OSEK OS

Session Speaker

Deepak V.

Session Objectives

- To understand the necessity of OSEK standard
- To find the Goal of OSEK
- To study the OSEK Implementation Language (OIL) Concept
- To know the meaning of conformance classes
- To study task properties and scheduling techniques

Session Topics

- OSEK/VDX overview
- Standards
- OS requirements, specifications
- OSEK Implementation Language
- Conformance Classes
- OS services
- OSEK Priority Ceiling Protocol
- Interrupt Processing
- Communication
- Error handling, Tracing and Debugging

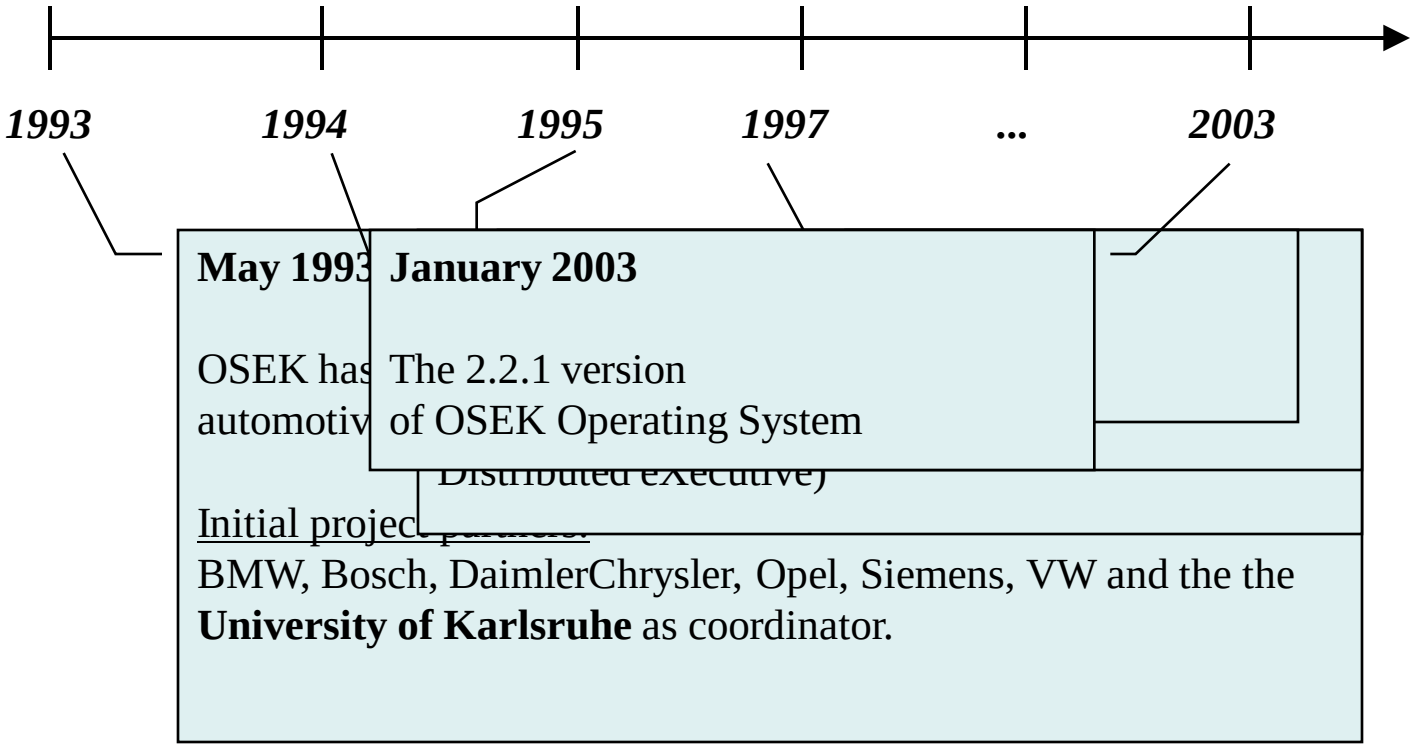
OSEK/VDX Overview

- **OSEK: "Open Systems and the Corresponding Interfaces for Automotive Electronics"**
  - "Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug"
- VDX is an acronym for Vehicle Distributed eXecutive

- OSEK/VDX™ (OSEK Vehicle Distributed eXecutive)
  - Operating System (OS)
  - Communications (COM)
  - Network Management (NM)

History

- History timeline of the OSEK standard



Motivation

- High, recurring expenses in the development and variant management of non-application related aspects of control unit software

- Incompatibility of control units made by different manufactures due to different interfaces and protocols

Goal of OSEK

- Support - portability, reusability of the application software by Specification of interfaces which are abstract and as application-independent as possible

- Specification of a user interface to be independent of hardware and network

- Efficient design of architecture: The functionalities shall be configurable and scalable, to enable optimal adjustment of the architecture to the application in question

Why OSEK?

- Clearly there are some excellent open source RTOS alternatives (FreeRTOS, eCos, Jaluna, etc). The problem with these operating systems, from an automotive perspective, is:
  - alternatives are quite bulky and their real-time performance is not quite "real" enough
  - The API of most alternatives (typically POSIX) is not ideally suited to the combination of synchronous and asynchronous task scheduling
  - The OSEK API was designed by automotive suppliers

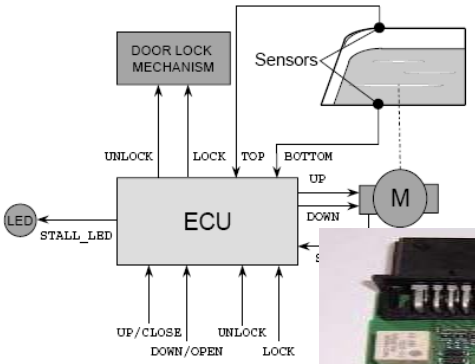
Advantages

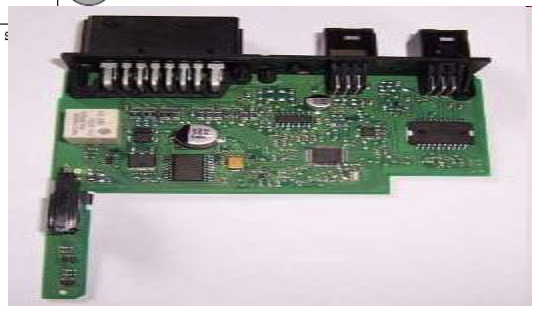
- Clear savings in costs and development time
- Enhanced quality of the software of control units
- Standardized interfacing features
- Utilization of the existing resources in the vehicle, enhance the performance of the overall system without requiring additional hardware
- Absolute independence with regards to individual implementation, as the specification does not prescribe implementation aspects
- Overhead in terms of memory usage and code footprint

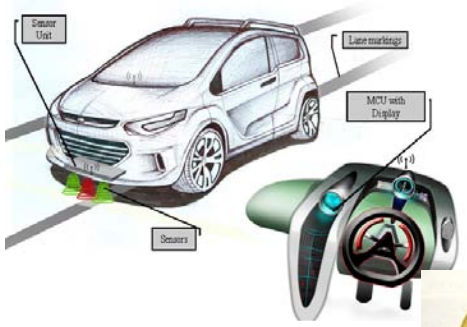
Standards

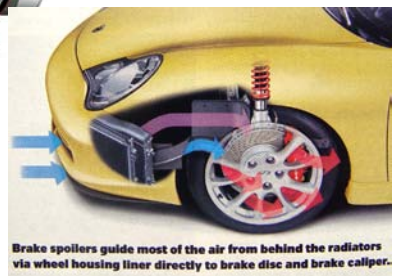
- **OS (Operating System) –** Provides a common API for OS features, it is configurable with OSEK OIL
- **OIL (OSEK Implementation Language) –** Provides system configuration and object description for the OS and COM implementations
- **COM (Communication) –** Provides the standard interfaces and protocols for data exchange in a network
- **NM (Network Management)**
- **ORTI (OSEK Run Time Interface):** The internal OS data is made available to the debugger and the debugger displays the information of OSEK system objects
- **OSEKtime OS**
- **FT COM** (Fault Tolerant Communication)

