OSEK/VDK Operating System (2)

3.5 Application modes ~ 3.13 Specification of OS Service

Contents

3.5 Application modes

What is 'application mode' ?

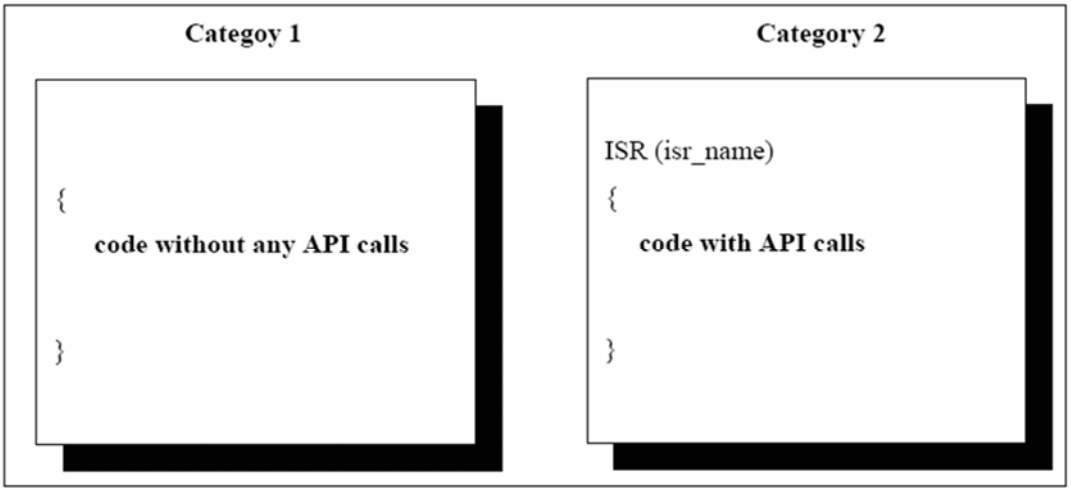
- Application modes are designed to allow an OSEK operating system to come up under different modes of operation.
- It is intended only for modes of operation that are totally mutually exclusive.
- An example of two exclusive modes of operation would be end-of-line programming and normal operation.
- The application mode is a means to structure the software running in the ECU according to those different conditions and is a clean mechanism for development of totally separate systems.

3.6 Interrupt processing

Interrupt processing

- ISR category 1:
  - does not use an operating system service.
  - After the ISR is finished, processing continues exactly at the instruction where the interrupt has occurred
  - has the least overhead.
- ISR category 2 :
  - The 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.

Interrupt processing



Interrupt processing

- ISR common features
  - Inside the ISR no rescheduling will take place.
  - The scheduling of interrupts is hardware dependent and not specified in OSEK(Interrupts are scheduled by hardware).
- Fast Disable/Enable API-functions
  - *EnableAllInterrupts/DisableAllInterrupts,*
  - *ResumeAllInterrupts/SuspendAllInterrupts*
  - *ResumeOSInterrupts/SuspendOSInterrupts*

