Linux et le monde de l'embarqué

M2 SETI — B4 Linux Embarqué

Guillaume Duc
guillaume.duc@telecom-paris.fr
2021–2022

Plan

Objectifs pédagogiques

- Être capable de mettre en œuvre un système Linux sur un système embarqué
  - Comprendre le mécanisme de démarrage (depuis le bootloader jusqu'aux applications)
  - Comprendre les différents éléments nécessaires et comment les générer (noyau, arbre des périphériques, système de fichiers initial, système de fichiers racine...)
  - Savoir écrire un pilote de périphérique simple

Pré-requis

- Programmation en langage C (pour l'écriture des pilotes de périphérique)
- Utilisation basique de Linux, de la ligne de commande, des outils de compilation classiques (`gcc`, `make`...)
- Architecture des ordinateurs et des systèmes embarqués (bus, interruptions, DMA, mémoire virtuelle...)
- Utilisation basique de `git`

Emploi du temps 2021-2022

- Mardi 14/12/21 13h30-17h00 : Séance 1
- Mardi 04/01/22 13h30-17h00 : Séance 2
- Mardi 11/01/22 13h30-17h00 : Séance 3
- Mardi 18/01/22 13h30-17h00 : Séance 4
- Mardi 25/01/22 13h30-17h00 : Séance 5
- Mardi 01/02/22 13h30-17h00 : Séance 5
- Mardi 08/02/22 13h30-17h00 : Séance 7
- Mardi 15/02/22 13h30-17h00 : Séance 8
- Mardi 01/03/22 13h30-16h30 : Examen

Travaux pratiques

- Thématiques
  - Compilation du noyau
  - Création d'un système de fichiers (distribution)
  - Création d'un pilote de périphérique...
- Utilisation de `qemu` pour simuler un système embarqué à base de processeur ARM Cortex-A
  - Pas besoin de maquettes de TP (travail à la maison, passage en distanciel...)
  - Plus simple à déboguer

Travaux pratiques

Matériel requis (si ordinateur personnel)

- Ordinateur sous Linux avec une distribution relativement récente
- 8 Go de RAM
- 50 Go de disque libre (sources du noyau, image disque, etc.)
- Outils de développement classiques (`gcc`, `make`, `git`...)
- Les autres logiciels nécessaires (`qemu`, `arm-linux-eabi-gcc`...) seront installés par la suite

Déroulement

- Séance 1
  - Cours Introduction
  - Cours Démarrage d'un système Linux (début)
- Séance 2
  - Cours Démarrage d'un système Linux (fin)
  - TP Noyau, init, initramfs, BusyBox et U-Boot
- Séance 3
  - Cours Système de fichiers racine
  - TP Construire sa distribution complète avec *buildroot*
- Séance 4
  - Cours introduction au développement noyau, premier module, mécanismes de débogage et allocation mémoire
  - TP Écrire votre premier module

Déroulement
Prévisionnel (suite)

- Séance 5
  - Cours Modèle de périphérique
  - TP Pilote de périphérique pour l'accéléromètre (partie communication avec le matériel)
- Séance 6
  - Cours Interface avec l'espace utilisateur
  - Cours Ordonnancement et attente
  - TP Accéléromètre (partie interface avec l'espace utilisateur)
- Séance 7
  - Cours Interruptions
  - TP Accéléromètre (partie interruptions)

Déroulement
Prévisionnel (suite)

- Séance 8
  - Cours Concurrence
  - Cours Accès aux périphériques mappés en mémoire
  - TP Accéléromètre (finalisation)

Modalités d'évaluation

- Note finale
  - 25 % rendus des TP sur le pilote de périphérique pour l'accéléromètre
  - 75 % examen final sur table
    - Questions type QCM et/ou questions ouvertes
    - Seul document autorisé : une page A4 avec ce que vous voulez inscrit dessus
    - Tout autre document, support de cours, ordinateur, téléphone, calculatrice... interdit
    - Pour l'instant, les conditions sont réunies pour effectuer cet examen en présentiel. Néanmoins, si la situation l'exigeait, cet examen pourrait avoir lieu à distance avec des modalités adaptées

Dernières remarques

- Cours repris depuis l'an dernier (assuré par un autre enseignant les années précédentes) et remonté à partir de zéro
- N'hésitez pas à signaler toute erreur que vous trouverez dans le cours, les textes des TP, etc.
- Votre retour sera précieux pour améliorer ce cours pour les prochaines années
- N'hésitez pas à poser des questions pendant le cours et les TP !

