

RTOS-Grundlagen und Anwendung: Mechanismen und deren Einsatz in Laufzeit-Architekturen für Embedded- und Echtzeitsysteme - Präsenz-Training

Angesichts steigender Komplexität in Embedded-Software-Applikationen und immer leistungsfähigerer Hardware werden auch immer mehr Echtzeitbetriebssysteme in die Software mit eingebunden. Der Einsatz von Echtzeit-Betriebssystemen stellt neue Herausforderungen an die Entwicklung.

Ziele - Ihr Nutzen

Sie kennen nach dem RTOS-Grundlagen-Training die Mechanismen moderner Echtzeit-Betriebssysteme und können damit neue Software-Laufzeitarchitekturen entwickeln und bestehende warten - unabhängig von dem konkreten Echtzeitbetriebssystem-Produkt.

Sie können Software-Laufzeitarchitekturen dokumentieren und kommunizieren und eine fundierte Betriebssystem-Auswahl treffen.

Mit dem Betriebssystem und zusätzlichen Kommunikationsstacks machen Sie Ihr System IoT-fähig.

Teilnehmer

Der RTOS-Kurs richtet sich an Programmierer, Software-Entwickler, Software-Designer und Software-Architekten, die aktuell oder zukünftig ein Echtzeitbetriebssystem in ihrer Embedded-Software-Applikation einsetzen.

Voraussetzungen

Grundkenntnisse über Mikrocontroller sowie Programmierkenntnisse in C.

RTOS-Grundlagen und Anwendung: Mechanismen und deren Einsatz in Laufzeit-Architekturen für Embedded- und Echtzeitsysteme - Präsenz-Training

Inhalt

Allgemeine Einführung in Echtzeitbetriebssysteme

- Wichtige Grundbegriffe (Betriebssystem, Echtzeit, Task, Multitasking, Scheduler)
- Klassifikation von Embedded-Systemen
- Klassifikation von Betriebssystem-Arten
- Anforderungen an Betriebssystem, Hardware, Entwicklungstools
- Lizenzmodelle
- Betriebssystem-Abstraktionsschicht (OSAL Operating System Abstraction Layer)
- POSIX (pThread)
- Nutzen, Vorteile und Nachteile beim Betriebssystem-Einsatz
- Praxisbeispiel: Aufteilung einer Applikation in Tasks

Prozess-/Thread-/Task-Management

- Differenzierung zwischen Prozess, Task und Thread
- Taskzustände und Übergänge
- Taskeigenschaften und Mehrfachinstanziierung
- Spezifische Tasks
- Task-Kontext-Umschaltung und Hook-Routinen
- Designaspekte für Tasks
- Scheduler und deren Algorithmen (Endless Loop, Time-triggered, Priority, Time-slice, Round-Robin, EDF Earliest Deadline First)
- Scheduler-Funktionalität und Designaspekte

- Die richtige Scheduler-Auswahl treffen
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren eine Task, instanzieren diese zweimal mit unterschiedlichen Prioritätskombinationen und werten das Verhalten aus

Interrupt Management

- Interrupt-Bearbeitung mit und ohne Betriebssystem
- Interrupt-Latenzzeit und Interrupt-Blockierzeit
- Priorisierung
- Interrupt-Serviceroutinen
- Interrupt-Threads/Tasks
- Designhinweise
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren eine Interrupt-Service-Routine und eine Task, um einen AD-Wandler zu servicen

Time Management

- Systemtick und Konfiguration
- Delay-, Timeout-, Intervall-, Software-Watchdog-, Alarm-Timer
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren eine Intervall-Timer-gesteuerte Taskausführung

Koordinationsmechanismen: Synchronisation

- Events, Signals: global, lokal, einzeln, als Gruppe, mit / ohne Parameter
- Semaphore, Promise und Future
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren eine Synchronisation zwischen einer Interrupt-Service-Routine und einer Task

Koordinationsmechanismen: Ressourcen-Management

- Race Conditions
- Ressource: Definition, Granularität und Blockierzeiten
- Semaphore, Mutex, Critical Section, Condition Variable, Spinlock
- Problemsituationen: Deadlock und Priority Inversion
- Lösungen: Priority Inheritance, Priority Ceiling u.a.
- Reader/ Writer Patterns
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren einen geschützten Zugriff von zwei Tasks auf eine gemeinsam genutzte Ressource

Kommunikationsmechanismen

- Nachrichtenkonzepte: System-lokal und System-übergreifend
- Shared-Memory, Mailbox, Queue, Message Buffer, Pipe, Message Based, Socket
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiele
- Übung: Sie programmieren die Kommunikation zwischen einer Interrupt-Service-Routine und einer Task sowie zwischen zwei Tasks mit dem Mailbox-Konzept

Speichermanagement

- Speichersegmente (BSS, Stack, Heap)
- Stack-Überwachung
- Dynamisches Speichermanagement
- Pool-Allocation-Pattern: Speicherpools und Speicherblöcke
- MPU (Memory Protection Unit) und MMP (Memory Management Unit) Unterstützung
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiel
- Risiken erkennen und Stolpersteine vermeiden

