CSEL

Rapports des TP

Fribourg, April 6, 2022

Auteurs
Macherel Rémy
remy.macherel@master.hes-so.ch
Raemy Mathis
mathis.raemy@master.hes-so.ch

Version: 1.0

Contents

1 TP 1 **1**
- 1.1 Résumé du travail pratique 1 . . . 1
- 1.2 Réponses aux questions . . . 1
  - 1.2.1 Comment faut-il procéder pour générer le U-Boot ? . . . 1
  - 1.2.2 Comment peut-on ajouter et générer un package supplémentaire dans le Buildroot ? . . . 1
  - 1.2.3 Comment doit-on procéder pour modifier la configuration du noyau Linux ? . . . 1
  - 1.2.4 Comment faut-il faire pour générer son propre rootfs ? . . . 2
  - 1.2.5 Comment faudrait-il procéder pour utiliser la carte eMMC en lieu et place de la carte SD ? . . . 2
  - 1.2.6 Dans le support de cours, on trouve différentes configurations de l'environnement de développement. Qu'elle serait la configuration optimale pour le développement uniquement d'applications en espace utilisateur ? . . . 2
- 1.3 Synthèse des connaissances acquises . . . 3
  - 1.3.1 Non acquis . . . 3
  - 1.3.2 Acquis, mais à exercer . . . 3
  - 1.3.3 Parfaitement acquis . . . 3
- 1.4 Infos utiles à retenir . . . 3
- 1.5 Feedback global . . . 3
  - 1.5.1 Problèmes rencontrés . . . 3

2 TP 2 **4**
- 2.1 Réponses aux questions . . . 4
  - 2.1.1 Trouvez la signification des 4 valeurs affichées lorsque l'on tape la commande cat /proc/sys/kernel/printk . . . 4
- 2.2 Feedback global . . . 5

3 TP 3 **6**
- 3.1 Résumé du TP3 . . . 6

Chapter 1

TP 1

1.1 Résumé du travail pratique 1

Dans ce TP, nous allons mettre en place un environnement de développement afin de faciliter les développements futures. Cet environnement utilise des container Docker (Docker-compose) pour assurer la compatibilité entre les différentes machines. Un container est composé de tous les outils nécessaire à la cross-compilation et l'autre pour la mise en place d'une partition réseau SAMBA afin de partager facilement les programmes cross-compilés avec le NanoPi.

1.2 Réponses aux questions

1.2.1 Comment faut-il procéder pour générer le U-Boot ?

On peut regénérer le u-boot uniquement avec la commande : make uboot-rebuild
See : https://github.com/ARM-software/u-boot/blob/master/Makefile
–> TODO : Ajouter 5-6 lignes qui expliquent ce que fais le make de U-Boot

1.2.2 Comment peut-on ajouter et générer un package supplémentaire dans le Buildroot ?

e ne comprends pas trop le sens de la question mais au sens général, il faut ajouter le binaire ainsi que des fichiers de configurations (qui définissent par exemple les dépendances). On peut par exemple importer depuis GitLab automatiquement un packet, ou encore ajouter des paquet pour le user space ou le kernel space et tout cela change les étapes a effectuer. Pour plus d'informations, la section 18 de la documentation.

1.2.3 Comment doit-on procéder pour modifier la configuration du noyau Linux ?

La commande make linux-menuconfig permet de configurer le noyau et on peut ensuite ajouter/enlever des modules du noyaux et finalement faire make linux-rebuild afin de reconstruire le noyau.

1.2.4 Comment faut-il faire pour générer son propre rootfs ?

Comme c'est Buildroot qui s'occupe de générer le Rootfs, c'est lui qu'il faut configurer
cd /buildroot
make menuconfig
On peut aussi par exemple modifier le skelette utilisé pour modifier la structure des dossiers du rootfs. On peut aussi ajouter des fichiers/scripts/dossiers par défaut dans le rootfs en ajoutant un rootfs_overlay */buildroot/board/friendlyarm/nanopi-neo-plus2/rootfs_overlay/*. Ils seront automatiquement inséré dans le rootfs après chaques générations.

1.2.5 Comment faudrait-il procéder pour utiliser la carte eMMC en lieu et place de la carte SD ?

Il faut commencer par modifier le script de démarrage en modifiant les deux lignes suivantes. Il faut remplacer le **0** par le numéro de MMC que l'on veut :
fatload mmc **0** $kernel_addr_r Image
fatload mmc **0** $fdt_addr_r nanopi-neo-plus2.dtb
On peut avoir des informations sur les mmc disponible en interrompant le U-Boot et en tappant la commande :

```
=> mmc list mmc@1c0f000: 0 (SD) mmc@1c10000: 2 mmc@1c11000: 1
```

Il faut ensuite recompiler le script avec la commande make et reflasher la carte. Enfin il ne faut pas oublier également de charger l'Image et le device tree depuis la carte SD vers la mmc interne (a faire une seul fois) ainsi que de modifier le script de démarrage pour utiliser celui que l'on vient de compiler avec la commande :

```
setenv boot_scripts boot.cifs
```

Finalement, on peut sauvegarder les modifications avec la commande :

```
saveenv
```

1.2.6 Dans le support de cours, on trouve différentes configurations de l'environnement de développement. Qu'elle serait la configuration optimale pour le développement uniquement d'applications en espace utilisateur ?

