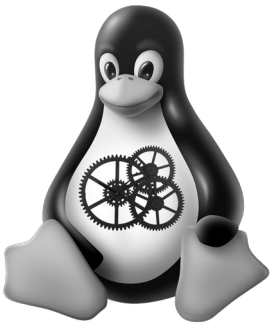
Logilin formations

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 noyau du système d'exploitation. Il est donc nécessaire pour le développeur amené à écrire ou à 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, bloc, 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 participants 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

Sessions en nos locaux : 4 jours (28 heures)

Sessions à distance : 3 jours (21 heures)

Travaux pratiques

Sessions en nos locaux : les exercices ont lieu sur PC Linux mis à votre disposition et sur des cartes Beagle Bone.

Sessions à distance : 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 : programmation noyau, mise au point et débogage, éléments temporels, gestion mémoire.

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

Communication avec le matériel : entrées-sorties, interruptions, projection et DMA.

Périphériques réseau : interface, enregistrement, communications.

Utilisation d'un bus : sous-système USB, driver en mode interrupt, aperçu des autres modes.

Plan détaillé

Le noyau Linux et ses modules

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

Travaux pratiques

Versions installées et disponibles, Manipulation des modules précompilés.

Développement en mode noyau : outils, compilation du noyau et des modules.

Travaux pratiques

Exemples de modules simples.

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

Travaux pratiques

Modules acceptant des paramètres. Dépendances entre modules.

Les appels-système Linux

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

Travaux pratiques

Observation des appels système invoqué par des applications.

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

Travaux pratiques

Écriture d'un module d'information sur la tâche appelante.

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

Travaux pratiques

Installation de fonctions *callback* sur des entrées de `/proc`.

A.P.I. interne du noyau Linux

Programmation noyau : chaînes, blocs , fonctions numériques, conversions.

Mise au point et débogage : avertissements, paniques, *debugfs*.

Travaux pratiques

Observation des effets des fonctions de mise au point.

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

Travaux pratiques

Mesure de précision des *timers*. Écriture d'un module d'horodatage.

Gestion mémoire : allocations, gestion des pages.

Travaux pratiques

Examen des adresses virtuelles, physiques et pages de mémoire allouée.

Driver en mode caractères

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

Travaux pratiques

Enregistrement classiques « Unix » vs. utilisation de la classe *misc*.

Méthodes fondamentales : ouverture et fermeture, lecture, écriture, paramétrage.

Travaux pratiques

Écriture d'un driver simple (compteur) proposant *read()*, *write()* et *ioctl()*.

Synchronisation entre appels système : nécessité, mutex.

Travaux pratiques

Nécessité de la synchronisation. Coût temporel d'un mutex.

Communication avec le matériel

Entrées-sorties : ports, projections, GPIO, bus PCIe.

Travaux pratiques

Écriture d'un *driver* pilotant led et bouton poussoir.

Interruptions : contextes, gestionnaires, traitements différés, synchronisation, attentes.

Travaux pratiques

Handlers d'interruption, mécanismes top-half et bottom-haf.

Communication par la mémoire : projections, DMA.

Travaux pratiques

Étude d'un *driver* PCIe utilisant le DMA.

Périphériques réseau

Périphériques réseau : interfaces, protocoles, enregistrements et adressage.

Travaux pratiques

Étude des échanges entre périphérique et pile de protocoles.

Communications réseau : émission et réception de paquets, statistiques d'utilisation.

Travaux pratiques

Écriture d'un *driver* virtuel supportant la communication en IPv4.

Utilisation d'un bus

Le sous-système USB : principe, interface USB de Linux ;

Travaux pratiques

Gestion des identifiants vendor/product, chargement automatique du driver.

Driver en mode *interrupt* : enregistrement du *driver*, *endpoints*, classes, URB.

Travaux pratiques

Écriture d'un *driver* supportant une carte d'entrées-sorties USB.

Aperçu de *drivers* pour autres modes : endpoints *bulk* et *control*..

Conclusion

Discussions libres sur l'ensemble des thèmes abordés.

Travaux pratiques

Expérimentations libres suivant les demandes des participants.