A Study on Embedded Operating Systems for Automotive Electronics

Xinbo ZHANG*, Feng LUO

Clean Energy Automotive Engineering Center*, School of Automotive Studies,* Tongji University, No. 4800 Cao'an Road, 201804, Shanghai, China

Abstract — **Embedded real time operating systems are becoming increasingly more popular in automotive electronics. At the same time, there is a protocol to define the features and implementation of real time operating systems. This paper studies the requirements of real time operating systems, their key features and how to implement them, also how an operating system is applied and validated. Finally the study compares and contrast several popular embedded operating systems, so that the most important features of operating systems in automotive electronics are summarized. The study results are very helpful for the new applications of operating systems in electronic controllers.**

Keywords- OS; Schedule; Automotive Electronics; Message; Robustness

I. MAIN EMBEDDED REAL TIME OPERATING SYSTEM INTRODUCTION

Over the last decade the functionality and complexity of embedded software systems has increased dramatically. Examples of such systems are electronic engine management, anti-lock braking systems and gear box control. Supported by the ever growing computing power of microcontrollers, this development has led to a rapid increase in the size and complexity of controller software. In the past, the achievable level of functionality was determined by the hardware and the performance of the microcontroller. Now the software development process is more and more becoming the limiting factor. To meet these challenges, a reliable and efficient real time operating system (OS) as a runtime environment is needed.

The embedded real time operating system [1] [2] in automotive electronic field follows OSEK protocol. Now more and more products comply with AUTOSAR protocol [3], in fact for AUTOSAR basic software definition, OS component still keeps OSEK definition.

At present popular OSEK OS solutions include: ERCOSEK/RTA-OSEK from ETAS, osCAN from Vector, OSEKWorks from Wind River, etc. And also these operating systems are integrated into AUTOSAR solution. Software architecture with real time operating system is like following figure 1.

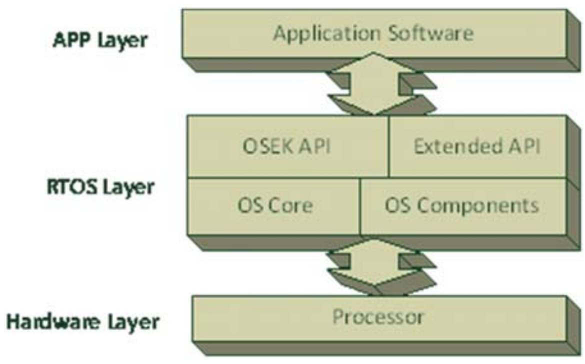

Figure 1. Software architecture with real time operating system.

II.. REQUIREMENTS AND FEATURES OF OS IN AUTOMOTIVE ELECTRONICS FIELD

A. *Real-time capability and support*

Real time OS manages the execution of concurrent application tasks which are subjected to real-time requirements [1] [4]. As a prerequisite the operating system responds within a guaranteed latency period to requests. Furthermore a priority-based scheduling of tasks serves to guarantee their respective timing requirements, which may cover a broad spectrum ranging e.g. from several 100 milliseconds down to a few microseconds. The pertaining features of OS are priority based scheduling and preemptive scheduling, guaranteed response times.

B. *Real-time capability and support*

OS provides means to recognize faults which may occur as a result of external and internal disturbances (e.g. EMC, overload, software errors). General fault tolerance features are provided to handle overload situations gracefully.

Furthermore, specific reactions may be triggered to handle faults. These are defined application-dependent. If a fault cannot be tolerated or handled by local measures, the system is transferred to a safe state (e.g. reset state). The pertaining features of OS are: system supervision, deadline monitoring, guaranteed minimum inter arrival period, exception handling.

The safety of operating system is also an important requirement about functional safety [5] [6]. With the increasing functionality and complexity of automotive electronics more risks of functional safety are introduced. It is necessary to perform the functional safety process throughout the safety lifecycle in this field. In addition, the release of functional safety standard ISO 26262 also make functional safety as an important requirement of the design and implementation process for automotive electronics products [7].

