RT1 - Programmation Temps-Réel et Multi-Cœurs

Comment éviter les pièges de la programmation temps réel et multi-processeur, en particulier sous Linux

Objectifs

- Découvrez les notions de multitâche temps réel
- Comprendre les problèmes liés aux CPUs multi-cœurs
- Maîtriser la programmation concurrente
  - Sur le même processeur
  - Sur un système multiprocesseur
- Comprendre les contraintes temps réel
  - Déterminisme
  - Préemption
  - Interruptions
- Interactions avec l'architecture du processeur
  - Cache
  - Pipeline
  - Optimisations
  - Multi-cœur et Hyperthreading
- Débogage d'applications temps réel
- Comprendre la structure d'un noyau temps réel

Ce cours apprend à maîtriser la programmation temps réel et multi-tâches en comprenant comment résoudre efficacement les problèmes rencontrés en fonction des primitives disponibles sur l'OS utilisé.

Matériel

- Un PC Linux par binôme
- Une carte cible sous Linux
- Compilateur et débogueur croisés
- Support de cours imprimé
- Présentation et solutions des exercices

Pré-requis

- Bonne connaissance de la programmation C embarquée
- Connaissance de base de l'architecture des processeurs

Démarche pédagogique

- **Les exercices s'attachent à mettre en œuvre les mécanismes disponibles pour résoudre des problèmes classiques: Readers-writers, producteurs-consommateurs, le repas des philosophes...**
- **Chaque exercice comprend une explication détaillée et un schéma, ce qui aide à comprendre le fonctionnement des algorithmes.**
- **Pour chaque exercice il est fourni un code quasiment complet avec des parties à compléter, ce qui permet, après une phase de compréhension du code fourni, de maîtriser des fonctionnalités qui habituellement prennent des heures à concevoir.**
- **Le cours comprend des exercices facultatifs destinés à approfondir la compréhension des sujets traités.**

Environnement du cours

- Cours théorique
  - Support de cours imprimé et au format PDF (en anglais).

    - 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.
    - Exemples de code, exercices et solutions
    - Un PC (Linux ou Windows) par binôme de stagiaires (si plus de 6 stagiaires) pour les activités pratiques avec, si approprié, une carte cible embarquée.
    - Le formateur accède aux PC des stagiaires pour l'assistance technique et pédagogique.
- Une machine virtuelle préconfigurée téléchargeable pour refaire 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.

Plan du cours

Premier jour

Introduction au temps réel

- Les concepts de base du temps réel
- Les contraintes du temps réel et de l'embarqué
- Multi-tâches et temps-réel
- Multi-coeur et Hyperthreading

Exercise : Installation de l'environnement de développement sur le système hôte (si nécessaire)
Exercise : Installation de l'environnement d'exécution sur la cible
Exercise : Création d'une routine de changement de contexte

Structures de données et parallélisme

- Nécessité des structures de données
- Structures de données et multi-tâches
    - Les listes (simple ou double liens)
    - Listes circulaires
    - FIFOs
    - Piles
- Preuves d'intégrité des structures de données
    - Assertions
    - Pré et post-conditions

Exercise : Construction d'un gestionnaire de listes chaînées génériques utilisable en environnement parallèle

Gestion mémoire

- Les algorithmes de gestion mémoire
    - Buddy system

Exercise : Écriture d'un gestionnaire de mémoire système simple utilisant l'algorithme "buddy"

    - Best fit
    - First fit
    - Gestion de pools

Exercise : Écriture d'un gestionnaire de mémoire générique, multi-niveau

- Les erreurs mémoire
    - Fuites mémoire

    - Accès à de la mémoire non allouée ou déjà libérée

Exercise : Amélioration du gestionnaire de mémoire pour la détection d'erreurs mémoire

    - Surveillance des piles

Exercise : Amélioration de la commutation de contexte pour surveiller l'utilisation de la pile

Second jour

Les composants d'un système temps réel

- Les tâches et les descripteurs de tâches
    - Contenu du descripteur de tâche
    - Listes de descripteurs de tâches
- Changement de contexte
- Mécanismes d'ordonnancement et de préemption
    - Ordonnancement avec ou sans tic d'horloge
- Modes d'ordonnancement et preuves d'ordonnançabilité
    - Ordonnancement à priorités fixes
    - Ordonnancement RMA et EDF
    - Ordonnancements adaptatifs

