Feature and Performance Analysis and Enhancements of the DECOS Core OS

Maximilian Rosenblattl[1], Andreas Wolf[1], and Bernhard Leiner[1]

TTTech Computertechnik AG
Schoenbrunner Strasse 7, 1040 Vienna, Austria,
phone: +43 1 5853434 0, fax: +43 1 5853434 90,
bernhard.leiner@tttech.com

Abstract. This paper presents an overview of the features of the Core Operating System (COS) of the Encapsulated Execution Environment (EEE) developed in DECOS (Dependable Embedded Components and Systems), an integrated project within the Sixth Framework Programme of the European Commission.
For all features, the current status of the tool support is shown, combined with how these features are used in current applications.
Additionally, this paper presents ideas to improve the performance of the COS by adapting the COS to the way current applications use it and by removing features currently not in use. Also, reducing the feature set simplifies the configuration and potentially enhances the reliability.

Keywords: Embedded Systems, Dependability, Virtualization.

1 Introduction

The Core Operating System (COS) of the Encapsulated Execution Environment (EEE) developed in DECOS [1] (Dependable Embedded Components and Systems), an integrated project within the Sixth Framework Programme of the European Commission, has a rich feature set while providing a high level of protection for the encapsulated partitions. When designing the COS, the focus was on building an operating system that ensures protection of user partitions, mainly driven by reliability and safety considerations [2]. However, some requirements specify huge flexibility of the execution environment at runtime by requiring reconfiguration of e.g. timers and external events.

Interestingly, dynamic reconfiguration is not used by the project partners and many operating system features are used very rarely because of the very low performance of system calls.

This paper presents the current feature set of the COS and discusses possible enhancements with an estimation of their impact on performance.

It is shown that a strictly deterministic time-triggered approach has advantages over the current implementation regarding performance, determinism and reliability, but reduces the feature set to a minimum.

2 Current Features of the COS

The design concept and most of the implementation details of the DECOS Encapsulated Execution Environment Core OS (COS) are described in [3], so this work will directly go into detail about the features with their tool support status and their usage in current applications.

Several features are rarely used by current applications [4]. One of the reasons is the non-negligible time for checking partition data passed to the COS via system calls. Since the COS has access to all memory areas, these checks ensure spatial protection of other partitions.

2.1 Timer Events

The COS offers timed notifications for every partition. These timer events may be periodic or not and may be based on any of the time bases *Coretime*, *Localtime* and *Partitiontime*. *Coretime* provides a time based on the core cycle, so a timer based on it will raise an event notification every core cycle at the same distance to the beginning of the core cycle. *Localtime* is a free-running time starting with the first core cycle. *Partitiontime*, finally, is the part of *Localtime*, which runs inside the partition. So, if the partition time is over, *Partitiontime* is stopped and continues running when the partition becomes active again.

In principle, all event handlers have a priority and the COS ensures that an event with lower priority does not interrupt an event handler with higher priority. In fact, the check cannot be performed before the timer interrupt arises and the COS is activated. So, a high-priority event handler is always interrupted if it does not end before the next timer event occurs.

Although all timer events are dynamically configurable, this is not used by most applications. It would be quite hard to prove that the system, from the application's point of view, acts still in a reliable and deterministic manner. The only deterministic way to configure a timer at runtime is disabling and enabling a timer event (e. g., to receive this timer event only every nth Core Cycle).

Tool support exists for periodic, non-configurable timers with time base *Coretime*, which is the most important case. It is used to position user tasks within the partition time windows. Additionally, all event handlers have the same priority and the integrator ensures (or tries to ensure) that every event handler has finished before the next one starts. Other time bases have no tool support yet but the integrator may modify the COS configuration to add additional timers for dynamic configuration during runtime.

2.2 Interrupt Events

Partitions may want to be notified if external interrupts occur, so the COS is able to raise partition events when interrupts arise, but only those interrupts the partition wants to receive. So, a partition without the need for interrupt notifications will not be interrupted.

