# Study of OSEK OS Possibilities for Usage in Railway Electronic Diagnostic Systems

Kristian Dilov Dilov and Emil Nikolov Dimitrov

*Abstract* – **OSEK is an abbreviation for the German term "Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug" (English: Open Systems and the Corresponding Interfaces for Automotive Electronics). It is a static operating system for fault free embedded applications. The aim is analysis of the OSEK specification, in order to design suitable OS for railway diagnostic proposes.**

*Keywords* – **OSEK, OSRT, OS, kernel, scheduling, ISR.**

## I. INTRODUCTION

OSEK is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.
A real-time operating system, software interfaces and functions for communication and network management tasks are thus jointly specified.

### *A. OSEK/VDX overview*

The OSEK operating system provides the necessary services to support distributed faulttolerant

highly dependable real-time applications (e.g., start-up of the system, message handling, state message interface, interrupt processing, synchronisation and error handling).

The operating system is built according to the user's configuration instructions at system generation time. The operating system cannot be modified later at execution time. That means it is static – its original form, in terms of occupied resources is kept up for lifecycle of the product without any modifications. The specified operating system services constitute a basis to enable the integration of software modules made by various manufacturers. To be able to react to the specific features of the individual control units as determined by their performance and the requirements of a minimum consumption of resources, the prime focus was not to achieve 100% compatibility between the application modules, but their direct portability.

The system service is the basic unit in OSEK. It is involved in tasks' states transition and execution. It is also responsible for resource and events management and interrupts handling. It is activated upon the following events:
- system time tick expiration;
- explicit request for task scheduling;
- system event status change;
- interrupt category 2 occurrence;

The main characteristics are:
- The OSEK operating system is configured and scaled statically. The user statically specifies the number of tasks, resources, and services required.
- The specification of the OSEK operating system supports implementations capable of running on ROM, i.e. the code could be executed from Read-Only-Memory.
- The OSEK operating system supports portability of application tasks.
- The specification of the OSEK operating system provides a predictable and documented behavior to enable operating system implementations, which meet automotive real time requirements.
- The specification of the OSEK operating system allows the implementation of predictable performance parameters.

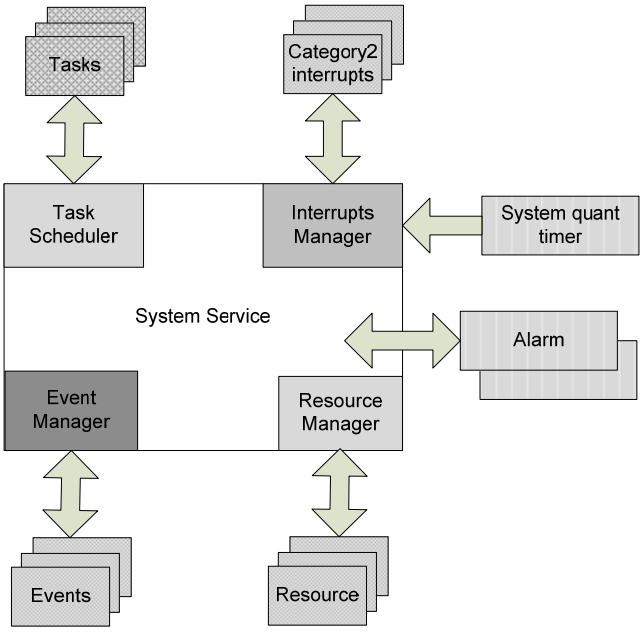


*Figure 1. OSEK – global view*

**Standardised interfaces**

The interface between the application software and the operating system is defined by system services. The interface is identical for all implementations of the operating system on various processor families.

System services are specified in an ISO/ANSI-C-like syntax, however the implementation language of the system services is not specified.

K. Dilov is with the Department of Electronics and Electronics Technologies, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: kdilov@kdilov.com

E. Dimitrov is with the Department of Electronics and Electronics Technologies, Faculty of Electronic Engineering and Technologies, Technical University - Sofia, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria, e-mail: edim@tu-sofia.bg

*B. Architecture of the OSEK operating system*

**Processing levels**

The OSEK operating system serves as a basis for application programs which are independent of each other, and provides their environment on a processor. The OSEK operating system enables a controlled real-time execution of several processes which appear to run in parallel.