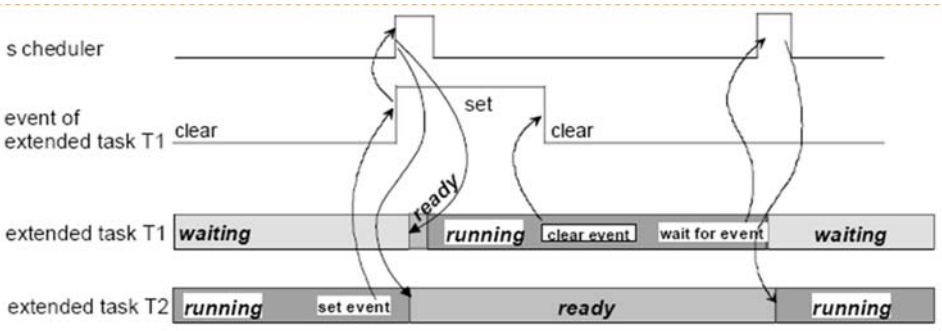
3.7 Event mechanism

Events

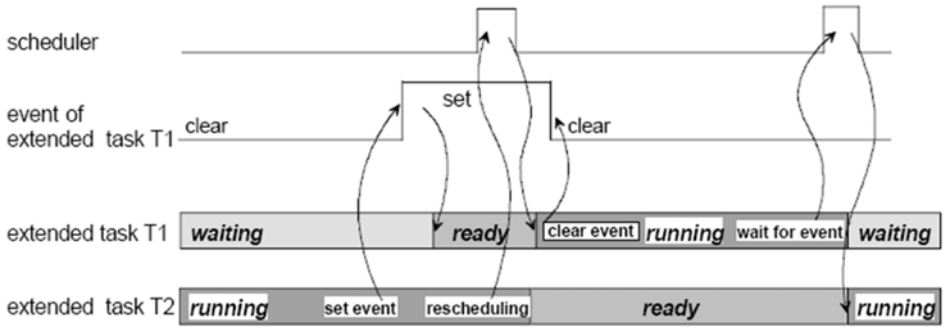
- The event mechanism
  - is a means of synchronization.
  - is only provided for extended tasks.
  - initiates state transitions of tasks to and from the *waiting state*.
- Features
  - Events can be used to communicate binary information to the extended task to which they are assigned.
  - The meaning of events is defined by the application, e.g. signaling of an expiring timer, the availability of a resource, the reception of a message, etc.
  - 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).

- Features
  - Any task or ISR of category 2 can set an event for a not suspended extended task.
  - Only the owner is able to clear its events and to wait for setting of its events.
  - Events are the criteria for the transition of extended tasks from the *waiting state into the ready state* if at least one event for which the task is waiting for has occurred.

Synchronization of preemptable/non-preemptable extended tasks



Synchronization of preemptable extended tasks



Synchronization of non-preemptable extended tasks

3.8 Resource management

Resource

- Kinds
  - Scheduler
  - Program sequences - CPU
  - Memory
  - Hardware areas
- Resource management ensures that
  - two tasks or interrupt routines 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*.

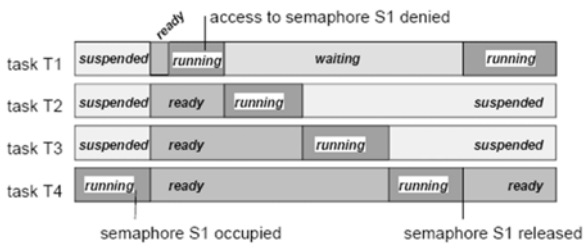
Resource management Features

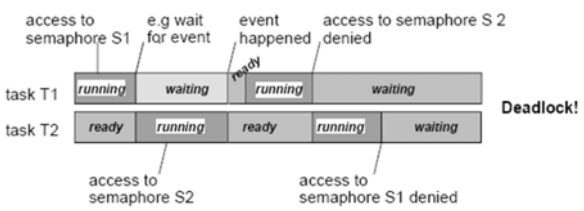
- *TerminateTask, ChainTask, Schedule, WaitEvent* shall not be called while a resource is occupied.
- The scheduler is treated like a resource which is accessible to all tasks. (RES_SCHEDULER)

