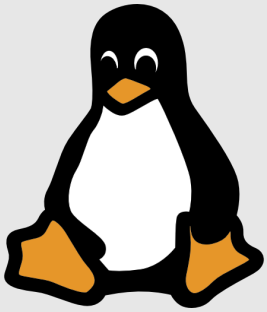


Formation développement noyau et pilotes Linux

Durée de la formation

5 jours – 40 h

Langue

Transparents	Anglais
Présentation	Français
	Anglais

Formateur

Un des ingénieurs suivants

- Grégory Clement
- Maxime Chevallier
- Miquèl Raynal
- Théo Lebrun

Contact

training@bootlin.com

+33 4 84 25 80 96

bootlin
bootlin.com

Formation en présentiel

Public visé

Ingénieurs développant des systèmes reposant sur le noyau Linux.
Ingénieurs supportant des développeurs Linux embarqué.

Objectifs opérationnels

- Être capable de configurer, compiler et installer le noyau Linux sur un système embarqué.
- Être capable de comprendre l'architecture générale du noyau Linux, et comment les applications user-space Linux interagissent avec le noyau Linux.
- Être capable de développer des pilotes de périphériques simples mais complets dans le noyau Linux, au travers du développement à partir de zéro de deux drivers, pour deux périphériques différents, qui illustrent les principaux concepts de la formation.
- Être capable de naviguer dans les différents mécanismes du noyau Linux pour les pilotes de périphériques : Device Tree, device model, infrastructures de bus.
- Être capable de développer des pilotes de périphériques qui communiquent avec des périphériques matériels.
- Être capable de développer des pilotes de périphériques qui exposent les fonctionnalités du matériel aux applications Linux user-space : périphériques caractères, sous-systèmes du noyau Linux pour les périphériques.
- Être capable d'utiliser les principaux mécanismes du noyau Linux pour le développement de pilotes de périphériques : gestion mémoire, verrouillage, gestion des interruptions, mise en sommeil et réveil de threads, DMA.
- Être capable de débugger des problèmes dans le noyau Linux, en utilisant différents outils et mécanismes de debug.

Prérequis

- **Expérience solide en programmation avec le langage C** : les participants doivent maîtriser l'utilisation de types de données et structures complexes, des pointeurs, pointeurs sur fonction et du pré-processeur C.
- **Connaissance et pratique des commandes UNIX ou GNU/Linux** : les participants doivent être à l'aise avec l'utilisation de la ligne de commande Linux. Les participants manquant d'expérience sur ce sujet doivent se former par eux-mêmes, par exemple en utilisant nos supports de formation.
- **Expérience minimale en développement Linux embarqué** : les participants doivent avoir une compréhension minimale de l'architecture d'un système Linux embarqué : rôle du noyau Linux par rapport à l'espace utilisateur, développement d'applications espace utilisateur en C. Suivre la formation Linux embarqué de Bootlin permet de remplir ce pré-requis.
- **Niveau minimal requis en anglais : B1**, d'après le *Common European Framework of References for Languages*, pour nos sessions animées en anglais. Voir la grille CEFR pour une auto-évaluation.

Méthodes pédagogiques

- Présentations animées par le formateur : 50% de la durée de formation
- Travaux pratiques réalisés par les participants : 50% de la durée de formation
- Version électroniques de supports de présentation, des instructions et des données de travaux pratiques. Les supports sont librement disponibles ici.

Modalités d'évaluation

Seuls les participants qui auront assisté à l'intégralité des journées de formation, et qui auront obtenu plus de 50% de réponses correctes à l'évaluation finale recevront une attestation individuelle de formation de la part de Bootlin.

Handicap

Les participants en situation de handicap qui ont des besoins spécifiques sont invités à nous contacter à l'adresse *training@bootlin.com* afin de discuter des adaptations nécessaires à la formation.