Currently, there is no tool support for interrupt events, but the integrator may modify the COS configuration to enable them. Most applications rather use polling to have more control over the execution and to reduce the interrupt latency time arising from the COS activation, although this has the disadvantage that the partition has to be set to the higher privileged user mode 1.

2.3 Message Channels

Message channels allow transferring data between partitions via COS system calls. They can be of type *Sampled*, where exactly one message is in the channel, which can be overwritten, but not deleted, or of type *Queued*, where they act as a message queue. For every message channel, reading and writing can be allowed or denied for every partition separately.

For performance reasons, current tools configure shared memory on the embedded computer or the Communication Controller (CC) instead of using message channels [5]. Here, the spatial encapsulation is ensured by hardware memory protection. For the partition, using internal, external or CC memory is completely transparent because of the unified address space.

Queued message channels are very useful for event messages that will not be produced periodically, but the use of *Sampled* message channels is limited to platforms where shared memory is not applicable. Since *Sampled* message channels are just a limited subtype of the *Queued* ones, both are provided.

2.4 Health Checks

Health checks provide extended watchdog functionality. They allow errors to be thrown when a partition reaches a defined state too early or too late (or not at all). There may be several watchdogs defined, where each of them has to be served by the system call function `tt_eee_si_serve_health_check`, which takes a lot of time. In addition, a very fast, but rather coarse-grained, check is provided for event handler functions: If an event handler function is still running at the end of a partition time window, a configurable error object is thrown.

Current tool support exists for the second functionality, which is, for performance reasons, the only one used by current applications.

2.5 Error Handling

Error objects are provided to notify either the partition or the COS that something has gone wrong. This could be a missed health check, an unfinished event handler, a system trap or some error condition detected by the partition. Upon raising an error, the subsequent actions could be cold-start, warm-start, shutdown or even stop the partition or the whole system or notify the current partition about that error. Every error object may be changed on runtime, but not to system-wide error actions.

Tool support exists for mandatory errors objects (system trap errors and a default error per partition). Application-specific error objects are not used by

current applications, but may be added to the COS configuration by the system integrator.

2.6 Partition State Functions

Five partition states exist for the COS: cold-start, warm-start, running, shutdown and stopped. For the first four of them, functions can be configured to handle the current state. The cold-start function is normally used to initialize a partition, whereas the warm-start function can only be entered after recovering from an error. The main function, handling the running state, should do all the partitions work but can be interrupted by timer or interrupt events. The shutdown function is called right before the partition or the system is shut down (which is equal to switching into the state stopped).

Currently tool support exists only for the cold-start and the main function, since no other functions are needed by current applications. The cold-start function can be programmed by the partition creator, who can even configure the expected duration for the cold-start function. But, in contradiction to the intended use, the main function is configured to return immediately, resulting in a system call after the return. This system call is necessary for the COS to get informed when the main function ends (see 2.7 for details). The whole functionality of the partition lies within the timer event handler functions.

2.7 Functions in general

Every function invoked by the COS, including state functions, error handlers and event handlers, is configured to return into a system call to inform the COS about function termination. In general, a function invocation is done by inserting a new Context Save Area (CSA) into the Context Save Area List [6] with the Program Counter (PC) set to the function's start address. To ensure that a notification system call is done after function termination, the COS inserts a special System Call CSA into the list right before the function's CSA, so a `RET` instruction will load this System Call CSA and perform a system call into the COS. The main need for these notification system calls is to check event handler priorities and to invoke or resume the event handler with the highest priority pending. If no event handlers are pending, the main function will be resumed.

3 Performance Enhancements

This section presents some ideas to increase the performance of the COS that arise from knowing that those features are not used by current applications. Of course, these improvements would require a modified configuration model, but they would not change the DECOS requirements of temporal and spatial encapsulation. When following all of the ideas presented here, the result would be a reduced COS (referred as RCOS in this work). Future projects will have the ability to use the RCOS with higher performance or the original COS with its larger feature set.