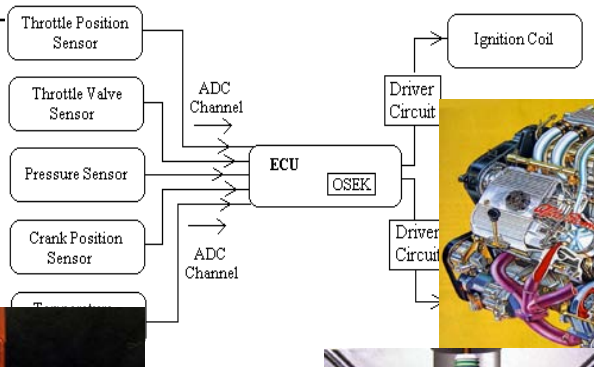
Applications



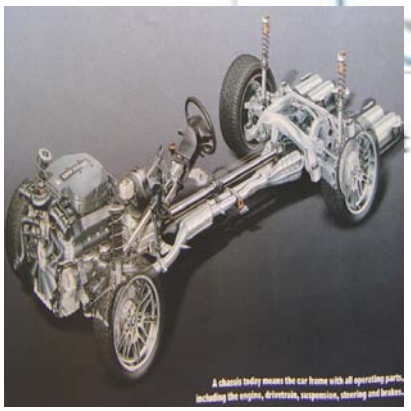








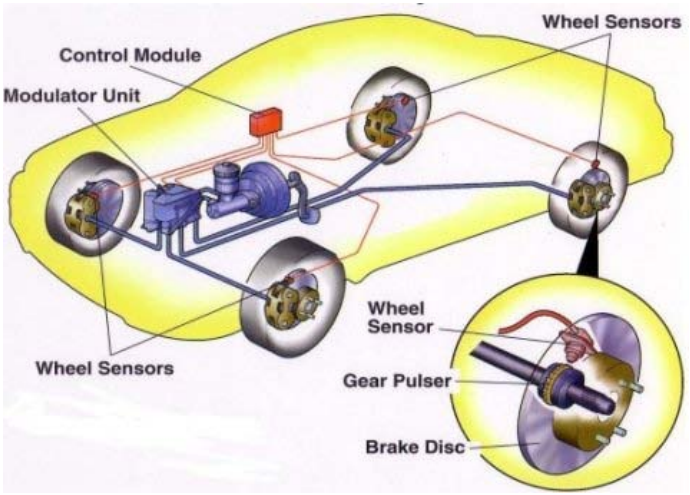
Belt Pretensioner

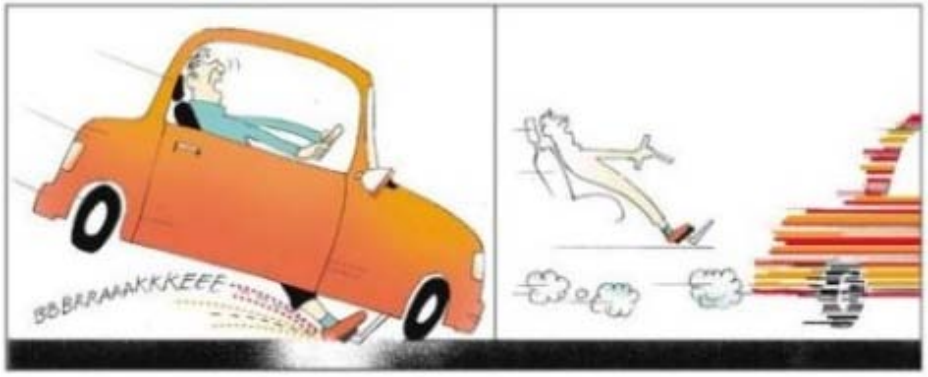


Application

Side airbag

Side curtain airbag





Antilock braking systems Airbag systems

OSEK/VDX OS Requirements

- Basis for the integration of software made by various manufacturers.
- Support for portability of software modules to allow for re-use
- Static configuration (scalability, scheduling policy)
- ROMable
- Require only a minimum of hardware resources
- Real-time multi tasking

OSEK Implementation Language (OIL) Concept

- A specially designed language for development of embedded applications based on OSEK concept
  - describe the application structure/configuration as a set of system objects
  - C-like text based configuration description for OSEK applications
  - have predefined structure and special (standard) grammar rules

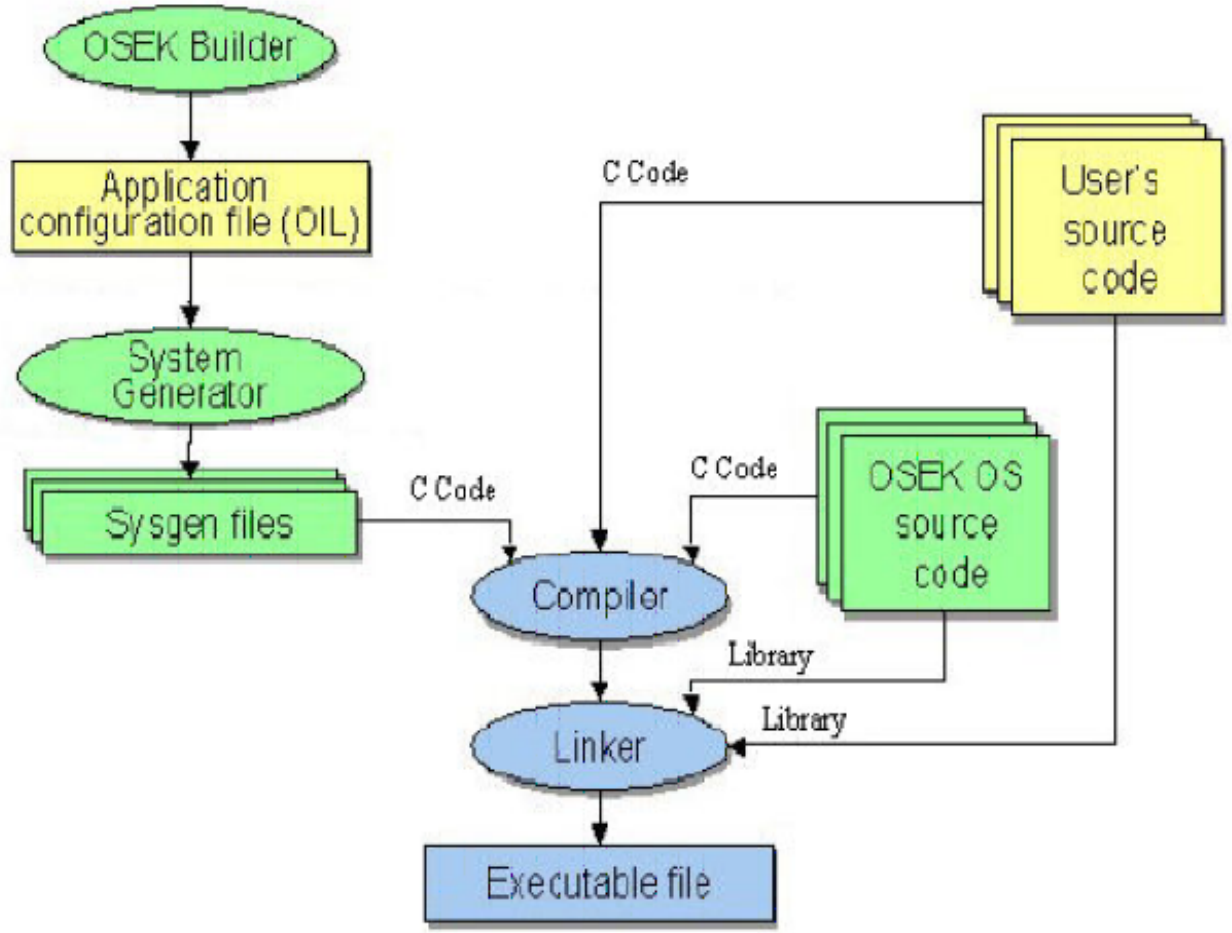
- The configuration of OS is object oriented, each object defines a certain set of attributes to define

- All system objects specified by OSEK and relationships between them can be described using OIL
  - OIL defines standard types for system objects
  - Each object is described by a set of attributes and references

OIL File

- The OIL file contains two parts
  - Implementation Definition - definition of implementation specific features
  - Application Definition - definition of the structure of the application located on the particular CPU
  - The application definition comprises a set of objects and the values for their attributes
- OS, COM and NM objects must be 1, other objects can be more
- Dependencies Between Attributes- infinite nesting

Application Development using OSEK Operating System



FEW of the OSEK Implementations

- Nucleus OSEK - Accelerated Technology (MPC555, MPC565)
- osCAN - Altera Corporation's (Altera Nios II Processor)
- PICOS18 – Pragmatec Inc. open source
- OSEKturbo – Metrowerks (Motorola HC08, HC12, MPC5xx, MPC52xx, MPC55xx architectures …)
- ProOSEK - Elektrobit
- RTA-OSEK - ETAS / Live Devices
- OSEKWorks – Metrowerks
- OpenSEK / FreeOSEK RTOS

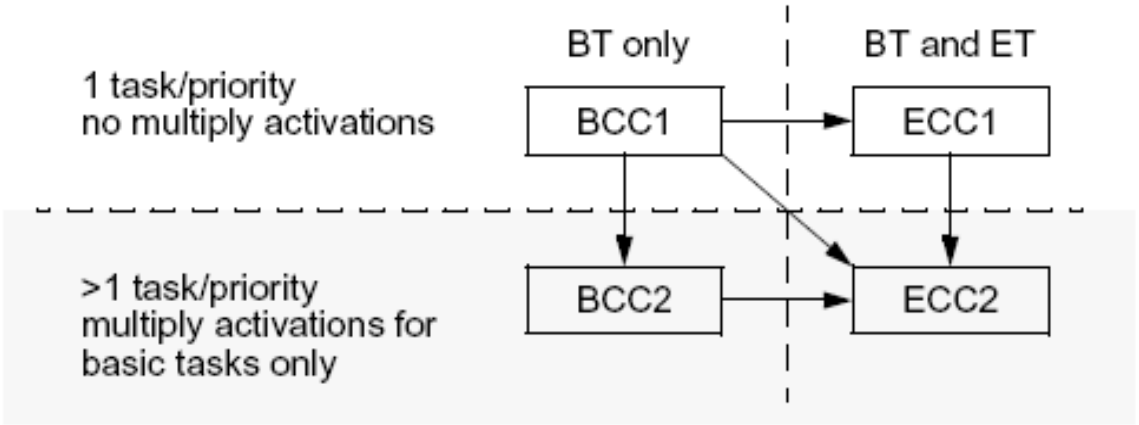
- OSEK hold about 70% of the world automotive operating system market share
- Other Information:
- Telelogic Releases Rhapsody OSEK Integration- UML 2.0/SysML-Based Modeling Environment to Produce OSEK Targeted Code
- OSEK-Shell for VxWorks

Conformance Classes

- Various requirements, capabilities of the application software demand different features of the operating system
  - (e.g. processor type, amount of memory)
- Operating system features are described as Conformance Classes (CC)
  - Differ in the number of services provided, their capabilities and different types of tasks
- Four different classes of operating systems
  - minimum requirements the system
  - Allows partial implementations (need not implement all features)
  - upgrade path

Conformance Classes

- Classified into- Basic-CC and Extended-CC
- The Conformance classes are determined by the following attributes:

