TP1

Objectif :

Obtenir une chaîne de compilation croisée basée sur la librairie uclibc-ng pour la cible Atmel SAMA5D3 Xplained, puis l'utiliser pour compiler un programme C.

Q1 : Quelle est l'architecture processeur de la carte SAMA5D3 Xplained ?
→ l'architecture de la carte SAMA5D3 est de l'ARMV7

Je télécharge un chaîne de compilation sur : **https://toolchains.bootlin.com/**
je choisit l'architecture armv7-eabihf, et la lib : uclibc
https://toolchains.bootlin.com/downloads/releases/toolchains/aarch64/tarballs/aarch64--glibc--stable-2020.02-2.tar.bz2

Q2 : Quelle est la version du compilateur gcc de votre chaîne de compilation croisée ?
→ la version de gcc est la : 8.4.0

Q3 : Quelle est la version de la librairie uclibc-ng ?
→ la version de uclibc-ng est la : 1.0.32

Q4 : Quelle est la version des linux-headers ?
→ la version des linux-headers est la : 1.0.32

Q5 : Dans quel répertoire (chemin absolu) se trouve les binaires principaux de votre toolchain ?
→ dans le repertoire armv7-eabihf--uclibc--stable-2020.02-2/bin/

Q6 : Quel est le nom (le nom de l'exécutable) du compilateur C ?
→ Le nom de l'exécutable est : arm-buildroot-linux-uclibcgnueabihf-gcc

Maintenant on vas ajouter l'executable a PATH, afin de pouvoir l'utiliser plus facilement.
J'ouvre le fichier bash rc avec la commande :
> sudo nano ~/.bashrc
j'ajoute a la fin de fichier la ligne :
« export PATH= '' $PATH:/home/yoda/Desktop/toolchain/armv7-eabihf—uclibc--stable-2020.02-2/bin/ ''
Puis je sauvegarde et ferme le fichier.

Je crée un petit programme, et le compile…

Q7 : Quelle est la taille de l'exécutable obtenu ?
→ La taille de l'executable est de : 7328 Bytes

Q8 : Que se passe-t-il si vous essayez d'exécuter l'exécutable obtenu ?
→ le programme ne s'exécute pas, et j'obtiens l'erreur : «cannot execute Binary file : exec format error»

Q9 : Comment pouvez-vous vous assurer que le code binaire généré correspond bien à l'architecture ARM (ELF 32-bit LSB executable) ?

→ j'utilise la commande file, qui me retourne le type d'executable :
ELF 32 bit LSB executable, ARM…

Q10 : Quelle est la taille de l'exécutable obtenu ?

→ En compilation static l'executable fait : 137 936 Bytes

Q11 : Comment pouvez-vous expliquez la différence de taille avec l'exécutable obtenu sans l'option -static ?

→ Car toutes les librairies sont compilé avec le programme.

TP2

Objectif :
L'objectif de ce travail pratique est d'abord de mettre en place une ligne de communication série, compiler, installer et utiliser le bootloader U-Boot. Ensuite, il s'agira de mettre en place une communication TFTP entre le système embarqué cible et la machine de développement (host).

Je télécharge sam-ba :
https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/SAM-BA%20In-system%20Programmer

Q1 : Parmis les binaires de sam-ba se trouvant dans l'archive, lequel s'exécute correctement ?
→ le fichier sam-ba_64

Q2 : Pourquoi n'est-ce pas le cas avec l'autre exécutable ?
→ C'est un binaire 32 bit pour les proc intel 80386

Rappel du cablage : TX cable vert → RX carte
RX cable blanc → TX carte
GND cable noir → GND carte

Ouverture de la communication Série :
> picocom /dev/ttyUSB0 -b 115200

Puis télécharger u-boot à l'adresse : ftp://ftp.denx.de/pub/u-boot/u-boot-2016.03.tar.bz2

je décompresse le dossier, puis je vais à l'intérieur avec la commande cd.