The OSEK operating system provides a defined set of interfaces for the user. These interfaces are used by entities which are competing for the CPU. There are two types of entities:

- Interrupt service routines managed by the operating system.
- Tasks (basic tasks and extended tasks)

The hardware resources of a control unit can be managed by operating system services. These operating system services are called by a unique interface, either by the application program or internally within the operating system.

OSEK defines three processing levels:

- Interrupt level
- Logical level for scheduler
- Task level

Within the task level, tasks are scheduled according to their user assigned priority. The run time context is occupied at the beginning of execution time and is released again once the task is finished. The following priority rules have been established:

- Interrupts have precedence over tasks.
- The interrupt processing level consists of one or more interrupt priority levels.
- Interrupt service routines have a statically assigned interrupt priority level.
- For task priorities and resource ceiling-priorities bigger numbers refer to higher.

priorities.

- The task's priority is statically assigned by the user
- Assignment of interrupt service routines to interrupt priority levels is dependent on implementation and hardware architecture

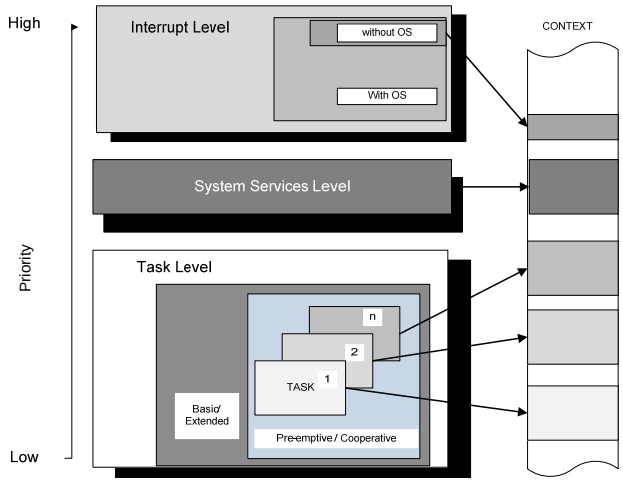


*Figure 2. Logical levels of processing in OSEK*

Processing levels are defined for the handling of tasks and interrupt routines as a range of consecutive values. Mapping of operating system priorities to hardware priorities is implementation specific.

Assignment of a priority to the scheduler is only a logical concept which can be implemented without directly using priorities.

*C. Tasks and task management*

**Task**

Complex control software can conveniently be subdivided in parts executed according to their real-time requirements. These parts can be implemented by the means of tasks. A task provides the framework for the execution of functions. The operating system provides concurrent and asynchronous execution of tasks. The scheduler organises the sequence of task execution.

The OSEK operating system provides a task switching mechanism including a mechanism which is active when no other system or application functionality is active. This mechanism is called idle-mechanism. Two different task concepts are provided by the OSEK operating system:

- basic tasks
- extended tasks

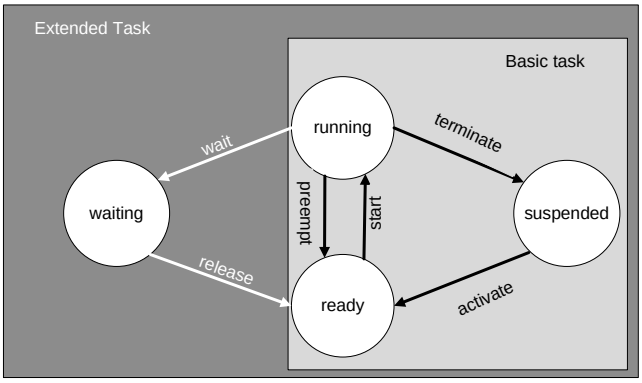


*Figure 3. Tasks types and their states*

**Task states**

A task has to change between several states, as the processor can only execute one instruction of a task at any time, while several tasks may be competing for the processor at the same time. The OSEK operating system is responsible for saving and restoring task context in conjunction with task state transitions whenever necessary. The OSEK operating system is responsible for saving and restoring task context in conjunction with task state transitions whenever necessary.

- running - in the running state, the CPU is assigned to the task, so that its instructions can be executed. Only one task can be in this state at any point in time, while all the other states can be adopted simultaneously by several tasks.
- ready - all functional prerequisites for a transition into the running state exist, and the task only waits for allocation of the processor. The scheduler decides which ready task is executed next.
- waiting - a task cannot continue execution because it has to wait for at least one event.
- suspended - in the suspended state the task is passive and can not be activated.

