D3 - Drivers Linux

Ecritures de drivers Linux

Objectifs

- Maîtriser les outils kernel de développement et de mise au point
- Découvrir la gestion du multi-core dans le noyau Linux
- Programmer les IO, les interruptions, les timers, le DMA
- Adapter les sources des drivers du marché
- Installer et intégrer les drivers dans un kernel linux
- Gérer les interfaces standard synchrones, asynchrones et ioctl
- Développer les structures des drivers caractères, blocs et réseaux
- Comprendre les spécificités des versions 2.6, 3.x et 4.x
- Maîtriser les techniques de debugging noyau avec les sondes jtag Lauterbach.

Les exercices se font sur des cartes cibles :

- *STM32MP15-DISCO basée sur un ARM Cortex/A7 dual cœur.*
- *NXP i.MX6 Sabrelite basée sur un ARM Cortex/A9 quadri cœur.*
- *NXP i.MX8MQ-EVK basée sur un ARM Cortex/A53 quadri cœur.*

Nous utilisons le dernier noyau supporté par le fournisseur du chip (4.x)

Public visé

- Ce cours s'adresse à des ingénieurs qui installent Linux sur une plateforme spécifique et ont à réaliser des pilotes de périphériques.

Matériel

- Un PC Linux par binôme
- Une carte cible par binôme
- Support de cours

Pré-requis

- Bonne maîtrise du langage C
- De préférence, connaissance de la programmation Linux en mode utilisateur (voir notre cours D0 - Programmation en mode utilisateur Linux)

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.

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

1er jour

Programmation noyau

- Développement dans le noyau Linux
- Allocation mémoire et comptage de référence
- Listes chaînées et fifos

Exercise : *Ecrire le module "hello world"*
Exercise : *Ajouter un module aux sources du noyau et au menu de configuration*
Exercise : *Utiliser les paramètres d'un module*
Exercise : *Ecrire des modules interdépendants, utilisant l'allocation mémoire, le comptage de référence et les listes chaînées*

Debug dans le noyau Linux

- Debug avec /proc et debugfs
- Traces
- L'interface de debug dynamique
- Détection dynamique des erreurs mémoire (kasan et kmemleak)
- Détection des dépendance à des comportements non-prédictibles
- Couverture de code lors des tests
- Debug avec kgdb
- Debug avec une sonde JTAG

Exercise : *Affichage de traces dynamiques sur un noyau en fonctionnementDebug a module using kgdb*
Exercise : *Debug de la fonction d'initialisation d'un module avec kgdb*

Multi-tâches noyau

- Gestion des tâches
- Programmation concurrente (mutex, spinlock, atomic_t, rw_semaphore, rwlock, seqlock, RCU)
- Timers
  - jiffies (timer_list)
  - timers haute résolution (hr_timer)

- threads noyau (kthread)

Exercise : *Corriger des "race conditions" de l'exercice précédent avec des mutex*

2ème jour

Introduction aux drivers Linux

- Accès au driver depuis une application en mode utilisateur
- Enregistrement du driver

Exercise : *Implémentation étape par étape d'un driver caractère:*

- *enregistrement du driver (réservation de major/minor) et création du fichier spécial (/dev)*

Pilotes caractère

- Ouverture/fermeture
  - Restriction à une seul ouverture/un seul utilisateur
  - différence entre "close" et "release"
- Transferts de mémoire entre espace noyau et espace utilisateur
  - espaces d'adressage des processus. Swap et pagination
  - fonctions de copie entre espaces
  - fonctionnement en "zéro copie" grâce au mapping d'adresses utilisateur dans le noyau
- Lecture et écriture
  - fonctions de base (read/write)
  - lecture/écriture combinées (readv/writev)
  - fonctions asynchrones en mode synchrone (aio_read/aio_write)
- Contrôle des périphériques
  - fonction ioctl
  - choix des codes de commandes
- Support de l'appel système mmap