3.1 Change the Definition of Partitions

In the COS approach, the *whole functionality* of a partition lies within the partition's *main function*, which could be *assisted* by the event handlers. This results in two timers for each partition time window, one to start it and one to end it, and the need for a main function being called every time window activation. Additionally, every function invoked by the COS, including the main function, is configured to return into a system call to inform the COS about function termination (see 2.7 on the preceding page for details). Since the main function is not used for code currently, this system call is completely useless.

The following simple redefinition would fit the current usage better and allow enhancements: The whole functionality of a partition lies within the event handlers, with a *background task* running, which can always be interrupted. This background task could, for example, perform an endless loop or calculate some checksums or initiate a switch to sleep mode.

The benefits of this different meaning are that only a single partition switching timer has to be used instead of two, the main function is no longer required to be started every partition activation.

Impact Switching out of a partition takes $18\mu s$, switching into a partition takes $20\mu s$ in the best case [3], resulting in a total partition-to-partition switching time of $130\mu s$. Changing this to a single partition switch would reduce the partition switching time to the time to switch out of a partition, $18\mu s$ (this is because the COS prepares the next partition but does not start it).

The simple definition change from the important but unused partition main function to the partition background task for lowest-priority activities eliminates the necessary system call at the end of the main function and reduces the overhead from $22\mu s$ system call duration to zero.

The total benefit would be $22\mu s$ of additional time for event handlers in each partition time window and $20\mu s$ less overhead between partitions.

3.2 Use only *Coretime*

At the moment, *Coretime* and *Partitiontime* are calculated from *Localtime*, although all current applications use a cyclic time base, which would be fit best by *Coretime*. A large amount of time is used to convert the different time bases into each other, which involves fairly slow 64-bit operations. If *Coretime* would be used as the only time base, 32-bit operations would be sufficient under most circumstances (if you create a core cycle longer than 28 seconds, you will probably not think about performance anyway).

3.3 Allow only a Single Task

The priorities of event handlers are of no use for current applications, so the RCOS should not allow event handlers with overlapping execution times. If it

comes to the condition that an event handler should be invoked when another event handler is still running, a preconfigured error should be thrown.

If only one concurrent event handler is allowed in the RCOS, the priority checking (see 2.7 on page 4) is no longer necessary, so it would be sufficient to return right into the background task (formerly main function). The RCOS just needs to check a running event handler when it attempts to invoke a new one and throw an error object if one is found. This check could be easily done by counting Context Save Areas.

Impact The restriction to a single event handler eliminates the necessary system call at the end of every event handler and reduces the overhead from $10\mu s$ system call duration to zero.

The total benefit would be $10\mu s$ of additional time for each event handler in each partition time window.

3.4 Deny Dynamic Modifications

If every timer event is static, the list of timer events could be sorted by time and the RCOS needs only to check one timer instead of walking through the whole list, which reduces the complexity from $O(n)$ to $O(1)$. The only dynamic modification should be enabling and disabling timer events.

By reducing the number of concurrent timer event handlers to one, it would even be possible to configure a Context Save Area (CSA) template and append a copy of it to the CSA list when the event handler should be called, saving the time for collecting all information to put into the CSA.

Finally, templates for the hardware memory protection sets of the Infineon TriCore 1796B [6] could be created statically and simply written to the memory protection unit on partition switch, saving time for collecting the partition protection information.

4 Estimated Impact

The biggest impact would be that dynamic modifications are no longer possible, which is not a big difference for current applications.

Since this paper presents ideas only, performance and code size gains or losses can only be estimated. By a first rough estimation, the code footprint would be reduced by 35 to 45 percent if all enhancements stated before are applied.

4.1 Performance Gain

An estimation of the performance gains is not that easy, but the minimum reduction would be about $64\mu s$ additional time for each partition (by definition change, see section 3.1 on the page before) and about $22\mu s$ per partition time window plus $10\mu s$, multiplied by the number of timer events raised in this partition time window, additional time for each partition time window. The gain of

the faster system calls is very hard to estimate, but a reduction of at least 20 percent is feasible just from reducing the number of spatial encapsulation checks, which are no longer necessary in the RCOS.