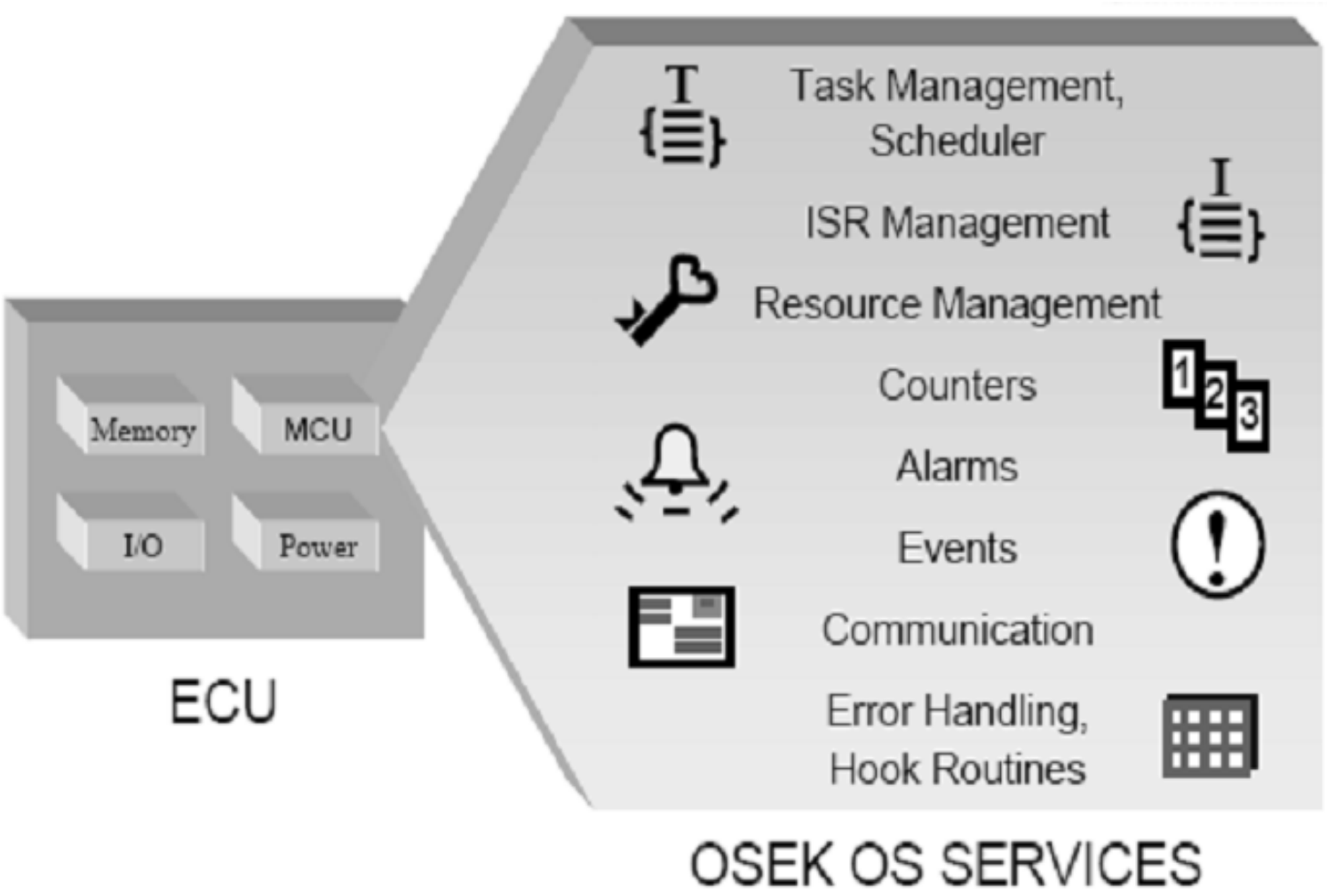


– Multiple requesting of task activation
– Task types
– Number of tasks per priority

OSEK OS Conformance Classes

	BCC1	BCC2	ECC1	ECC2
Multiple activation of tasks	no	yes	BT: no, ET: no	BT: yes, ET: no
Number of tasks which are not in *suspended* state	>=8		>= 16, any combination of BT/ET	
Number of tasks per priority	1	>1	1 (both BT/ ET)	>1 (both BT/ ET)
Number of events per task	-		BT: no ET: >= 8	
Number of task priorities	>=8		>=16	
Resources	only Scheduler	>= 8 resources (including Scheduler)		
Internal Resources	>=2			
Alarm	>= 1 single or cyclic alarm			
Messages	possible			

OS Services

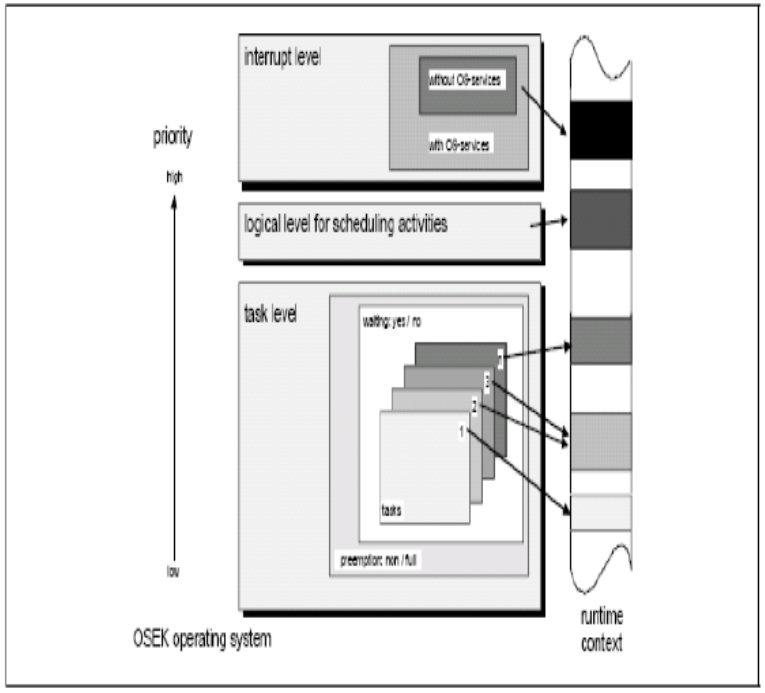

Memory
MCU
I/O
Power
ECU
Task Management,
Scheduler
ISR Management
Resource Management
Counters
Alarms
Events
Communication
Error Handling,
Hook Routines
OSEK OS SERVICES

Processing Levels

- The OSEK operating system enables a controlled real-time execution of several processes which appear to run in parallel
- Entities which are competing for the CPU are:
  - Interrupt service routines
  - Tasks
- OSEK defines three processing levels
  - Interrupt level
  - Logical level for scheduler
  - Task level

Priority rules:

- Interrupts have precedence over tasks



Task Management

- A task is an independent thread of execution that can compete with other concurrent tasks for processor execution time. A task is schedulable
  - A task provides the framework for the concurrent and asynchronous execution of functions
- Task in an application are generated using the TASK system generation object with the set of properties in OIL file
- The OSEK operating system provides two kind of tasks:
  - Basic tasks
  - Extended tasks

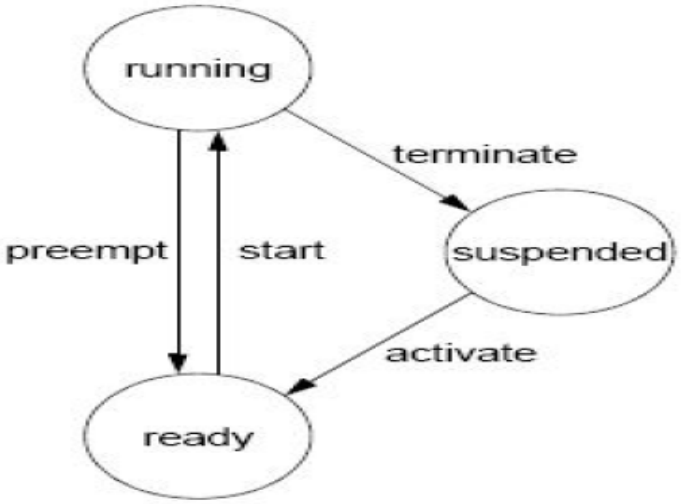
Basic Tasks (BT)

- A basic task runs to completion unless preempted by a higher priority task or an interrupt means
- The Basic Tasks release the processor only if:
  - they are being terminated
  - the OSEK OS is executing higher-priority tasks
  - interrupts have occurred
- Advantages
  - Uses minimal resources
  - Used when interprocess synchronization is not required

Status Model with Task Transitions for a BT

Basic task can exist in one of the states namely

- Running
- Ready
- Suspended



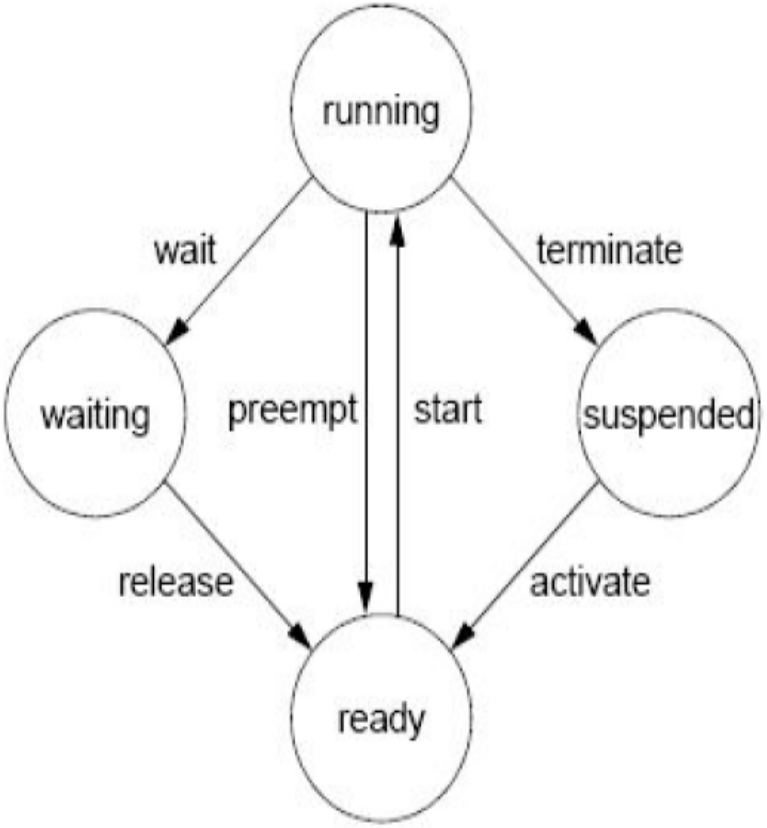
Transition	Former state	New state	Description
activate	*suspended*	*ready*[4]	A new task is set into the *ready* state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler 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.

