Écriture de pilotes Linux

Objectifs

- Maîtriser les outils de développement et de débogage du Kernel
- Découverte de la programmation multi-cŒur dans le Kernel Linux
- Programmation des E/S, des interruptions, des timers et du DMA
- Installer et intégrer des pilotes dans le Kernel Linux
- Gérer les E/S synchrones et asynchrones et les ioctl
- Ecrire un pilote en mode caractère
- Comprendre les spécificités des versions 2.6 et 3.x
- Maîtriser les techniques de débogage du Kernel avec les sondes JTAG de Lauterbach

Les travaux pratiques sont menés sur des tableaux cibles, qui peuvent être :
Cartes "STM32MP15-DISCO" à base de deux Cortex/A7 de STMicroelectronics.
Cartes "SabreLite" de NXP basées sur le Quad Cortex/A9.
Cartes "imx8q-evk" de NXP basées sur le Quad Cortex/A53.
Nous utilisons un kernel linux récent (4.x), tel que supporté par le fournisseur de la puce.

Public visé

- Ce cours s'adresse aux ingénieurs qui installent Linux sur une plate-forme personnalisée et doivent créer des pilotes de périphériques spécifiques

Pré-requis

- Bonnes connaissances en programmation C
- De préférence, connaissance de la programmation utilisateur Linux (voir nos cours D0 - Programmation en mode utilisateur Linux ou cours oD0 - Programmation en mode utilisateur Linux)

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.
  - Exemples de code, exercices et solutions.
  - Un PC Linux en ligne par stagiaire pour les activités pratiques.
  - 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 refaire les activités pratiques après le cours
- Au début de chaque session 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 : 24 heures
- 4 sessions de 6 heures chacune (hors temps de pause)
- De 40% à 50% du temps de formation est consacré aux activités pratiques
- Certains laboratoires peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante

Modalités d'évaluation

- Les prérequis indiqués ci-dessus sont évalués avant la formation par l'encadrement technique du stagiaire dans son entreprise, ou par le stagiaire lui-même dans le cas exceptionnel d'un stagiaire individuel.
- Les progrès des stagiaires sont évalués de deux façons différentes, suivant le cours:
  - Pour les cours se prêtant à des exercices pratiques, les résultats des exercices sont vérifiés par le formateur, qui aide si nécessaire les stagiaires à les réaliser en apportant des précisions supplémentaires.
  - Des quizz sont proposés en fin des sections ne comportant pas d'exercices pratiques pour vérifier que les stagiaires ont assimilé les points présentés
- En fin de formation, chaque stagiaire reçoit une attestation et un certificat attestant qu'il a suivi le cours avec succès.
  - En cas de problème dû à un manque de prérequis de la part du stagiaire, constaté lors de la formation, une formation différente ou complémentaire lui est proposée, en général pour conforter ses prérequis, en accord avec son responsable en entreprise le cas échéant.

Plan

Première session

Linux kernel programming

- Development in the Linux kernel
- Memory allocation
- Linked lists

Exercise : *Writing the "hello world" kernel module*
Exercise : *Adding a driver to kernel sources and configuration menu*
Exercise : *Using module parameters*
Exercise : *Writing interdependent modules using memory allocations, reference counting and linked lists*

Linux kernel debugging

- The /proc and debugfs filesystems
- Traces
- The kernel Dynamic Debugging interface
- The Kernel Address Sanitizer
- Debugging memory problems with kmemleak
- Using the Undefined Behavior Sanitizer
- Code coverage using gcov
- Debugging with kgdb
- Debugging with a JTAG probe

Exercise : *Display dynamic traces on the running kernel*
Exercise : *Debug a module initialization using kgdb*

Kernel multi-tasking

- Task handling
- Concurrent programming

- Timers
- Kernel threads

Exercise : *Fixing race conditions in the previous lab with mutexes*

Deuxième session

Introduction to Linux drivers

- Accessing the device driver from user space
- Driver registration

Exercise : *Step by step implementation of a character driver:*

- *driver registration (major/minor reservation) and device special file creation (/dev)*

Driver I/O functions

- Kernel structures used by drivers
- Opening and closing devices
- Data transfers
- Controlling the device
- Mapping device memory

Exercise : *Step by step implementation of a character driver:*

- *Implementing open and release*
- *Implementing read and write*
- *Implementing ioctl*
- *Implementing mmap*

Troisième session

Synchronous and asynchronous requests

- Task synchronization
- Synchronous request
- Asynchronous requests

Exercise : *implementation of a pipe-like driver:*

- *implementing waiting and waking*
- *adding non-blocking, asynchronous and multiplexed operations (O_NONBLOCK, SIGIO, poll/select)*

Input/Output and interrupts

- Memory-mapped registers
- Interrupts
- Gpios
- User-level access through /sys or the GPIO character driver

Exercise : *Polling gpio driver with raw register access*
Exercise : *Interrupt-based gpio driver with raw register access*
Exercise : *gpio driver using the gpiolib*

Busses

- Plug-and-Play management
- Static devices declaration
  - in the BSP code
  - in the device tree
- Platform bus
- PCI
- SPI
- Power management
  - System sleep

- Implementing power management in drivers
- Remote wakeup

Exercise : *Implementing a platform driver and customizing the device tree to associate it to its device (a serial port)*
Exercise : *Implementing power management in the previous driver*
Exercise : *Implementing remote wakeup in the previous driver*

Quatrième session

Linux Driver Model

- Linux Driver Model Architecture
  - Overview
  - Classes
  - Busses
- Hot plug management
  - Plugging devices
  - Removing devices
- Writing udev rules

Exercise : *Writing a custom class driver*
Exercise : *Writing a misc driver*

DMA

- Direct Memory Access
  - DMA scenarios
  - Buffer access
- DMA programming
  - Bus master DMA
  - Slave DMA
- Memory barriers

Exercise : *Implementing slave DMA in a serial port driver*

Annexes

USB Drivers

- The USB bus
- USB devices
- User-space USB interface
- USB descriptors
- USB requests
- USB device drivers

Exercise : *Writing a USB host driver*

Network drivers

- structures
  - network interface representation (struct net_device)
  - network packet (struct sk_buff)
- scatter/gather
- interface
  - receiving packets
  - sending packets
  - lost packets management
  - network interface statistics
- New network API (NAPI)
  - "interrupt mitigation" (suppression of unneeded IRQs)
  - "packet throttling" (suppression of packets in the driver itself when system is overwhelmed)

Memory management

- Virtual Memory
- Memory Allocation
  - Free page management
  - Normal memory allocation
  - Virtual memory allocation
  - Huge allocations

Renseignements pratiques

Renseignements : 24 heures