8.4 General problems with synchronization mechanisms

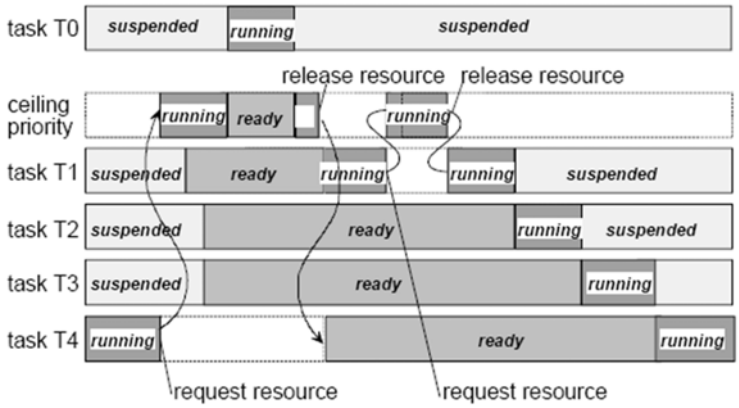
Pro 1.Explanation of priority inversion

Pro 2. Deadlocks



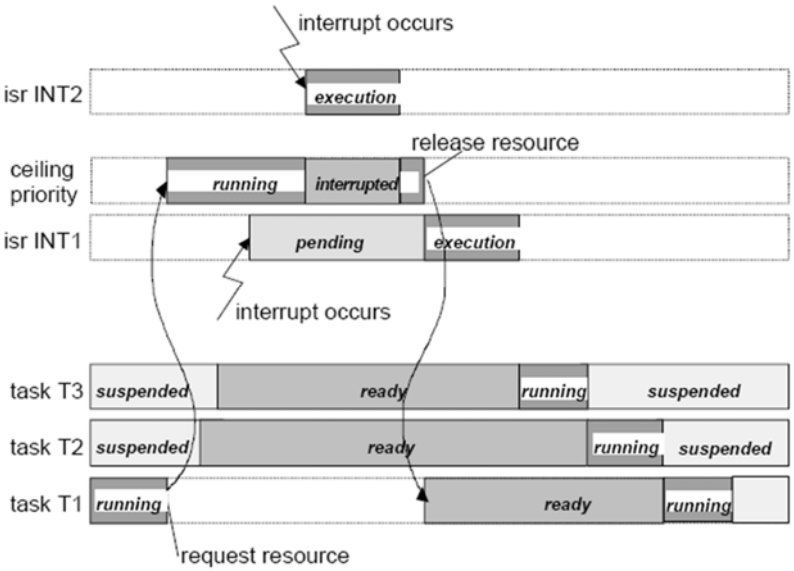


8.5 OSEK Priority Ceiling Protocol



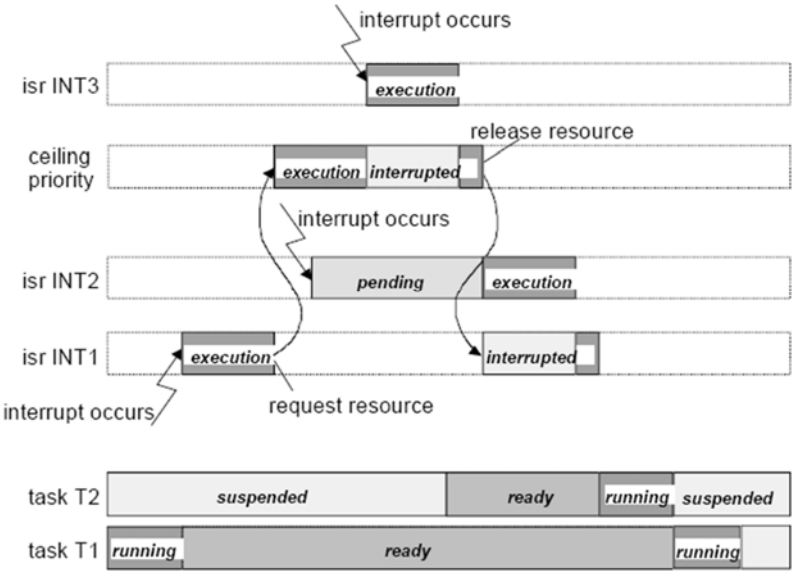
Resource assignment with priority ceiling between preemptable tasks

8.6 OSEK Priority Ceiling Protocol with extensions for interrupt levels(1)



Resource assignment with priority ceiling
between preemptable tasks and interrupt services routines

8.6 OSEK Priority Ceiling Protocol with extensions for interrupt levels(2)



Resource assignment with priority ceiling between interrupt services routines

8.7 Internal Resources

- not visible to the user
- can not be addressed by the system functions *GetResource and ReleaseResource.*
- **Instead,** they are **managed strictly internally** within a clearly defined set of system functions
- **the behavior** of internal resources is **exactly the same** as **standard resources.**
- **At most one** internal resource can be assigned to a task during system generation.

3.9 Alarms

1. Counter

- A counter is represented by a counter value, measured in ”ticks”, and some counter specific constants.
- The OSEK OS offers at least one counter that is derived from a (hardware or software) timer.