Extended Tasks (ET)

- Extended tasks differ with basic – have a waiting state
  - allows the processor to be released and to be reassigned to a lower-priority task (no need to terminate the running extended task)
  - in principal, more complex than management of basic tasks and requires more system resources

- Advantage
  - Handle coherent job in a single task

States and Status Transitions for an ET



Transition	Former state	New state	Description
activate	suspended	ready	A new task is entered into the *ready* list by a system service.
start	ready	running	A *ready* task selected by the scheduler is executed.
wait	running	waiting	To be able to continue an operation, the *running* task requires an event. It causes its transition into *waiting* state by using a system service.
release	waiting	ready	Events have occurred which a task has been waiting for.
preempt	running	ready	The scheduler decides to start another task. The *running* task is put into *ready* state.
terminate	running	suspended	The *running* task causes its transition into *suspended* state by a system service.

Task Properties

- Priorities :
  - 0 is considered the lowest task priority
  - Bigger numbers - higher task priorities
  - Interrupts - separate priority scale
  - Priority is statically assigned and cannot be changed at run-time
- Stack :
  - statically allocated stack
  - The minimal size depends on:
    - the scheduling policy (non-preemptable or preemptable task)
    - the services used by the task
    - the interrupt and error handling policy
    - the processor type

Task Properties

- **Task activation**: performed using the operating system services

  ActivateTask or ChainTask

- **Termination of Tasks:** In the OSEK operating system, a task can only terminate itself

  TerminateTask or *ChainTask*

Task Properties

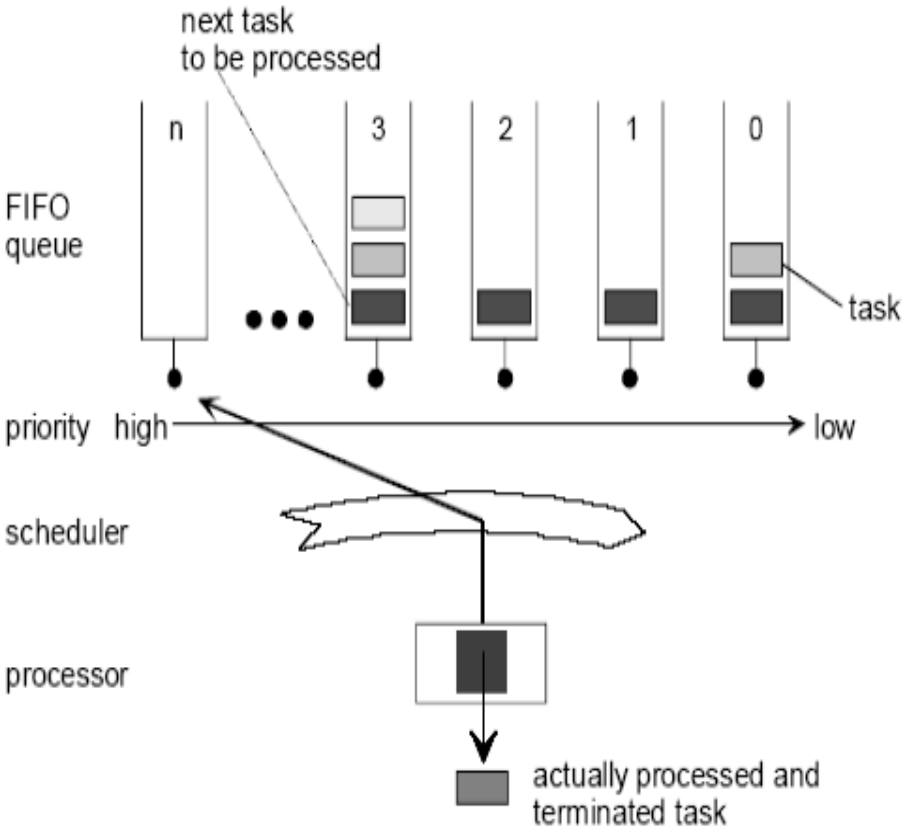
Object Parameters	Possible Values	Description
Standard Attributes		
PRIORITY	integer [0...0x7FFFFFFF]	Defines the task priority. The lowest priority has the value 0
SCHEDULE	FULL, NON	Defines the run-time behavior of the task
AUTOSTART	TRUE, FALSE	Defines whether the task is activated during the system start-up procedure or not
APPMODE	name of APPMODE	Defines an application mode in which the task is auto-started
ACTIVATION	1	Specifies the maximum number of queued activation requests for the task (the OSEKturbo does not allow multiply activations)

Scheduler

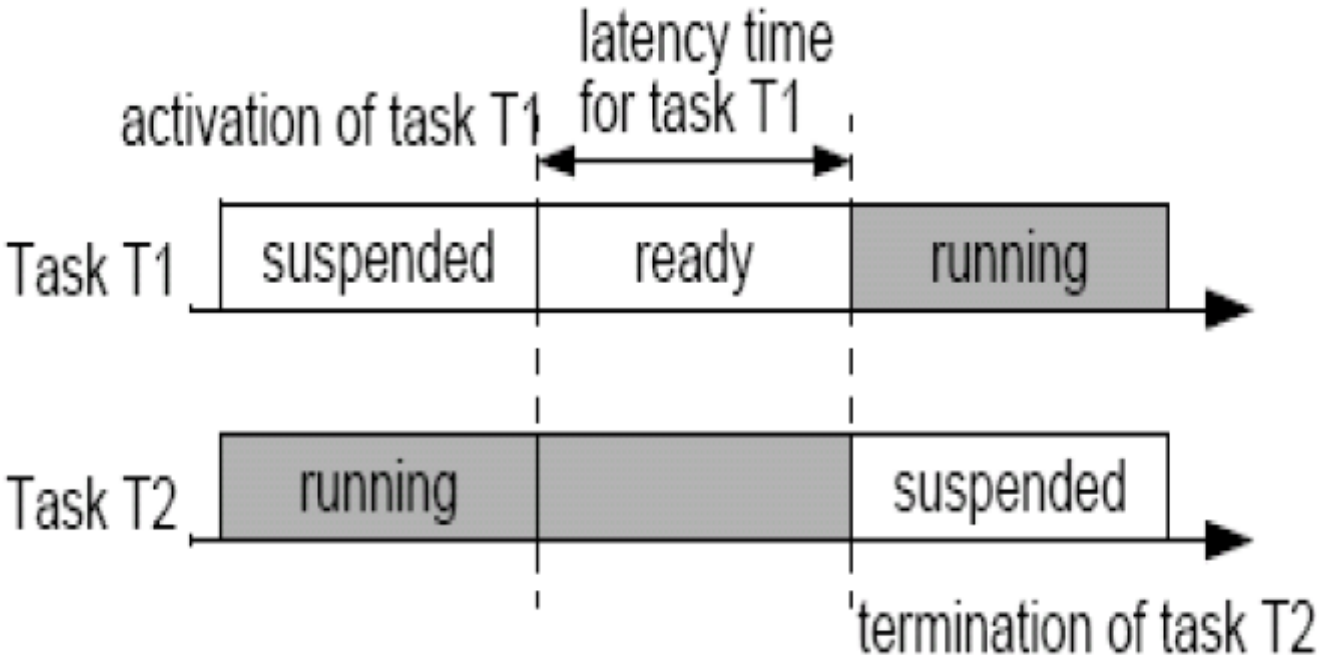
- The algorithm deciding which task has to be started and triggering all necessary OSEK Operating System internal activities is called scheduler

The scheduling policy determines –

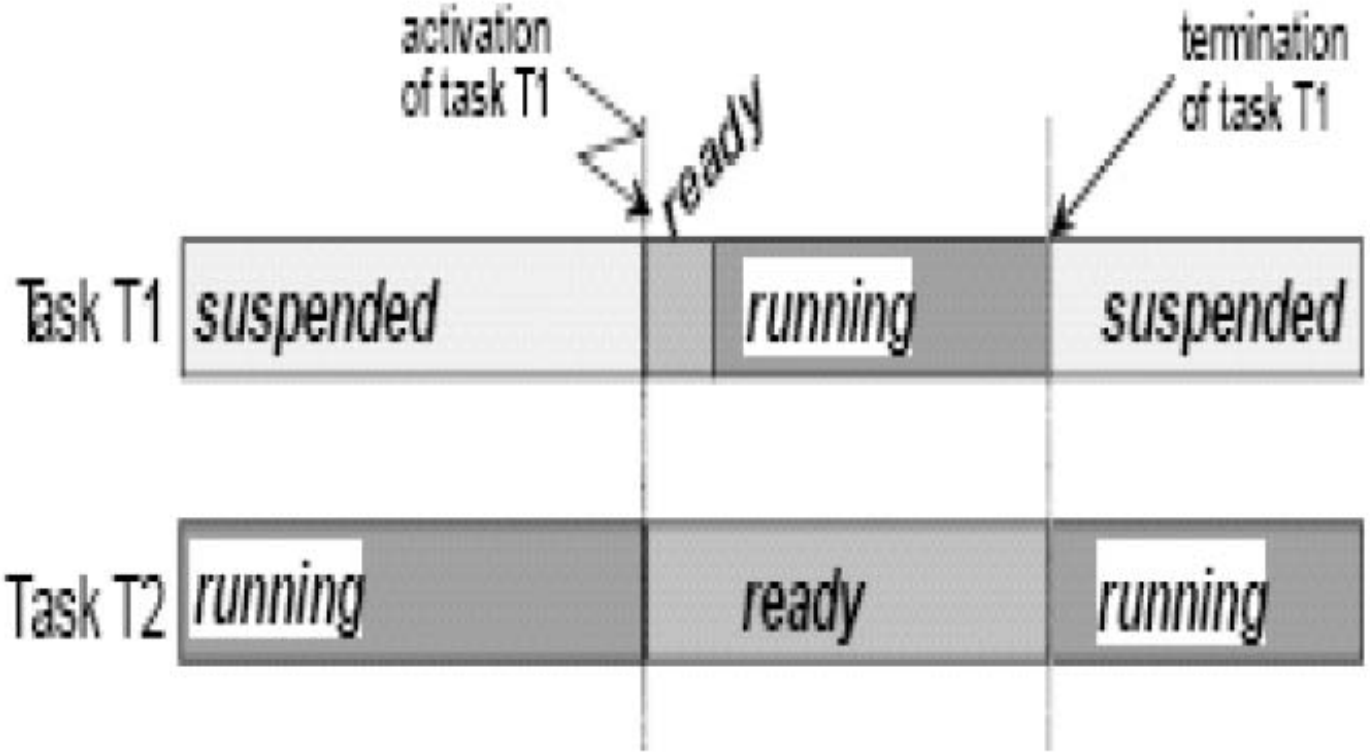
- execution of a task may be interrupted by other tasks or not
- affects the system performance and memory resources
- full, non and mixed-preemptive scheduling policies



