Security and Safety Considerations for the DECOS Core OS

Andreas Wolf[1], Maximilian Rosenblattl[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 safety and security considerations for 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.
It is shown that security and safety is well considered in the COS and a high level of security and safety can be achieved when systems using the COS are designed properly.
Keywords: Embedded Systems, Security, Virtualization.

1 Introduction

Reliable embedded systems have high safety requirements, especially in the automotive, aerospace or industrial control domains. As embedded systems are getting more complex with each generation and more interactivity and network connectivity are proposed, security considerations must be taken. While traditional embedded systems work in a well-defined, separated network with its own communication links and predefined environment, modern systems shall be able to work in heterogeneous networks and shall be configured dynamically. Here, security becomes a big issue as hardware and software components of different vendors are mixed and standard communication systems are used.

In the DECOS project (Dependable Embedded Components and Systems), an integrated project within the Sixth Framework Programme of the European Commission, mixed criticality subsystems are integrated into the same embedded hardware node [1, 2]. While safety and reliability were covered by design requirements, security and protection of intellectual property were not explicitly mentioned.

This paper covers security considerations, which also apply to the protection of intellectual property of the integrated subsystems by analyzing the design and implementation of the Core Operating System (COS) of the DECOS Encapsulated Execution Environment (EEE).

The security considerations cover bogus and malicious subsystems on the same node as well as robustness to security attacks from outside while still considering the safety of the whole system.

As a result of the analysis, an overview of vulnerabilities is given and possible solutions as well as enhancements of the COS are presented.

2 Definitions

The terms security, safety and intellectual property are defined in different ways in literature, depending on the context in which they are used. To assure correct understanding, these terms are defined in detail regarding their meaning in this paper. According to [3, 4], safety and security are properties of dependability.

2.1 Safety

Safety refers to the ability of a system to avoid erroneous behavior and damage.

2.2 Security

Security is in some ways related to safety, although this term is usually used as protection against intentional manipulation or intrusion. According to [5], security addresses confidentiality, integrity and availability. While confidentiality is a classical security issue, integrity is partly and availability mostly covered by safety requirements.

2.3 Intellectual Property

In this paper, intellectual property is understood as precious goods such as program code, data or other aspects of a subsystem that needs to be protected against unauthorized use. Therefore, the protection of intellectual property is a subset of security.

3 System Architecture

The architecture of the DECOS EEE Core Operating System (COS) is on the one side quite hardware-independent, but most security-related modules are designed for the specific hardware platform that is used. Therefore, the features used on the hardware platform are presented first:

3.1 Hardware Features

The used hardware, an Infineon TriCore 1796B [6, 7], provides numerous security features that are rather untypical of embedded platforms. It uses a unified memory address space, applying memory protection with read, write and execution rights. Three user access modes exist, in which different levels of access rights to I/O peripherals and internal system components and configuration registers

are applied (`superuser`, `user1` and `user0`). Although I/O, internal system components and configuration registers are mapped to the unified memory address space, the memory protection is not applied to them so only the non-configurable access rights of the user access modes are applied.

The internal flash memory has ECC support, detecting two-bit errors and correcting single bit errors. Internal SRAMSs support parity checks. Both features are currently not used by the COS.

The TriCore architecture uses context save areas (CSAs) instead of the stack to save the context of a process. To be exact, the higher half of the register file (upper context) is saved on every function invocation and linked to the context list of the current process. The upper context contains the return address register, the stack pointer, the program status word and the link word (links to the previous CSA in the list) as well as the upper data and address registers. The CSAs are stored by the CPU into the LDRAM, a special memory with a fast connection to the register file, enabling the saving and restoring of an upper context within six CPU cycles.

The TriCore supports two hardware protected software tasks (that are referred as `task0` and `task1`) at a time with different access modes and memory protection sets. Every task has its own set of context save area lists and pointers.