Dans le dossier u-boot-2016.03/ j'exécute la commande :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm menuconfig

J'obtiens une erreur lors de la compilation, il faut que j'installe le paquet : libncurses-dev/focal
> sudo apt install libncurses-dev/focal

Ensuite pour compiler le binaire de u-boot, il faut exécuter la commande :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm sama5d3_xplained_nand_flash_defconfig

Q3 : Quelle configuration U-Boot avez-vous utilisée ?
→ j'ai utiliser la configuration sama5d3_xplained_nand_flash_defconfig

Q4 : Quels fichiers ont été générés et où se trouvent-ils ? Voir réponse question 5
Q5 : Quelle est la taille de ceux-ci ?
la compilation à générer 3 fichier :
Dans le dossier : u-boot-2016.03/
- u-boot → 4,5 [Mo]
- u-boon.bin → 516 [ko]
- u-boot.cfg → 8.0 [ko]

AT91BootStrap :
Maintenant il faut télécharger la version 3.9.3 de AT91BootStrap sur le repos :
git://github.com/linux4sam/at91bootstrap

Ensuite dans le dossier, executer la commande :
> make menuconfig

Ensuite dans le menu, il faut configurer AT91BootStrap, pour notre carte :
→ la carte : SAMA5d3_xplained
→ CPU à 528 [MHz]
→ 256 [MB] de RAM
→ dans u-Boot image size, mettre : 0x101000

Ensuite, il faut compiler :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm

Q6 : Quels fichiers ont été générés et où se trouvent-ils ?
6 fichier ont été généré dans at91bootstrap.../bin
→ boot.bin
→ at91bootstrap.bin
→ sama5d3_xplained_nandflash_boot-uboot-3.9.3.bin
→ sama5d3_xplained_nandflash_boot-uboot-3.9.3.elf
→ sama5d3_xplained_nandflash_boot-uboot-3.9.3.pmecc
→ sama5d3_xplained_nandflash_boot-uboot-3.9.3.map

Maintenant, il faut flasher AT91bootstrap, et U-boot sur la carte :
je suis la démarche écrite dans le cours :

1. Enlever le jumper JP5 (NAND CS) de la carte : ceci désactive l'accès à la NAND, empêchant ainsi le système de booter depuis la NAND.
2. Effectuer un reset de la carte (boutton pressoir RESET).
3. A ce moment là, le message "RomBoot" devrait s'afficher sur la ligne série et un nouveau périphérique série devrait être détecté par le kernel Linux : `/dev/ttyACM0`.
4. Remettre alors le jumper JP5 en place.
5. Lancer `sam-ba`, puis séléctionnez l'interface série `/dev/ttyACM0` ainsi que la carte correspondant à votre cible. Cliquer ensuite sur "Connect".
6. Dans l'onglet "NandFlash" :
   — Choisir le script "Enable NandFlash" → "Execute"
   — Choisir ensuite le script "Erase All" → "Execute"
   — Choisir ensuite le script "Enable OS PMECC parameters" → "Execute"
   — Choisir ensuite le script "Pmecc configuration" → "Execute". Ceci permet de changer les paramètres ECC de la NAND afin que cela corresponde aux besoins de RomBOOT. Vérifiez que "ECC bits" est à 4 et "ECC offset" est à 36.
   — Choisir le script "Send Boot File" puis envoyer le bootloader AT91Bootstrap; cette opération "flash" AT91Bootstrap dans la NAND de la carte.
   — Dans la zone de l'onglet où est indiqué "Download / Upload File", sélectionnez le fichier U-Boot compilé précédemment. Spécifiez ensuite l'adresse de destination dans le champs "Address" puis cliquez sur "Send File". Cette opération "flash" U-Boot dans la NAND de la carte.

Ensuite, appuyer sur le bouton reset de la carte, et elle devrait booter.

Mise en place du serveur FTP sur le PC :
1. Installer tfptd-hpa
> sudo apt install tftpd-hpa

2. Configurer tftpd :
2.1 crée un dossier «tftp_host» dans $HOME
2.2 modifier le fichier /etc/default/tftpd_hpa, et modifier la ligne Directory=’home/yoda/tftpd_host’
2.3 redémarrer le serveur : > systemctl restart tftpd_hpa

3. Activer le serveur au démarrage du PC:
> sudo systemctl enable tftpd_hpa

4. mettre un fichier hello.txt dans tftp_host pour tester le serveur

Q7 : Comment pouvez-vous vous assurer que votre serveur TFTPD est bien en attente de connexions (indice : netstat. . .) ?
netstat -na | grep udp
On peux voir que le port 69 est utiliser, donc le serveur est actif

