



## L71 - Programmation temps réel

### La programmation embarquée et temps réel en C, C++ et Java(TM).

Java est une marque déposée de Sun Microsystems

#### Objectifs

- Découvrir les concepts de base du multi-tâches et du temps réel
- Apprendre à lire des diagrammes de conception temps-réel UML-RT
- Comprendre le fonctionnement d'une chaîne de compilation
- Analyser les options d'optimisation de code du compilateur
- Développer des applications pour accéder aux ports d'E/S et gérer les interruptions
- Interfacer C et assembleur pour mettre en œuvre des instructions spécifiques du processeur cible
- Maîtriser les difficultés de la programmation concurrente
- Connaître les standards applicables
- Découvrir les contraintes temps réel (déterminisme, interruptions, préemption...)
- Comprendre les implications des architectures des processeurs en contexte temps-réel (cache, pipeline,...)
- Mettre au point des applications multi-tâches
- Utiliser le langage C pour implémenter un noyau temps-réel minimal

*A la demande des développeurs auxquels elle est particulièrement destinée cette formation met principalement l'accent sur de nombreux exercices pratiques allant jusqu'à la réalisation effective d'un système temps-réel embarqué.*

*Les personnes désireuses d'un cours couvrant de façon plus large les problèmes d'utilisation d'UML/RT dans le contexte projet peuvent regarder également notre cours référence [L70 - Les projets temps réel](#)*

#### Matériel

- Un PC et une carte ColdFire par binôme
- Chaîne de compilation croisée et sonde d'émulation BDM
- Machine virtuelle Java
- Manipulations et exercices en environnements natif et croisé
- Un support de cours ainsi que la disquette contenant les exemples

#### Pré-requis

- Connaissance de la programmation en C, C++ ou Java (niveau cours L2, L3 ou L4)
- Connaissance d'un microprocesseur souhaitée
- Connaissance de la programmation embarquée utile

## Plan

### Introduction au temps réel

- concepts temps réel de base
- contraintes particulières du temps réel
- programmation structurée et objet
- apports des techniques objets

### L'approche temps réel avec UML

- genèse d'UML
- modèles UML standards
- cycle de développement Objet
- contraintes liées à l'interprétation des diagrammes
- comment lire les diagrammes UML

### Analyse des éléments constitutifs d'une chaîne de compilation

- Explication des étapes du processus de génération de code en natif et en croisé
- Rôle du compilateur, de l'assembleur et du linker
- Paramétrage en fonction d'un mapping mémoire
- Structure d'un programme source C, distinction des parties essentielles : préprocesseur, déclaration des variables globales, fonctions
- Découpage d'une application en fichiers distincts
- Le préprocesseur
- Les instructions define et include
- Ecriture de macros
- Précautions à prendre dans les headers pour éviter les redéclarations de variables
- Notion de projet, réalisation de librairies

*Exercice : mise en œuvre de la chaîne de compilation et fabrication d'une librairie*

### Particularités de la programmation dans le contexte embarqué

- Analyse d'une instruction de transfert assembleur pour comprendre l'accès à la mémoire
- Mise en évidence de la distinction adresse / contenu
- Calculs d'adresses, opérations mêlant pointeurs et adresses absolues
- Les tableaux de pointeurs

*Exercice : allocation d'un pointeur sur un port d'I/O*

- Accès aux champs d'une structure
- Déclaration de variables et de pointeurs sur type structuré
- Explication du padding imposé par les règles d'alignement : options pack du compilateur
- Les formats big et little endian
- Les structures à champ de bits : modélisation des périphériques
- Les unions : une même zone mémoire peut être envisagée de différentes manières

*Exercice : modélisation de l'UART du ColdFire au moyen d'une union et d'une structure à champs de bits et communication avec un terminal*

- Utilité des tableaux de pointeurs sur des fonctions

*Exercice : lancement d'une fonction à partir d'un tableau de pointeurs*

- Distinction entre adressage absolu et adressage relatif pour la relogeabilité du code et des données
- Réentrance des handlers d'exception

- Cstart : initialisation du pointeur de pile et mise à 0 des variables non initialisées
- Inconvénients des fonctions Setjmp et Longjmp
- Gestion de la mémoire
  - Algorithmes
  - Gestion des fuites mémoire
- Mise en EPROM d'une application

*Exercice : mise en évidence et détection de fuites mémoire*

### Principe de fonctionnement d'un système Temps réel et embarqué

- Notion de tâche
- Cadencement des tâches selon leur priorité, préemption
- Sauvegarde de contexte
- Nécessité d'un tick temps réel pour déclencher les commutations de tâches

### Les traitements concurrents

- Recensement des champs d'un descripteur de tâche
- Réalisation d'une structure chaînée des tâches en attente d'exécution
- Insertion d'un nouveau descripteur lors du chargement d'une nouvelle tâche
- Exemple de règles d'ordonnement: priorité évoluant en fonction du temps
- Réorganisation de la file lors de l'invocation de l'ordonnanceur: préemption

### Les interruptions

- Nécessité des interruptions dans un système embarqué
- Distinction entre déclenchement sur front et sur niveau
- Acquiescement logiciel
- Ecriture d'un gestionnaire d'interruption : distinction des 3 étapes prologue / corps / épilogue
- Table de vecteurs
- Ecriture des fonctions d'installation et de lecture de vecteur

*Exercice : lancement d'une action suite à une interruption*

### Mise au point

- Communication avec la cible
- Les différents niveaux de mise au point : C, assembleur
- Les fenêtres du debugger : source, mémoire, pile et variables
- Positionnement de points d'arrêt
- Analyse de la pile et extraction des stacks frames correspondant aux fonctions imbriquées
- Les procédés de mise au point sur les processeurs récents munis de cache : synchronisation avec un analyseur logique

### La programmation dans le contexte multi-tâches

- Structures de données:
  - Listes simplement chaînées
  - Listes doublement chaînées
  - Listes circulaires
  - Files d'attentes
  - Piles

*Exercice : réalisation de listes chaînées utilisables en contexte multi-tâches*

- Noyau temps-réel minimal
  - descripteurs de tâches
  - scheduling

*Exercice : réalisation d'un scheduler simple, "fair scheduling"*

- Synchronisation entre tâches
  - Modes de synchronisation
  - Primitives de synchronisation

*Exercice : gestion d'un sémaphore, "fixed scheduling"*

- Gestion des accès concurrents
  - Variable simple
  - Structure de données
  - Entre tâches
  - Entre tâches et routines d'interruption

*Exercice : synchronisation et communication entre tâches*

- Gestion de la mémoire
  - Algorithmes
  - Gestion des fuites mémoire

*Exercice : mise en évidence et détection de fuites mémoire*

## Renseignements pratiques

**Durée : 4 jours**  
**Prix : 1850 € HT**



SARL au capital de 15400€ - SIRET 449 597 103 00026 - RCS Nanterre - NAF 722C - Centre de Formation : 19, rue Pierre Curie - 92400 Courbevoie  
Siège social et administration : 21, rue Pierre Curie - 92400 Courbevoie - Tél. 01 41 16 80 10 - Fax. 01 41 16 07 78

Dernière mise à jour du site: mar 22 mai 2012 10:50:29 CEST

<http://www.ac6-formation.com/>