TABLE 1. TASKS' STATES TRANSITION

| Transition | Former state | New state | Description |
|---|---|---|---|
| activate | suspended | ready | A new task is set into the ready state by a system service. |
| start | ready | running | A ready task selected by the scheduler is executed |
| preempt | running | ready | The scheduler decides to start another task. The running task is put into the ready state. |
| terminate | running | suspended | The running task causes its transition into the suspended state by a system service |
| wait | running | waiting | The executed task is topped until event is activated |
| release | waiting | ready | At least one event has occurred which a task has waited for. |

**Basic Tasks**

Basic tasks only release the processor, if
- terminate;
- the OSEK operating system switches to a higher-priority task;
- an interrupt occurs which causes the processor to switch to an interrupt service routine (ISR).

**Extended Tasks**

Extended tasks are distinguished from basic tasks by being allowed to use the operating system call WaitEvent, which may result in a waiting state. The waiting state allows the processor to be released and to be reassigned to a lower-priority task without the need to terminate the running extended task. In view of the operating system, management of extended tasks is, in principal, more complex than management of basic tasks and requires more system resources.

**Task's descriptor**

The task's descriptor is used by the system service for management of the tasks. It is a RAM structure, where the characteristics of the tasks are kept during OSEK execution – such like priority, state, task's stack pointer, task type(basic or extended).

*D. System dispatching*

Unlike conventional sequential programming, the principle of multitasking allows the operating system to execute various tasks concurrently. The entity deciding which task has to be started and the triggering of all necessary OSEK operating system internal activities is called scheduler. The scheduler is activated whenever a task switch is possible according to the implemented scheduling policy.

**Full preemptive scheduling**

Full preemptive scheduling means that a task which is presently running may be rescheduled at any instruction by the occurrence of trigger conditions pre-set by the operating system. Full preemptive scheduling will put the running task into the ready state, as soon as a higher priority task has got ready.

The task context is saved so that the preempted task can be continued at the location where it was preempted. With full preemptive scheduling the latency time is independent of the run time of lower priority tasks. Certain restrictions are related to the increased (RAM) memory space required for saving the context, and the enhanced complexity of features necessary for synchronization between tasks.

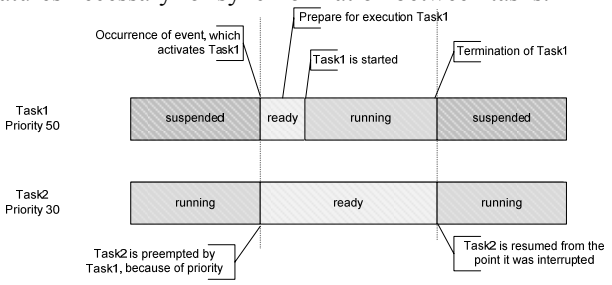


*Figure 4. Extended task activation*

As each task can theoretically be rescheduled at any location, access to data which are used jointly with other tasks has to be synchronized.

Summarized, rescheduling is performed in all of the following cases:
- Successful termination of a task.
- Successful termination of a task with explicit activating of a successor task.
- Activating a task at task level - message notification mechanism, alarm expiration.
- Explicit wait call if a transition into the waiting state takes place (extended tasks only).
- Release of resource at task level.

**Non preemptive scheduling**

The scheduling policy is described as non pre-emptive, if task switching is only performed via one of a selection of explicitly defined system services - explicit points of rescheduling. Non pre-emptive scheduling imposes particular constraints on the possible timing requirements of tasks. Specifically the non pre-emptive section of a running task with lower priority delays the start of a task with higher priority up to the next point of rescheduling.

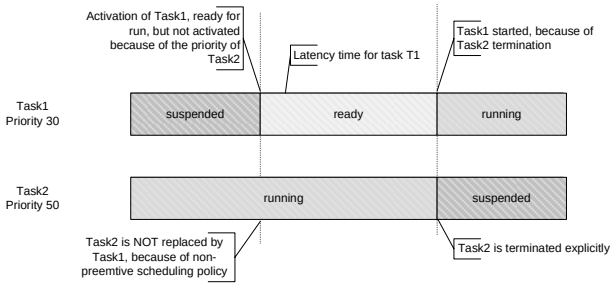


*Figure 5. Cooperative tasks swithcing*

*E. Alarms*

The OSEK operating system provides services to activate tasks, set events or call an alarm callback routine

when an alarm expires. An alarm-callback routine is a short function provided by the application.

