Formation Drivers Linux: Ecritures de drivers Linux - Systèmes d'Exploitation: Android

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)

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 D0 - Linux and uClinux Programming course)

Plan

1er jour

Programmation noyau

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

- Listes chaînées et fifos

Exercice : *Ecrire le module "hello world"*
Exercice : *Ajouter un module aux sources du noyau et au menu de configuration*
Exercice : *Utiliser les paramètres d'un module*
Exercice : *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

Exercice : *Affichage de traces dynamiques sur un noyau en fonctionnementDebug a module using kgdb*
Exercice : *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)

Exercice : *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

Exercice : *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

Exercice : *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

Exercice : *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)

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

Accès au matériel et interruptions

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

Exercice : *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"

Exercice : *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

Exercice : *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

Exercice : *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

Exercice : *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

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

Exercice : *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

Exercice : *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)

Exercice : *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)

Renseignements pratiques

Durée : 4 jours
Prix : 2300 € HT
Prochaines sessions : du 12 au 15 novembre 2019