Input/Output Management

- Softwareschichten-Architektur
- Treiberkonzepte
- Beispiele mit serieller und Ethernet-Kommunikation
- Typische Anwendungen in Embedded-Software-Applikationen
- API (Application Programming Interface) Beispiel

Debugging auf der Ebene von Betriebssystem-Mechanismen

- Trace auf Betriebsmittelebene
- Vorstellung und Bewertung verschiedener Trace-Möglichkeiten und Darstellungsformen
- Vorführung mit Logik-Analyzer und professionellen Trace Tools (Perceptio Tracealyzer und ARM Keil MDK)

Vorgehensweise beim Entwurf von Embedded- und Echtzeitssoftware

- Von der Idee zur fertigen Laufzeitarchitektur
- Laufzeitarchitektur-Pattern und deren Einsatz für konkrete Aufgabenstellungen
- Transformation einer bestehenden Software-Architektur ohne Betriebssystem auf eine mit Betriebssystem unter optimaler Ausnutzung der Betriebssystem-Mechanismen
- Vorstellung und Vergleich verschiedener Laufzeitarchitektur-Konzepte - mit, aber auch ohne Betriebssystem
- Vorhersagbarkeit und Berechenbarkeit der verschiedenen Laufzeitarchitektur-Konzepte
- Auswahlhilfen für das "richtige" Laufzeitarchitektur-Konzept
- Praxisbeispiel Messgeräte-Applikation
- Übung: Sie entwickeln auf Basis von ausformulierten Anforderungen und einer Software-Architektur eine dazu passende Laufzeitarchitektur für ein real existierendes Embedded-System

Multicore- und Multiprozessor-Aspekte

- Hardware- und Software-Architekturen
- Aufgabenverteilung
- Möglichkeiten des Betriebssystem-Einsatzes
- Virtualisierung und Hypervisor
- Interrupt- und Treiber-Konzepte
- Wichtige Designaspekte

Dokumentation und Kommunikation

- Die geeignete Darstellungsform einer Laufzeitarchitektur
- Auszüge aus der UML (Unified Modeling Language)
- Praxistipps
- Übung: Sie nutzen Notationen und Diagramme der UML zur Darstellung der Laufzeitarchitektur

Betriebssystem-Abstraktion (OSAL Operating System Abstraction Layer)

- Nutzendarstellung, Vor- und Nachteile
- Programmierung
- Praxisbeispiel mit FreeRTOS™

Betriebssystem-Auswahlhilfen und aktuelle Produktübersicht

- Leitfaden zur Betriebssystem-Auswahl
- Praxistipps zum Vergleich von Betriebssystemen
- Aktuelle Produktübersicht für Embedded-Software
- Checkliste mit wichtigen Auswahlkriterien

Praktische Übungen

- In der durchgängigen Programmierübung nutzen Sie Betriebssystem-Mechanismen, um Schritt für Schritt eine Messgeräte-Applikation zu entwickeln.
- Zur Übungsdurchführung verwenden Sie das Arm Keil MDK (Microcontroller Development Kit) zusammen mit einer realen Hardware basierend auf einem Arm Cortex® Mikrocontroller.
- Als Echtzeit-Betriebssystem wählen Sie zwischen FreeRTOS™ oder dem CMSIS-RTOS Standard als Betriebssystem-Abstraktion.
- Sie entwickeln und dokumentieren eine Laufzeitarchitektur für eine komfortable Elektromotor-Steuerung und setzen dazu Betriebssystem-Mechanismen ein.
- Zur Übungsdurchführung nutzen Sie wahlweise den Enterprise Architect von Sparx Systems oder Papier und Bleistift.

MicroConsult Plus:

- Sie erhalten von uns Ihre Übungsverzeichnisse und Lösungsbeispiele für alle Übungsaufgaben.
- Sie erhalten zur Messgeräte-Applikation den Programmcode und ein UML-Modell sowie zur Elektromotor-Steuerung ebenfalls ein UML-Modell.
- Sie erhalten eine Tool- und Software-Komponentenübersicht inklusive einer aktuellen Betriebssystem-Übersicht.
- Sie erhalten zudem eine Checkliste mit Betriebssystem-Auswahlkriterien.
- Sie bekommen hilfreiche Notationsübersichten für UML (Unified Modeling Language) und SysML (Systems Modeling Language).

Präsenz-Training

Termin	Preis *	Dauer
19.10.2026 – 22.10.2026	2.600,00 €	4 Tage

* Preis je Teilnehmer, in Euro zzgl. USt.

Anmeldecode: RTOS-AR

Live-Online - Deutsch

Termin	Dauer
27.07. – 30.07.2026	4 Tage

Präsenz-Training - Englisch

Dauer
4 Tage

Live-Online - Englisch

Termin	Dauer
27.07. – 30.07.2026	4 Tage

Coaching

Unsere Coaching-Angebote bieten den großen Vorteil, dass unsere Experten ihr Wissen und ihre Erfahrungen direkt in Ihren Lösungsprozess einbringen und damit unmittelbar zu Ihrem Projekterfolg beitragen.

Für Ihre Anfrage oder weiterführende Informationen stehen wir Ihnen gern zur Verfügung.