oRT1 - Linux Real-Time and Multi-Core programming

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

Objectives

- 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 optimizations
  - Multicore and Hyperthreading
- Debug real time applications
- Understand the structure of a real time kernel

Labs are conducted QEMU ARM-based board

Prerequisite

- Good C programming skills (see our L2 course)
- Basic understanding of processor architecture

Course Environment

- Theoretical course
  - PDF course material (in English).
  - Course dispensed using the Teams video-conferencing system.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance through the Teams video-conferencing system.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One Online Linux PC per trainee for the practical activities.
  - The trainer has access to trainees' Online PCs for technical and pedagogical assistance.
  - Eclipse environment and GCC compiler.
  - QEMU Emulated board or physical board connected to the online PC (depending on the course).
  - Some Labs may be completed between sessions and are checked by the trainer on the next session.
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

Duration

- Total: 30 hours
- 5 sessions, 5 hours each (excluding break time)
- From 40% to 50% of training time is devoted to practical activities
- Some Labs may be completed between sessions and are checked by the trainer on the next session

Target Audience

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

Evaluation modalities

- The prerequisites indicated above are assessed before the training by the technical supervision of the traineein his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
  - For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
  - Quizzes are offered at the end of sections that do not include practical exercises to verifythat the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
  - In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites,in agreement with their company manager if applicable.

Plan

First Session

Introduction to real time

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

Exercise: *Prepare the environment*
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
  - Best fit
  - First fit
  - Pool management

- Memory management errors
  - memory leaks
  - using unallocated/deallocated memory
  - 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*
Exercise: *Enhance the context switching infrastructure to monitor stack use*

Second Session

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
  - 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
- Interrupts and scheduling

Exercise: *Write a basic interrupt manager*
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
  - Out-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 Session

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*

Synchronization primitives

- Waiting and waking up tasks
- Semaphores
- Mutual exclusion
  - Spinlocks and interrupt masking
  - Mutexes or semaphores
  - Recursive and non-recursive mutexes
  - The priority inversion problem
  - Priority inheritance (the automagic answer)
  - Priority ceiling (the design centric answer)
- Mutexes and condition variables
- Mailboxes

Exercise: *Implement Semaphores by direct interaction with the scheduler*
Exercise: *Implement the mutex mechanism*
Exercise: *Check proper nesting of mutexes and recursive/non-recursive use*
Exercise: *Implement a priority ceiling mechanism*
Exercise: *Add Condition variable support to the mutex mechanism*

Fourth Session

Avoiding sequencing problems

- The various sequencing 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*

Working with Pthreads

- The pthread standard
  - threads
  - mutexes and condition variables
  - Thread local storage
- POSIX semaphores
- Scheduling
  - context switches
  - scheduling policies (real-time, traditional)
  - preemption

Exercise: *Solve the classic readers-writers problem with POSIX threads*
Exercise: *Maintain per-thread static data for the readers-writers problem*

Fifth Session

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

Exercise: *Create a kernel-mode execution barrier using kernel synchronization primitives*
Exercise: *Create a kernel event synchronization object, using basic synchronization 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 demonstration*

Renseignements pratiques

Inquiry : 30 hours