Non-Preemptive Scheduling



Full Pre-emptive Scheduling



Resource Management

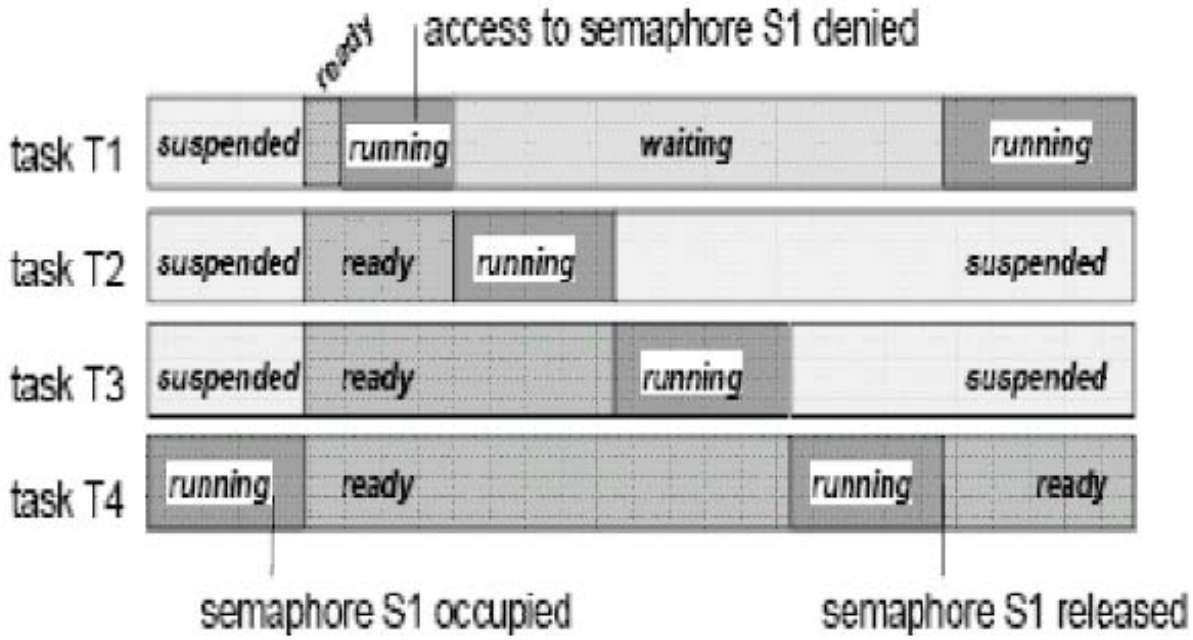
- Used to coordinate concurrent access of several tasks or/and ISRs to shared resources
  - critical sections, memory or hardware areas, management entities (scheduler)
- Is provided in all Conformance Classes
- The resource management ensures that
  - two modules (tasks /ISRs) cannot occupy the same resource at the same time
  - priority inversion cannot arise while resources are used
  - deadlocks do not occur
  - Task/ISR must request/release resources following the LIFO principle (stack)

Resource Management

- **Access to Resource**
  - It must be defined by the user at system configuration stage through the ***RESOURCE*** definition
  - *GetResource* and *ReleaseResource*
  - ***Ceiling Priority*** - resources are ranked by priority (is calculated automatically)
  - ***Ceiling Priority*** is higher or equal to the highest task or ISR priority with access to this resource
- **Restrictions**
  - *TerminateTask*, *ChainTask*, *Schedul*e and *WaitEvent* must not be called
  - interrupt service routine cannot be completed with the resource occupied
  - nested access to same resource
- **Scheduler as a Resource (**RES_SCHEDULER)

Problems with Synchronization Mechanisms

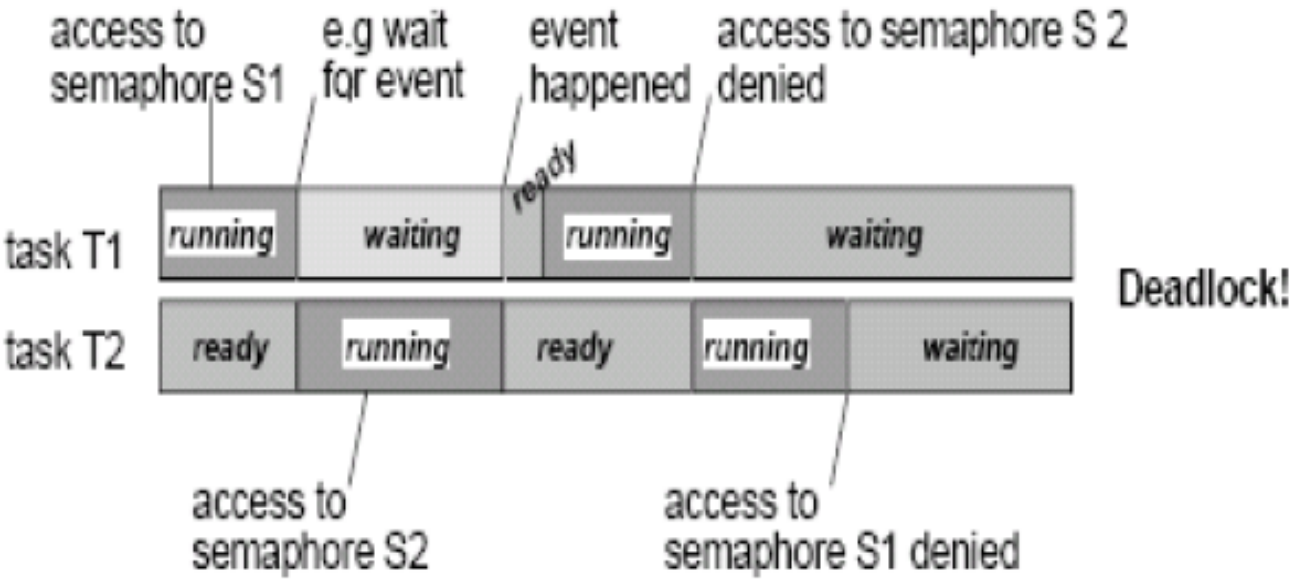
- **Priority Inversion**



- missed time deadlines
- gaps in data acquisition
- time gaps in the control of external devices
- In conjunction with a watchdog timer, these delays might even trigger system resets

Deadlocks

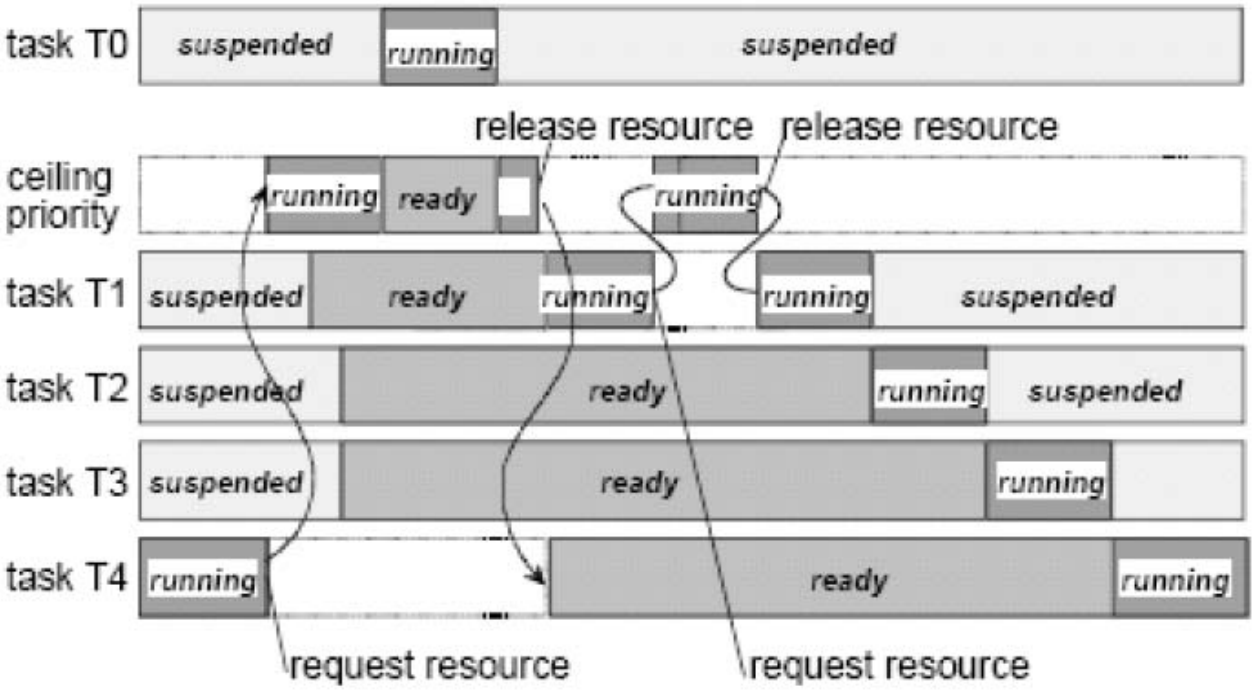
- A typical problem of common synchronization mechanisms, such as the use of semaphores
- Means the impossibility of task execution due to infinite waiting for mutually locked resources