Exercise : *Implémentation étape par étape d'un driver caractère:*

- *Implémenter open et release*
- *Implémenter read et write*
- *Implémenter ioctl*
- *Implémenter mmap*

3ème jour

Entrées sorties synchrones et asynchrones

- Synchronisation des tâches
  - files d'attente
  - mise en attente/réveil d'une tâche
  - évènement de complétion
- Entrées/sorties synchrones

Exercise : *Ecriture des fonctions de lecture/écriture synchrones*

- Entrées/sorties asynchrones
  - requêtes non bloquantes
  - asynchrone multiplexé (select et poll)
  - asynchrone notifié (signal SIGIO)
  - asynchrone vrai (totalement parallèle)

Exercise : *Ajout des fonctions de gestion des E/S asynchrones*

Accès au matériel et interruptions

- Registres mappés en mémoire
  - mapping des registres dans un driver

Exercise : *Driver GPIO accédant directement aux registres, en mode polling*

- Interruptions

  - contexte d'exécution des gestionnaires d'interruption
  - "top half" et "bottom halves" (softirq, tasklet, work_queue)
  - irqs threadés
  - synchronisation entre code noyau, "top half" et "bottom halves"

Exercise : *Driver GPIO accédant directement aux registres, sur interruptions*

- Gpios
  - accès aux gpios depuis le noyau (gpiolib)
  - accès aux gpios depuis le mode utilisateur via /sys ou le driver caractère GPIO

Exercise : *Driver GPIO utilisant la gpiolib*

Bus

- Gestion du Plug-and-Play dans Linux
- Déclaration des périphériques statiques
  - dans le code du BSP
  - dans le device tree
- Bus platform

Exercise : *Implémentation d'un driver platform et customisation du device tree pour l'associer à son périphérique (un port série)*

- PCI
- SPI
- Power management
  - mise en veille du système
  - implémentation de la mise en veille et du réveil dans un driver
  - configurer le réveil du système par un périphérique

Exercise : *Implémentation de la mise en veille et du réveil du système dans le driver précédent, et réveil du système par le périphérique*

4ème jour

Modèle driver de Linux

- Architecture des drivers
  - Classes et bus
  - kernel events
  - sysfs
- Gestion du Plug-and-Play
  - Connecter un périphérique
  - Retirer un périphérique
- Classes
  - création automatique de fichier spécial (/dev)
  - créer sa propre classe
  - classe misc
- Ecrire des règles udev

Exercise : *Ecrire un driver qui crée sa propre classe. Le driver se charge automatiquement au boot et le fichier spcial est automatiquement c dans /dev*

Exercise : *Utiliser la classe misc*

DMA

- Utilisations du DMA (direct IO, transferts asynchrones)
- Accès au buffer (problèmes de cache et d'adressabilité)
- DMA bus master
- DMA esclave
- Barrières mémoires

Exercise : *Implémentation d'un driver pour un port série utilisant le DMA esclave*

Annexes

Pilotes USB

- La norme USB
  - notion de configuration
  - notion d'interface (rôle d'un périphérique)
  - notion de terminaison (canal de communication)
  - types des terminaisons (contrôle, interruption, bloc, isochrone)
- Drivers USB host
  - requêtes synchrones (directes)

Exercise : *Examen d'un pilote USB host*

Pilotes réseaux

- structures
  - représentation d'une interface réseau (struct net_device)
  - paquet réseau (struct sk_buff)
- Cas du "scatter/gather"
- interface
  - réception de paquet
  - envoi de paquet
  - gestion des paquets perdus
  - statistiques de l'interface
- Nouvelles API réseau (NAPI, nouveau en 2.6)
  - "interrupt mitigation" (suppression des IRQ inutiles)
  - "paquet throttling" (désengorgement des couches protocolaires)

Memory management

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

Renseignements pratiques

Renseignements : 4 jours