Plan

Le monde de l'embarqué

Caractéristiques très variées

- Cœur : d'un petit microcontrôleur 8 bits à plusieurs processeurs multi-cœurs/multi-threads 64 bits
- RAM : de quelques ko à plusieurs Go
- Stockage de masse : quelques ko de ROM à plusieurs Go/To de flash ou disque dur
- Protection mémoire : rien, MPU (*Memory Protection Unit*), MMU (*Memory Management Unit*)
- Contraintes : coût, consommation, temps réel, sûreté, sécurité, etc.
- Nous verrons par la suite les caractéristiques minimales pour faire tourner Linux

Le rôle d'un système d'exploitation (OS)

- Partage, de façon sécurisée, les ressources du support d'exécution entre les différentes tâches
  - Processeur(s) : répartit le temps processeur entre les différentes tâches souhaitant s'exécuter (avec éventuellement gestion des priorités, préemption, etc.)
  - Mémoire : distribue la mémoire aux tâches et s'assure de l'isolation entre celles-ci (en se basant sur du matériel : MPU, MMU)
  - Périphériques
- Fournit
  - Une abstraction du matériel pour les tâches
  - Des mécanismes de communication et de synchronisation entre les tâches

Les systèmes d'exploitation dans l'embarqué
Classification

- Licence et accès au code source : Propriétaires / *open-source*
- Généralistes / dédiés
- Autres critères spécifiques
  - Temps réel
  - Certification sûreté ou sécurité

Les systèmes d'exploitation dans l'embarqué
Exemples

- Généralistes : Linux, *BSD (FreeBSD, NetBSD...), Windows...
- Dédiés
  - Windows IoT (ex. Windows Embedded, propriétaire), QNX (propriétaire), iOS (propriétaire), Android (*open source*)...
  - Temps réel : VxWorks (propriétaire), FreeRTOS (*open source*), ChibiOS/RT (*open source*), INTEGRITY (propriétaire, classifié), RTX (propriétaire)...

Linux dans le monde de l'embarqué

- D'après [1], 65 % des systèmes embarqués utilisent un système d'exploitation
  - 42 % un système *open-source* sans support commercial
  - 24 % un système commercial (en baisse ces dernières années)
  - 19 % un système développé maison
  - 16 % une distribution commerciale d'un système *open-source*

Linux dans le monde de l'embarqué (suite)

- D'après [1], systèmes d'exploitation utilisés dans l'embarqué :
  - **Embedded Linux : 21 %**
  - Maison : 19 %
  - FreeRTOS : 18 %
  - **Ubuntu : 14 %**
  - **Debian : 13 %**
  - **Android : 13 %**
  - Windows 10 : 10 %
    - *Réponses multiples autorisées, donc somme > 100 %*
- La part de marché de Linux (toutes distributions confondues) est donc très importante dans le monde de l'embarqué

Pourquoi choisir Linux ?

- Licence libre et code source ouvert
- Support de nombreuses architectures
- Nombreux pilotes de périphériques
- Écosystème dynamique
- Viabilité à long terme

Pourquoi ne pas choisir Linux ?

- Licence libre et code source ouvert
- Absence de support commercial
- Responsabilité en cas de problème
- Écosystème foisonnant
- Pas adapté à tous les systèmes embarqués (voir caractéristiques requises plus loin)

Plan

Licences libres

Droits et obligations

- Point important à vérifier, notamment si vous distribuez ou vendez un système embarqué contenant du logiciel libre
- Certaines licences imposent des contraintes au niveau de l'accès au code source, des brevets ou de la possibilité de mettre à jour le logiciel
- Principales licences utilisées
  - Licences *Copyleft* (*gauche d'auteur*)
    - GNU General Public License (GPL) v2
    - GNU General Public License (GPL) v3
    - GNU Lesser General Public License (LGPL)
  - Licences permissives
    - BSD, MIT, Apache

GNU General Public License (GPL) v2 [3]

- Utilisée par le noyau Linux, de nombreux outils (`busybox` par exemple) et quelques bibliothèques (GNU `Readline` !)
- Droits (pour l'utilisateur)
  - Utiliser le logiciel
  - Étudier son fonctionnement
  - Copier et distribuer le logiciel
  - Modifier le logiciel et distribuer ces modifications

