RT kernel.

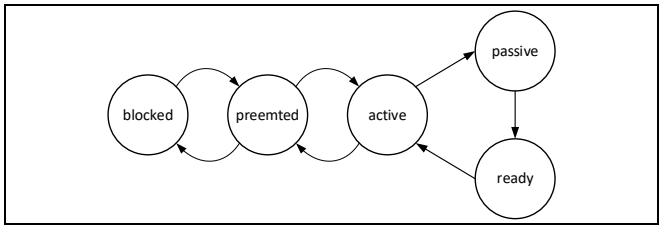

Fig 1. Task status diagram of HARTEX kernel

The HARTEX kernel has the following task state diagram (see Figure 1). The HARTEX kernel operates its task on interruptible manner. That is why it has the five main statuses. The first four of them are:

- active – the task is occupying CPU
- passive – the task is inactive and does not request any CPU time
- ready - the task is ready for activation and waits for CPU time
- preempted – the task has been on an active state but CPU was attached to a more privileged task and execution is suspended for a while

Additionally, there is one more task status – blocked task.

Blocked is a task which misses some resource and cannot continue its work, e.g. task, waiting on a semaphore.

Basic characteristic of HARTEX is its task registering and scheduling mechanism. Usually, task's status is written in the corresponding Task Control Block (TCB). In this case task selection need to go through the TCBs to select the task for some system action (e.g. CPU occupation or similar). To avoid TCB search in HARTEX is implemented other mechanism – the Boolean vector processing. The idea to use Boolean vectors for task status registration is based on the fact that usual cyber-physical system embedded in some device has – 1) limited; 2) unchangeable number of tasks. Additionally, this number is not very big. Here "big" means not more that 20-30 task.

HARTEX uses fixed tasks priority mechanism: every single task has a unique priority. To register 32 tasks, we need 32-bit Boolean vector. Today's ARM processors mainly offer internal 32-bit architecture, so tasks status words will fit processor registers. In case of really big number of tasks (up to 64) two registers will be needed.

The fast process managing is based on a distributed task status hold in the following Boolean vectors:

- active task vector (ATV)
- pre-empted tasks vector (PTV)
- ready task vector (RTV)
- blocked tasks vector (BTV).

The number of the bit in the Boolean vector represents the task priority, e.g. bit 1 means priority level 1. The bigger bit number means higher priority.

All tasks ready for execution are marked in RTV. All pre-empted task are marked in PTV. If a task is blocked this is marked in BTV. In the ATV in marked the task currently occupying CPU. This vector status words make possible to select the active task after fixed number of Boolean operations over the status vectors.

Some previous versions of HARTEX were built on dual priority mechanism where some "soft real-time" tasks have fixed priorities but some other "hard real-time" tasks have normal and high (promoted) priorities.

On Figure 2 are presented tasks status Boolean vectors.

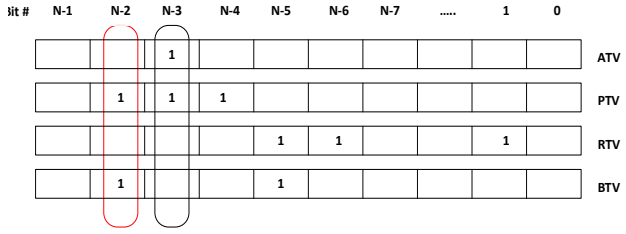

Fig. 2. Task status Boolean vectors

One of the main characteristics of HARTEX is the fact that the kernel and tasks share one address space. Depending on processor some versions of HARTEX use system and user priority for accessing memory regions and program resources but some other do not. Every task has its own stack space. Every task has its own list of systems resources used by it. Here resource is everything sharable between more than one task and/or having its own context. We will not discuss here resource access discipline because this is out of scope of this paper.

HARTEX MODIFICATIONS FOR MULTI-CORE IMPLEMENTATION

As it was discussed before there are several ways to implement OS for multicore processor. The discussed here implementation of HARTEX is oriented to the STM32H745 single chip dual core controller. The motivation to use this system has several elements but the main one is – it is Cortex-based fully real-time oriented.

According to the original documentation [3] this system has two processor cores – one Cortex-M4 and one Cortex-M7. They share one and the same address space. There is one exclusion – part of this space is accessible only for the M4 core and some other part – for M7 core. Here is used the word "core" but it is a "processor" itself. In first assumption we have the so-called symmetrical system architecture (see Figure 3). here we have the so-called heterogeneous architecture. This is because processors are not from one type.

If they were identical, we will have “homogeneous” architecture.

