



oRT1 - Programmation Linux temps-réel et multi-cœurs

Programmation du système Linux temps-réel et multi-cœurs, en évitant les pièges courants

Objectifs

- Découvrir les concepts du multitâche temps-réel
- Comprendre les spécificités des processeurs multicœurs
- Maîtriser la programmation concurrente
 - sur un même processeur
 - sur un système multiprocesseur
- Comprendre les contraintes du temps réel
 - Déterminisme
 - Prémption
 - Interruptions
- Interactions avec l'architecture du processeur
 - Cache
 - Pipeline
 - Optimisations E/S
 - Multicore et "Hyperthreading"
- Déboguer des applications temps-réel
- Comprendre la structure d'un noyau temps-réel

Les travaux pratiques sont effectués sur une carte ARM QEMU

Pré-requis

- Bonnes connaissances en programmation C (voir notre cours L2)
- Compréhension de base de l'architecture des processeurs

Environnement du cours

- Cours théorique
 - Support de cours au format PDF (en anglais)
 - Cours dispensé via le système de visioconférence Teams
 - 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 Linux en ligne par stagiaire pour les activités pratiques
 - Exemples de code, exercices et solutions
 - Le formateur a accès aux PC en ligne des stagiaires pour l'assistance technique et pédagogique
 - Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante
- Une machine virtuelle préconfigurée téléchargeable pour les activités pratiques après le cours
- 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

Audience visée

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

Durée

- Totale : 30 heures
- 5 sessions de 6 heures chacune (hors temps de pause)
- De 40% à 50% du temps de formation est consacré aux activités pratiques
- Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante

Course Outline

Première 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

Deuxième 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

Troisième 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

Quatrième 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

Cinquième 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