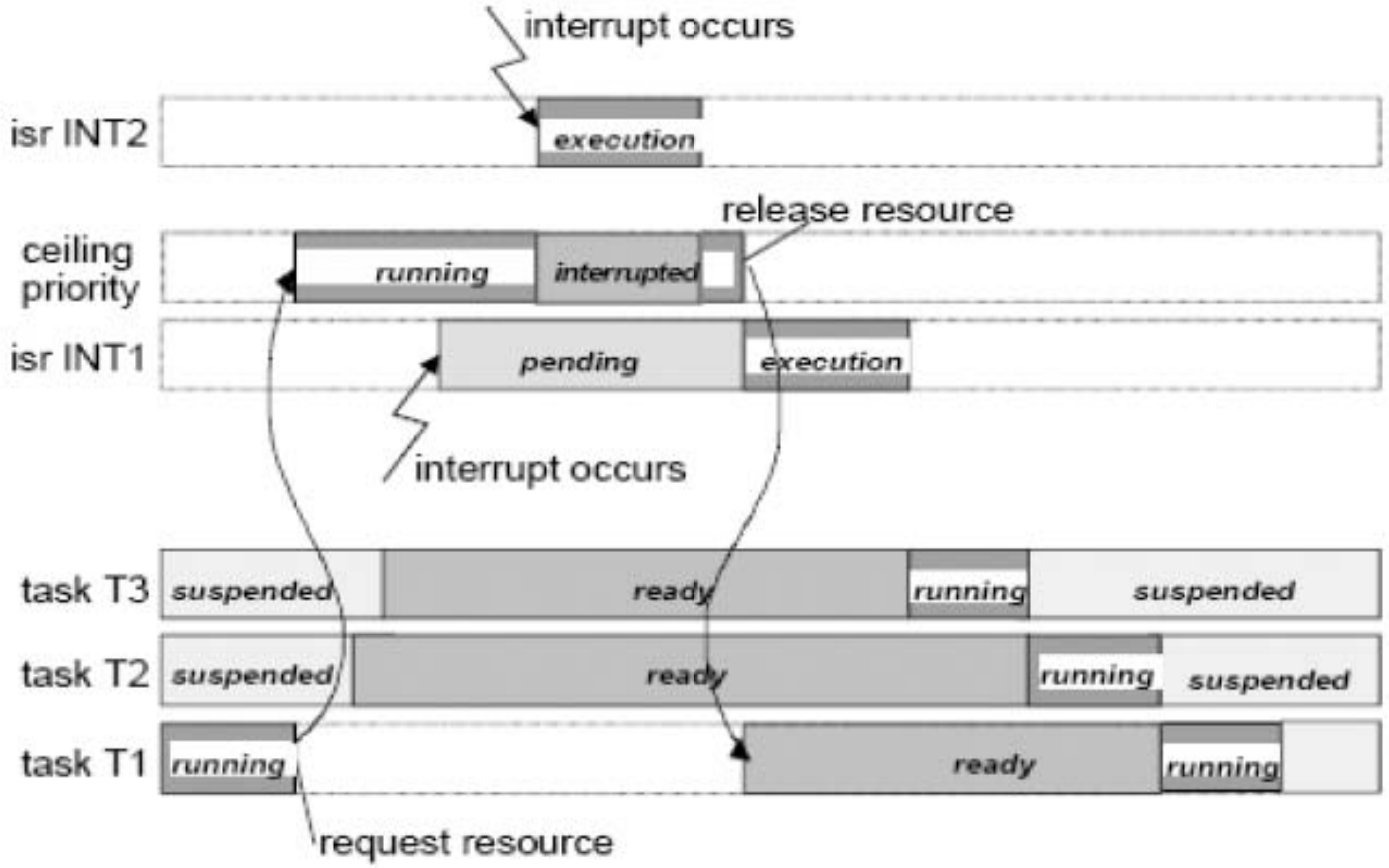
OSEK Priority Ceiling Protocol

- At system generation - to each resource its own ceiling priority is statically assigned.
- The ceiling priority - set at least to the highest priority of all tasks accessing a resource
- The ceiling priority - lower than the lowest priority of all tasks that do not access the resource
- What happens when task requires or releases a resource ?
    - priority

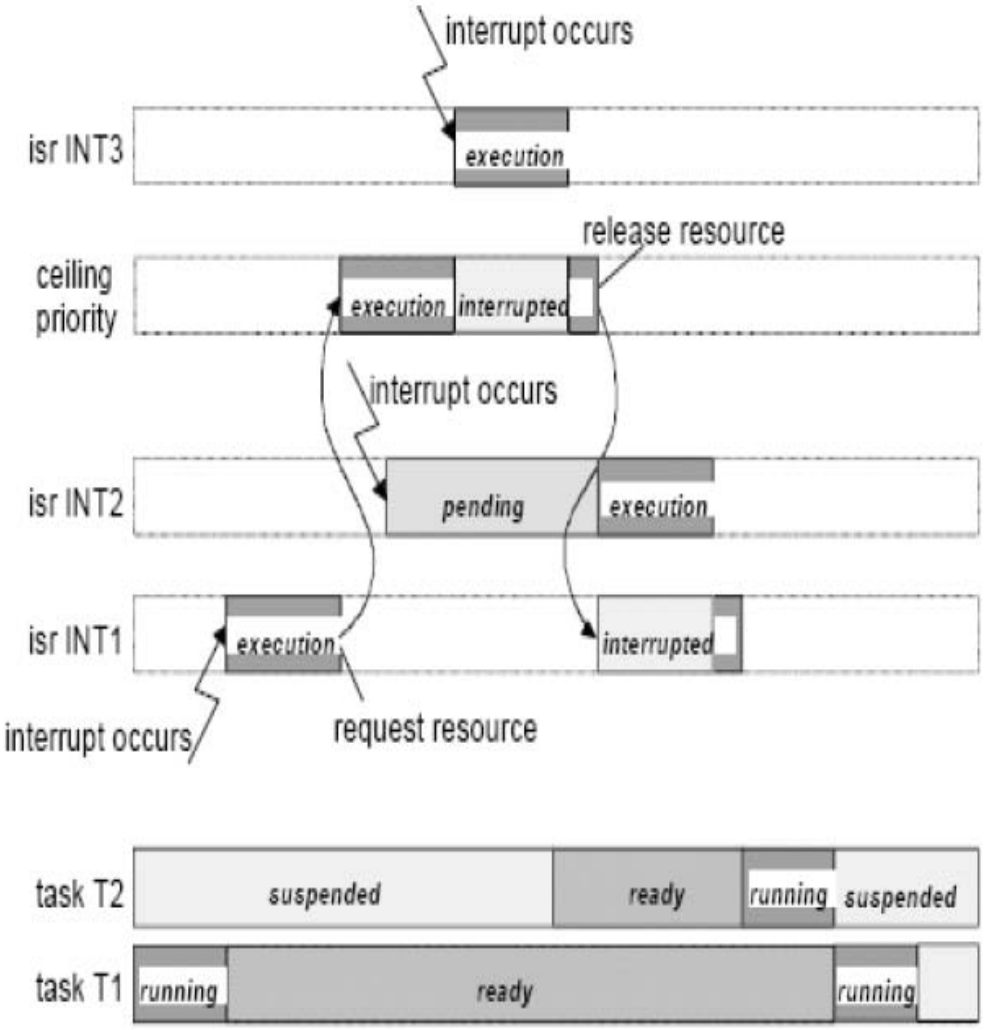
OSEK Priority Ceiling Protocol



Pre-emptable Tasks and ISR



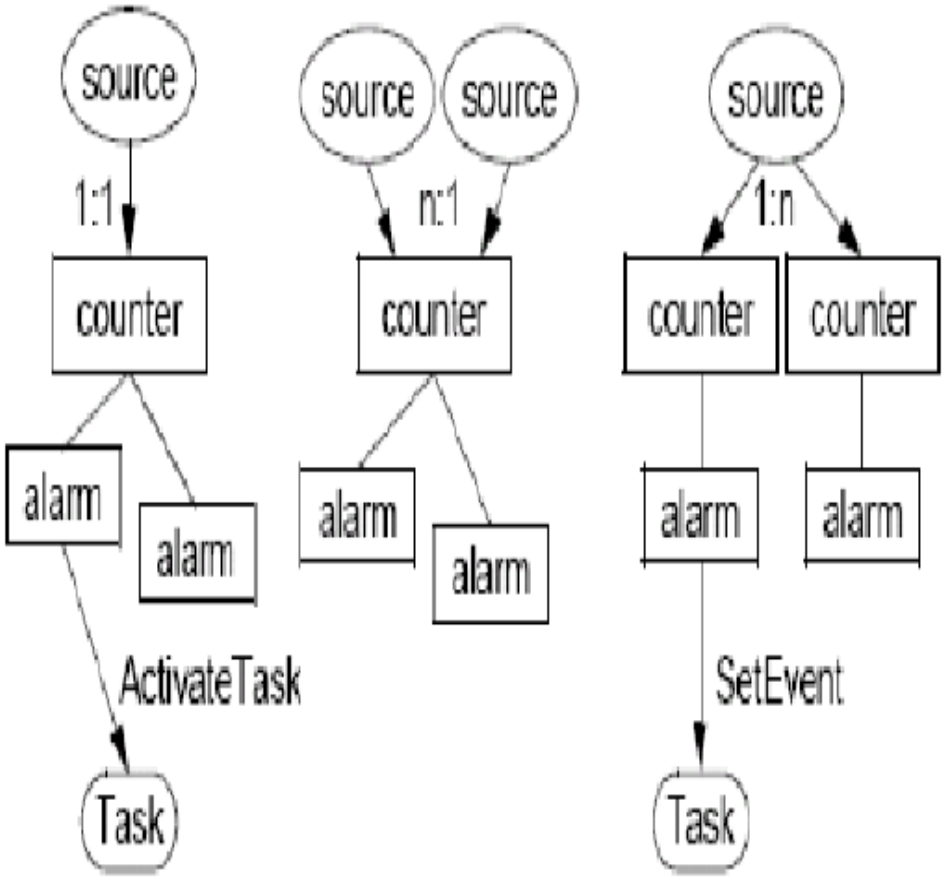
Priority Ceiling between ISR



Resource assignment with priority ceiling between interrupt services routines

Counters and Alarms

- In embedded systems, system tasks and user tasks often schedule and perform activities after some time has elapsed or to carry out recurring operations.
- The OSEK operating system provides a two-stage concept to process such events



