Real Time College

Course: Real-Time Embedded O.S. Systems Implemented by FreeRTOS

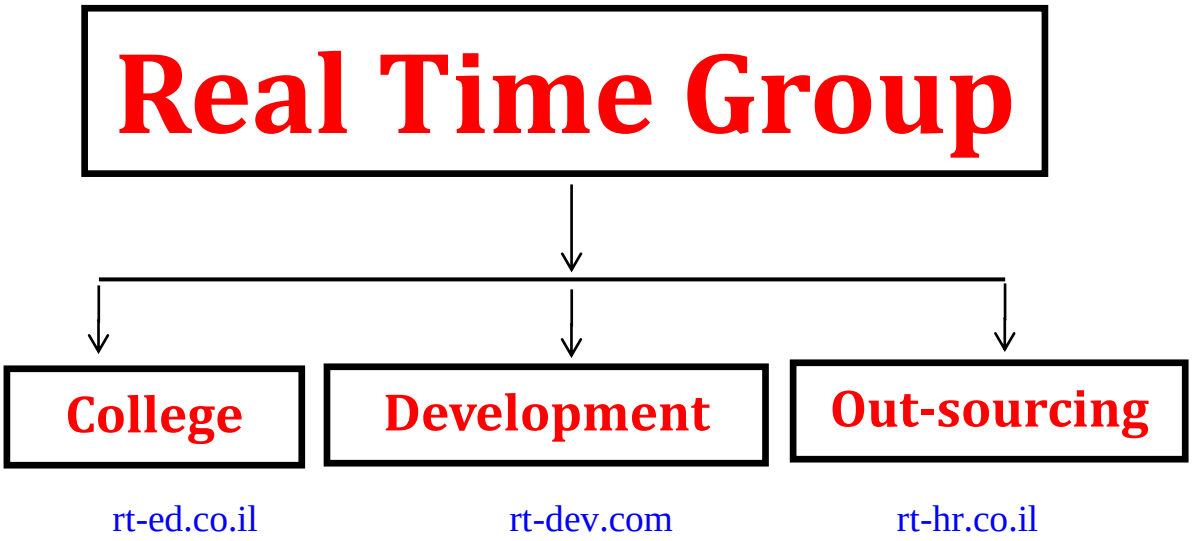
Duration: 40 Hours

Hands-On-Training: 60%

Real Time Group is a multi-disciplinary dynamic and innovative Real-Time O.S. and Embedded Software Solutions Center, established in 2007.

Providing Bare-Metal and Embedded Linux solutions, professional services and consulting, end-to-end flexible system infrastructure, outsourcing, integration and training services for Hardware, Software and RT-OS \ Embedded Systems.

The company is divided into the following three Divisions:



Training Division:

Professional Training Services for Hardware, Software, RT-OS and Embedded systems industries.

We provide the knowledge and experience needed to enable professional engineers to Develop, Integrate and QA Hardware, Software and Networking Projects.

In order to insure experience, all courses are practical – hands-on-training.
The latest Development, QA and Automation equipment which are adopted by the industry are used.

All students are supplied with Development-Boards for home-work and course projects.

Course Overview:

This is a Hands-on course for RT **Embedded O.S. Systems** in **FreeRTOS** and **Embedded Linux** contex.

The course aims to teach subjects shared by many OS's, including VxWorks, Micrium, Embedded Linux and others, it focuses on how to use Kernel objects and services for programming embedded multitasking application whether they are RT \ non-RT Systems efficiently.

We begin with introduction to RTOS, then we'll explores the building blocks of RTOSs, key kernel services such as Tasks, scheduling policies (preemptive ver Non–preemptive multitasking), synchronization and Locking mechanisms, Exceptions & Interrupts implementations, later we'll dive into Linux based Kernel services such as IPC, File-system, Network stack and much more. Each Theoretical lesson will we followed by practical FreeRTOS example.

This is a **Practical Hands-on-Course**, all exercises will be implemented on **Embedded FreeRTOS \ Linux based boards**.
Throughout the course, student will use these boards for home exercises as well.

The course covers all major areas for RT Embedded programming specifically in FreeRTOS, reveling the knowledge and "know-how" needed in order to understand, design and program such devices.

Who should attend:

- Suitable for embedded systems software development engineers, software system architects, project managers and technical consultants.
- Hardware and software engineers who need to quickly speed up with developing multi-tasking applications.