Exercise : Écriture d'un ordonnanceur simple, à priorités fixes

Gestion d'interruptions dans les systèmes temps-réel

- Besoin d'interruptions dans un système temps réel
    - Interruptions de timer
    - Interruptions de périphériques
- Notion d'interruption sur niveau ou sur front
- Acquittement matériel et logiciel
- Vectorisation des interruptions

Exercise : Écriture d'un gestionnaire d'interruption simple

- Interruption et ordonnancement

Exercise : Extension de l'ordonnanceur pour supporter un ordonnancement en ronde (round-robin)

Interactions entre coeurs en multicoeurs

- Cohérence de cache
    - Snooping
    - Snoop Control Unit: transferts cache à cache
    - Machine d'état MOESI
- Ordonnancement et cohérence mémoire
    - Ordre des accès
    - Ordonnancement mémoire
    - Barrières mémoire
    - Cohérence de données et DMA
- Accès aux données et multicoeurs
    - Instructions Read-Modify-Write
    - Linked-Read/Conditional-Write
- Synchronisation en multicœurs
    - Spinlocks
    - Interruptions inter processeurs

Exercise : Écriture d'un spinlock

Troisième jour

Ordonnancement multicoeurs

- Ordonnancement multicœurs
  - Affectation d'interruptions à un cœur
  - Ordonnanceur multicœurs
- Optimisations multicœurs
  - Optimisation de l'utilisation des caches
  - Éviter les "faux" partages
  - Éviter la saturation des caches

Exercise : Étude d'un ordonnanceur multicœurs

Les primitives de synchronisation

- Mise en attente et réveil des tâches
- Sémaphores

Exercise : Mise en œuvre des sémaphores par interaction directe avec l'ordonnanceur

- Exclusion mutuelle
  - Spinlocks et masquage d'interruptions
  - Mutex ou sémaphores

Exercise : Mise en œuvre du mécanisme de mutex

  - Mutex récursifs et non récursifs

Exercise : Vérifier la bonne imbrication des mutex récursifs et l'utilisation de mutex non récursifs

  - Le problème de l'inversion de priorité
  - L'héritage de priorité (le mécanisme automagique)
  - Le plafond de priorité (la réponse centrée sur la conception)

Exercise : Mise en place du mécanisme de plafond de priorité

- Mutexes et variables condition

Exercise : Ajout du support des variables condition aux mutex

- Les boites aux lettres

Quatrième jour

Solutions aux problèmes de parallélisme

- Les divers problèmes de la programmation parallèle
  - Accès concurrent non maîtrisé

Exercise : Le problème "producteur-consommateur", ou une illustration d'accès concurrents et sa solution

  - Deadlocks (étreinte fatale)
  - Livelocks (blocage vivant)
  - Starvation (famine)

Exercise : Le problème du "dîner des philosophes", illustration des risques de deadlocks, livelocks et de famine

Les Pthreads sous Linux

- Le standard pthread
  - threads
  - mutexes et variables condition

Exercise : Résolution du problèmes des lecteurs et des écrivains avec des threads POSIX

  - Variables spécifiques à un thread

Exercise : Maintient de statistiques par thread pour le problème des lecteurs et écrivains

- Sémaphores POSIX
- Ordonnancement sous Linux
  - changements de contexte
  - politiques d'ordonnancement (temps réel, classique)

- préemption

Cinquième jour

Multi-tâches dans le noyau Linux

- La gestion de mémoire du noyau
  - Les algorithmes d'allocation de mémoire "buddy" et "slab"
- Gestion des tâches sous Linux
- Threads noyau Linux
  - Création
  - Terminaison
- Programmation concurrente dans le noyau
  - Opérations atomiques
  - Spinlocks
  - Spinlocks en lecture/écriture
  - Sémaphores et sémaphores en lecture/écriture
  - Mutexes
  - Verrous séquentiels
  - Le mécanisme "Lecture/Copie/Mise-à-jour" (RCU)
  - Hardware spinlock

Exercise : Création d'un mécanisme de barrière d'exécution à partir des primitives de synchronisation du noyau Linux

- Les mécanismes de base de synchronisation de threads
  - Les files d'attente
  - Les événements de complétion
- Les timers matériels
  - Clockevents
- Les timers logiciels
  - Délais d'exécution
  - Timers noyau
  - Timers haute résolution

Exercise : Création d'un évènement de synchronisation, à partir des mécanismes de synchronisation de base

Multiprocessing asymétrique

- Aperçu sur AMP
  - Architecture
  - Mémoire partagée
  - Comparaison avec SMP
- Communication entre les processeurs
- Framework OpenAMP
  - Remoteproc
  - Rpmsg

Exercise : Envoi de messages entre les cŒurs AMP