Formation Linux : écriture de drivers et programmation noyau

Réf. SF34727

Développez des pilotes de périphériques fiables pour le noyau Linux, du premier module au driver qui pilote du matériel réel, gère les interruptions et s'intègre aux sous-systèmes réseau et USB.

4 j Durée	2 600 € HT Inter / participant	0 Sessions à venir

Objectifs pédagogiques

✓ Compiler, charger et déboguer des modules pour le noyau Linux, y compris en cross-compilation pour une cible embarquée

✓ Développer un driver en mode caractère exposant lecture, écriture, ioctl, poll et mmap à l'espace utilisateur

✓ Implémenter la synchronisation et les attentes adaptées au contexte d'exécution du code noyau

✓ Piloter du matériel depuis un driver : accès aux registres et aux GPIO, gestion des interruptions et traitement différé

✓ Intégrer un driver au modèle de périphériques du noyau à l'aide du Device Tree, des platform drivers et de sysfs

✓ Développer des drivers pour les sous-systèmes réseau et USB

Public visé & prérequis

Public viséDéveloppeurs, Testeurs logiciels

Prérequis- Bonne maîtrise du langage C : pointeurs et pointeurs de fonctions, structures, allocation mémoire, compilation séparée - Pratique de Linux en ligne de commande, de make et d'un outil de gestion de version (Git) - Notions d'architecture des ordinateurs (mémoire virtuelle, interruptions, bus) et de programmation concurrente (threads, sections critiques)

Programme détaillé

Noyau Linux, modules et outils de développement

– Architecture du noyau Linux : noyau monolithique modulaire, modes superviseur et utilisateur, rôle des pilotes
– Cycle de développement et licence : branches stables et LTS, patchs, licence GPL et exportation des symboles
– Sources et chaîne de construction : organisation de l'arborescence, Kconfig, Kbuild et Makefile, compilation du noyau et des modules
– Cross-compilation : construction d'un noyau et de modules pour une cible embarquée ARM
– Structure d'un module : fonctions d'initialisation et de sortie, macros, paramètres de module, métadonnées
– Chargement et dépendances : insmod, modprobe, depmod, chargement automatique et paramètres de boot
– Mise au point du code noyau : printk et debug dynamique, lecture d'un oops, ftrace, débogage avec GDB/KGDB sous QEMU

Activités pratiques :

– Configuration, compilation et démarrage d'un noyau dans une machine virtuelle
– Écriture, compilation et chargement d'un premier module avec paramètres
– Mise en œuvre de deux modules dépendants par exportation de symboles
– Analyse d'un oops provoqué volontairement et traçage d'une fonction avec ftrace

Appels système et API interne du noyau

– Appels système : mécanisme de passage en mode noyau, suivi des appels d'une application avec strace
– Contexte d'exécution : tâches et processus, pointeur current, contexte processus et contexte d'interruption, préemptibilité
– Espaces d'adressage : séparation utilisateur / noyau, échanges sécurisés avec copy_to_user et copy_from_user
– Services du noyau sans libc : chaînes, blocs mémoire, conversions, gestion des erreurs
– Structures de données du noyau : listes chaînées, kfifo, macro container_of
– Gestion du temps : jiffies, timers haute résolution, attentes, actions différées et threads noyau
– Interface /proc : exposition d'informations sur les processus et le système

Activités pratiques :

© 2026 Sparks Formation. Tous droits réservés. Fiche générée depuis https://www.sparks-formation.com/formation/linux-drivers-programmation-noyau/

– Observation des appels système invoqués par une commande utilisateur
– Module d'information sur les structures internes d'un processus exposé via /proc
– Mesure de la granularité d'horodatage et de la précision des timers
– Création d'un thread noyau périodique

Driver en mode caractère

– Fichiers spéciaux et VFS : place du driver entre l'application, le Virtual File System et le matériel
– Numéros majeur et mineur : réservation dynamique, enregistrement du périphérique caractère (cdev)
– Classes de périphériques : création automatique des nœuds /dev avec udev
– Méthodes file_operations : open, release, read, write et gestion de la position
– Paramétrage par ioctl : définition des commandes, échange de structures, contrôle des arguments
– Misc devices : enregistrement simplifié pour les pilotes à nœud unique

Activités pratiques :

– Écriture progressive d'un driver caractère pour un périphérique virtuel
– Création d'une classe personnalisée et génération automatique du nœud /dev
– Implémentation d'un ioctl de configuration et d'une application de test

Concurrence, mémoire et entrées-sorties avancées

– Nécessité de la synchronisation : préemption, multiprocesseur, mise en évidence des race conditions
– Mutex, spinlocks et variables atomiques : choix du mécanisme selon le contexte d'exécution
– Files d'attente (waitqueue) : appels système bloquants et non bloquants
– Multiplexage des entrées-sorties : implémentation de poll pour select, poll et epoll
– Allocation mémoire : kmalloc, vmalloc, allocation de pages et caches slab, adresses virtuelles et physiques
– Projection en espace utilisateur : implémentation de mmap pour partager un tampon sans copie
– Diagnostic du driver : exposition d'informations de mise au point avec debugfs

Activités pratiques :

– Mise en évidence d'une race condition puis correction par mutex
– Périphérique virtuel « file de messages » avec lectures bloquantes et support de poll
– Implémentation de mmap et accès au tampon du driver depuis une application
– Test des limites d'allocation selon les différentes fonctions