Because all cores have access to the memory, this enables implementation of OS kernel possible to dispatch tasks between cores. To have this dispatching possible the first requirement is to have possibility to execute binaries

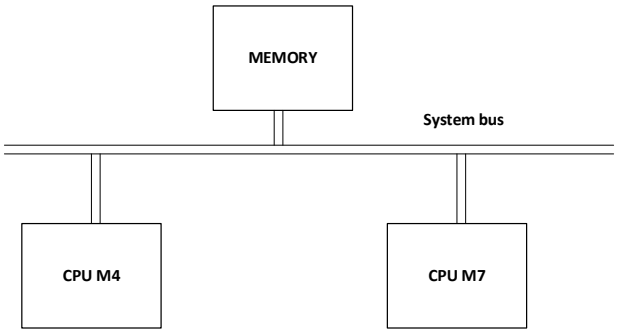

Fig. 3. Simplified logical structure of STM32H745

compiled for every core on the other core. This requirement is covered only on 50%. Binaries for M4 (according to ARM documentation) can be executed on M7 core. The other is not possible in general.

More detailed logical structure of the STM32H745 is presented on Figure 4. In details – local memories are part of the global address space but they are accessible only for the respective core. This structure enables implementation of specific customizable applications but they are not prat of the OS kernel and will not be discussed in this paper.

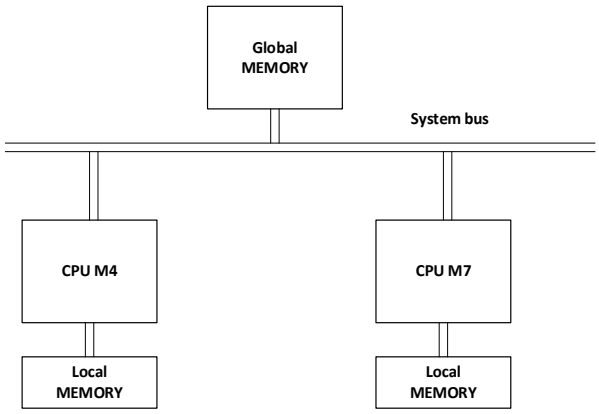

Fig. 4. Detailed logical structure of STM32H745

Experiments, testing the ability to run Cortex-M4 codes on Cortex-M7 core were implemented and they confirmed this assumption. Experiments follow the very important note [9]. For multicore implementation HARTEX is redesigned to have two stubs – one for each core. Every task is compiled for the core it has to use. Every task is marked for which core it is compiled.

- This allows tasks to be distinguished and redirected to a core on which they can be executed.
- The system scheduler and all other OS elements are compiled for the smaller core - M4.

The main difference from the single-core version of HARTEX is that now there are two active task words (ATV) - one for each core. In this situation Figure 2 is converted the latter way (Figure 5). For each core there is a word for the active task. Due to the heterogeneous structure of the system, it is necessary to know which task on which core can be executed. Basically, this is described in the TCB already when configuring the system information for each task. To maintain system performance, an additional status word "M7_only" is introduced. it is formed once at system startup and indicates which tasks can only be executed on the M7 core.

The OS kernel code is a single instance. It is used by both cores in mutually exclusive mode. The hardware facilities of the STM32H745 allow the implementation of such code. Actually, the kernel calls happen from the stubs from each core and the mutual exclusion is performed in this part of the code. The kernel works with a single copy of the tasks state Boolean vectors.

The most significant in terms of execution is the moment of the start (or continuation) of a particular task. Tasks compiled for the M7 core are only executable on it and in that sense they are simpler for start/continuation management.

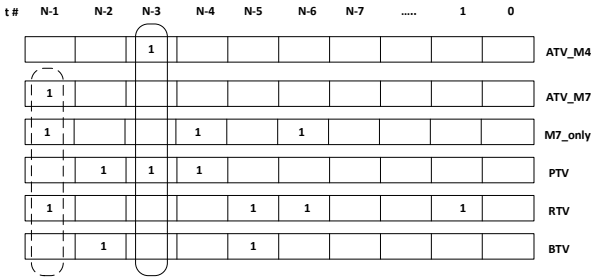

Fig. 5. Task status Boolean vectors for multicore

More problematic is the redirection of a task compiled for an M4 core onto M7. Technically, redirection of the code is not very difficult. The OS knows starting address of the task. It prepares the effective processor’s context and starts the task calling on its entry (TaskMAIN) function. The crucial point is that due to the difference in the internal structure, the context of the two cores is different and for that, such a task can after running on an M7 core only be completed there. This requires temporarily assigning the "M7_only" property to such a task. When an M4 task is started on an M7 core, this task is also marked in the Boolean vector "M7_only". Upon its completion, the OS kernel checks the TCB of that task (of every task generally) and if it is not marked for execution on M7 only, the tag in "M7_only" is removed. As a conclusion for this part, we can say - if a task for M4 is preempted, its execution continues on the core it was working on before the preemption. This allows consistent restore of the processor’s context.

