Linux Device Drivers

Cours d'introduction

Guillaume Duc
guillaume.duc@telecom-paris.fr
SE758 2019–2020

Objectifs de la semaine

- À la fin de la semaine, vous serez capables d'écrire un pilote de périphérique pour le noyau Linux

Pré-requis

- Langage C
- Développement en C sous GNU/Linux (utilisation de la ligne de commande, gcc, Makefiles)
- Périphériques
  - Registres
  - MMIO/PMIO
  - Interruptions

GNU/Linux

- Un système GNU/Linux est composé :
  - Du noyau Linux proprement dit
  - De bibliothèques regroupant des fonctions sur lesquelles peuvent s'appuyer les applications
  - D'applications
- Le GNU de GNU/Linux vient du projet GNU qui fournit la majeure partie des bibliothèques de base et des applications importantes (*libc*, *gcc*, *emacs*)...

Le noyau Linux

- En 1991, alors étudiant à Helsinki (Finlande), *Linus Torvalds* (âgé de 21 ans), commence à écrire un petit noyau destiné à fonctionner sur un processeur 80386
- Le 25 août 1991, il annonce son projet sur le newsgroup `comp.os.minix`

```
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486)
AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the
file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get
something practical within a few months, and I'd like to know what features most people would want. Any
suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386
task switching etc), and it probably never will support anything other than AT-harddisks, as that's all
I have :-(.
```

Le noyau Linux

- Septembre 1991 : version 0.01 publiée sur un serveur FTP, environ 10.000 lignes de code
- Décembre 1992 : version 0.99 distribuée sous licence GNU GPL
- Grâce à la démocratisation d'Internet dans la début des années 90, Linux fédère une communauté grandissante de développeurs
- Mars 1994 : Version 1.0.0 (175.000 lignes de code)
- Janvier 1999 : Version 2.2.0 (1.800.000 lignes de code)
- Janvier 2001 : Version 2.4.0 (3.400.000 lignes de code)
- Décembre 2003 : Version 2.6.0 (6.000.000 lignes de code)

Le noyau Linux

- Version 5.5 (mars 2020)
  - ~28 millions de ligne de code
  - Environ 14.000 contributeurs
  - Architectures supportées : Alpha, ARC, ARM, ARM64, c6x, H8/300, Hexagon, Itanium, m68k, Microblaze, MIPS, NDS32, Nios II, OpenRISC, PA-RISC, PowerPC, s390, SuperH, SPARC, um (User Mode), Unicore32, x86, Xtensa

Questions

- Quels sont les rôles d'un noyau (et en particulier celui du noyau Linux) ?
- Quels sont les particularités du développement pour le noyau Linux ?

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)
- *Quiz* : 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 : interruption logicielle pour pouvoir faire un *appel 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 C (sous Linux, en général la GNU *libc*)
  - Indépendance par rapport à l'architecture
  - Vérifications

Les particularités du développement noyau

- Programmation en C (avec un peu d'assembleur pour les parties fortement liées au matériel)
- Pas de bibliothèque standard 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 périphérique de communication 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`
- *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

Programme de la semaine

- Explorons le noyau Linux
- Écriture d'un premier module
- Modèle de périphérique (*Device model*)
- Interface avec l'espace utilisateur
- Gestion de la mémoire
- Accès aux périphériques
- Ordonnancement et attente
- Interruptions
- Accès concurrents aux ressources
- Debug noyau

Déroulement de la semaine

- Format
  - Pas de cours magistraux, récapitulatifs des points clés en fin de journée
- Tout le texte est sur eCampus
- Suivez-le à votre rythme, posez-vous des questions, l'objectif est de comprendre (un planning est donné à titre indicatif pour couvrir les parties essentielles)
- N'hésitez pas à solliciter vos camarades si vous ne trouvez pas de réponse à vos interrogations

Pratique

- Les parties pratiques se basent sur les cartes DE1-SoC (une page décrit comment mettre en place tout l'environnement)
- L'objectif va être de développer des pilotes pour plusieurs périphériques de la carte
  - Accéléromètre I^2C
  - Et d'autres si vous avez le temps (notamment UART)
- Vous disposerez tous d'un dépôt git dans lequel vous mettrez les réponses aux questions ainsi que votre code

Évaluation

- L'évaluation de la semaine se basera sur vos réponses, sur votre code (dans le dépôt git) ainsi que sur votre implication
- Petit bonus si vous trouvez des erreurs (technique, style, orthographe) dans le texte...