An alarm will expire when a predefined counter value is reached. This counter value can be defined relative to the actual counter value (relative alarm) or as an absolute value (absolute alarm). For example, alarms may expire upon receipt of a number of timer interrupts, when reaching a specific angular position, or when receiving a message.
Alarms can be defined to be either single alarms or cyclic alarms. In addition the OS provides services to cancel alarms and to get the current state of an alarm. More than one alarm can be attached to a counter. An alarm is statically assigned at system generation time to:
- one counter;
- one task or one alarm-callback routine.

*F. Events*

The event mechanism
- is a means of synchronization;
- is only provided for extended tasks
- initiates state transitions of tasks to and from the waiting state.

Events are objects managed by the operating system. They are not independent objects, but assigned to extended tasks. Each extended task has a definite number of events. This task is called the owner of these events. An individual event is identified by its owner and its name or mask. When activating an extended task, these events are cleared by the operating system. Events can be used to communicate binary information to the extended task to which they are assigned to. The meaning of events is defined by the application, e.g. signal for a timer expiration, the availability of a resource, the reception of a message.

Events are the criteria for the transition of extended tasks from the waiting state into the ready state. The operating system provides services for setting, clearing and interrogation of events and for waiting for events to occur.

*G. Resource management*

The resource management is used to co-ordinate concurrent accesses of several tasks with different priorities to shared resources, e.g. management entities (scheduler), program sequences, memory or hardware areas.

The resource management can optionally be extended to coordinate concurrent accesses of tasks and interrupt service routines. Resource management ensures that:
- two tasks cannot occupy the same resource at the same time;
- priority inversion can not occur;
- deadlocks do not occur by use of these resources;
- access to resources never results in a waiting state;

If the resource management is extended to the interrupt level it assures in addition that two tasks or interrupt routines cannot occupy the same resource at the same time.

The functionality of resource management is useful in the following cases:
- pre-emptive tasks;
- cooperative tasks, if the user intends to have the application code executed under other scheduling policies too;
- resource sharing between tasks and interrupt service routines;
- resource sharing between interrupt service routines;

*H. Interrupt processing*

The functions for processing an interrupt (Interrupt Service Routine: ISR) are subdivided into two ISR categories:
- ISR category 1 - The ISR does not use an operating system service. After the ISR is finished, processing continues exactly at the instruction, where the interrupt has suspended the execution, i.e. the interrupt has no influence on task management. ISRs of this category have the least overhead. It is not allowed to call OS services from context of this kind of interrupts.
- ISR category 2 - OSEK operating system provides an ISR-frame to prepare a run-time environment for a dedicated user routine. During system generation the user routine is assigned to the interrupt. The meaning of this time frame is that the OS activates a task with higher priority, and preempts the currently executed task, with no respect to the current scheduling policy.

## II. CONCLUSION

OSEK provides the necessary possibilities for vehicle control and diagnostic electronic systems development. It allows distributing the application in the respective environment organized by the OS, in terms of time, synchronization, based on:
- sequenced utilization of the existing resources, distributed in the vehicle, to enhance the performance of the overall system without requiring additional hardware;
- precise scheduling of the tasks by alarms;
- possibilities to execute slow, time-consuming tasks parallel to the fast time critical tasks;
- possibilities for fast reaction on event with different set of interrupts handling options;
- possibilities for control and instrumentation of the tasks execution time.

The static character of the system ensures fault free system and robustness.

## REFERENCES

[1] М. Луканчевски. *Системно програмиране за едночипови микрокомпютри*
[2] *OSEK/VDX - Operating System Specification 2.2.22*
[3] J.K Stankovic, J. K. Ramamritham. *The design of the Spring kernel.* Real time systems symposium San Jose 1987
[4] Shao B., R. Wang, *EMBEDDED REAL-TIME SYSTEMS TO BE APPLIED IN CONTROL SUBSYSTEMS FOR ACCELERATORS* - Tsinghua University, Beijing
[5] http://portal.osek-vdx.org/
[6] http://www.autosar.org/