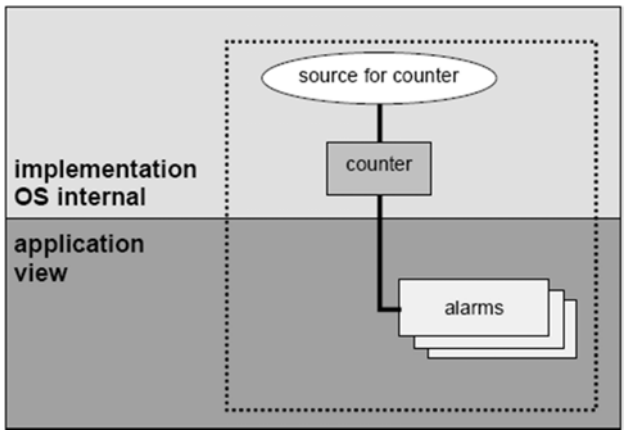


Fig 1. Layered model of alarm management

2. Alarm management(1)

- An alarm will expire when a predefined counter value is reached.
- Feature
  - Relative alarm - be defined relative to the actual counter
  - Absolute alarm - be defined as an absolute value
  - single/cyclic alarm
  - More than one alarm can be attached to a counter.
  - Counters and alarms are defined statically.
  - The assignment of alarms to counters and the action to be performed when an alarm expires, are defined statically.

2. Alarm management(2)

- Dynamic :
  - counter value
  - the period for cyclic alarms
- is assigned statically at system generation time to
  - one counter
  - one task or alarm-callback routine
- When the alarm expires
  - alarm-callback routine is called
  - task is activated
  - an event is set for this task

3. Alarm-callback routines

- An alarm-callback routine is a short function provided by the application.
  - Prototype

    ALARMCALLBACK(AlarmCallbackRoutineName);
  - Example :

```
ALARMCALLBACK(BrakePedalStroke)
{
/* do application processing */
}
```


Messages

Messages

- Part 4. OSEK/VDK Communication(COM)

Error Handling, Tracing and Debugging

Contents

- "Error Handling, Tracing and Debugging"
  1. Hook routines
  2. Error handling
  3. System start-up
  4. System shutdown
  5. Debugging

11.1 Hook routines(1)

- The OSEK OS provides system specific hook routines to allow user-defined actions within the OS internal processing.
- Hook routines are
  1. called by the operating system
  2. higher prior than all tasks
  3. not interrupted by category 2 interrupt routines.
  4. part of the operating system
  5. implemented by the user with user defined functionality
  6. standardized in interface, but not standardized in functionality therefore usually hook routines are not portable.
  7. are only allowed to use a subset of API functions
  8. mandatory, but configurable via OIL

11.1 Hook routines(2)

- Purpose:
  - system start-up
  - system shutdown
  - tracing or debugging
  - error handling

2. Error handling

- Purpose : to handle temporarily and permanently occurring errors within the OSEK OS.
- Feature :
  - Its basic framework is predefined and shall be completed by the user.
  - So efficient centralized or decentralized error handling

- Two kinds of errors
  - Application errors
  - Fatal errors
- All those error services are assigned with a parameter that specifies the error.

- Two levels of error checking
  - standard status
    - return : "E_OK" or “Too many task activations”
  - extended status
    - return : "E_OK“, “Too many task activations”, "Task is invalid" or "Task still occupies resources", etc.

Error hook routine

- When 'error hook routine (***ErrorHook***) ' is called?
  - *if a system service returns a StatusType value not equal to E_OK.*
  - if an error is detected during task activation or event setting
  - a recursive call of error hook never occurs.

Error management

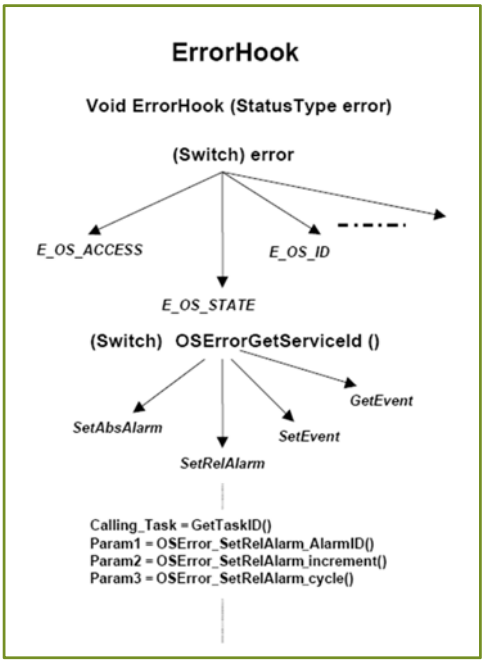


Fig 2. Example of centralised error handling (extended status)

- OSErrorGetServiceId() provides the service identifier where the error has been risen.
- Possible values : OSServiceId_xxxx
  - (xxxx is the name of the system service)

