Introduction au développement d'un pilote de périphérique

M2 SETI B4 / MS SE SE758

Guillaume Duc
guillaume.duc@telecom-paris.fr
2020–2021

Plan

Les rôles d'un noyau

- Gérer l'accès aux ressources matérielles
  - Processeur(s) : répartit le temps processeur entre les différentes applications (multitâche)
  - Mémoire : distribue la mémoire aux applications et s'assure de l'isolation entre les processus
  - Périphériques
- Il fournit également une abstraction du matériel pour les applications

Espace utilisateur / Espace noyau

Kernel space (land) / User space (land)

- Deux niveaux d'exécution
  - *Espace noyau* : tous les privilèges sur le matériel ; noyau et pilotes de périphériques
  - *Espace utilisateur* : pas de privilèges, pas d'accès direct au matériel ; bibliothèques et les applications
- Cette séparation permet de forcer les applications à utiliser les services du noyau pour accéder aux ressources et d'isoler les applications du noyau et les applications entres-elles (avec l'aide de la MMU)
- Cette séparation est assurée par le processeur qui fournit différents modes d'exécution (ou anneaux de protection)
- *Question* : Une applications lancée avec les droits super-utilisateur (*root*) s'exécute en espace noyau ou en espace utilisateur ?

Espace utilisateur / Espace noyau

Kernel space (land) / User space (land)

- Le passage de l'espace utilisateur vers l'espace noyau à l'autre se fait principalement grâce aux *interruptions*
- Lorsqu'une interruption survient, le processeur bascule en mode noyau
- Les gestionnaires d'interruption sont intégrés au noyau
- Sources des interruptions
  - Matériel : périphérique signalant la survenue d'un événement (opération terminée, données reçues...), y compris l'horloge (utile pour le multitâche préemptif)
  - Processeur : problèmes avec le programme en cours d'exécution (instruction invalide, faute de page, division par zéro...)
  - Logiciel : *appels système* notamment

Appels systèmes

- Méthode utilisée par les applications pour demander un service au système d'exploitation
- Le noyau fournit plusieurs centaines d'appels systèmes
- Les appels systèmes ne sont en général pas réalisés directement par les applications mais par la bibliothèque standard du C (sous Linux, en général la *GNU C library*)
  - Indépendance par rapport à l'architecture
  - Vérifications

Plan

Quelques rappels

Le développement d'un pilote de périphérique pour Linux

Les particularités du développement noyau

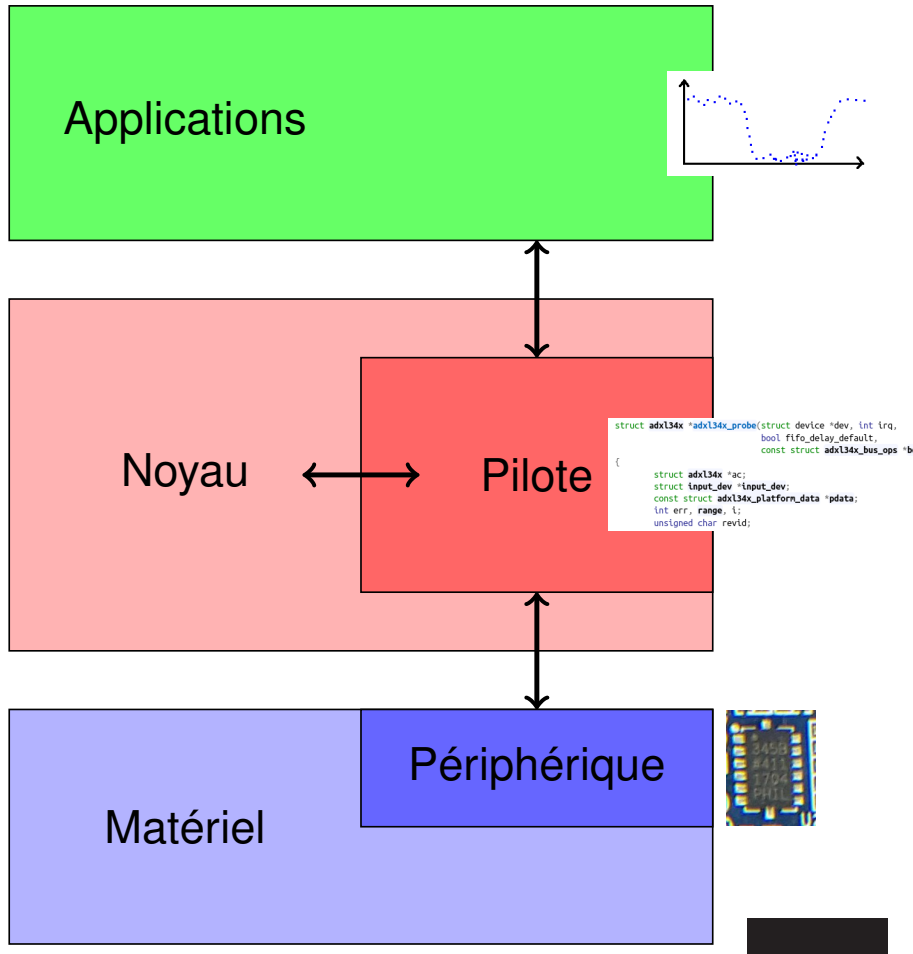
- Programmation en C (avec un peu d'assembleur pour les parties fortement liées au matériel)
- Pas de bibliothèque standard du C (adieu `printf`, `scanf`, `malloc`, `strcpy`...)
- Pas de support des nombres flottants
- Espace noyau : attention à ce que vous faites, risque de failles de sécurité très important
- Pile de taille (très) limitée
- L'API fournie par le noyau peut changer d'une version à l'autre
- Portabilité (attention notamment à l'*endianness*)
- Licence GNU GPL v2 (les modules sont une zone grise)