Équipement nécessaire

Pour les sessions en présentiel délivrées chez nos clients, notre client devra fournir :

- Projecteur vidéo
- Un ordinateur sur chaque bureau (pour une ou deux personnes), avec au moins 16 Go de RAM et Ubuntu Linux 24.04 installé dans une partition dédiée d'au moins 30 Go.
- Les distributions autres que Ubuntu Linux 24.04 ne sont pas supportées, et l'utilisation de Linux dans une machine virtuelle n'est également pas supportée.
- Connexion à Internet rapide et sans filtrage : au moins 50 Mbit/s de bande passante en téléchargement, et pas de filtrage des sites Web et protocoles.
- Les ordinateurs contenant des données importantes doivent être sauvegardés avant d'être utilisés dans nos sessions.

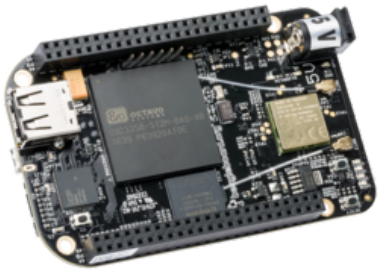
Pour les sessions en présentiel organisés dans les locaux de Bootlin, Bootlin fournit l'ensemble de l'équipement.

Plateforme matérielle pour les travaux pratiques

BeagleBone Black

Carte **BeagleBone Black** ou **BeagleBone Black Wireless**

- Un processeur ARM AM335x de Texas Instruments (à base de Cortex-A8), avec accélération 3D, etc.
- 512 Mo de RAM
- 2 ou 4 Go de stockage eMMC
- USB hôte et device
- Sortie HDMI
- Connecteurs à 2 x 46 broches, pour accéder aux UARTs, aux bus SPI, aux bus I2C, et à d'autres entrées/sorties du processeur.
- Ethernet ou WiFi



BeaglePlay

Carte **BeaglePlay**

- SoC Texas Instruments AM625x (CPU 4xARM Cortex-A53)
- SoC avec accélération 3D, MCU intégré et de nombreux autres périphériques.
- 2 GB de RAM
- 16 Go de stockage eMMC
- USB hôte et device, microSD, HDMI
- WiFi 2.4 and 5 GHz, Bluetooth et aussi Ethernet
- 1 Header MicroBus (SPI, I2C, UART, ...), connecteurs OLDI et CSI.

Programme de la formation

Jour 1 - Matin

Cours	Introduction au noyau Linux	▪ Fonctionnalités et rôle du noyau. ▪ L'interface noyau / espace utilisateur (/proc et /sys). ▪ Architecture générale ▪ Versions du noyau Linux ▪ Organisation du code source
TP	Téléchargement du noyau Linux	▪ Téléchargement du noyau Linux en utilisant Git
Cours	Le code source du noyau	▪ Spécificités du développement noyau ▪ Conventions de codage ▪ Stabilité des interfaces ▪ Aspects juridiques : license ▪ Organisation de la communauté du noyau Linux ▪ Le processus de développement : versions bêta, versions stables, versions long-terme, etc.
TP	Code source du noyau	▪ Effectuer des recherches dans les sources du noyau Linux : recherche de définitions C, de paramètres de configuration et d'autres informations. ▪ En utilisant la ligne de commande UNIX et des outils de navigation dans le code.

Jour 1 - Après-midi

Cours	Configuration, compilation et démarrage du noyau Linux	▪ Configuration du noyau. ▪ Compilation native et croisée. Fichiers générés. ▪ Démarrage du noyau. Paramètres de démarrage. ▪ Montage du système de fichiers racine par NFS.
TP	Configuration, compilation croisée et démarrage sur NFS	▪ Configuration, compilation croisée et démarrage du noyau Linux avec support de NFS.
Cours	Modules noyau Linux	▪ Pilotes de périphériques Linux ▪ Un module simple ▪ Contraintes de programmation ▪ Chargement et déchargement de modules ▪ Dépendances entre modules ▪ Ajouter du code source à l'arbre du noyau
TP	Développement de module	▪ Écriture d'un module noyau offrant quelques fonctionnalités ▪ Accès aux informations internes du noyau depuis le module ▪ Mise en place de l'environnement de compilation