3. System start-up

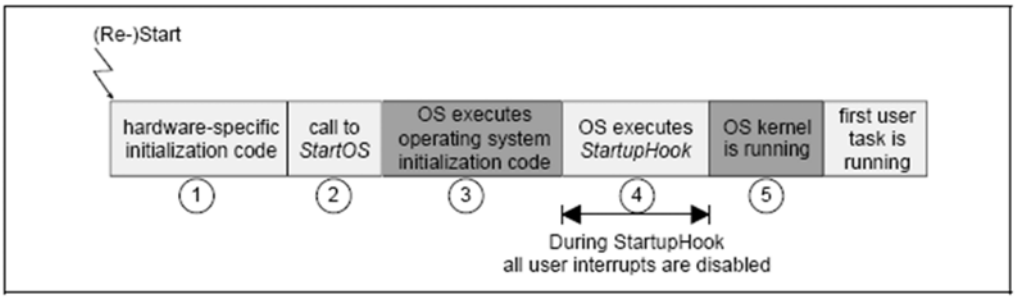


Fig 3. System start-up

4. System shutdown

- *ShutdownOS* : a service to shut down the operating system
  - called by application or OS due to a fatal error

(This case can be applied only OSEK OS)

5. Debugging

- When task context switches,

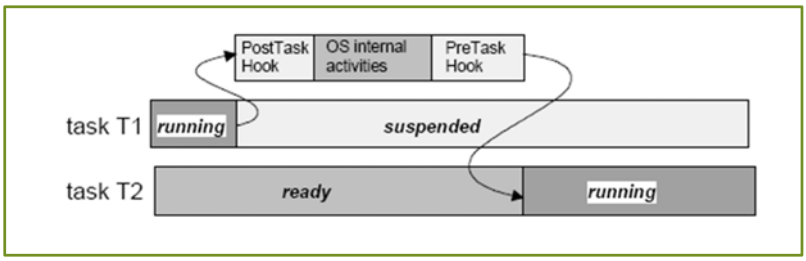


Fig 4. PreTaskHook and PostTaskHook

- Two hook routines may used for debugging or time measurement (including context switch time)

Description of system services

Contents

- "Description of system services"
  1. Definition of system objects
  2. Conventions

1. Definition of system objects

- all system objects have to be determined statically by the user.
  - The actual creation of the objects (unique names and specific characteristics) is done during the system generation phase.
  - The names are used as identifiers within the system services

2. Conventions

1. Type of calls
   - The system service interface is ISO/ANSI-C.
   - Its implementation is normally a function call.
2. Legitimacy of calls
   - System services are called from tasks, interrupt service routines, hook routines, and alarm-callbacks.
   - Depending on the system service, there may be restrictions regarding the availability.
   - Further restrictions are imposed by the conformance classes.

Service	Task	ISR category 1	ISR category 2	ErrorHook[12]	PreTaskHook	PostTaskHook	StartupHook	ShutdownHook	alarm-callback
ActivateTask	✓		✓						
TerminateTask	✓								
ChainTask	✓								
Schedule	✓								
GetTaskID	✓		✓	✓[13]	✓	✓			
GetTaskState	✓		✓	✓	✓	✓			
DisableAllInterrupts	✓	✓	✓						
EnableAllInterrupts	✓	✓	✓						
SuspendAllInterrupts	✓	✓	✓	✓	✓	✓			✓
ResumeAllInterrupts	✓	✓	✓	✓	✓	✓			✓
SuspendOSInterrupts	✓	✓	✓						
ResumeOSInterrupts	✓	✓	✓						
GetResource	✓		✓						
ReleaseResource	✓		✓						
SetEvent	✓		✓						
ClearEvent	✓								
GetEvent	✓		✓	✓	✓	✓			
WaitEvent	✓								
GetAlarmBase	✓		✓	✓	✓	✓			
GetAlarm	✓		✓	✓	✓	✓			
SetRelAlarm	✓		✓						
SetAbsAlarm	✓		✓						
CancelAlarm	✓		✓						
GetActiveApplicationMode	✓		✓	✓	✓	✓	✓	✓	
StartOS									
ShutdownOS	✓		✓	✓			✓		

Fig 5. API service restrictions

- [12] Behavior of system services is only defined for the services marked in the table.
- [13] It may happen that currently no task is *running*. *In this case the service returns the task ID INVALID_TASK*