Si on développe uniquement dans le userspace, on pourrait imaginer que l'on flash la eMMC comme décrit à la question précédente pour s'affranchir de carte SD et qu'on développe sur notre ordinateur. Pour le partage des programmes une partition NFS ou la partition CIFS sont très bien et permettent de rendre transparent le transfert des programmes.

1.3 Synthèse des connaissances acquises

1.3.1 Non acquis

Aucune

1.3.2 Acquis, mais à exercer

Les Makefile, debugging (GDB/HDB Server), Principe CIFS

1.3.3 Parfaitement acquis

Le reste comme par exemple ce qui est commande, connexion, Linux (déjà globalement vu dans un cours précédent)

1.4 Infos utiles à retenir

Pour monter automatiquement un disque réseau cifs :

```
1  echo "//<server_ip>/workspace /workspace cifs username=<username>,
2  password=<password>,port=1445,noserverino" >> /etc/fstab
```

Lorsqu'on créer une nouvelle partition, il faut l'aligner sur 2048 bits. On peut calculer le début de la prochaine partition par rapport à la précédente comme ceci :
<prev_end_sector> + (2048 - <prev_end_sector> % 2048)

1.5 Feedback global

Dans l'ensemble laboratoire très intéressant et bon point de départ pour un cours comme celui-ci. Assez dirigé pour nous indiquer la bonne voie mais pas trop pour qu'on puisse tester des choses. Nous aurions bien souhaité un peu plus de maitère sur les Makefiles.
Nous avons également pour le flash de la carte SD développé un script permettant ceci de manière automatique (à voir dans le dossier script le FlashSDCard.sh). Ce script permet donc le flash de la SD Card avec les fichiers nécessaires sans utiliser le logiciel *Balena*. Afin de l'utiliser, il suffirait de modifier le chemin d'accès à l'image (ligne 2 du script). Il a également été très intéressant de découvrir les differentes méthodes de debugging possibles lors du développement de systèmes embarqués. Parmis celles vues en cours, nous avons particulièrement testé la méthode utilisant GDB / GDB Server. La capacité de cette méthode à réaliser du débugging à distance était pour nous un point fort. Cette méthode permettait entre autres d'exécuter le code présent sur la cible pas à pas et de découvrir de potentielles erreurs.

1.5.1 Problèmes rencontrés

Pour un des membres du groupe, une fois le CIFS mis en place, le nanoPi était incapable de démarrer et semblait relancer sa séquence de boot indéfiniment. Nous avons par la suite déterminé que ceci était dû à une trop faible alimentation via un HUB USB qui faisait que lorsque le nanoPi tentait de se connecter via le réseau la tension n'était pas suffisante et celui-ci s'éteignait puis rebootait directement.

Chapter 2

TP 2

2.1 Réponses aux questions

2.1.1 Trouvez la signification des 4 valeurs affichées lorsque l'on tape la commande cat /proc/sys/kernel/printk

Cela affiche les console_loglevel (niveaux de logs de la console) actuels, dans notre cas sur le nanopi les valeurs affichées sont 7 4 1 7, comme le montre la documentation l'ordre des chiffres correspond à :

1. current
2. default
3. minimum
4. boot-time default

Name	String	Alias function
KERN_EMERG	"0"	`pr_emerg()`
KERN_ALERT	"1"	`pr_alert()`
KERN_CRIT	"2"	`pr_crit()`
KERN_ERR	"3"	`pr_err()`
KERN_WARNING	"4"	`pr_warn()`
KERN_NOTICE	"5"	`pr_notice()`
KERN_INFO	"6"	`pr_info()`
KERN_DEBUG	"7"	`pr_debug()` and `pr_devel()` if DEBUG is defined
KERN_DEFAULT	""	
KERN_CONT	"c"	`pr_cont()`

Figure 2.1: Console log levels correspondance

2.2 Feedback global

Nous avons initialement dans l'exercice 7 peiné à utiliser les fonctions wait_event_interruptible et wake_up_interruptible. Nous avions tout d'abord pensé qu'il n'était pas nécessaire d'utiliser un verrou atomique car nous pensions que la fonction wake_up_interruptible générait le signal permettant de réveiller le wait, cependant il s'est avéré que ce n'était pas le cas et nous avons donc dû (comme dans la solution fournie) implémenter un verrou permettant de bloquer et de signaler au wait que celui-ci doit se réveiller et afficher son message. Nous avons donc conclus que la queue était présente pour que le thread se mette en attente si la condition est fausse. Dans notre cas, comme au début nous avions mis une condition à true en permanence cela tournait dans une boucle infinie car le fait que la condition soit vraie lui permettait de toujours passer par dessus la queue. Le wake_up sert donc juste à indiquer au wait qu'il doit re-vérifier sa condition. Ces informations furent découvertes en faisant des tests ainsi qu'en discutant lors de la séance de laboratoire.

Chapter 3

TP 3

3.1 Résumé du TP3

Salut