Jour 2 - Matin

Cours	Description des périphériques matériels	▪ Matériel découvrable dynamiquement : USB, PCI ▪ Matériel non-découvrable dynamiquement ▪ Présentation détaillée du Device Tree : syntaxe, propriétés, principes de conception, exemples ▪ Explication des bindings en YAML permettant de définir comme la description matérielle doit être écrite dans un Device Tree.
TP	Description des périphériques matériels	▪ Créer son propre fichier Device Tree ▪ Configurer des LEDs connectées à des GPIOs ▪ Décrire un périphérique connecté sur I2C dans le Device Tree

Jour 2 - Après-midi

Cours	"Pin muxing" (multiplexage d'entrées-sorties)	▪ Comprendre l'infrastructure *pinctrl* du noyau. ▪ Comprendre comment configurer le multiplexage des entrées/sorties.
TP	Pin-muxing	▪ Configurer le pin-mux pour le bus I2C utilisé pour communiquer avec le Nunchuk ▪ Valider que la communication I2C est fonctionnelle en utilisant des outils en espace utilisateur
Cours	Le "device model" de Linux	▪ Comprendre comment le noyau est conçu pour supporter les pilotes de périphériques ▪ Le "device model" ▪ Connexion entre périphériques et pilotes. ▪ Périphériques "platform", le "Device Tree" ▪ Interface en espace utilisateur avec `/sys`

Jour 3 - Matin

Cours	Introduction à l'API I2C	▪ Le sous-système I2C du noyau ▪ Détails sur l'API fournie aux pilotes du noyau pour interagir avec les périphériques I2C.
TP	Communiquer avec le Nunchuk via I2C	▪ Explorer le contenu de `/dev` et `/sys` et les périphériques disponibles sur le système embarqué ▪ Implémenter un driver qui s'enregistre comme driver I2C ▪ Communiquer avec le Nunchuk et lire des données depuis le Nunchuk
Cours	Infrastructures du noyau	▪ Périphériques de type bloc et caractère ▪ Interaction entre applications en espace utilisateur et le noyau ▪ Détails sur les pilotes caractère, `file_operations`, `ioctl()`, etc. ▪ Échange de données vers ou depuis l'espace utilisateur ▪ Le principe des infrastructures du noyau

Jour 3 - Après-midi

Cours	Le sous-système input	▪ Principe du sous-système *input* du noyau ▪ API offerte aux pilotes du noyau pour exposer des fonctionnalités de périphériques d'entrée aux applications en espace utilisateur. ▪ API en espace utilisateur offerte par le sous-système *input*
TP	Exposer la fonctionnalité du Nunchuk en espace utilisateur	▪ Extension du pilote du Nunchuk pour exposer les fonctionnalités du Nunchuk aux applications en espace utilisateur, comme un périphérique d'entrée. ▪ S'assurer du bon fonctionnement du Nunchuk via `evtest`
Cours	Gestion de la mémoire	▪ Linux : gestion de la mémoire. Espaces d'adressages physique et virtuel, séparation noyau et espace utilisateur. ▪ Implémentation de la gestion de la mémoire dans Linux. ▪ Allocation avec `kmalloc()`. ▪ Allocation par pages. ▪ Allocation avec `vmalloc()`.

Jour 4 - Matin

Cours	Entrées-sorties avec le matériel	▪ Enregistrement des plages de mémoire d'E/S. ▪ Accès aux plages de mémoire d'E/S. ▪ Barrières mémoire.