GNU General Public License (GPL) v2 [3]

- Obligations (lors de la distribution du logiciel)
  - Conserver les notices de copyright
  - Donner accès au code source
  - Les modifications apportées au logiciel doivent être distribuées sous les mêmes termes (et donc distribuer le code source des modifications)
- Le lien avec une bibliothèque statique ou dynamique sous GPL est considéré comme un travail dérivé et donc impose une publication du logiciel complet sous licence GPL
- Un module compilé en dehors de l'arbre des sources du noyau peut être distribué sous une autre licence (mais il n'aura pas accès à certaines fonctionnalités du noyau, et le noyau sera marqué comme *tainted*)

GNU General Public License (GPL) v3 [4]

- Principaux ajouts à la v2
  - En cas de distribution matériel + logiciel, obligation de fournir les informations nécessaires pour que l'utilisateur puisse faire tourner une version modifiée du logiciel (*anti tivoïsation*)
  - En cas de distribution du logiciel, fourniture d'une licence sur les brevets nécessaires à l'utilisateur pour exercer ses droits
  - En cas d'attaque pour violation de brevet, l'attaquant perd les droits accordés par la licence

GNU Lesser General Public License (LGPL) [5]

- Similaire à la GPL, spécialement dédiée aux bibliothèques
- Facilite l'utilisation d'une bibliothèque LGPL par une application non GPL/LGPL
  - En cas de liaison statique, il faut permettre à l'utilisateur de modifier la bibliothèque LGPL et de la lier de nouveau avec votre application (donc fournir votre application sous forme objet par exemple)
  - En cas de liaison dynamique, si la bibliothèque LGPL est déjà présente sur l'ordinateur de l'utilisateur, il n'y a pas besoin de distribuer le code de celle-ci. Si la bibliothèque LGPL est distribuée avec l'application, il faut fournir le code source de la bibliothèque

Licences permissives (BSD, MIT, Apache...)

- Contrairement aux licences *copyleft*, n'imposent pas la distribution des travaux dérivés sous la même licence
- En général, en cas de redistribution, elles obligent simplement à mentionner l'usage du composant, sa licence et ses auteurs, et l'absence de garantie

Licences libres
Conclusion

- Faire très attention aux licences : si vous avez un doute, demander à un juriste spécialisé
- Point important : la quasi totalité des licences libres mentionnent explicitement l'absence de garanties sur le fonctionnement du composant logiciel
  - Vous ne pourrez pas vous retourner contre les auteurs d'un tel composant logiciel si un bug dans ce dernier a causé une défaillance critique de votre système
  - La légalité de cette exclusion est débattue dans certains pays

Plan

Le noyau Linux

- *Stricto sensu*, Linux désigne juste le noyau
- C'est un système d'exploitation généralise, destiné à s'exécuter sur un très grand nombre d'architectures et de systèmes (embarqué, ordinateur personnel, serveur, super-calculateur)

Le noyau Linux

Bref historique

- 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 protabl
(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

Bref historique

- 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

Bref historique

- Juillet 2011 : Version 3.0 (abandon de la numérotation 2.x avec x pair pour la branche stable et impair pour la branche de développement)
- Avril 2015 : Version 4.0
- Mars 2019 : Version 5.0
- Décembre 2021 : Version 5.15
  - ~29 millions de ligne de code
  - Environ 14.000 contributeurs

Le noyau Linux

Matériel nécessaire minimal

- Processeur 32 bits ou plus (voir *Embeddable Linux Kernel Subset* [2] pour un support minimal des processeurs 16 bits)
  - Architectures supportées (v5.9) : DEC Alpha, ARC, ARM, ARM 64, C6x, C-SKY, Renesas H8, Qualcomm Hexagon, Intel IA-64, Freescale 68k, Xilinx Microblaze, MIPS, Andes NDS32, Altera Nios II, OpenRISC, HP PA-RISC, PowerPC, RISC-V, IBM System/390, SuperH, SPARC, x86, x86-64, Tensilica Xtensa
- *Memory Management Unit (MMU)* (voir µClinux pour les systèmes sans MMU)
- Plusieurs Mo de RAM (le minimum dépend de la configuration du noyau, aux alentours de 4–8 Mo, mais il faut ensuite compter les exécutables)