Maintenant, il faut configurer le port ethernet qui est connecter a la carte :
→ IP: 192.168.1.1 static (pas de DHCP)
→ Masque : 255.255.255.0
→ GateWay ./../../

Ensuite, on configure la carte :
> setenv ipaddr 192.168.1.42
> setenv serverip 192.168.1.1
> setenv ethaddr 00:01:02:03:04:05

On sauvegarde les paramètre dans la nand :
> saveenv

Puis on télécharge le fichier hello.txt sur la carte :
> tftp 0x22000000 192.168.1.1:hello.txt

Q8 : Quelle commande avez-vous utilisé pour afficher le contenu de la RAM ?
→ la commande md permet de voir la memoire :
> md 0x22000000
Ce qui permet de voir hello world contenu dans hello.txt

TP3

Objectif :
Le but de ce travail pratique est, dans un premier temps, d'obtenir les sources du kernel Linux, de les patcher, puis d'effectuer la compilation croisée pour une cible ARM. Ensuite, il s'agira de flasher le kernel fraîchement compilé sur la carte Sama5D3 Xplained afin de le charger et l'exécuter avec U-Boot.

Je me rend sur le site : https://www.kernel.org/

Q1 : Quelle est la dernière version stable du kernel ?
→ La dernière version stable est la : 5.8.14

Q2 : Quelle est la version du kernel sur votre machine de développement ?
j'exécute la commande :
> journalctl -k | uname -r
Et j'obtiens : 4.4.215

Q3 : Quelle est la version des headers du kernel de votre chaîne de compilation croisée ?
→ Réponse 4 du TP1 → 4.4.215

Q4 : Quelle version officielle du kernel avez-vous téléchargée ?
→ Je télécharge la version 4.4.238

Q5 : Quel est le nombre total de fichiers sources (.c, .h, et .S) du kernel que vous venez de télécharger ?
Pour compter le nombre de fichier source, j'utilise plusieurs commande :
> find -name *.c | wc -l
retourne : 22249

> find -name *.h | wc -l
retourne : 17289

> find -name *.S | wc -l
retourne : 1426

Ce qui fait un total de 40964 fichier source.

Q6 : Quelles est le nombre d'architectures supportées par le kernel et comment avez-vous déterminé cette valeur ?
→ 31 Architecture sont supporté par le kernel
> cd arch
> ls -l | grep dr | wc -l

Q7 : Quels fichiers patch devez vous donc télécharger ?
→ je télécharge le kernel V4.3 →
https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/snapshot/linux-4.3.tar.gz

Puis je télécharge le patch de la version 4.4 :
http://cdn.kernel.org/pub/linux/kernel/v4.x/patch-4.4.xz

Ensuite je télécharge le patch de la version 4.4.238 :
http://cdn.kernel.org/pub/linux/kernel/v4.x/patch-4.4.238.xz

Q8 : Comment est représenté le code ajouté, enlevé ou modifié d'un version à l'autre ?
→ patching file, suivit de tous les fichier qui se font patcher
Pour appliquer les patch :
> cd linux-4.3
> xzcat ../patch-4.4.xz | patch -p1
Ensuite un applique le second patch :
> xzcat ../patch-4.4.238.xz | patch -p1

maintenant il faut renommer le dossier linux-4.3 en linux-4.4.238

Compilation du kernel :

Q9 : Quelle configuration avez-vous choisie ?
→ la configuration : sama5_defconfig
Pour charger la configuration :
> make sama5_defconfig CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm

Ensuite, j'utilise la commande make menuconfig pour verifier que la bonne configuration est chargée :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm menuconfig

Q10 : Comment pouvez-vous vous assurez avec une assez bonne confidence qu'il s'agit de la bonne configuration ?

Je compile le kernel avec la commande :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm

Q11 : Comment pouvez-vous accélerer le processus de compilation ?
→ En utilisant plusieurs processus:
> make -j n
Ou n est le nombre de processus à utilisé.