TP	Pilote "platform" minimal et accès à la mémoire d'E/S	▪ Réalisation d'un pilote "platform" minimal ▪ Modification du Device Tree pour ajouter un nouveau port série. ▪ Réservation des adresses d'E/S utilisées par le port série. ▪ Lecture et écriture des registres du périphérique, pour envoyer des caractères sur le port série.
Cours	Le sous-système misc	▪ Utilité du sous-système *misc* du noyau ▪ API du sous-système *misc*, à la fois du côté du noyau, et du côté de l'espace utilisateur.
TP	Pilote de port série en écriture seule	▪ Extension du pilote commencé dans le TP précédent, en enregistrant celui-ci dans le sous-système *misc* ▪ Implémentation de l'écriture vers le port série en utilisant le sous-système *misc* ▪ Tests d'écriture depuis l'espace utilisateur

Jour 4 - Après-midi

Cours	Processus, ordonnancement, sommeil et interruptions	▪ Gestion des processus dans le noyau Linux. ▪ L'ordonnanceur du noyau Linux et la mise en sommeil des processus. ▪ Gestion des interruptions dans les pilotes de périphérique : enregistrement et développement des gestionnaires d'interruption, exécution différée de tâches.
TP	Mise en sommeil et gestion d'interruptions dans un pilote de périphérique	▪ Ajout de la fonctionnalité de lecture au pilote caractère développé précédemment. ▪ Enregistrement d'un gestionnaire d'interruption. ▪ Attente de la disponibilité de données dans l'opération `read()` ▪ Réveil lorsque les données deviennent disponibles.
Cours	Verrouillage	▪ Problématique de l'accès concurrent à des ressources partagées ▪ Primitives de verrouillage : mutexes, sémaphores, spinlocks. ▪ Opérations atomiques. ▪ Problèmes typiques de verrouillage. ▪ Utilisation du validateur de verrouillage pour identifier les sources de problèmes.
TP	Verrouillage	▪ Ajout de mécanismes de verrouillage au pilote en cours

Jour 5 - Matin

Cours	DMA : Direct Memory Access	▪ *Peripheral DMA* par rapport à l'utilisation de contrôleur DMA ▪ Contraintes du DMA : cache, adressage ▪ APIs du noyau Linux pour le DMA : `dma-mapping`, `dmaengine`, `dma-buf`
TP	DMA : Direct Memory Access	▪ Mise en place de *streaming mappings* avec l'API `dma-mapping` ▪ COnfiguration du contrôleur DMA avec l'API `dmaengine` ▪ Configurer le matériel pour lancer les transfers DMA ▪ Attendre la fin d'un transfert DMA

Jour 5 - Après-midi

Cours	Techniques de débogage noyau	▪ Débogage avec les fonctions d'affichage ▪ Utilisation de debugfs ▪ Analyse d'un oops noyau ▪ Utilisation de kgdb, un débogueur noyau ▪ Utilisation des commandes SysRq

TP	Investigation de bugs noyau	▪ Étude d'un pilote incorrect. ▪ Analyse du message d'erreur et recherche du problème dans le code source.
Cours	Gestion de l'énergie	▪ Vue d'ensemble des fonctionnalités de gestion d'énergie du noyau Linux. ▪ Sujets abordés : horloges, mise en veille et réveil, ajustement automatique de la fréquence, économie d'énergie dans la boucle idle, "runtime power management", régulateurs, etc.
Cours	S'il reste du temps	▪ mmap

Travaux pratiques

Les travaux pratiques de cette formation font appel aux périphériques matériels suivants, pour illustrer le développement de pilotes de périphériques pour Linux :

- Une manette Nunchuk pour console Wii, qui est connectée à la BeagleBone Black via le bus I2C. Son pilote utilisera le sous-système *input* du noyau Linux.
- Un port série (UART) supplémentaire, dont les registres sont mappés en mémoire, et pour lequel on utilisera le sous-système *misc* de Linux.

Bien que nos explications cibleront spécifiquement les sous-systèmes de Linux utilisés pour réaliser ces pilotes, celles-ci seront toujours suffisamment génériques pour faire comprendre la philosophie d'ensemble de la conception du noyau Linux. Un tel apprentissage sera donc applicable bien au delà des périphériques I2C, d'entrée ou mappés en mémoire.