FORMATIONS LINUX

MNIS – Tour de l'Horloge - 4, place louis Armand – 75012 Paris

TEL : 0950 070814

LDI

NOYAU LINUX ET DEVELOPPEMENT DE DRIVERS

Durée : 5 jours

Prix : 1740€

GROUPE DE FORMATIONS

La formation fait partie du groupe de formation « Développement»

Développement			
ARM	Assembleur et architecture ARM	5	
LGC	Langage C	5	DEBUTANT
LIS	Développement d'applications sous Linux	5	
PTR	Programmation temps réel	4	
SHL	Ecrire des scripts Shell	4	DEBUTANT
KSH	Ecrire des scripts Korn Shell	4	
LDI	Noyau Linux et développement de drivers	5	AVANCÉ
DRN	Développement de drivers réseau sous Linux	5	AVANCÉ
TCP	Développement TCP-IP sous UNIX	5	AVANCÉ

QUEL OBJECTIF

Cette formation permet de passer en revue l'ensemble des outils disponible dans le noyau Linux pour écrire un driver Linux et d'apprendre à connaître les différents outils utiles pour la mise en œuvre et le debug du noyau et des pilotes de périphériques.

POUR QUI

Cette formation est adaptée aux techniciens et ingénieurs, désirant perfectionner leurs connaissances sur le noyau Linux ou ayant besoin de développer des pilotes de périphérique pour Linux.

PRE-REQUIS

Connaissance générale des systèmes d'exploitation. Et de préférence, connaissance de la programmation en langage C.

POUR QUOI

Vous développez une nouvelle carte pour PC ou un système embarqué et vous avez besoin d'écrire un pilote de périphérique pour interfacer Linux avec votre nouvelle carte.

DEROULE DE LA FORMATION

PRESENTATION DU NOYAU

Rappels sur le fonctionnement du système d'exploitation Linux
Cycles de développement du noyau, les patchs, GIT.
Organisation des sources (Include/linux, Arch, Kernel, Documentation, ...).
Principe de compilation du noyau et des modules, les dépendances et symboles.
Outils de développement (Gcc, Kbuild, Kconfig et Makefile, ...) et de debug KGDB, ftrace, LTT
Le chargement du noyau (support, argument,...).
Travaux pratiques : Compilation et installation d'un noyau 3.x. , développement d'un module simple

GESTION DES THREADS, SCHEDULING

Représentation des threads (structure task_struct, thread_info, ...).
Les threads, contexte d'exécution.
Les ordonnanceurs de Linux et la préemption.
Création d'un thread noyau (kthread_create, wakeup_process, ...).
Travaux pratiques : Ecrire un module qui créé un thread noyau et un autre qui fourni des informations sur les structures internes des processus.

GESTION DE LA MEMOIRE, SYNCHRONISATIONS

L'organisation mémoire pour les architectures UMA et NUMA.
L'espace d'adressage utilisateur et noyau.
Allocations mémoire, buddy allocator, kmalloc, slabs et pools mémoire.
La gestion des accès à la mémoire (les caches et la MMU).
La gestion de pages à la demande (demand paging), copy-on-write, allocations lazy, hugepages
Les problèmes liés à la sur-réservation de la mémoire (OOM killer).
Synchronisations et attentes dans le noyau, spinlocks, waitqueues, mutex et les completions.
Travaux pratiques : Ecriture d'un driver en mode « loopback » pour la mise en œuvre des synchronisations

GESTION DU TEMPS

L'horloge temps réel, RTC (real Time Clock), implémentation des timers, les jiffies.
Interface timers haute résolution, estampilles.
Les outils spécifiques au noyau, listes chaînées, kfifo et container_of.
Travaux pratiques : Utilisation des timers et des estampilles. Implémentation d'un accès au procfs. Mise en oeuvre de l'allocation mémoire dans le noyau et optimisation à l'aide des slabs.

LES INTERRUPTIONS, DRIVER CARACTERE

Ecriture de pilotes de périphériques caractère.
Le VFS (Virtual File System).

Les méthodes associées aux périphériques caractères.
Gestion des interruptions, du DMA et des accès au matériel.
Enregistrement des pilotes de périphériques de type caractère et optimisations.
Travaux pratiques : Ecriture progressive d'un pilote périphérique en mode caractère. Implémentation des synchronisations d'entrée-sortie entre threads et avec la routine d'interruption. Implémentation de l'allocation mémoire. Enregistrement d'une interuption partagée.

LINUX DRIVER FRAMEWORK – SYSFS ET PROCFS

L'interface noyau avec /proc par le procfs. Gestion mémoire par le procfs.
Présentation du framework, kobject, kset et kref. Interaction avec le sysfs (/sys).
Les objets drivers, device driver, bus et class.
Utilisation et génération des attributs présentés dans le sysfs.
Interface avec le hotplug, méthodes match, probe et release.
Gestion du firmware.
Gestion de l'énergie, méthodes de gestion de l'énergie.
Travaux pratiques : Implémentation d'un bus, d'un driver et d'un device driver. Adaptation du pilote de périphériques caractère. Exemple d'utilisation de l'interface.

PERIPHERIQUE EN MODE BLOC ET SYSTEMES DE FICHIER

Principe des périphériques en mode bloc. Enregistrement du driver.
Callback de lecture et écriture. Support du formatage et opérations avancées.
Ordonnanceur des entrées-sorties par bloc du noyau.
Conception des systèmes de fichiers.
Enregistrement d'un nouveau système de fichiers.
Travaux pratiques Exemple de pilote complet de périphérique block virtuel. Exemple d'un système de fichiers personnalisé.

INTERFACES ET PROTOCOLES RESEAU

Gestion des interfaces réseau sous Linux.
Utilisation des skbuff.
Les hooks netfilter.
Intégration d'un protocole.
Travaux pratiques : Exemple de driver réseau pour périphérique virtuel. Implémentation de protocole réseau.

AUTRES DRIVERS

Principe des périphériques USB. Interface avec le module USB-core.
Construction d'un URB (USB Request Block).
Les gadgets USB.
Les drivers audio
Les drivers vidéo, Linux Frame Buffer
les drivers d'entrée, drivers input.
Travaux pratiques : Driver pour périphérique d'entrée (input)