As of 23.05.2026

RTOS Basics and Application: RTOS Mechanisms and their Application in Runtime Architectures for Embedded and Real-Time Systems - Live Online Training

The growing complexity of embedded software applications and the ever increasing performance of hardware has resulted in more and more real-time operating systems being included in software. The use of real-time operating systems poses new challenges for developers.

Objectives

After the RTOS basic training, you know the mechanisms of sophisticated real-time operating systems (RTOS) and can use them to develop new software runtime architectures and maintain existing ones - independent of the concrete real-time system product.

You are able to document and communicate software runtime architectures and make an informed choice about operating systems.

Make your system fit for IoT by means of the RTOS and additional communication stacks.

Participants

The RTOS training addresses programmers, software developers, software designers and software architects who are using or are planning to use a real-time operating system in their embedded software application.

Requirements

Basic knowledge of microcontrollers as well as C programming skills.

Live-Online-Training

27.07. – 30.07.2026	2.600,00 €	4 Days

* Price per attendee, in Euro plus VAT

Training code: LE-RTOS-AR

Face-To-Face - English

Duration
4 days

Live Online - German

Date	Duration
27.07. – 30.07.2026	4 days

Face-To-Face - German

Date	Duration
19.10. – 22.10.2026	4 days

As of 23.05.2026

RTOS Basics and Application: RTOS Mechanisms and their Application in Runtime Architectures for Embedded and Real-Time Systems - Live Online Training

Content

General Introduction: Real-Time Operating Systems

- Key terminology (operating system, real-time, task, multitasking, scheduler)
- Classification of embedded systems
- Classification of operating system types
- Requirements for the operating system, hardware, development tools
- License models
- OSAL Operating System Abstraction Layer
- POSIX (pThread)
- Benefits, advantages and disadvantages when using an operating system
- Practical example: Splitting an application into tasks

Process/Thread/Task Management

- Differentiation between process, task and thread
- Task states and transitions
- Task properties and multiple instantiation
- Specific tasks
- Task context switch and hook routines
- Design aspects for tasks
- Schedulers and their algorithms (endless loop, time-triggerd, priority, time slice, round robin, EDF earliest deadline first)
- Scheduler functionality and design aspects
- Selecting the right scheduler
- Typical use in embedded software applications
- API (application programming interface) examples
- Exercise: Programming a task, instantiating it twice with different priority combinations; subsequent analysis of the behavior

Interrupt Management

- Interrupt processing with and without operating system
- Interrupt latency and interrupt blocking time
- Prioritization
- Interrupt service routines
- Interrupt threads/tasks
- Design notes
- API (application programming interface) examples
- Exercise: Programming an interrupt service routine and a task to service an AD converter

Time Management

- System tick and configuration
- Delay, timeout, interval, software watchdog, alarm timer
- Typical use in embedded software applications
- API (application programming interface) examples
- Exercise: Programming interval timer controlled task execution

Coordination Mechanisms: Synchronization

- Events, signals: global, local, single, as group, with/ without parameters
- Semaphore, promise and future
- Typical use in embedded software applications
- API (application programming interface) examples
- Exercise: Programming synchronization between an interrupt service routine and a task

Coordination Mechanisms: Resource Management

- Race conditions
- Resource: definition, granularity and blocking times
- Semaphore, mutex, critical section, condition variable, spinlock
- Problem scenarios: deadlock and priority inversion
- Solutions: priority inheritance, priority ceiling etc.
- Reader/ writer patterns

- Typical use in embedded software applications
- API (application programming interface) examples
- Exercise: Programming protected access to a resource shared by two tasks

Communication Mechanisms

- Message concepts: system local and cross-system
- Shared memory, mailboxes, queues, message buffer, pipes, message based, socket
- Typical use in embedded software applications
- API (application programming interface) examples
- Exercise: Programming communication between an interrupt service routine and a task and between two tasks using the mailbox concept

Memory Management

- Memory segments (BSS, stack, heap)
- Stack monitoring
- Dynamic memory management
- Memory pools and memory blocks
- Pool allocation pattern: memory pools and memory blocks
- MPU (memory protection unit) and MMP (memory management unit) support
- Typical use in embedded software applications
- API (application programming interface) example
- Identifying risks and avoiding pitfalls

Input/Output Management

- Software layer architecture
- Driver concepts
- Examples with serial and Ethernet communication
- Typical use in embedded software applications
- API (application programming interface) example

Debugging at Operating System Mechanism Level

- Tracing at operating system level
- Introduction and assessment of tracing and visualization alternatives
- Presentation with logic analyzer and professional trace tools (Percepio Tracealyzer and ARM Keil MDK)

Embedded and Real-Time Software Development Procedure

- From the idea to the final runtime architecture
- Runtime architecture patterns and their use for concrete tasks
- Transformation of an existing software architecture without operating system to an architecture with operating system, with optimized utilization of the OS mechanisms
- Introduction and comparison of different runtime architecture concepts - with and without operating system
- Predictability and calculability of various runtime architecture concepts
- Guidelines for selecting the "right" runtime concept
- Practical example: Measurement device application
- Exercise: Development of a suitable runtime architecture for a real embedded system based on drawn up textual requirements and a software architecture

Multicore and Multiprocessor Aspects

- Hardware and software architectures
- Task assignment
- Possible operating system applications
- Virtualization and hypervisor
- Interrupt and driver concepts
- Important design aspects

Documentation and Communication

- Suitable modeling formats for a runtime architecture
- Excerpts from UML (Unified Modeling Language)
- Practical tips
- Exercise: Using UML notations and diagrams for modeling the runtime architecture

Operating System Abstraction Layer OSAL

- Description of benefits, advantages and disadvantages
- Programming
- Practical example with FreeRTOSTM

Operating System Selection Guidelines and Product Overview

- Operating system selection guidelines
- Practical tip: Operating system comparison
- Current product overview for embedded software
- Checklist with important selection criteria

Practical Exercises

- Throughout the programming exercise, you will use operating system mechanisms for developing a measurement device application step by step.
- The exercise is performed using the Arm Keil MDK (microcontroller development kit) with real hardware based on an Arm Cortex® microcontroller.
- For the real-time operating system, you choose between FreeRTOS™ and the Arm Keil implementation of the CMSIS-RTOS standard.
- You develop and document a runtime architecture for an electric motor control application using operating system mechanisms.
- The exercises are performed using Enterprise Architect (Sparx Systems) or paper and pencil.

MicroConsult PLUS:

- We will provide you with a download link for your exercises and the solutions developed by MicroConsult from this workshop.
- For the measurement device application, you get the program code and a UML model as well as a UML model for the electric motor application.
- You get a tool and software component overview including a current operating system overview.
- You also get a checklist with operating system selection criteria.
- You get helpful notation overviews for UML and SysML.