CERMET: A LIGHTWEIGHT MULTITHREADING BASED ON THE COMPLETELY EXPLICIT RESOURCE MANAGEMENT FOR REAL-TIME SYSTEMS

Yauhen Klimiankou
Department of Software for Information Technologies, BSUIR
Minsk, Belarus
E-mail: klimenkov@bsuir.by

Design and development of embedded and real-time applications are challenging. Multithreading allows to split of the sophisticated applications into multiple tasks with clearly defined priorities of execution. We design a new subsystem of multithreading for microkernels, that is targeted to the embedded, real-time and mission-critical operating systems – CERMet. CERMet is based on the ideas of completely explicit resource management.

I. Introduction

The experience accumulated during decades of production and usage of embedded and real-time applications tells that their design and development are challenging. Reasons for this are a stringent resource constraints coupled with demanding requirements imposed to such systems. Real-time systems require high level of determinism, predictability and robustness provided in the hardware environment with low-power processor and small amounts of memory. Complexity of embedded and real-time applications grows continuously forcing developers rely more and more on the operating system support.

Multithreading allows to split of the sophisticated application into multiple tasks with clearly defined priorities of execution. In conjunction with real-time scheduling policies multithreading constitute a main tool for management of the system temporal behavior. While dynamical multithreading is beneficial, creation and destruction of threads in run time is still the one of the sources of unpredictability in the system and introduces performance penalty.

We design a new subsystem of multithreading for microkernels, the *CERMet*, targeted to the embedded, real-time and mission-critical operating systems. It is based on the idea of the completely explicit management and control on the system resources. The multithreading subsystem of microkernel performs only transformation of system resources and provides a guarantee of their strict isolation. Full control on the system resources using during thread creation and destruction on the level of user applications leads to the higher level of efficiency, determinism and predictability with shortening and simplification of kernel code.

In this paper, we introduce the lightweight multithreading based on completely explicit resource management – *CERMet*. Specifically, we describe the underlying ideas of the *CERMet* to show its ability to support design and development of more deterministic, predictable and efficient applications in the context of limited resources of embedded, real-time and mission-critical systems.

II. Typical model of multithreading

In this section, we give an overview of the classical thread management model which is typical for the modern operating systems.

Multithreading is a well-known concept which has matured and popularized more than two decades ago in the world of *NIX family of OSes [1]. Its popularity was recently significantly boosted by wide spreading of multicore microprocessors and microcontrollers. Models of multithreading with explicit support from the OS kernel (1:1 and 1:M) are dominant nowadays. Due to this, kernels with multithreading support are de facto standard in the world of modern operating systems. This rule affects not only well-known desktop OSes [2], but also a populous world of embedded and real-time OSes [3].

Concept of threads is especially beneficial for the real-time systems [4], because such systems take special care about predictability, which they reach, in high degree, through careful scheduling of all activities going in the system and special interrupt handling techniques. Multithreading allows to perform different activities concurrently and execute each of them with its own priority. Thus, multithreading, with using special scheduling policies and careful prioritization of activities, allows to achieve more predictable timing behavior. At the same time, concept of threads reduces performance penalty and latency of task switching, which are inherent to the classical multiprocessing model. Combination of both advantages gives system developers enough design space to build advanced real-time architectures like threaded interrupt handling.

Classical thread management model inherited by POSIX from early UNIX implementations is built around three operations: creation of a thread, waiting a thread termination and termination of a thread [1]. Creation of a thread is a synchronous operation, during which kernel implicitly transforms basic system resources into new high-level system resource – child thread. Destruction of a thread, including its disassembling into basic system resources, is performed in two steps: termination (performed by child thread) and disassembling (performed by parent thread), and requires rendezvous between parent and child threads. Parent thread states to the kernel its intention to block itself and wait child termination using *Wait Thread* operation. On the opposite side, child thread invokes *Exit Thread* operation to terminate itself execution. Once parent is in waiting state and child in the exit state, the rendezvous is established. Reaching the rendezvous leads to unblocking of the parent thread and disassembling of the child thread into basic system resources. This disassembling performed by kernel implicitly.

III. Approach used by the *CERMet*

Thread is a high-level system resource which is built from physical memory, virtual memory (slot in scheduler table of threads) and CPU time share. *CERMet* exploits the ideas of explicit resource management introduced by seL4 for management of all three kinds of system resources.

Determinism. Management of system resources that is hidden in the kernel creates a lack of determinism on the application side. Indeterminacy is a result of lack of visibility of kernel state from the application and lack of the strict inter-process (inter-thread) resources isolation. Applications in such systems run with a permanent risk of failure due to the exhaustion of system resources (Deny of Service attack).

Predictability. Actual number of processor ticks spending in creation and destruction of the thread strongly depend on the actual state of shared resources in the moment, when operation is performed, and from the adopted by kernel resource allocation/deallocation policies. Due to this, it is hard to provide guarantees about maximum number of processor cycles required for each call of the operation in the dynamical system, as well as, maintain such guarantees in long-term perspective with constantly evolving OS kernel. Lack of performance isolation of thread management operations between processes of the system and complex nature of its dependency from the OS kernel creates significant challenge for real-time applications.

Thread construction and termination. In *CERMet* all three kinds of basic system resources involved into thread management are explicitly managed by user space applications. Due to this, application is able to allocate and to reserve all required system resources before start of actual functioning or die explicitly.

Construction of the thread consumes computer resources of all three kinds which are owned by parent. Destruction of the thread releases them back to the parent allowing their reuse in future. *CERMet* moves lookup and accounting of the thread table slot and physical memory from the scope of the kernel responsibilities into scope of application responsibilities. Thus, kernel path of the thread management operations become straightforward and strictly bounded in time. On the another hand, *CERMet* allows application which have full info about resource management policies to make entire Thread Create operation strictly bounded in time and thus completely predictable and deterministic.

CERMet splits the *wait()* operation into two independent operations: waiting of thread termination notification and disassembling of the thread into basic system resources. Such splitting provides to the application full control on the time of disassembling and released resources.

Kernel designed with *CERMet* performs only resource ownership control and functions of resource transformations and does not contain any resource allocation policies. Explicit identification of the resources which are converting into the thread allows eliminate generally unbounded lookup operations. As a result, real-time application becomes able to high level of resource and performance isolations between threads, and thus high level of determinism and predictability, and full control on the owned resources. Finally dependency of the application time behavior from the kernel becomes straightforward, thus maintaining of the application in the context of the constantly evolving OS kernel becomes simple.

IV. Conclusion

CERMet multithreading subsystem based on the ideas of explicit resource management, significantly improves control of application on the resources, temporal behavior and determinism of the system. Furthermore, it simplifies design and implementation of the multithreading subsystem of the operating system kernel. Embedded and real-time systems can gain especial benefits from the improved level of determinism, predictability, resource isolation, performance and minimalist footprint of the *CERMet*. At the same time *CERMet* can find its application in the domain of general purpose microkernels. We are currently working on improving the application level API and runtime library for our experimental kernel which will support *CERMet* on applications side.

1. D. Marshall. Programming in C. https://users.cs.cf.ac.uk/Dave.Marshall/ C/, March 1999. [Online; accessed 09-October-2016].
2. R. Love. *Linux System Programming: Talking Directly to the Kernel and C Library*. O'Reilly Media, Inc., 2007.
3. *eCos Reference Manual*, 1.3.1 edition, 2008.
4. D. Beal. Linux As a Real-Time Operating System. *Freescale Semiconductor White Paper*, 2005.
5. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.