Linux, drivers et programmation noyau

Cours Pratique de 4 jours - 28h
Réf : LDI - Prix 2024 : 2 550€ HT

Cette formation vous permettra de maîtriser le développement de pilotes de périphériques (drivers) robustes et adaptés aux différentes distributions de Linux. Vous verrez les différents types de périphériques, la gestion de la mémoire, l'implémentation de protocole réseau ainsi que les périphériques USB.

OBJECTIFS PÉDAGOGIQUES

À l'issue de la formation l'apprenant sera en mesure de :

Maîtriser le développement de pilotes de périphériques

Comprendre en détail les mécanismes internes du noyau

Savoir développer et intégrer de nouveaux éléments dans le noyau Linux

Ecrire un pilote périphérique en mode caractère ou bloc

TRAVAUX PRATIQUES

Les nombreux exercices et études de cas progressifs seront réalisés sur un réseau de serveurs Linux. Tous les programmes réalisés en TP existent sous forme de squelettes que les participants complètent eux-mêmes.

LE PROGRAMME

dernière mise à jour : 01/2018

1) Présentation du noyau

- Vue d'ensemble du système et rôle du noyau.
- Les sites de référence.
- Spécificités des noyaux 3.x et 4.x.
- Cycles de développement du noyau, les patchs.
- Mode de fonctionnement (superviseur et utilisateur). Appels système.
- Organisation des sources (Include/linux, Arch, Kernel, Documentation...).
- Principe de compilation du noyau et des modules.
- Les dépendances et symboles.
- Les exportations de symboles.
- Le chargement du noyau (support, argument...).

Travaux pratiques : Compilation et installation d'un noyau 3.x.

2) Les outils utilisables

- Outils de développement (Gcc, Kbuild, Kconfig et Makefile...).
- Outils de débogage (GDB, KGDB, ftrace...).
- Environnement de débogage (Linux Trace Toolkit...).
- Outil de gestion de version (Git...).
- Tracer les appels système (ptrace...).

Travaux pratiques : Installer l'ensemble des outils et des sources pour générer un module pour le noyau. Configurer le système pour effectuer le chargement automatique de module au boot. Ecriture et test de modules simples.

3) Gestion des threads, scheduling

- Les différents types de périphériques.
- Contextes de fonctionnement du noyau. Protection des variables globales.

PARTICIPANTS

Développeurs Linux/Unix.

PRÉREQUIS

Bonnes connaissances de Linux/Unix et de la programmation C.

COMPÉTENCES DU FORMATEUR

Les experts qui animent la formation sont des spécialistes des matières abordées. Ils ont été validés par nos équipes pédagogiques tant sur le plan des connaissances métiers que sur celui de la pédagogie, et ce pour chaque cours qu'ils enseignent. Ils ont au minimum cinq à dix années d'expérience dans leur domaine et occupent ou ont occupé des postes à responsabilité en entreprise.

MODALITÉS D'ÉVALUATION

Le formateur évalue la progression pédagogique du participant tout au long de la formation au moyen de QCM, mises en situation, travaux pratiques…
Le participant complète également un test de positionnement en amont et en aval pour valider les compétences acquises.

MOYENS PÉDAGOGIQUES ET TECHNIQUES

• Les moyens pédagogiques et les méthodes d'enseignement utilisés sont principalement : aides audiovisuelles, documentation et support de cours, exercices pratiques d'application et corrigés des exercices pour les stages pratiques, études de cas ou présentation de cas réels pour les séminaires de formation.
• À l'issue de chaque stage ou séminaire, ORSYS fournit aux participants un questionnaire d'évaluation du cours qui est ensuite analysé par nos équipes pédagogiques.
• Une feuille d'émargement par demi-journée de présence est fournie en fin de formation ainsi qu'une attestation de fin de formation si le stagiaire a bien assisté à la totalité de la session.

MODALITÉS ET DÉLAIS D'ACCÈS

L'inscription doit être finalisée 24 heures avant le début de la formation.

ACCESSIBILITÉ AUX PERSONNES HANDICAPÉES

Vous avez un besoin spécifique d'accessibilité ? Contactez Mme FOSSE, référente handicap, à l'adresse suivante psh-accueil@orsys.fr pour étudier au mieux votre demande et sa faisabilité.

- Représentation des threads (état, structure task_stru, thread_info...).
- Les threads, contexte d'exécution.
- Le scheduler de Linux et la préemption.
- Création d'un thread noyau (kthread_create, wakeup_process...).

Travaux pratiques : Créer un module qui crée un thread noyau lors de l'insertion et le décharge lors du rmmod. Ecriture d'un module d'horodatage d'événements à haute précision. Ecriture d'un module d'information sur les structures internes des processus.

4) Gestion de la mémoire, du temps et de proc

- L'organisation mémoire pour les architectures UMA et NUMA.
- L'espace d'adressage utilisateur et noyau.
- La gestion de pages à la demande (demand paging).
- Allocations mémoire, buddy allocator, kmalloc, slabs et pools mémoire.
- La gestion des accès à la mémoire (les caches et la MMU).
- Les problèmes liés à la sur-réservation de la mémoire.
- Gestion de la mémoire sur x86 et ARM, utilisation des Hugepages.
- Optimisation des appels systèmes (IAPX32, VDSO).
- Synchronisations et attentes dans le noyau, waitqueues, mutex et les completions.
- Les ticks et les jiffies dans Linux.
- L'horloge temps réel, RTC (real Time Clock), implémentation des timers.
- Interface timers haute résolution, estampilles.
- Les outils spécifiques au noyau, listes chaînées, kfifo et container_of.
- L'interface noyau avec /proc par le procfs.

Travaux pratiques : Utilisation des timers et des estampilles. Implémentation d'un accès au procfs. Mise en œuvre de l'allocation mémoire dans le noyau et optimisation à l'aide des slabs.

5) Périphérique en mode caractère

- Ecriture de pilotes de périphériques caractère.
- Le VFS (Virtual File System).
- Les méthodes associées aux périphériques caractères.
- Gestion des interruptions DMA et 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.

6) Linux Driver Framework - sysfs

- Présentation du framework, kobject, kset et kref.
- 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.

7) Périphérique en mode bloc et systèmes de fichiers

- 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 virtuel. Exemple d'un système de fichiers personnalisé.

8) Interfaces et protocoles réseau

- 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.

9) Drivers pour périphériques USB

- Principe des périphériques USB. Interface avec le module USB-core.
- Interaction du périphérique avec le noyau Linux.
- Construction d'un URB (USB Request Block).
- Les gadgets USB.

Travaux pratiques : Enregistrement d'un driver USB. Ecriture d'un driver en mode isochrone.

LES DATES

CLASSE À DISTANCE
2024 : 11 juin, 17 sept., 17 déc.

PARIS
2024 : 04 juin, 10 sept., 10 déc.