



Université Paris 8

Rapport du projet

"Construction d'un noyau Linux pour l'embarqué"

Réalisé par :

- Rajmagan BALAKICHENIN
- Kevin TOLEON
- Nouredine DJERROUD

Année universitaire: 2018-2019

Table des matières

I.Introduction.....	1
II. Étapes de construction du projet.....	1
II.1 Acquisition des outils :.....	1
II.2 Ajout d'un module	1
II.3 Manières de configurer le noyau.....	2
II.4 Buildroot	2
II.5 Configuration noyau	2
III. Partitionnement de la carte SD.....	3
IV. Problème survenue	4
V. Travail non effectué :	4

I. Introduction

Notre projet consiste à pactiser un module de notre choix dans une distribution buildroot. Le choix du module et de la distribution buildroot est un n'est pas imposé, c'est à nous de choisir. Le projet doit est opération sur une Raspberry Pi3 dès son démarrage. La date limite du projet est pour le 06/12/2018 à 7 H. Dans ce rapport, nous allons expliquer les étapes du projet, de l'installation d'un module jusqu'au démarrage du projet sur la raspberry Pi3. Mais aussi détailler sur les problèmes rencontrés, et pourquoi il y a eu ces problèmes.

II. Étapes de construction du projet

II.1 Acquisition des outils :

Pour réaliser ce projet nous devons tous d'abord télécharger un buildroot. Un buildroot est un outil qui permet d'automatiser la construction complète d'un système Linux embarqué. Nous avons donc téléchargé un buildroot qui contenait déjà tous les fichiers de base dans les dossiers boot, kernel, ... Pour utiliser le buildroot, il faut télécharger les outils et les dépendances qui le permettent de l'utiliser comme les dépendances de 'make', 'make config', que nous verrons plus tard.

Les commandes utilisées sont :

- apt-get install make
- apt-get install bison puis apt-get install flex. C'est deux dépendances sont utilisées pour la configuration du noyau.
- apt-get install libncurses5-dev. Pour utiliser make menuconfig que nous verrons aussi plus tard.
- Nous devons aussi acquérir une Raspberry Pi3 avec une carte SD, une Raspberry est un micro-ordinateur fonctionnant sur un processeur ARM.

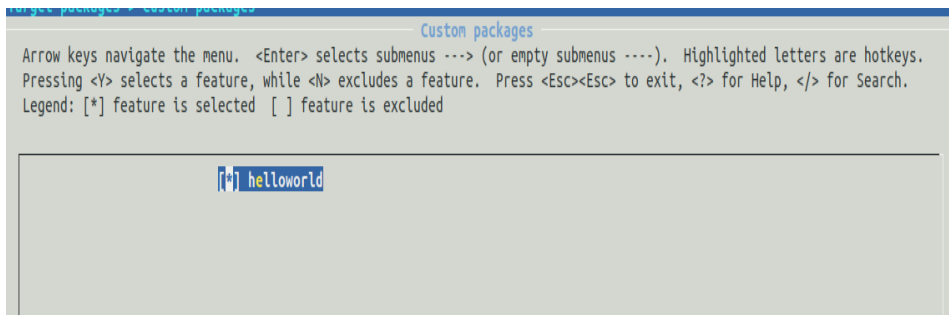
II.2 Ajout d'un module

Notre module sera téléchargé sur internet, comme notre but était simplement de l'implémenter sur une Raspberry. Notre module sera un programme qui contiendra uniquement un hello world [1]. Nous avons installé le module dossier helloworld que nous avons créé dans le dossier package. Dans ce dossier packagé que contiennent tous les modules que nous souhaitons. Pour connaître le chemin de nos modules, il faut les indiquer dans ce fichier: package/config.in, comme ceci :

```
menu "Custom packages"
    source "package/helloworld/Config.in"
endmenu
```

Pour vérifier que les modules soient reconnus lors du noyau, il faut l'activer dans la configuration du noyau.

Donc il faut faire : `make menu config`, aller dans `target` et activer les modules.



L'image ci-dessus prouve que notre module `helloworld` s'est bien attaché à notre noyau.

II.3 Manières de configurer le noyau

Pour pouvoir utiliser notre `buildroot` afin de la booter sur notre Raspberry Pi3, nous avons choisi de la configurer par défaut avec cette commande :

`make raspberrypi_defconfig`. “_Defconfig” qui permet de laisser la configuration par défaut de l'ensemble des `makefiles`. Ensuite il faut configurer notre noyau, pour cela nous avons utilisé : `make menu config`. Il y a plusieurs commandes pour configurer notre noyau, `make config`, `make menu config`, `make config`;

Nous avons choisi la commande : `make menuconfig`, car nous voulions activer une option du noyau. De plus, `make menuconfig`, nous offre une interface graphique, qui rend plus agréable les configurations. Mais surtout elle permettait d'activer le module `helloworld` installé mentionné plutôt.