C. *Efficiency*

OS allows for a most efficient use of the controller resources without compromising maintainability and extendibility of the software. The pertaining features of OS are: concept of tasks as static schedule sequences of processes, cooperative scheduling, static system configuration and tool support.

III. IMPLEMENTATION OF OS FEATURES

A. *Scheduling*

Scheduling [8] [9], i.e. the planning of the execution order of processes, is a core function of real-time multitasking operating systems. Dynamic or run-time scheduling is needed to enable a fast and flexible handling of requests while efficiently utilizing the processing power of the pertaining CPU.

The most fundamental distinction with scheduling is whether it is static or dynamic. With static scheduling, the scheduling algorithm has complete knowledge of all tasks and their constraints. A dynamic scheduling algorithm on the other hand has only knowledge of the ready processes but it has no knowledge of future activation times. Since new processes may become ready spontaneously, the scheduler has to decide at runtime which process has to be selected among the ready processes.

The advantage of dynamic scheduling over static scheduling is its flexibility to react on external events. Especially the effectiveness of static scheduling decreases with a decreasing latency period. Disadvantages of dynamic scheduling are higher computational requirements and the memory demand for the management of processes. By supporting static as well as dynamic scheduling, OS allows the implementation of combined strategies which are optimal with respect to the application requirements for response time and memory demand.

Dynamic scheduling of tasks is managed according to a task state model, cf. Figure 2. Upon activation, a task is transferred to the ready state. If its priority is higher than that of the running task and a task switch is possible, it is assigned or started2, respectively, while the latter task is preempted, i.e. transferred to the ready state. Upon termination of the running task, i.e. upon its transition to the suspended state, the ready task with highest priority and the top position in a pertaining FIFO queue is started or continued (the latter if it had been preempted).

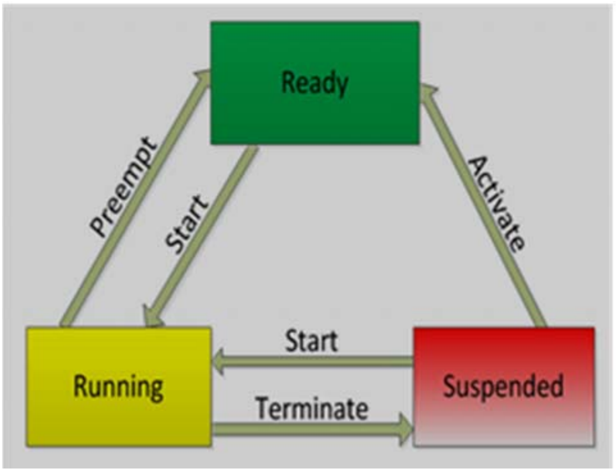

Figure 2. Task states and transitions

There are two possible strategies when to switch from an executing task to a ready task with higher priority. The first strategy, called cooperative scheduling [8] [9], switches execution at predefined points in the software. These predefined points are the borders between processes within a task. If necessary, it is also possible to implement switching points by operating system calls. Since the scheduler has to wait until the running process is ready and thus cooperates with the application software, the scheduler is said to be cooperative, see Figure 3.

With the second strategy, called preemptive scheduling, execution can be switched within processes at the boundary of machine instructions (under the assumption that interrupts are not disabled). The scheduler is therefore able to suspend the currently executing process within a task and to start the execution of a task with higher priority, see Figure 4.