Les alternatives au développement d'un pilote noyau

- Respect d'un standard ou d'un protocole préétabli pour lequel il existe un pilote générique
  - USB *Human Interface Device* (HID) par exemple pour tous les périphériques d'interface avec l'utilisateur (clavier, souris, joystick, bouton, LED, capteurs simples, etc.)
- Accès au bus ou à la mémoire du périphérique directement depuis l'espace utilisateur
  - Bus USB via *libusb*
  - Bus I^2C via `/dev/i2c-x`
  - Mémoire via `/dev/mem` (non recommandé !)
- *Userspace IO driver* : mécanisme prévu dans le noyau (voir `Documentation/DocBook/uio-howto`)
  - Un tout petit pilote à mettre dans le noyau
  - Le reste peut être réalisé en espace utilisateur

Positionnement de votre pilote

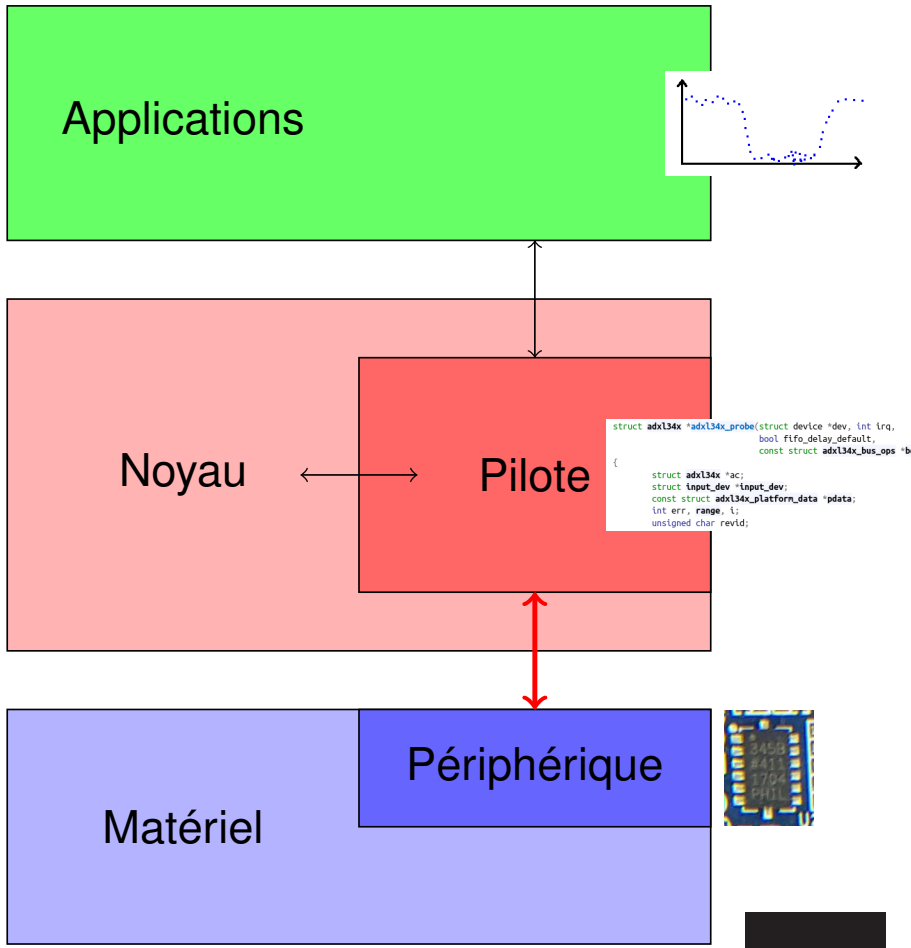
- Un pilote de périphérique sous Linux s'exécute en espace noyau
- Il doit s'interfacer
  - Avec votre périphérique matériel
  - Avec les applications
  - Avec le reste du noyau