3.2 Core Operating System

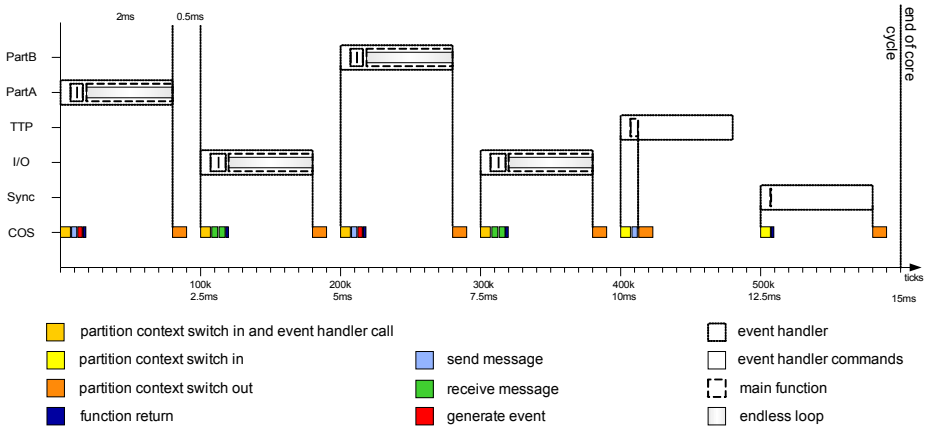

Fig. 1. Example of a Core Schedule with Runtime Profile

The Core Operating System (COS) [8] of the DECOS EEE is designed as an exo kernel [9], supporting only basic functionality and separating user partitions in time and space. As no further hardware abstraction is provided, the partitions need to access the hardware they need directly. It was usually planned to permit a partition exclusive access to particular system components, ensuring

the protection from unauthorized access. Unfortunately, the TriCore protection system does not support such access restrictions, as access to peripherals and system components is only restricted by the user mode of a task.

Using the abilities of the TriCore architecture, the following design decisions were made: Partitions may only run in user mode 1 (`user1`) or user mode 0 (`user0`). The first permits access to peripherals and system components. Also, the global interrupt-enabled flag may be changed. In user mode 0, no access to peripherals or system components is possible. Therefore, untrusted partitions may only run in user mode 0.

The COS itself runs as `task0` in superuser mode (`superuser`), in which full access to all memory locations as well as to configuration data is permitted. The COS manages all partitions as tasks with separated context save area lists. To execute code of a partition, `task1` is configured to have the appropriate access rights, user mode and context save area list. Also, the invocation of the right function is enforced by the COS and interrupts are enabled or disabled according to the configuration of the partition.

System calls, interrupts and system traps always invoke the COS (see figure 1 on the preceding page). Once the COS is active it cannot be interrupted by other incoming interrupts. They are handled after the COS system function returns. Multiple occurrences of an interrupt may be lost.

As the COS has full access rights, parameters of incoming system calls need to be checked for validity regarding their values. Data pointers need to point to a memory area the partition is permitted to access, function pointers must point to code the partitions is allowed to execute and so on.

Partitions itself have one or more predefined time windows within a schedule of fixed length. Within those time windows the partition may execute code and receive events such as timer events, partition events or external events (interrupts). Interrupts are not handled directly by the partition, because all interrupts are received by the COS. Only those interrupts the current partition is configured to receive are enabled and may be detected. The COS is responsible for the invocation of the event handler. Every task within a partition has a priority level, meaning that only events with higher priority than the currently running task can be invoked instantly. Otherwise, they are delayed until their priority level is the highest of all waiting tasks of the partition. Nevertheless, the execution of the current task of the partition is interrupted by the COS because of the interrupt handling and priority checking, which is done when the interrupt is received by the COS. Partitions may change event priorities, but as every partition has its own event set with its own priority values other partitions are not affected.

Inter partition communication may be done through the COS, which provides message channels for sampled and queued messages. Sampled message channels hold only the latest value written to them, enabling to read the value at any time. There may exist more than one reader. Queued message channels have one sender and one consumer. The message is removed from the queue when read. There may exist more than one consumer, but the message is consumed

at the first read and therefore can only be received by one of the consumers. As for every communication operation a system call needs to be performed and access rights and message data need to be checked, message channels are very resource-consuming and currently not used by user applications.

The high-performance solution of inter-partition communication is using shared memory. Using the memory protection, this enables maximum performance while keeping a high level of security, as every partition may have read and write memory areas where it may exchange data with other partitions. Unfortunately, the maximum number of memory areas with access rights is limited by hardware. It would be possible to change the set of memory areas on runtime, but this would need a system call, which is also a strong performance penalty. Therefore, it is not implemented yet, but may be possible to implement with only a few modifications to the COS.