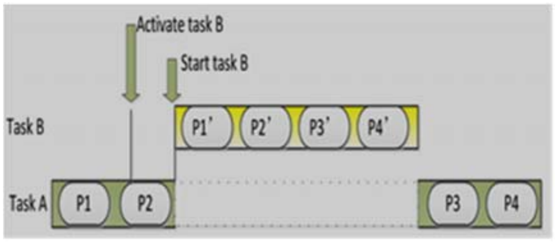

Figure 3. Cooperative task scheduling

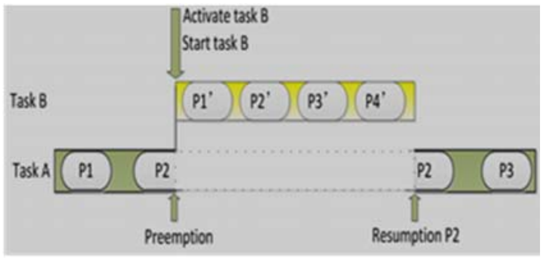

Figure 4. Preemptive task scheduling

OS therefore supports a combination of cooperative and preemptive scheduling. This allows selecting the most effective combination for a certain application. Typically, only a small amount of the application has short latency requirement which requires preemptive scheduling while the large remainder can be scheduled cooperatively. The necessary amount of memory resources can thus be minimized while guaranteeing the application specific real-time requirements. This is realized by a hierarchical scheduler concept where the cooperative scheduler is subordinated under the preemptive scheduler. The cooperative scheduler is treated as a single task at the lowest priority level of the preemptive scheduler. The operating principle of the hierarchical scheduler is shown in Figure 5.

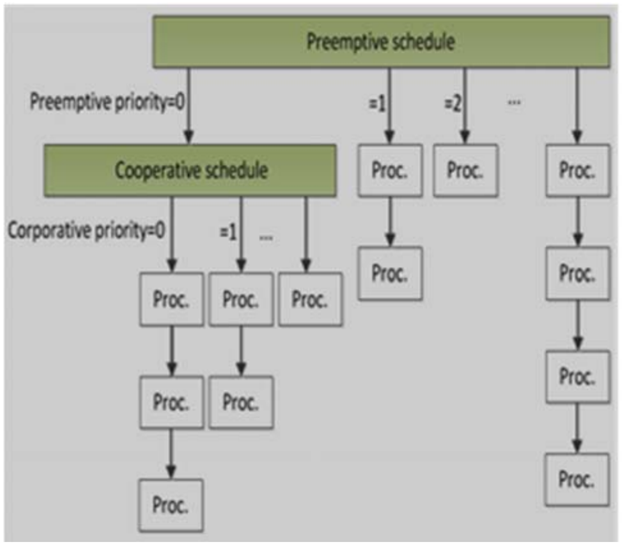

Figure 5. Combined preemptive/cooperative scheduler

B. Messages

Real-time systems typically support preemptive scheduling to guarantee short latency periods. This may lead to cases where the execution of some low priority process is preempted by a higher priority process. Under the assumption that the preempted process reads an object attribute and that the preempting process writes the same object attribute data inconsistency may arise if the preemption occurs between two consecutive read operations, see Figure 6.

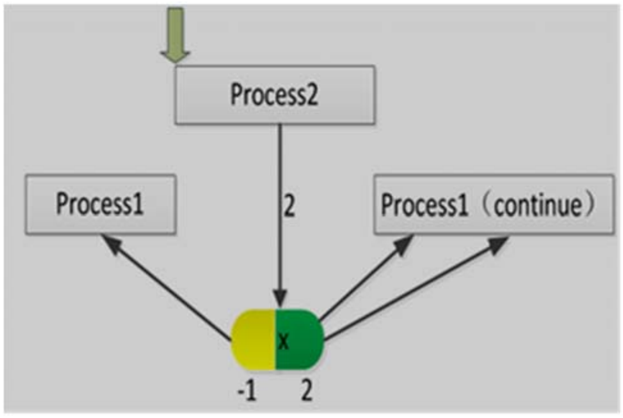

Figure 6. Data inconsistency through object attributes implemented by variables

After preemption through Proc 2, Proc 1 reads the object attribute x a second and a third time and gets a different result (x = 2) than before the preemption (x = -1). This example shows that the data accessed by Proc 1 becomes inconsistent. As a consequence, it may happen that Proc 1 fails.