Q12 : Une fois la compilation du kernel terminée, où se trouve le kernel compilé et quelle est sa taille ?
→ le kernel se trouve dans arch/arm/boot/Zimage
le fichier fait : 3'650'768 [Bytes]

Q13 : Aussi, quel est le fichier de Device Tree Blob pour votre carte Sama5D3 Xplained ?
→ le DTB est le : at91-sama5d3_xplained.dtb
qui se trouve dans le dossier : /*arch/arm/boot/dts/*

Q14 : Quelle est la nouvelle taille du kernel et quel pourcentage en espace disque avez-vous ainsi gagné ?
→ le nouveau fichier fait 2'633'648 [Bytes]
le gain d'espace est de ~27,86 %

Q15 : Comment avez vous désactivé le boot automatique ?
→ Pour désactiver le boot automatique j'ai utiliser la commande :
> setenv bootdelay -1

Téléchargement du kernel sur la carte :
je deplace le kernel, et le dtb dans le dossier tftp_host

> tftp 0x21000000 192.168.1.1:zImage
> tftp 0x22000000 192.168.1.1 :at91-sama5d3_xplained.dtb

Ensuite pour demarrer le kernel :
> bootz 0x21000000 – 0x22000000

Q16 : Pourquoi est-ce que le kernel crash ?
→ Car le kernel ne trouve pas le volume rootfs.

Configuration du démarrage automatique :
> setenv downloadkernel tftp 0x21000000 192.168.1.1:zImage
> setenv downloaddtb tftp 0x22000000 192.168.1.1 :at91-sama5d3_xplained.dtb
> setenv bootcmd run downloadkernel ; run downloaddtb ;bootz 0x21000000 – 0x22000000
> setenv bootdelay 4
> saveenv

Q17 : Quelle est la taille, en KB, réservée au DTB ?
→ La taille réservé est de :131'072 [kB] → 0x20000

Q18 : Quelle est la taille, en MB, réservée au kernel ?
→ la taille réservé est de : 5'242'880 [MB] → 0x500000

Effacer la NAND :
DTB :
> nand erase 0x140000 0x20000
Kernel :
> nand erase 0x160000 0x500000

Q19 : Comment pouvez-vous vous assurer que la NAND a bien été effacée ?
→ la commande md 0x140000 affiche que des 0xFF, donc la mémoire est effacée.

Écriture du kernel et du DTB dans la NAND :
Je fais des script intermédiaire, afin de rendre le code plus lisible :
Effacement de la nand :
> setenv cleardtb nand erase 0x140000 0x20000
> setenv clearkernel nand erase 0x160000 0x500000

Puis écriture des script **update_dtb**, et **update_kernel** :
> setenv update_dtb run downloaddtb ;run cleardtb;nand write 0x22000000 0x140000 0x20000
> setenv update_kernel run downloadkernel;run clearkernel ; nand write 0x21000000 0x160000 0x500000

Ensuite je sauvegarde les script dans la mémoire :
> saveenv

Je fais aussi un script qui permet de charger le kernel, et le dtb, de la NAND, a la ram :
> setenv load_nand_kernel_dtb nand read 0x21000000 0x160000 0x500000 ; nand read 0x22000000 0x140000 0x20000
> saveenv

Mise en place du démarrage automatique depuis la nand :

Q20 : Quelles sont les commandes U-Boot (complètes) que vous avez utilisées ?
→ les commande sont les suivante :

> setenv bootcmd run load_nand_kernel_dtb;bootz 0x21000000 – 0x22000000

Tout fonctionne, quand je reset la carte, elle essaie de démarrer sur le kernel.