Programme de la suite du cours

Interface avec le matériel

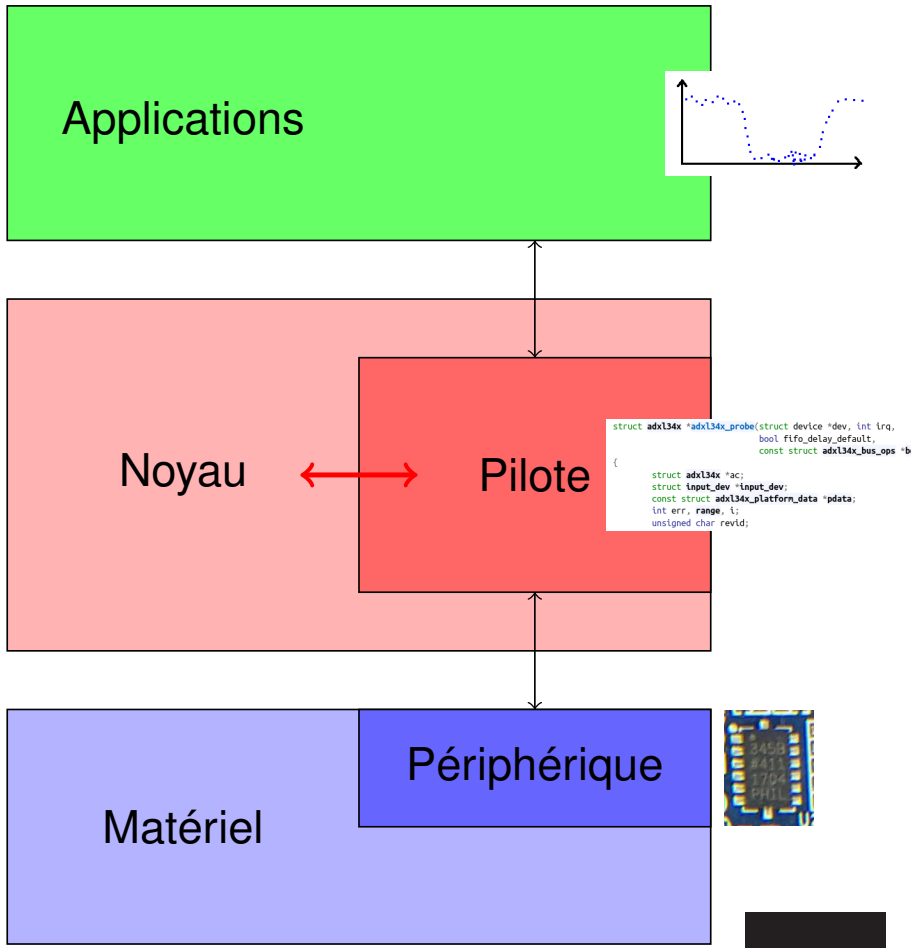
- Modèle de périphérique
  - Découverte des périphérique
  - Communication via un bus (I^2C, USB...)
- Interruptions
- Communication directe vers les périphériques mappés en mémoire



Programme de la suite du cours

Interface avec le noyau

- Notion de module
- Allocation mémoire
- Accès concurrents aux ressources
- Outils de debug



Programme de la suite du cours

Interface avec les applications

- Abstraction du périphérique sous forme d'un fichier spécial
- Notion de framework
- Ordonnancement et attente