Consider the case where Proc 1 calculates the absolute value of x by the algorithm:

```
if(x<0)
{y= -x;}
else
{y=x;}.
```

Since the sign of x may change between the comparison and the assignment operation, Proc 1 may fail. Correct function therefore depends on the timing and the sequence of preemptions in a certain system, as this example shows. This, however, conflicts with the basic requirements for software reuse, modularity, maintainability and extendibility. It also conflicts with the goals of object-orientation where objects should be encapsulated and their correct function should be independent of the environment. To avoid software failures which are timing and system configuration dependent data consistency has to be guaranteed.

OS therefore provides state-message semantics with a highly optimized implementation. Based on the OS object model, a process receives input data, performs processing actions and sends output data (input – process - output). Send and receive operations are mapped onto message objects. Between processes there is no direct method of information exchange by attributes (or variables). Conceptually, the state-message implementation of OS provides a message copy for each receiver process of a message. During startup of a process, all input messages are copied to the private area for message copies. Having finished, all output messages which are held in a message copy are copied to the global message area. The functioning principle of state-messages is shown in Figure7.

Upon start, process Proc 1 copies its input message msg to the private message copy msg(1). All subsequent read operations to the message are done from the private copy. Even though Proc 1 gets interrupted by Proc 2 which changes the contents of msg from -1 to 2 this change does not affect Proc 1. It is therefore guaranteed that Proc 1 perceives the same contents of its input message during the whole execution. This ensures data consistency.

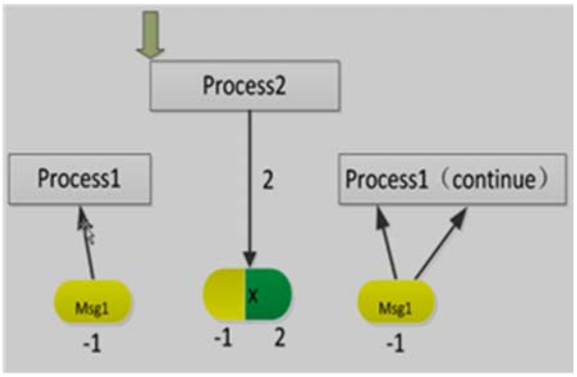

Figure 7. Data consistency through state-messages

C. Fault tolerance and exception handling

Robustness and fault-tolerance are important properties for real-time systems since they have to respond to state changes in the environment with a guaranteed latency period. Such guarantees, however, can only be given for a specified peak load scenario. The peak load scenario defines the maximum arrival rate and pattern of events that has to be handled by the system[10] [11].

Firstly guarantee minimum interarrival period. For task activations, the peak load scenario can be specified in terms of a minimum interarrival period. Individually for each task this parameter specifies the minimum time interval between two consecutive activations, see Figure 8.

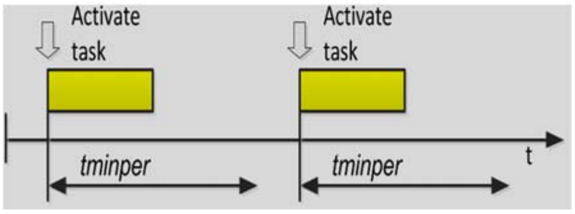

Figure 8. Minimum interarrival period

In the case of sensor faults or a wrongly estimated peek load scenario it is possible that two consecutive task activations are closer together than specified. To prevent system overloads, OS provides a mechanism that rejects task activations which are too early. If tasks are activated by software, the operating system checks the duration since the last activation. It rejects the task activation if the elapsed time is shorter than the minimum interarrival period. In the case of tasks that are activated by hardware interrupts a different strategy has to be used since the activation is not done by an operating system call. To ensure that the minimum interarrival period is not violated, the operating system disables the activating interrupt after the task activation for the duration of the minimum interarrival period. Figure 9 shows the handling of early interrupts.