A very special way of inter-partition communication is enabled by the used communication controller, which applies a hardware fault-tolerant layer (HFTL). The memory of this controller is also memory-mapped an the memory protection of the TriCore is applied. The HFTL controller has its own schedule and is able to copy data based on a predefined schedule. This data may be incoming messages from the field bus (currently LTTP or FlexRay) or from other partitions. The configuration of the HFTP controller is done by writing a message description list (MEDL) into the configuration memory area of the controller.

4 Security Analysis

During runtime, a partition may be well separated from other partitions, strongly depending on the configuration data of the COS. A partition may only access memory areas it is allowed to. By default, a partition even cannot read its own executable code. Also, the stack is only used for function local variables. The return address, the stack pointer and so on are saved in the context save area list, which is stored by the CPU to special memory location where no partition but the COS itself has access to.

I/O access may be prohibited by setting the access mode to user mode 0. In that case, the partition may need another partition with user mode 1, which performs I/O access for it and communicate with that partition through shared memory or COS message channels. Partitions with user mode 1 have comprehensive access to I/O and may even deactivate global interrupts. This is a serious problem as the COS depends on the system timer interrupts, which provide the time base for all time-triggered actions. So it is possible for a partition to prevent the COS from being instantiated, literally freezing it and all other partitions. This may not be a security, but more a safety issue as it mostly concerns availability.

By the possible strong encapsulation it is hard, if not impossible for a partition to manipulate or intrude into other partitions. This limits the possible damage if a malicious partition may be present or a erroneous partition may be exploited from outside. The effectiveness strongly relies on the configuration of

the COS, which defines memory protection and access rights for the partitions. The configuration generated by the used tools is quite restrictive and provides a high level of possible security. Nevertheless, it must be ensured that the configuration data is not manipulated.

Information flow security is only partly provided. To be exact, only basic functionality is provided by message channels. As mentioned in [10], security policies are hardly implementable and no assertions can be made upon the completeness of such a policy.

5 Conclusion

The security level provided by the COS mainly relies on the used configuration. A restrictive configuration allows a maximum of possible security. Nevertheless, it may be necessary to grant partitions access rights that enable partitions to violate security restrictions. This is mainly due to performance issues and hardware limitations and cannot be easily compensated by a modification of the operating system design.

Anyway, there exist some ideas to possibly further enhance the security level by reducing weak points:

- Using the watchdog timer to detect if a partition with user mode 1 (`user0`) has disabled the system timer.
- Apply some minimum value range checks for protection set entries, such as read, write and execution rights of partitions. Partitions must not have access rights to COS memory areas, therefore the configuration could be checked for violation of these requirements on startup.
- Encryption of the configuration and possibly parts of the COS code would raise the protection against offline manipulation. Startup code, encryption routines and encryption keys could be placed into locked flash areas (one time programmable). These startup routines will then place the decrypted code and data into code and data RAM locations and proceed execution there.

References

1. 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.
2. Hermann Kopetz, Roman Obermaisser, Philipp Peti, and Neeraj Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical report, Vienna University of Technology, Austria, and Darmstadt University of Technology, Germany, 2004.
3. J.C. Laprie. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, 1992.
4. A. Avižienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January–March 2004.

5. Stewart Lee. Essay about computer security. Technical report, Centre for Communications Systens Research Cambridge, 1999.
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_Architecture.pdf, October 2005. V1.3.6.
7. Infineon Technologies AG. TC1796 32-Bit Single-Chip Microcontroller Volume 1 (of 2): System Units, User's Manual. http://www.infineon.com/upload/Document/cmc_upload/documents/011/9544/tc1796_um_v1.0_2005_06_sys.pdf, June 2005. V1.0.
8. M. Schlager, W. Herzner, A. Wolf, O. Gründonner, M. Rosenblattl, and E. Erkinger. Encapsulating application subsystems using the DECOS core OS. In *The 25th International Conference on Computer Safety, Security and Reliability (SAFECOMP)*, pages 386–397, Gdansk, Poland, September 2006.
9. 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.
10. J. Rushby. Partitioning in avionics architectures: Requirements, 1998. J. Rushby. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance. Draft technical report, Computer Science Laboratory, SRI International, October 1998. http://citeseer.ist.psu.edu/article/rushby99partitioning.html.