Le noyau Linux

- Il n'est pas temps réel
  - Patch `PREEMPT_RT` pour rendre le noyau capable de respecter des contraintes temps-réel dur
  - RTLinux : micro-noyau temps réel faisant tourner Linux comme une de ses tâches

Les composants logiciels minimums d'un système Linux

- Le noyau Linux a besoin d'un certain nombre de composants logiciels supplémentaires (il ne peut pas faire grand chose seul)
- Un mécanisme pour lui permettre d'identifier le matériel présent (processeur, mémoire, périphériques...)
  - Arbre des périphériques (*device tree*) ou structure de données passée par le *bootloader* (ATAG...)
  - Description statique de la plate-forme compilée avec le noyau
  - Mécanisme automatique pour identifier les périphériques (tables ACPI, énumération des périphériques USB...)

Les composants logiciels minimums d'un système Linux (suite)

- Un chargeur d'amorçage (*bootloader*)
  - Initialisation de base de la plate-forme (arbre d'horloge, caches, contrôleur de DRAM...)
  - Chargement du noyau depuis un support de stockage de masse (disque dur, mémoire flash...)
  - Mise en place en mémoire des structures complémentaires pour le démarrage du noyau (arbre des périphériques, disque mémoire initial (*initrd*...))

Les composants logiciels minimums d'un système Linux (suite)

- Un système de fichiers racine
  - Soit un système minimal en RAM mis en place à partir d'une image fournie au noyau lors du démarrage (`initrd` ou `initramfs`)
  - Soit un système de fichiers plus complet monté depuis un support de stockage de masse ou depuis le réseau
  - Remarque : dans beaucoup de systèmes les deux méthodes ci-dessus sont utilisées l'une après l'autre (sera détaillé dans la suite)

Les composants logiciels minimums d'un système Linux (suite)

- Un processus `init` (lancé depuis `/sbin/init`)
  - Premier processus exécuté par le noyau
  - Ancêtre de tout les autres processus
  - Une fois ce processus lancé, la phase de démarrage du noyau est terminée
  - C'est ce processus qui va être en charge de la suite du démarrage du système

Les autres composants usuels

- Fichiers de configuration et exécutables pour le démarrage (`SysV init` ou `systemd`)
- Bibliothèques partagées (au minimum `libc`)
- Shell et utilitaires classiques (`sh`, `mount`, `test`...)
  - La plupart des bibliothèques et des outils essentiels proviennent du projet GNU (*GNU's Not Unix*), d'où le nom GNU/Linux
  - C'est moins vrai dans le domaine de l'embarqué (Newlib, BusyBox...)
- Démons (`udev`, `sshd`...)
- Une console (clavier/écran, série)

Construction d'un système Linux complet

- Utilisation d'une distribution Linux (générique ou spécialisée)
  - Debian
  - Ubuntu
  - Raspberry Pi OS (anciennement Raspbian), basée sur Debian
  - ...
- Avantages : tout prêt, maintenance (sécurité) sur le long terme
- Inconvénients : difficile à adapter pour une situation particulière, difficile à optimiser

Construction d'un système Linux complet (suite...)

- Générateur de distribution Linux
  - Yocto Project : `https://www.yoctoproject.org/`
  - OpenEmbedded : `http://www.openembedded.org`
  - Buildroot : `https://buildroot.org/`
- Avantages : flexibilité et optimisation accrues
- Inconvénients : plus difficile à mettre en place qu'une distribution toute prête, temps de compilation

Construction d'un système Linux complet (suite...)

- Création des différents éléments à la main
- Avantages : flexibilité totale, compréhension totale du système
- Inconvénients : très long à mettre en place, problèmes de dépendance entre outils ou versions, courbe d'apprentissage difficile

Plan

Références I

[1] Aspencore.
2019 embedded markets study.
https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf, March 2019.

[2] Embeddable linux kernel subset.
https://github.com/jbruchon/elks, 2020.

[3] GNU.
General Public License, version 2.
https://www.gnu.org/licenses/old-licenses/gpl-2.0.html, June 1991.

[4] GNU.
General public license, version 3.
https://www.gnu.org/licenses/gpl-3.0.html, June 2007.

[5] GNU.
Lesser general public license, version 3.
https://www.gnu.org/licenses/lgpl-3.0.html, June 2007.