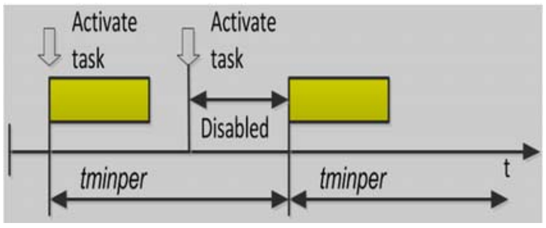

Figure 9. Handling of early interrupts

Secondly bound number of concurrently ready task instances. For software activated tasks it is under some circumstances necessary to allow multiple activations of tasks. This results in multiple concurrently ready instances of a task which can be managed by the operating system in FIFO manner. Hence it is possible to implement queuing semantics. Again, a peek load scenario is required to guarantee that task activations are handled correctly by the operating system. OS therefore allows specifying the maximum number of concurrently ready instances of a task. This parameter serves two purposes: Firstly, during the system configuration an analysis tool reads all the task definitions and generates data structures which are sufficient to hold all possible concurrently active task instances. And secondly, the operating system checks upon each task's activation whether the maximum number of concurrently active instances will be exceeded. In this case the task activation is rejected. This method guarantees that the operating systems data structures are sufficient to handle the worst case scenario. There is no need for experimental buffer size estimations. Furthermore, the operating system is capable of tolerating software faults at runtime by rejecting excessive task activations.

Then monitor deadline. Real-time systems are characterized by the fact that they have to respond to events in the environment with a bounded latency period. This latency period, called deadline, specifies whether a result is delivered timely, and thus being correct or not. OS provides a mechanism to implement deadline checking. This allows the detection of late and incorrect system responses and enables the user to react on behalf of these faults. The deadline mechanism provides functions to start deadline supervision and to check whether a started deadline has expired. Furthermore, the operating system provides consistency checks to detect corruption of the internal data structures of the deadline monitoring service.

At last handle exception. Besides faults in the timing of task activations as mentioned above there is a broad variety of faults which are detected by the operating system at runtime. Among these faults are stack over- and underflows or corruption of data structures. Furthermore, there are possible faults which are detected by the user software. To handle these faults in accordance to the application requirements the operating system provides an exception mechanism. Upon occurrence of a fault an exception is raised either by the operating system or by the user. In accordance to the application requirements it is possible to bind an exception handling routine to an exception. This binding is done statically at system generation time and allows binding of user written exception handlers.

IV. REAL TIME OPERATING SYSTEM APPLICATION

A. *Configure operating system with tool*

As well known the operating system should be easily ported between different CPUs. So normally operating systems is divided into two parts. First a static kernel provides a set of essential run-time services and second an external program exists, the so called Offline-Tools (OLT) [2] [4], which provides a system generation functionality (Let each do what it does best). The offline tools have various tasks. Within the offline tools (OLT) a target processor independent programming interface exists. Therefore the offline tools provide a platform that encapsulates the specifics of the OS runtime kernel.

Offline tools need perform a lot of tasks. Firstly should support code reuse, include data consistency in interruptions, separation of time response (priority) and function, determination of SW interfaces and visibility, analysis of the SW interfaces for completeness and consistency. Secondly automatically configure the operating system, include creation of the necessary data structures, memory reservation. Thirdly optimize operating system functions, include message concept (data exchange between processes), resource concept (protection for critical code sections). Figure 10 shows the two parts of OS.