Counters and Alarms

- A counter is an OS object - keeps track of the number of ticks that have occurred i.e., the value is measured in ticks
- Recurring events (sources) are registered by implementation specific counters
- Counter specific constants are defined in the OIL configuration file
- The counter is used by alarms as a System timer

  Few API used

  – *GetCounterInfo and GetCounterValue, CounterTrigger*

Alarms

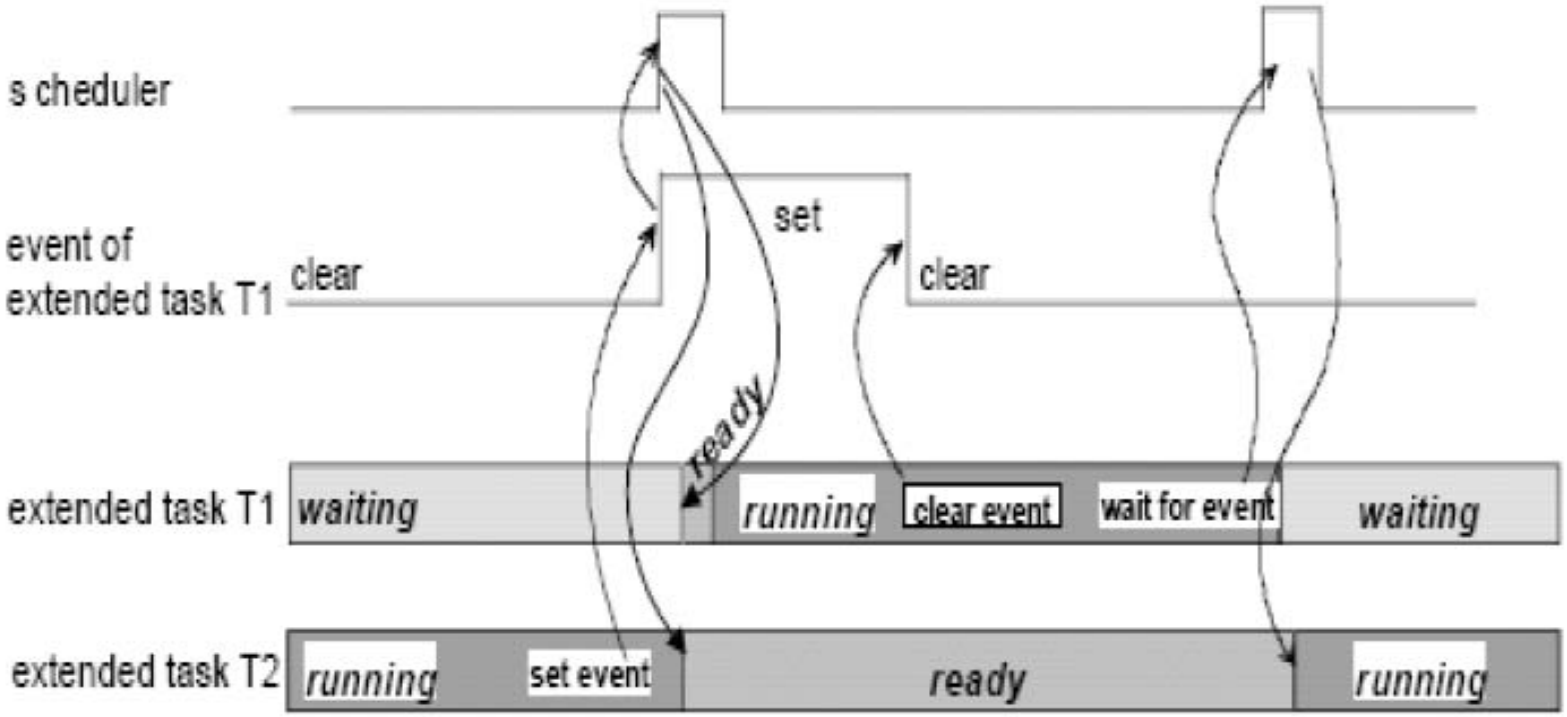
- The alarm management is built on top of the counter management
- Allows the user to link task activation or event setting or a call to callback function to a certain counter value
- Alarms can be either single (one-shoot) or cyclic alarms
- The user can set alarms, cancel alarms and read information out of alarms by means of system services

  SetAbsAlarm or *SetRelAlarm* are used to start an alarm
- User can define alarm callback function for each alarm

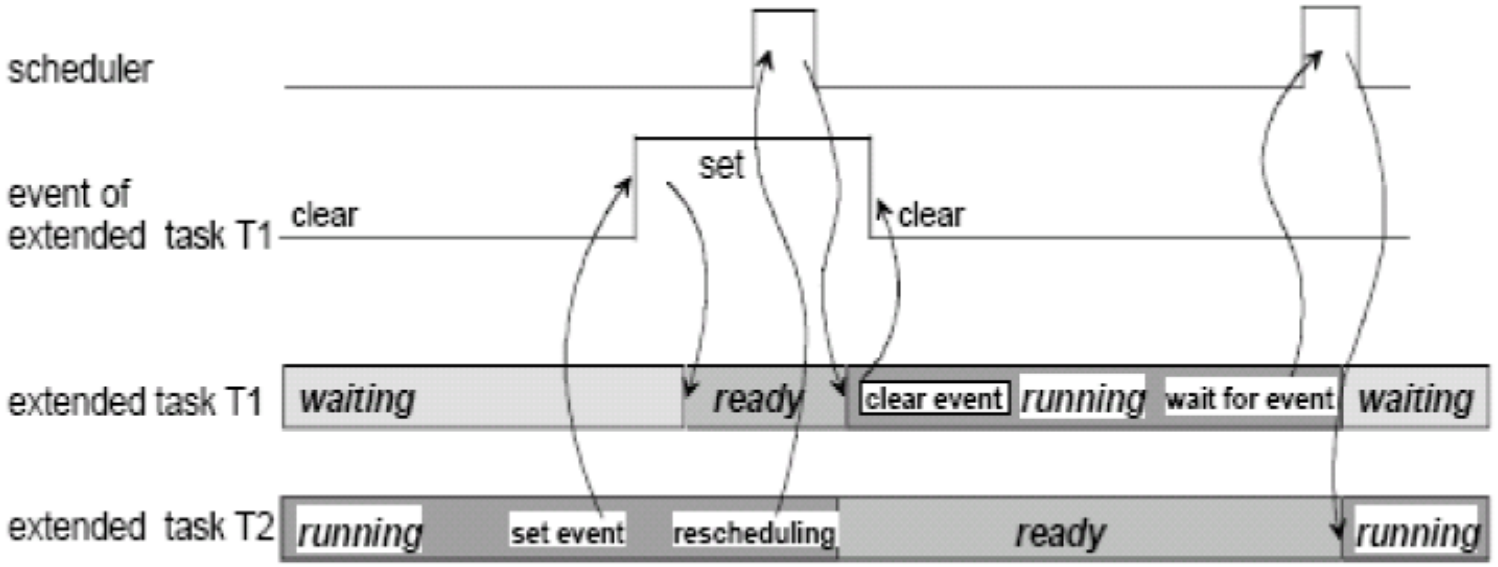
Event Mechanism

- Events are the criteria for the transition of extended tasks from the waiting state into the ready state
  - Means of synchronization
  - Only for extended tasks
  - Initiates state transitions of tasks to and from the waiting state
  - Events are "owned" by an extended task
  - Any task, including basic tasks, can set an event. But cleared by the owner only
  - Only owner task can clear the event or wait for the event
  - Extended Task can wait for several events
- Examples are: the signaling of a timer's expiry ( ) , the availability of data (data available on an GPS device), the receipt of a message (data available on the in vehicle network), etc

Synchronization of Pre-emptable ET



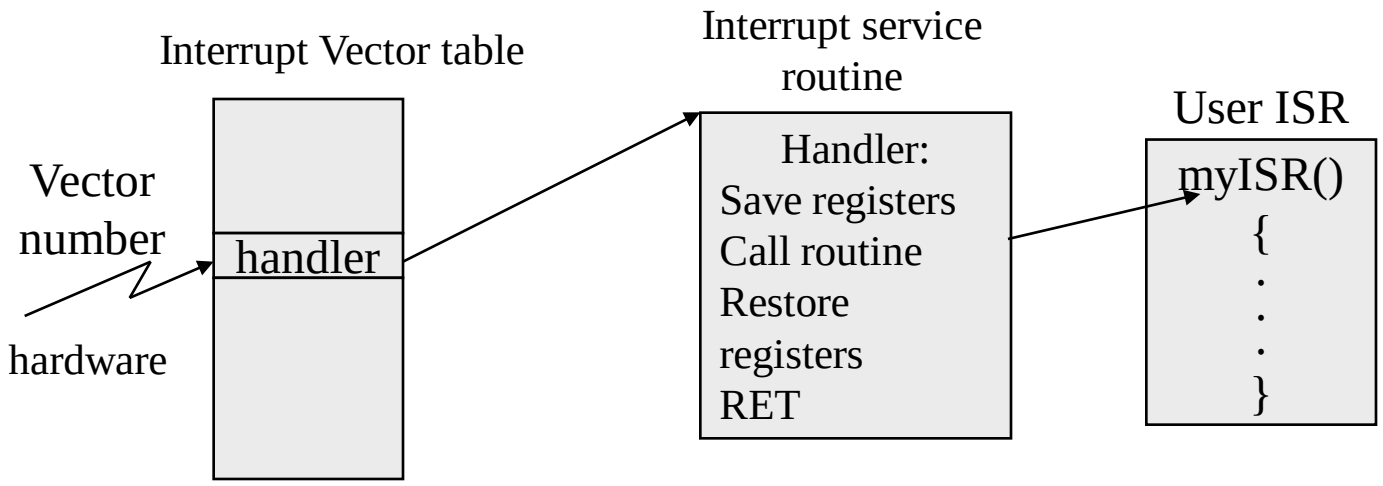
Synchronization of Non-Preemptable ET



- If non preemptive scheduling is supposed, rescheduling does not take place immediately after the event has been set

