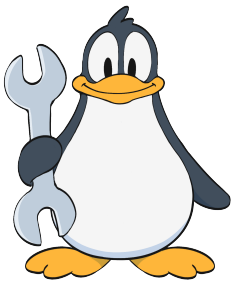
logilin
Logilin formations

Écriture de drivers pour Linux



Le support pour les périphériques est assuré sous Linux par des ***drivers*** (pilotes) dont le code se déroule dans le **kernel** (noyau) du système d'exploitation. Il est donc nécessaire pour écrire ou tester des *drivers* de périphériques de maîtriser les concepts propres à la **programmation noyau**.

Ce cours propose une approche originale, s'appuyant sur l'écriture progressive de *drivers* de différents types, pour appréhender les mécanismes parfois complexes (préemptibilité, multiprocesseur, support d'architectures différentes, etc.) inhérents au code exécuté en mode noyau.

Outre les périphériques classiques (caractère, réseau), on étudie certains sous-systèmes du noyau tels que les périphériques PCIe ou le bus USB.

Organisation

Audience

Nous limitons habituellement le nombre de stagiaires dans nos sessions à 4 personnes au maximum pour garantir des échanges fluides et conviviaux.

Les sessions à distance se déroulent sur **plateforme Zoom**. Le seul matériel nécessaire est un ordinateur avec une connexion Internet et un micro. Nous conseillons un ensemble casque + micro pour limiter le bruit de fond. Nous suggérons également l'emploi d'une webcam si l'environnement le permet.

Pré-requis

Connaissance de Linux (niveau utilisateur). Notions de langage C.

Durée

4 jours (28 heures)

Travaux pratiques

Les exercices se déroulent sur des PC Linux et des cartes Beagle Bone accessibles à distance (connexion SSH / PuTTY / Tera Term),

Thèmes abordés

Le noyau Linux et ses modules : noyau Linux, développement, écriture de modules.

Les appels-système Linux : appels-système, environnement du noyau, `/proc`.

A.P.I. interne du noyau Linux : fonctions habituelles, mise au point et débogage, éléments temporels, gestion mémoire.

Driver en mode caractères : enregistrement, méthodes fondamentales, synchronisation.

Communication avec le matériel : entrées-sorties, interruptions, attente d'événement, projection mémoire et DMA.

Périphériques réseau : interface, enregistrement, communications.

Utilisation d'un bus : sous-système USB core, implémentation d'un driver en mode *interrupt*.

Plan détaillé

Le noyau Linux et ses modules

Noyau Linux : modèle, versions, évolutions du *kernel*, modules.

Développement en mode noyau : outils, compilation du noyau, programmation en mode noyau.

Écriture de modules kernel : *headers*, macros, licence, messages, *Makefile*, dépendances.

Travaux pratiques

Manipulation des modules précompilés, écriture, compilation et test d'un module, tests de modules avec dépendances.

Les appels-système Linux

Appels-système : principe, suivi d'un appel système, préemptibilité du noyau.

Environnement du noyau : contexte de tâche, espaces d'adressage.

Système de fichiers /proc : ajout d'entrées, méthodes de lecture et d'écriture.

Travaux pratiques

Appels système invoqués par une application, échange de données entre espace utilisateur et *kernel*.

A.P.I. interne du kernel

Fonctions habituelles de la libC : chaînes, mémoire, fonctions numériques.

Mise au point et débogage : *warning*, *panic*.

Éléments temporels : *ticks*, mesure du temps, attentes, actions différées.

Gestion mémoire : allocations, gestion des pages.

Travaux pratiques

Mesure de granularité de l'horodatage, de précision des *timers*, appels-système de base, manipulation d'adresses virtuelles, physiques et de pages mémoire.

Driver en mode caractères

Enregistrement d'un driver : principe, numéros, modèles de drivers, classes.

Méthodes du driver : *open()*, *close()*, *read()*, *write()*, *ioctl()*.

Synchronisation entre appels système : nécessité d'une synchronisation, *mutex*.

Travaux pratiques

Numéros majeurs et mineurs, appel des méthodes du driver, classe personnalisée, écriture d'un driver simple. Implémentation d'un *ioctl()*.

Accès au matériel

Entrées-sorties : ports dédiés, *mapping* en adresses virtuelles, GPIO, bus PCIe.

Interruptions : contexte d'interruption, *handlers* bas-niveau, activation désactivation, routine de service

Traitement différé : *tasklet*, *workqueue*, *threaded interrupt*.

Synchronisation et attentes d'événements : *spinlock*, *waitqueue*, opérations bloquantes et non-bloquantes.

Projection de mémoire : méthode *mmap()*, DMA.

Travaux pratiques

Interactions par GPIO, écriture d'un *handler* d'interruption, driver virtuel en mode caractères, synchonisation par *mutex* et *spinlock*, utilisation d'une *waitqueue*, implémentation de *mmap()*.

Driver réseau

Périphériques réseau : interfaces, protocoles, enregistrements et adressage.

Driver réseau : enregistrement, activation, *socket buffer*, statistiques d'utilisation.

Travaux pratiques

Enregistrement d'un driver *netdevice* miroir, gestion des statistiques.

Utilisation d'un bus

Le sous-système USB : principe, environnement USB Core;

Implémentation d'un driver: enregistrement, identifiants, *endpoints*, classes, URB.

Travaux pratiques

Communication avec un bus USB, énumération, écriture et lecture sur un device USB

Conclusion

Discussions libres sur l'ensemble des thèmes abordés.

Travaux pratiques

Expérimentations libres suivant les demandes des participants.

MàJ 02/08/2024