II.4 Buildroot

`Buildroot` est techniquement un ensemble de `Makefiles` définissant, en fonction des options paramétrées par l'utilisateur, la manière de compiler chaque paquet sélectionné avec des options particulières. Il construit finalement une distribution complète et cohérente dont chaque composant a été compilé. Il possède un outil confortable de configuration, basé et très similaire à celui du noyau Linux: `menuconfig` et qui peut être utilisé dans tout projet.

II.5 Configuration noyau

Tout ceci sera compilé avec une cross-compilation avec la configuration adaptée à notre système. Grâce à une `toolchain` qui est un ensemble de paquet utilisé lors d'une compilation du noyau que nous avons configuré au préalable avec le `make menuconfig`, cela a pour effet de créer un fichier `.config` dans le répertoire actuel. Ce fichier répertorie la configuration de la cible. Il peut être transmis à un tiers qui pourra ainsi construire une `toolchain` identique. La compilation est lancée avec la commande `build`.

L'ajout d'un module se fait avant la configuration du noyau. Il peut être chargé dynamiquement sans avoir besoin de recompiler le noyau (avec la commande `insmod` ou `modprobe`) ou de redémarrer le système.

Une fois que tout est configuré, l'outil de configuration génère un fichier `.config` qui contient l'ensemble de la configuration. Effectuer un `make` démarrera la compilation

La commande make effectue généralement les étapes suivantes :

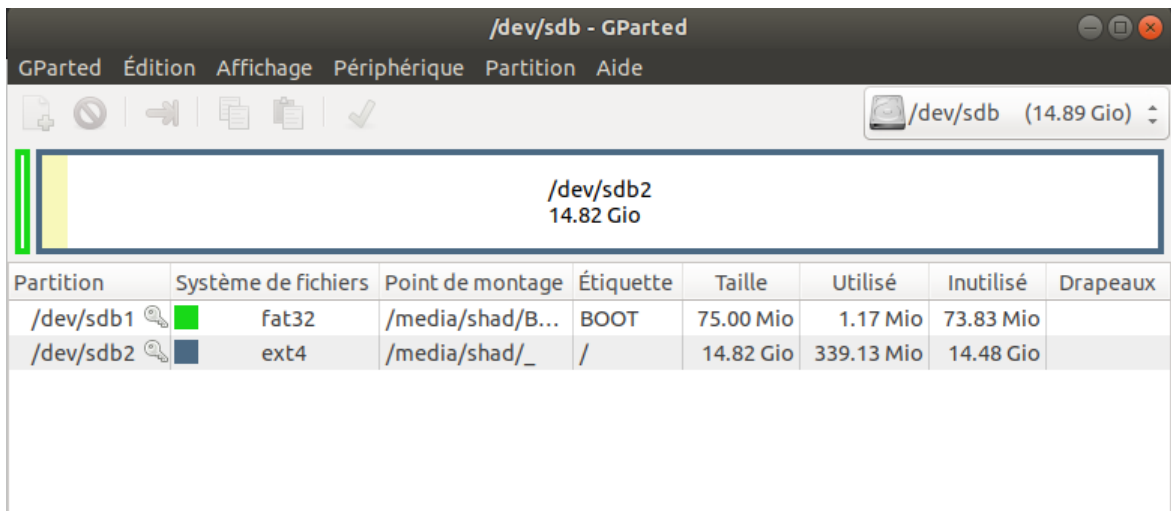
- téléchargement des fichiers sources
- configuration de la cross-compilation
- correction et installation des paquets cibles
- construction d'une image du noyau
- construction d'une image du bootloader
- création d'un système de fichier racine nommé

Le résultat de la construction est stocké dans un répertoire qui contiendra plusieurs sous répertoires :

- images/ : toutes les images (bootloader, noyau, système de fichier racine)
- build/ : tous les composants sont construits ici avec leurs sous répertoires respectifs.
- Staging/ : contient la hiérarchie des fichiers système de la racine : les entêtes et les bibliothèques de la chaîne de compilation.
- target/ : contient le système de fichiers rootfs
- host/ : contient l'installation des outils compilés pour l'hôte qui sont nécessaires pour la bonne exécution de Buildroot, y compris la chaîne d'outils de compilation croisée

III. Partitionnement de la carte SD

Nous avons partitionné la carte comme ceci en respectant l'espace conseillé d'indiquer sur les tutoriels du web. La partition fat32 devait recevoir les fichiers ZImage qui sont l'image du noyau sur lequel la Raspberry devra booter et recevoir aussi le dossier rpi-firmware qui contient tous les fichiers nécessaires au boot de la carte. La partition ext4 contient les dossiers rootfs qui contiennent la racine et tous les fichiers de bases.



