

RT3 - Programmation temps réel avec FreeRTOS

Formation FreeRTOS

Objectifs

- Obtenir une vue d'ensemble de l'architecture Cortex-M4
- Découvrir les concepts du multitâche temps réel
- Comprendre les contraintes temps réel
 - Déterminisme
 - Prémption
 - Interruptions
- Comprendre l'architecture de FreeRTOS
- Découvrir les différents services et APIs de FreeRTOS
- Apprendre à développer des applications FreeRTOS
- Apprendre à déboguer les applications FreeRTOS

Environnement du cours

- Cours théorique
 - Support de cours imprimé et au format PDF (en anglais)
 - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique
- Activités pratiques
 - Les activités pratiques représentent de 40% à 50% de la durée du cours
 - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
 - Un PC pour deux stagiaires pour les activités pratiques
 - Une plateforme cible (base STM32) pour deux stagiaires (sauf en cas d'utilisation de qemu)
 - Accès à un serveur cloud privé
 - Exemples de code, exercices et solutions
 - Le formateur assiste les stagiaires pendant les exercices
- Au début de chaque demi-journée une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Pré-requis

- Familiarité avec les concepts et la programmation du C embarqué
- Connaissance de base des processeurs embarqués

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus.

Course Outline

Premier jour

Cortex-M resources used by RTOS

- Cortex-M Architecture Overview
 - Two stacks pointers
 - Different Running-modes and Privileged Levels
 - MPU Overview
 - SysTick Timer Description
- Exception / Interrupt Mechanism Overview
 - Interrupt entry and return Overview
 - SVC / PendSV / SysTick Interrupt Presentation
- Developing with the IDE

Exercise : Interrupt Management on Cortex-M4

Element of a real time system

- Base real time concepts
- The Real Time constraints
- Multi-task and real time
- Tasks and Task Descriptors
 - Content of the task descriptor
 - List of task descriptors
- Context Switch
- Task Scheduling and Preemption
 - Tick based or tickless scheduling
- Scheduling systems and schedulability proof
 - Fixed priorities scheduling
 - RMA and EDF scheduling
- Scheduling through FReeRTOS
 - Deterministic preemptive scheduling
 - Scheduling strategies
 - Cooperative scheduling
 - Hybrid scheduling

Exercise : Analyse a Context Switch

Task Management

- Creating Tasks
- Task Priorities
- Task States
- The idle task
- Delays
- Changing Task Priority
- Deleting Tasks
- Suspending Tasks
- Kernel Structures
- Thread Local Storage
- Kernel Interrupts on Cortex-M4
- Scheduling Traces
- Visual trace diagnostics using Tracealyzer

Exercise : Task Management

Exercise : Periodic Tasks

Exercise : Task Statistics

Deuxième jour

FreeRTOS Memory Management

- FreeRTOS Memory Managers
- Out of Memory management
- Stack Overflow Management

Exercise : Check stack usage in existing programs

Resource Management

- Mutual exclusion through FreeRTOS
 - Critical sections (interrupt masking)
 - Suspending (locking) the scheduler
 - Mutexes
- Mutexes concepts
 - Mutex or Semaphore
 - Recursive or not recursive mutexes
 - Priority inversion problem
 - Priority inheritance (the automatic answer)
 - Priority ceiling (the design centric answer)
- Gatekeeper tasks

Exercise : Implement mutual exclusion between tasks

Synchronization Primitives

- Introduction
 - Waiting and waking up tasks
 - Semaphores
 - Events
 - Mailboxes
- Binary Semaphores through FreeRTOS
 - Give a Binary Semaphore
 - Take a binary Semaphore
- Queue Management through FreeRTOS
 - Creation
 - Sending on a queue
 - Receiving from a queue
 - Data management
 - Sending compound types
 - Transferring large data
- Event groups
- Task Notifications
- Stream Buffers and Message Buffers

Exercise : Synchronizing a task with another one through binary semaphores

Exercise : Synchronizing a task with another one through queues

Exercise : Task Notifications

Exercise : Properly use stream Buffers

Exercise : Message Buffers

Parallelism Problems Solution

- Parallel programming problems

- Uncontrolled parallel access
- Deadlocks
- Livelocks
- Starvation

Exercise : The producer-consumer problem, illustrating (and avoiding) concurrent access problems

Exercise : The philosophers dinner problem, illustrating (and avoiding) deadlock, livelock and starvation

Troisième jour

Interrupt Management

- Need for interrupts in a real time system
 - Software Interrupt
 - Time Interrupts
 - Device Interrupts
- Level or Edge interrupts
- Hardware and Software acknowledge
- Interrupt vectoring
- Interrupts and scheduling
- Deferred interrupt processing through FreeRTOS
 - Tasks with interrupt synchronization
 - Using semaphores within an ISR
 - Counting semaphores
 - Using queues within an ISR
- FreeRTOS interrupt processing
 - Writing ISRs in C
 - Interrupt safe functions
 - Interrupt nesting

Exercise : Synchronize Interrupts with tasks

Software Timer

- The Timer Daemon Task
- Timer Configuration
- One-shot / Auto-reload Timer
- Software Timer API
- Deferred interrupt handling

Exercise : Implement Soft Timers

FreeRTOS-MPU

- The Cortex/M MPU
 - User and privileged modes
 - Access permissions
- Defining MPU regions
 - Overlapping regions
 - Predefined regions
 - Programmer-defined regions
- Needed linker configuration
- Practical usage tips

Exercise : Implement protected memory regions

Annexes

Data structures

- Need for specific data structures
- Data structures
 - Linked lists
 - Circular lists
 - FIFOs
 - Stacks
- Data structures integrity proofs
 - Assertions
 - Pre and post-conditions
- Thread safety

Exercise : Build a general purpose linked list

Memory Management

- Memory management algorithms
 - Buddy System
 - Best fit / First Fit
 - Pools Management
- FreeRTOS-provided memory allocation schemes
 - Allocate-only scheme
 - Best-fit without coalescing
 - Thread-safe default malloc
- Checking remaining free memory
- Adding an application-specific memory allocator
- Memory management errors
- Stack monitoring

Exercise : Write a simple, thread safe, buddy system memory manager

Exercise : Write a generic, multi-level, memory manager

Exercise : Enhance the memory manager for memory error detection