4.2 Different System Interface (SI)

The System Interface (SI) of the COS needs a lot of time to check all access rights for the calling partition. If the SI functions have no need to deal with pointers or return values, it would be faster and much easier. The only SI functions provided should be:

- `tt_eee_si_enable_timer(<timer ID>)` enables the given timer object. Only a check if the timer belongs to the calling partition is necessary
- `tt_eee_si_disable_timer(<timer ID>)` the opposite of the above
- `tt_eee_si_sleep()` would enter a sleep mode out of the partition's background task until a timer event is raised or the partition's current time window ends
- `tt_eee_si_raise_error(<error ID>` raises an error. Only a check if the partition is allowed to raise the error is necessary.
- `tt_eee_si_set_sync_offset(<duration>)` is needed by a privileged partition to tell the sync partition what to do.

It is recommended that the new SI functions do not return anything (type `void`), so the RCOS has no need to check the access rights for the return value's destination address, which was necessary for spatial encapsulation. The only requirement would be that the partition functions are called with the current *Coretime* by the RCOS and that they get the current time from somewhere else than the RCOS. This is not such a big difference, since the current SI functions cannot return an appropriate time value because of their latency.

4.3 Feature Reduction

The following features are removed completely, since they are not in scope of the RCOS, but remain in the COS.

- Health Checks (see 2.4 on page 3), they are replaced by the faster but more coarse-grained checks introduced in section 3.3 on page 5.
- Message Channels (see 2.3 on page 3), shared memory is used instead of the COS message system.
- Interrupt Events (see 2.2 on page 2), current applications rather use polling over interrupts.
- The partitions' *main functions*, *warm-start functions* and *shutdown functions* (see 2.6 on page 4). Current applications use neither warm-start not shut-down functionality.

5 Conclusion

The current implementation of the DECOS Encapsulated Execution Environment (EEE) Core OS (COS) has quite poor performance due to temporal and spatial encapsulation issues combined with a rich feature set (for an exokernel [7]).

It is shown that most of the implemented features are not used by current applications, especially the features of dynamic reconfiguration on runtime. In section 3 on page 4, ideas are presented to improve the performance by adapting the COS's feature set to the current usage.

When followed, the ideas presented in this work can lead to a Reduced COS (RCOS) with improved performance compared to the COS. RCOS would retain the temporal and spatial encapsulation requirement of DECOS, and could be used without major changes to current applications or tools. Future projects would have the ability to use the RCOS with higher performance or the original COS with its larger feature set.

References

1. R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (Journal of the Austrian professional institution for electrical and information engineering)*, 3, March 2006.
2. Martin Schlager, Erwin Erkinger, Wilfried Elmenreich, and Thomas Losert. Benefits and Implications of the DECOS Encapsulation Approach. In *Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems*, pages 13–18, September 2005.
3. Martin Schlager, Wolfgang Herzner, Andreas Wolf, Oliver Gründonner, Maximilian Rosenblattl, and Erwin Erkinger. Encapsulating application subsystems using the decos core os. In Janusz Górski, editor, *SAFECOMP*, volume 4166 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 2006.
4. B. Huber, P. Peti, R. Obermaisser, and C. El Salloum. Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems*, May 2005.
5. R. Obermaisser and P. Peti. Realization of virtual networks in the DECOS integrated architecture. In *Proc. of the Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS)*. IEEE, April 2005.
6. Infineon Technologies AG. TriCore 1 32-Bit Unified Processor Core Volume 1 (of 2): V1.3 Core Architecture, User's Manual. `http://www.infineon.com/upload/Document/TriCore_1_um_vol1_Core_Architec%ture.pdf`, October 2005. V1.3.6.
7. Dawson R. Engler, M. Frans Kaashoek, and James OToole Jr. Exokernel: an operating system architecture for application-level resource management. M.I.T. Laboratory for Computer Science. Cambridge, MA 02139, March 1995.