



RT1 - Real Time and Multi-Core programming

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

Goals

- Discover the concepts of real time multitasking
- Understand the specificities of multicore processors
- Master concurrent programming
 - on the same processor
 - on a multiprocessor system
- Understand real time constraints
 - Determinism
 - Preemption
 - Interruptions
- Interactions with processor architecture features
 - Cache
 - Pipeline
 - I/O optimisations
 - Multicore and Hyperthreading
- Debug real time applications
- Understand the structure of a real time kernel

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

Introduction to real time

- Base real time concepts
- The real time constraints
- Multi-task and real-time
- Multi-core and Hyperthreading

Exercise: Install the development environment on the host system (if needed)

Exercise: Install the execution environment on the target system

Exercise: Create a simple context switch routine

Thread safe data structures

- Need for specific data structures
- Thread safe data structures
 - Linked lists (simple or double links)
 - Circular lists
 - FIFOs
 - Stacks
- Data structure integrity proofs
 - Assertions
 - Pre and post-conditions

Exercise: Build a general purpose thread safe doubly linked list

Memory management

- Memory management algorithms
 - Buddy system

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

- Best fit
- First fit
- Pool management

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

- Memory management errors
 - memory leaks
 - using unallocated/deallocated memory

Exercise: Enhance the memory manager for memory error detection

- stack monitoring

Exercise: Enhance the context switching infrastructure to monitor stack use

Second day

Elements of a real time system

- Tasks and task descriptors
 - Content of the task descriptor
 - Lists of task descriptors
- Context switch
- Task scheduling and preemption
 - Tick based or tickless scheduling
- Scheduling systems and schedulability proofs

- Fixed priority scheduling
- RMA and EDF scheduling
- Adaptative 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

Third 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

Fourth day

Avoiding sequencing problems

- The various sequencing problems
 - Uncontrolled parallel access

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

- Deadlocks
- Livelocks
- Starvation

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

Working with Pthreads

- The pthread standard
 - threads
 - mutexes and condition variables

Exercise: Solve the classic readers-writers problem with POSIX threads

- Thread local storage

Exercise: Maintain per-thread static data for the readers-writers problem

- POSIX semaphores
- Scheduling
 - context switches
 - scheduling policies (real-time, traditional)
 - preemption

Fifth day

Multi-tasking in the Linux kernel

- Kernel memory management
 - "buddy" and "slab" memory allocation algorithms
- Kernel task handling
- Linux kernel threads
 - creation
 - termination
- Concurrent kernel programming
 - atomic operations
 - spinlocks
 - read/write locks
 - semaphores and read/write semaphores
 - mutexes
 - sequential locks
 - read-copy-update
 - hardware spinlock

Exercise: Create a kernel-mode execution barrier using kernel synchronisation primitives

- Basic thread synchronisation

- waiting queues
- completion events
- Hardware clocks
 - clockevents
- Software clocks
 - delayed execution
 - kernel timers
 - high resolution timers

Exercise: Create a kernel event synchronisation object, using basic synchronisation primitives

Asymmetric multiprocessing

- AMP overview
 - Architecture
 - Shared memory
 - Challenges comparing to SMP
- Inter-processor communication
- OpenAMP framework
 - Remoteproc
 - rpmsg

Exercise: Sending messages between AMP cores