Prerequisite:

- Knowledge in programming C language.
- No previous experience of OS programming is necessary.

Real-Time O.S. Systems Implemented by FreeRTOS

1. **Introduction to Real-Time Operating Systems**
   a. RT Systems
   b. Soft real time systems
   c. Hard real time systems
   d. Differences
   e. Applications

2. **Components of RTOS**
   f. History of Embedded Operating Systems
   g. Building blocks of Os's

3. **The Scheduler**
   a. How does a Scheduler work (go through basic implementation)
   b. Round Robin ver Preemptive Scheduling
   c. Free-ROS Scheduler
   d. Linux Scheduler

   Lab Exercise: Free-RTOS Scheduler

4. **Tasks**
   a. What happens when a task is first created.
   b. Task states (Idle state \ Running state \ Blocking state\ idle task).
   c. Loading and Execution of Programs
   d. Preemptive Multitasking
   e. Task priorities
   f. Task context Switching
   g. Task Control Blocks
   h. De-Scheduling

Lab Exercises: Task Management

1. **Creating tasks\Deleting tasks.**
2. **The Endless-loop pattern**
3. **Task Priorities**
4. **The idle task**

5. Task Starvation

a. What is Task Starvation

b. What can it cause?

c. Can we avoid it ?

Lab Exercises:

5. **Implementing Task Starvation**
6. **Avoiding Task Starvation**

6. Critical sections and Thread Synchronization

a. Race Conditions

b. Semaphore Types

   1. Binary Semaphores
   2. Mutexes
   3. Counting Semaphores

c. Priority Inversion

d. Dead Locks

Lab Exercises: Task Synchronization through FreeRTOS

7. **Synchronizing a task with another one through binary semaphores**
8. **Mutual exclusion through FreeRTOS**
9. **Critical sections (interrupt masking)**
10. **Suspending (locking) the scheduler**

7. Message Queues

a. Defining \ Creating Message Queues
b. Receiving \ Sending Task Waiting lists
c. Message Queue States
d. Receiving \ Sending Messages to Message Queue
e. Message Attributes
f. Full \ Empty Message Queues

Lab Exercises: Queue Management through FreeRTOS

11. Creating and Sending \ Receiving on\from a queue

12. Synchronizing a task with another one through queues

8. Exceptions and Interrupts

a. What are Interrupts \ Exceptions
b. Interrupt vector and triggers for exception handling
c. Software based interrupts
d. Hardware based interrupts
e. Interrupt Controllers
f. IRQ's - Level or Edge interrupts or MSI\MSIX
g. Nested Interrupts
h. Interoperating tasks with interrupts
i. FreeRTOS interrupt processing
    1. Writing ISRs in C
    2. Interrupt safe functions
    3. Interrupt nesting

Lab Exercises: Interrupt Management through FreeRTOS

13. Synchronizing Interrupts with tasks

14. Deferred interrupt processing through FreeRTOS

1. Using semaphores within an ISR
2. Counting semaphores
3. Using queues within an ISR

Linux Based O.S. Systems

9. IPC (Inter Processes Communication) in RT & non RT OS's

a. Message Queues
b. Signals
c. Shared memory
d. Pipes and FIFOs
e. Memory Mapping
f. Synchronization primitives- Semaphore, Spinlock, Barrier, Mutual exclusion

10. Other RTOS Services (Based on Client's request & time constraints)

g. Networking Stack
h. File System Stack

11. Timer and Timer Services (Based on Client's request & time constraints)

a. The HW\SW Timers
b. Timer Configuration
c. One-shot / Auto-reload Timer
d. Software Timer API
e. Interrupt derived handling

12. Virtual memory concepts (Based on Client's request & time constraints)

f. paging and segmentation
g. MMU and TLB
h. Address mapping.

13. Virtual storage management (Based on Client's request & time constraints)

a. page replacement strategies.
b. blocking and buffering,
c. file descriptor,
d. directory structure

14. **File and Directory structures** **(Based on Client's request & time constraints)**
    a. blocks and fragments,
    b. directory tree,
    c. inodes,
    d. file descriptors,
    e. UNIX file structure.

Appendix: (Based on Client's request & time constraints)

15. **I/O Subsystem**
16. **Parallel programming principles**
17. **Common Design Problems**
18. **The Linux Kernel and Device Drivers**