MC4 - Multi-Core Programming with OSEK/VDX and AutoSAR

Programming real-time and multi-core systems, avoiding common pitfalls

Goals

- Understand the specifics of programming multi-core processors
- Master concurrent programming
  - on the same processor
  - on a multiprocessor system
- Interactions with processor architecture features
  - Cache
  - Pipeline
  - I/O optimizations
  - Multicore and Hyperthreading
- Understand the structure of a real time kernel
  - OSEK/VDX
  - Multicore Autosar

This course helps you master multitask and real-time programming, understanding how to effectively solve problems using the primitives provided by the underlying Operating System.

Course material

- Linux PC for each group of 2 trainees
- Embedded target board
- Cross compiler toolchain and debugger
- Course slides hardcopy
- Labs manual hardcopy

Prerequisite

- Good knowledge of embedded C programming
- Basic understanding of processor architecture

Pedagogic strategy

- **The exercises focus on using the mechanisms available to solve traditional problems: Readers-writers, producer-consumer, the dining philosophers, ...**
- **Each exercise includes a detailed explanation and a diagram which helps to understand how the algorithm works.**
- **For each exercise there an almost complete code is provided, with parts to complete; this allows, after a phase of understanding of the provided code, to implement features that usually take hours to design.**
- **The course includes optional exercises to deepen understanding.**

Target Audience

- Any embedded systems engineer or technician with the above prerequisites.

Course Outline

First day

Tasks and scheduling in embedded systems

- Tasks and task descriptors
- Context switch

Exercise: Write a context switch routine

- Task scheduling and preemption
  - Tick based or tickless scheduling
- Scheduling systems and schedulability proofs
  - Fixed priority scheduling
  - RMA and EDF scheduling
  - Adaptive scheduling

Exercise: Write a simple, fixed priority, scheduler

Interrupt management in real time systems

- Need for interrupts in a real time system
  - Time interrupts
  - Device interrupts
- Level or Edge interrupts
- Hardware and software acknowledge
- Interrupt vectoring

Exercise: Write a basic interrupt manager

- Interrupts and scheduling

Exercise: Extend the scheduler to also support real-time round-robin scheduling

Multicore interactions

- Cache coherency
  - Snooping basics
  - Snoop Control Unit: cache-to-cache transfers
  - MOESI state machine
- Memory Ordering and Coherency
  - ut-of-order accesses
  - Memory ordering
  - Memory barriers
  - DMA data coherency
- Multicore data access
  - Read-Modify-Write instructions
  - Linked-Read/Conditional-Write
- Multicore synchronization
  - Spinlocks
  - Inter-Processor Interrupts

Exercise: Writing a spinlock implementation

Second day

Multicore scheduling

- Multicore scheduling
  - Assigning interrupts to processors

    - Multi-core scheduling
- Multicore optimization
    - Cache usage optimization
    - Avoiding false sharing
    - Avoiding cache spilling

Exercise: Study of a multi-core scheduler

Synchronisation primitives

- Waiting and waking up tasks
- Semaphores

Exercise: Implement Semaphores by direct interaction with the scheduler

- Mutual exclusion
    - Spinlocks and interrupt masking
    - Mutexes or semaphores

Exercise: Implement the mutex mechanism

    - Recursive and non-recursive mutexes

Exercise: Check proper nesting of mutexes and recursive/non-recursive use

    - The priority inversion problem
    - Priority inheritance (the automagic answer)
    - Priority ceiling (the design centric answer)

Exercise: Implement a priority ceiling mechanism

- Mutexes and condition variables

Exercise: Add Condition variable support to the mutex mechanism

- Mailboxes

Third day

Avoiding sequencing problems

- The various sequencing problems
    - Uncontrolled parallel access

Exercise: The producre-consumer problem, illistrating (and avoiding) concurrent access problems

    - Deadlocks
    - Livelocks
    - Starvation

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

Osek/VDX tasking architecture

- Task management
    - Basic tasks
    - Extended tasks
    - Scheduling policies
    - Task activation and termination
- Interrupt processing
- Events
- Resources

Autosar and Multicore programming

- Autosar multicore architecture
- Autosar Locatable Entities
- Enhancements to the OSEK scheduling
- Autosar spinlocks
- The Inter OS-Application Communicator
- Migrating Autosar application to multicore