Partition	Système de fichiers	Point de montage	Étiquette	Taille	Utilisé	Inutilisé	Drapeaux
/dev/sdb1	fat32	/media/shad/B...	BOOT	75.00 Mio	1.17 Mio	73.83 Mio	
/dev/sdb2	ext4	/media/shad/_/	/	14.82 Gio	339.13 Mio	14.48 Gio	

IV. Problème survenue

Les problèmes rencontrés :

- le noyau ses greffés à coté de mon noyau. Nous avons téléchargé toutes les dépendances nécessaires.

```
tree-vec-data-refs.c(.text+0xc95e) : référence indéfinie vers « alt_dump_file »
tree-vec-data-refs.c(.text+0xca06) : référence indéfinie vers « alt_dump_file »
tree-vec-data-refs.o:tree-vec-data-refs.c(.text+0xca19) : encore plus de références indéfinies suivent vers « alt_dump_file »
tree-vec-data-refs.o : Dans la fonction « vect_enhance_data_refs_alignment(_loop_vec_info*) » :
tree-vec-data-refs.c(.text+0xca93) : référence indéfinie vers « dump_file »
tree-vec-data-refs.c(.text+0xcaad) : référence indéfinie vers « dump_printf_loc(int, unsigned int, char const*, ...) »
tree-vec-data-refs.c(.text+0xcb03) : référence indéfinie vers « alt_dump_file »
tree-vec-data-refs.o : Dans la fonction « vect_get_data_access_cost(data_reference*, unsigned int*, unsigned int*, vec_stmt_info_for
201_cost, va_heap, vl_ptr*) » :
tree-vec-data-refs.c(.text+0x618) : référence indéfinie vers « dump_printf_loc(int, unsigned int, char const*, ...) »
./collect2: error: ld returned 1 exit status
/gc../gcc/c/Make-lang.in:85: recipe for target 'cc1' failed
make[3]: *** [cc1] Error 1
make[3] : on quitte le répertoire « /home/rajubu/Téléchargements/buildroot-2018.11/output/build/host-gcc-final-7.3.0/build/gcc »
GCCMakefile:4208: recipe for target 'all-gcc' failed
make[2]: *** [all-gcc] Error 2
make[2] : on quitte le répertoire « /home/rajubu/Téléchargements/buildroot-2018.11/output/build/host-gcc-final-7.3.0/build »
Makefile:875: recipe for target 'all' failed
make[1]: *** [all] Error 2
make[1] : on quitte le répertoire « /home/rajubu/Téléchargements/buildroot-2018.11/output/build/host-gcc-final-7.3.0/build »
package/pkg-generic.mk:229: recipe for target '/home/rajubu/Téléchargements/buildroot-2018.11/output/build/host-gcc-final-7.3.0/.stamp_built' failed
./p_built' failed
make: *** [/home/rajubu/Téléchargements/buildroot-2018.11/output/build/host-gcc-final-7.3.0/.stamp_built] Error 2
rajubu@rajubu-VirtualBox:~/Téléchargements/buildroot-2018.11$ ls
```

- Nous avons rencontré des problèmes avec la compilation du noyau les chemins spécifier aboutissent à des erreurs de compilation car certaines références sont indéfinies ce qui produit des erreurs comme Dump-files.
- Nous avons pensé qu'il fallait mettre à jours les dépendances cependant le problème n'est pas lié à la version.

Nous pensons que le problème de compilation est du à cause du Makefile, les erreurs reportées dans la compilation viennent des lignes du Makefile. Dans ce fichier, il est précisé qu'il faut avoir une version gcc supérieur ou égale à 5, Or notre version du gcc est à 5.

Certaines références ne sont pas définies, ce qui est utile pour compiler certain fichier.c. Le fichier treevec-data-refs.c a un problème de lien vers une autre ressource.

V. Travail non effectué :

- Inclusion des pilotes des périphériques
- boot sur Raspberry PI3

Références Bibliographiques

<http://langevin.univ-tln.fr/CDE/LEXYACC/Lex-Yacc.html>

<http://www.linuxembedded.fr/2011/03/ajouter-un-package-dans-buildroot-en-5-minutes/>

<http://silanus.fr/sin/wp-content/uploads/2016/03/TP3.pdf>

<https://buildroot.org/downloads/manual/manual.html>

<https://www.blaess.fr/christophe/2014/07/22/ajouter-un-module-noyau-personnel-dans-buildroot/>