Interrupt Processing

- Alter the sequence of instructions executed by a process
- Divert the processor to code from the normal flow of control
- Provides an efficient way to handle unanticipated events



Interrupt Processing in OSEK

- In the OSEK Operating System two types of Interrupt Service Routines
- ISR Category 1
  - Can not use operating system service
  - The interrupt has no influence on task management
  - Have the least overhead
- ISR Category 2
  - During system generation the user routine is assigned to the interrupt
  - Rescheduling takes place on termination of the ISR category 2 and according to OSEK scheduling points
- Separate Interrupt Stack

ISR Processing and Restrictions

- ISRs can communicate with tasks in the OSEK OS by the following means:
    - Activate a task; get task ID
    - Send/receive an messages of certain kind only
    - trigger a counter
    - Get state of the task;
    - Set event for a task;
    - Manipulate alarms

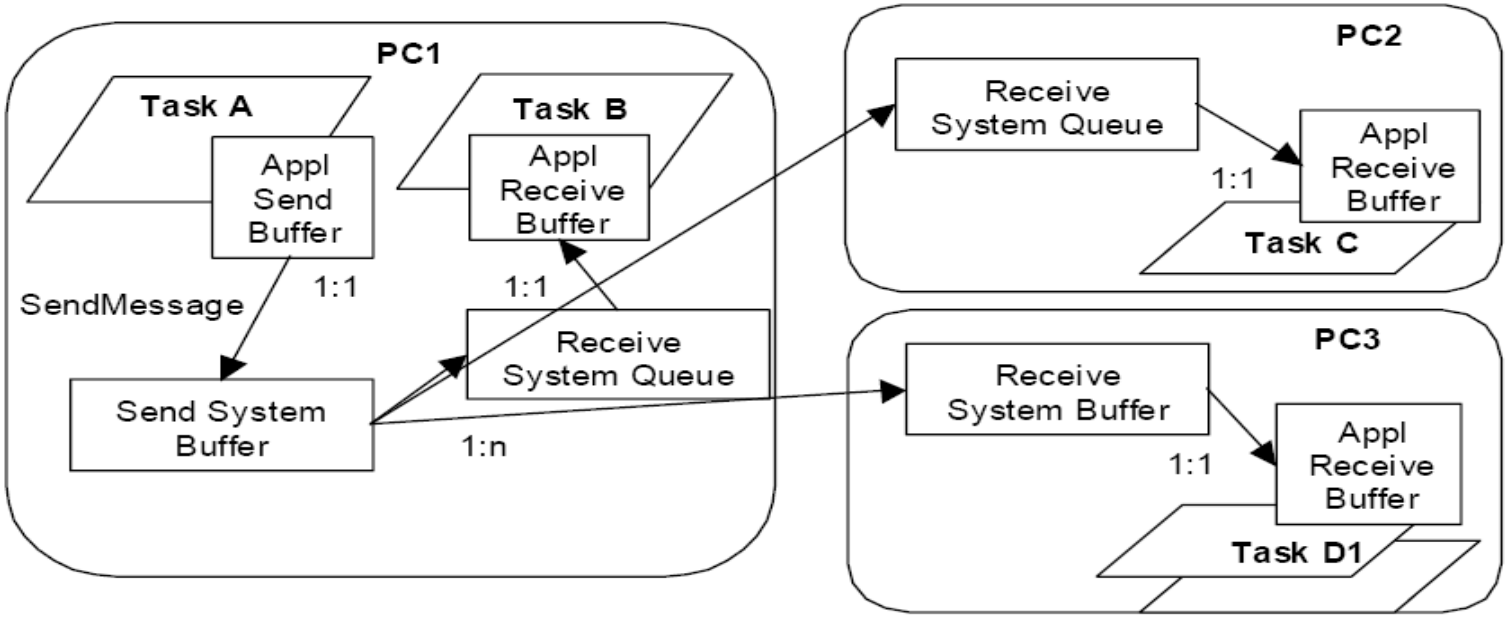
- Inside the ISR no rescheduling will take place

Services Allowed for Use in ISRs of Category 2

Service Name	Description
ActivateTask	Activates the specified task (puts it into the *ready* state)
GetTaskId	Gets reference to a task
GetTaskState	Gets state of the task
GetResource	Occupies a resource
ReleaseResource	Releases a resource

Message Communication Model

Communication between application tasks and/or ISRs takes place via messages



- Initiation of the communication with a SendMessage
- Transfer of the message data from the sending system buffer to the receiving system buffer
- Message receipt notification action in form of a task dispatch

Communication

- OSEK OS supports CCCB (Communication Conformance Class)
  - ***Unqueued***
  - ***Queued Messages***

- **Tasks and ISRs -** personal assessors to read/write message
- **Accessor type :** SENT and RECEIVED
- **Accessing Messages:** directly through the message buffer or indirectly through the message copy

Unqueued Messages

- Unqueued Message : represents the current value of a system variable, e.g. engine temperature, wheel speed, etc
  - Send operation : overwrites the current value of a message
  - Receive operation : reads the current value of message, message data is not consumed
- Allocation of memory for message copies depends on the *MessageCopyAllocation*1 attribute

Queued Messages

- **Queued Messages:** messages are stored in a FIFO buffer and they are read by the application in the order they arrived

- Only **tasks** (not ISRs) have personal accessors to read/write message
  - **Send operation:** end of the message queue
  - **The receive operation:** first value of a Queued Message then removes it from the queue. Queued Message can be read only once

Error Handling, Tracing and Debugging

- The OSEK operating system provides system specific hook routines to allow user-defined actions within the OS internal processing
- Hook routines are
  - Called by the operating system, in a special context
  - Higher priority than all tasks
  - Not interrupted by category 2 interrupt routines
  - Part of the operating system
  - Implemented by the user with user defined functionality
  - Standardised in interface, but not standardised in functionality
  - Are only allowed to use a subset of API functions (like ISR)
  - Configurable via OIL

Hook Routines

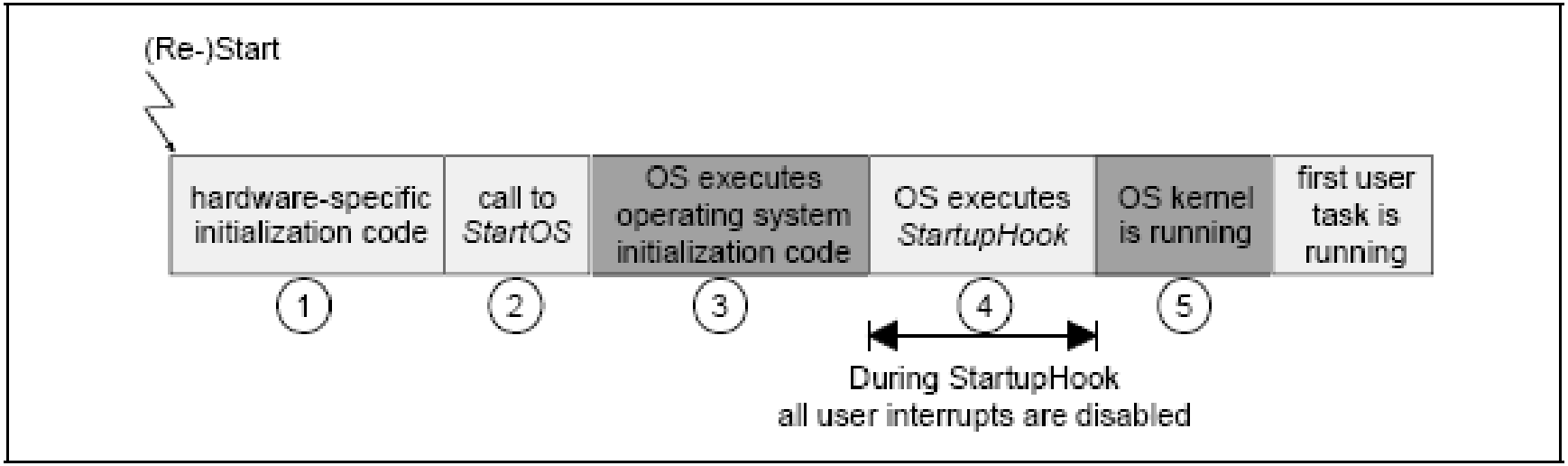
- The OSEKturbo OS provides the following hook routines:

- *ErrorHook*, *PreTask*Hook, *PostTaskHook*, *StartupHook*, *ShutdownHook* and *IdleLoopHook*

- In the OSEK operating system hook routines may be used for:
  - System start-up
  - System shutdown
  - Tracing or application dependent debugging
  - Error handling

System Start-up

- OSEK OS offers support for a standardised way of initialisation
- OSEK OS does not force the application to define special tasks which shall be started after the operating system initialisation, but it allows the user to specify autostart tasks and autostart alarms during system generation
- After a reset of the CPU, hardware-specific application software is executed



System Shutdown

- The OSEK OS specification defines a service to shut down the operating system, *ShutdownOS*
- This service can be requested by the application or by the operating system due to a fatal error
- When *ShutdownOS* is called the operating system will call the hook routine *ShutdownHook* and shut down afterwards

Summary

- Task in an application are generated using the TASK system generation object with the set of properties in OIL file
- A counter is an OS object which keeps the track of the number of ticks that have occurred
- Events are the criteria for the transition of extended tasks from the waiting state into the ready state
- Interrupts provides an efficient way to handle unanticipated events
- The OSEK operating system provides system specific hook routines to allow user-defined actions within the OS internal processing
- When ShutdownOS is called the operating system will call the hook routine ShutdownHook and shut down afterwards