LOW LEVEL ADAPTATION FOR MULTICORE EXECUTION

To avoid race conditions in OS execution, as it was pointed to before, the OS code is a single instance. It is not re-entrant in general. This is normal situation because OS resources are unique and does not provide concurrent (and this context really parallel) access. The serialisation of the OS calls is based on the embedded in ARMs event/signal mechanism

(read-test-modify-write) instructions. This allows system calls to be serialized no matter which core and which task they come from. Understanding the core type, making system call is based on the SWI (software interrupt) instruction mechanism. Each core has its own interrupt table and thus corresponding to the core ISR (interrupt service routine) functions are attached to it. The ARM documentation makes a difference between interrupts and events, but in this context, we will not get in details here.

The system API has independent subsystem that can be accesses simultaneously. This has been taken into account when creating the serialization procedures for system calls. They are separated according to their mutual dependencies.

As it was described before, all system code and user tasks are executed in supervisor more. This is dangerous in general, but usual implementations of real-time operated cyber-physical system differ much from general purpose programs and this risk is acceptable. This decision makes possible to avoid some additional loses of context saving coming from mode switching. The process of interrupt handling in ARMs is described in ARM®v7-M Architecture Reference Manual [10] and in a number of other internet and printed materials.

CONCLUSION AND FUTURE WORK

The presented here HARTEX real-time microkernel is oriented to application having relatively low number of tasks (20-30 or less) requiring fast and predictable task switching. It operated on fixed priority scheduling discipline. Multicore implementation enables task management with using all available processor resources. This increases the computer performance by increase of cores utilization.

The other use of this multi-core hard real-time kernel is for educational purposes. It is small size, observable and easy for modifications and use. Now it is oriented for students in higher Batchelor or Master degree courses oriented to cyber-physical systems and underlying real-time control systems.

The achieved results in this research and implementation include – designing of extended version of the OS kernel, enabling activation tasks on both CPU cores. A singleton kernel code controls both cores. It is executed in mutual exclusion mode on the core, making system call. Thus, system calls are executed in parallel with the user tasks occupying different cores.

Future work on this research is full implementation of HARTEX system onto the dual core computer. The very important part is preparation of adaptation of schedulability analysis theory for multicore systems with migrating between cores task and corresponding to it available HARTEX schedulability analyzes tools [11].

ACKNOWLEDGEMENT

This research is partially supported by the TU Sofia project 242ПД0023-08/2024.

REFERENCES

[1] Angelov C. K. and I. E. Ivanov (1999). High-Performance Task Management for Hard Real-Time Systems. Proc. of the Technical University of Sofia, Vol. 50-2, pp. 190-197.

[2] C. K. Angelov, I. E. Ivanov, A. Burns, HARTEX—a safe real-time kernel for distributed computer control systems, Software Practice and Experience 32(3)., pp. 209-232, DOI:10.1002/spe.435

[3] ST Microelectronics, RM0399 Reference manual, https://www.st.com/resource/en/reference_manual/rm0399-stm32h745755-and-stm32h747757-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[4] Mastering the FreeRTOS™ Real Time Kernel, https://www.freertos.org/Documentation/Mastering-the-FreeRTOS-Real-Time-Kernel.v1.0.pdf

[5] µC/OS-III , https://docs.silabs.com/micrium/latest/micrium-general-concepts/

[6] RTX Real-Time Operating System, Https://developer.arm.com/Tools and Software/Kei MDK/RTX5 RTOS

[7] Symmetric Multiprocessing (SMP) with FreeRTOS, https://www.freertos.org/symmetric-multiprocessing-introduction.html

[8] A new approach to software is needed to unleash the full power of multicore processing, https://community.arm.com/arm-community-blogs/b/high-performance-computing-blog/posts/new-approach-to-software-full-power-cluster-processing

[9] J. Yiu and R. Boys, Migrating Application Code from ARM Cortex-M4 to Cortex-M7 Processors, www.keil.com/appnotes/docs/apnt_270.asp

[10] ARM®v7-M Architecture Reference Manual, https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m7

[11] Angelov C. K., I. E. Ivanov and I. J. Haratcherev. Schedulability Analysis of Real-Time Systems under Dual-Priority Scheduling, Proc. of the International Conference "Automation & Informatics'2000", Oct. 2000, Sofia, Bulgaria, pp. 20-23, ISBN 954-9641-19-8