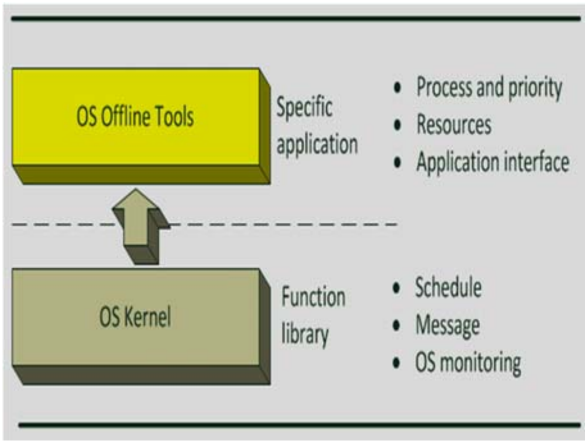

Figure 10. Offline tools and kernel of OS

B. *Validation of operating system*

After the configuration of an operating system, must validate the OS. For validation should consider three perspectives: function, performance and robustness. [10] [11]

To validate the function of a process, there are two ways. In every process record the system time and write it into an array, then read it out with debug tool. Or in each process change the status of a port and output it onto an oscilloscope, then analyze the wave. To validate the performance of operating system should test the accuracy and calculate whether the performance is acceptable. Robustness validation, an important feature of fuctional safety requirement [12] [13], need check OS' behavior in failure case such as: stack overflow, data structure is wrong, unexpected interrupt or trap happens, continuously reset happens, etc.

V. CONCLUSIONS

An operating system specifically designed for automotive requirements must have real-time capability, robustness and efficiency. Through the reasonable design of schedule, message and fault tolerance can fulfill the requirements. Normally with configuration tool can configure an operating system. After sufficient validation the OS can be used on certain automotive controller.

ACKNOWLEDGMENTS

The authors thank the reviewers who gave a through and careful reading to the original manuscript. Their comments are greatly appreciated and have help to improve the quality of this paper.

REFERENCES

[1] Specification OSEK OS. OSEK group, Version 2.2.3. 2005.
[2] ERCOS user's guard. ETAS GmbH. Version 4.3. May 2004.
[3] KPIT Cummins Offers Autosar Stack Compatible to R4.0.3. Health & Beauty Close - Up. 2013 03.

[4] Einarbeitung von Betriebssystem ERCOS (Embedded Real-time Control Operating System). Steigmüller, Robert Bosch GmbH. Oct 1997.

[5] Herpel, T. and R. German. A simulation approach for the design of safety-relevant automotive multi-ECU systems. in System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on. 2009.

[6] Gall, H. Functional safety IEC 61508 / IEC 61511 the impact to certification and the user. Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on Digital Object Identifier: 10.1109/AICCSA.2008.4493673. Publication Year: 2008 , Page(s): 1027 – 1031.

[7] Lee, S. ; Yamada, Y. ; Ichikawa, K. ; Matsumoto, O. ; Homma, K. ; Ono, E. Safety-Function Design for the Control System of a Human-Cooperative Robot Based on Functional Safety of Hardware and Software. Mechatronics, IEEE/ASME Transactions on Volume: PP , Issue: 99. Digital Object Identifier: 10.1109/TMECH.2013.2252912. Publication Year: 2013 , Page(s): 1 – 11.

[8] S. Saewong and R. Rajkumar. Cooperative scheduling of multiple resources. In IEEE RTSS'99, pages 90–101, Dec. 1999.

[9] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In ACM/SPIE Multimedia Computing and Networking, Jan. 2002.

[10] Real-time operating system. http://en.wikipedia.org/wiki/RTOS.

[11] Real time OS basics. http://wenku.baidu.com.

[12] P raprotnik, O., et al. A Test Suite for System Tests of Distributed Automotive Electronics. in Advances in Circuits, Electronics and Micro-electronics, 2009. CENICS '09. Second International Conference on. 2

[13] Navinkumar, V.K. ; Archana, R.K. Functional safety management aspects in testing of automotive safety concern systems (electronic braking system). Electronics Computer Technology (ICECT), 2011 3rd International Conference on Volume: 1. Digital Object Identifier: 10.1109/ICECTECH.2011.5941627. Publication Year: 2011 , Page(s): 381 - 384

.