Modèle de périphériques, Device Tree et platform drivers

– Linux Device Model : bus, devices, drivers et classes, comptage de références avec kobject et kref
– sysfs : lecture et création d'attributs pour configurer un driver depuis l'espace utilisateur
– Hotplug et uevents : notification des ajouts et retraits de périphériques, règles udev
– Device Tree : description du matériel non découvrable, nœuds, propriétés et chaîne de compatibilité
– Platform drivers : association driver / périphérique, fonctions probe et remove
– Ressources managées (devm_*) : libération automatique à la déconnexion pour des drivers plus robustes
– Firmware et gestion d'énergie : chargement de firmware, principes du suspend/resume et du runtime PM

Activités pratiques :

– Conversion du driver caractère en platform driver instancié depuis le Device Tree
– Exposition d'attributs de configuration dans sysfs
– Ajout d'un nœud Device Tree sur une cible ARM émulée et vérification du probe

Accès au matériel, interruptions et DMA

– Accès aux registres : ports d'entrée-sortie, I/O mappées en mémoire (ioremap), fonctions d'accès portables
– GPIO : pilotage par l'API à descripteurs (gpiod) et déclaration dans le Device Tree
– Gestion des interruptions : enregistrement d'un handler, contraintes du contexte d'interruption, activation et désactivation
– Traitement différé : threaded interrupts et workqueues ; tasklets en héritage dans le code existant
– Latence et temps réel : priorités des threads d'interruption, mesure de la latence
– DMA : principes des transferts directs, allocation de tampons cohérents et API DMA
– Bus PCI Express : détection et enregistrement d'un driver, Base Address Registers, interruptions MSI

Activités pratiques :

– Driver d'entrées-sorties GPIO sur périphérique émulé ou carte embarquée
– Écriture d'un gestionnaire d'interruption avec traitement différé par threaded IRQ

© 2026 Sparks Formation. Tous droits réservés. Fiche générée depuis https://www.sparks-formation.com/formation/linux-drivers-programmation-noyau/

– Mesure de la latence des interruptions et influence des priorités temps réel
– Étude d'un driver PCIe pour un périphérique émulé sous QEMU

Drivers réseau et périphériques bloc

– Architecture réseau de Linux : interfaces bas niveau, pile IP et protocoles
– Périphérique net_device : allocation, enregistrement, activation et adressage
– Socket buffers (sk_buff) : émission et réception de paquets, trajet des données dans la pile IP
– Statistiques et état de l'interface : compteurs d'utilisation, gestion du lien
– Périphériques bloc : principe, enregistrement d'un disque générique, traitement des requêtes avec blk-mq
– Stockage et VFS : partitionnement, formatage, montage et rôle des caches disque

Activités pratiques :

– Écriture progressive d'un driver réseau virtuel (interface miroir) utilisable en IPv4
– Gestion des statistiques d'interface et observation du trajet des paquets
– Driver de disque virtuel en mémoire, puis partitionnement, formatage et montage

Drivers USB

– Sous-système USB de Linux : USB core, énumération, descripteurs, interfaces et endpoints
– Enregistrement d'un driver USB : table d'identifiants, fonctions probe et disconnect
– URB (USB Request Blocks) : construction, soumission et fonctions de complétion
– Types de transferts : Control, Bulk, Interrupt et Isochrone selon les usages
– Robustesse du driver : déconnexions à chaud, accès concurrents et écritures successives rapides

Activités pratiques :

– Énumération et identification d'un périphérique USB
– Écriture d'un driver USB de lecture et d'écriture vers un périphérique physique ou émulé
– Gestion de la déconnexion d'un périphérique en cours d'utilisation

© 2026 Sparks Formation. Tous droits réservés. Fiche générée depuis https://www.sparks-formation.com/formation/linux-drivers-programmation-noyau/

Pourquoi choisir Sparks Formation ?

20 ans d'expertise IT, des formateurs experts et des sessions garanties contre l'annulation.

20 ans d'expertise	1 068+ formations	11 178 apprenants formés/an	+67 Net Promoter Score

Avis des participants

Note moyenne des participants de la filière Administration Systèmes Unix/Linux : 4,9/5 — 28 avis

Alexander A. — Globalement, j'ai trouvé cette formation très intéressante et enrichissante. Le contenu était pertinent et m'a permis de mieux comprendre plusieurs notions de Linux. Le formateur était disponible et…

Alexander A. — Formation LPIC-101 très enrichissante. Le formateur est clair, compétent et possède une grande expérience terrain. J'ai beaucoup appris tout au long de la formation. C'était vraiment agréable de…

Ils nous font confiance

Cette année notre filière Administration Systèmes Unix/Linux a accompagné plus de 51 entreprises dont :

SOPRA STERIA I2S SOPRA STERIA GROUP CIMPA SAS CS GROUP FRANCE ADISTA ECONOCOM SERVICES & SOLUTION ECONOCOM WORKPLACE INFRA INNOVATION SULLY GROUP ARPEGE CEGEDIM ACTIV +41 autres

Financement & certifications

Cette formation est éligible aux financements OPCO et FNE-Formation. Nos équipes vous accompagnent dans le montage de votre dossier.

Qualiopi processus certifié RÉPUBLIQUE FRANÇAISE

SILVER | Top 15% ecovadis Sustainability Rating JAN 2026

Réservez votre place dès maintenant

20 ans d'expertise · 1 000+ formations · NPS +67 · réponse sous 24h ouvrées

0 805 950 800

demande@sparks-formation.com

© 2026 Sparks Formation. Tous droits réservés. Fiche générée depuis https://www.sparks-formation.com/formation/linux-drivers-programmation-noyau/