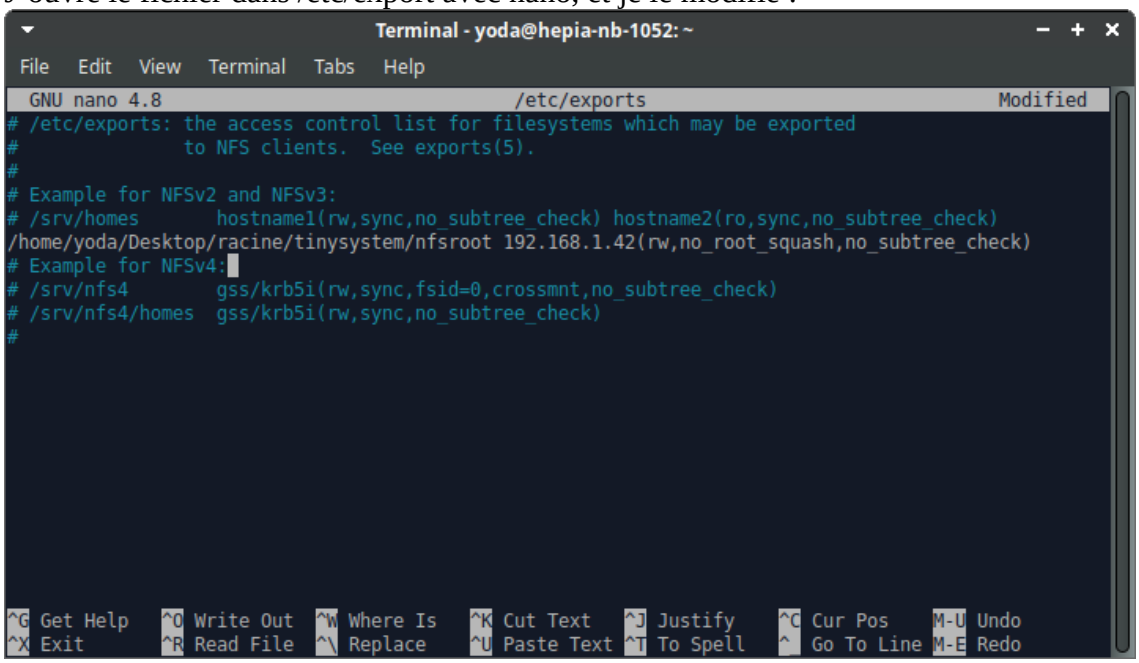
TP4

Dans mon cas l'option NFS était déjà activé dans le kernel. Donc je n'ai pas eut besoin de l'activer.
Le réglage se trouve dans :
File systems > Network File System > NFS

je crée un répertoire tinysystem dans le répertoire racine. Et dans ce répertoire, je fais un dossier nfsroot. (racine/tinysystem/nfsroot)

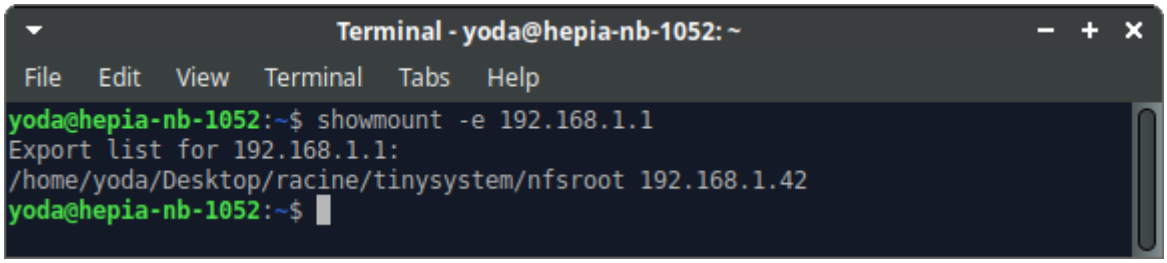
Ensuite j'installe le paquet permettant de crée un serveur NFS.
> sudo apt install nfs-kernel-server
Pour l'activer au démarrage, j'utilise la commande :
> sudo systemctl enable nfs-kernel-server

Configuration de nfs-kernel-serveur :
J'ouvre le fichier dans /etc/export avec nano, et je le modifie :



Puis je relance le serveur :
>sudo systemctl start nfs-kernel-server

Le serveur est bien demarré :



<u>Q1 : Pourquoi a-t-on intérêt à spécifier les options rw et no_root_squash dans le fichier /etc/exports ?</u>
→ wr signifie que le répertoire est en lecture et écriture (WRITE/READ)

L'option no_root_squash donne les droit du répertoire du serveur NFS.

Je redémarre la carte SAMA5d3, et l'empêche de démarrer le kernel.
Je configure la carte avec l'argument bootargs, pour utiliser nfs :
> setenv bootargs 'root=/dev/nfs ip=192.168.1.42:::::eth0 nfsroot=192.168.1.1:/home/yoda/Desktop/racine/tinysystem/nfsroot,v3 rw'

Le système plante, car il n'y a pas de dossier dev, je crée donc un dossier dev.
Mais le système plante toujours :

Q2 : Pourquoi ? (lisez les messages du kernel)
→ le système ne boot pas, il lui manque beaucoup de fichier, comme le daemon init

Q3 : Comment résoudre ce problème ?
→ Pour résoudre ce problème, il faut installer busybox, afin les fichier qui permettront au système de booter correctement.

Je crée un répertoire busybox dans racine/tinysystem/
Je récupère la version 1.30.1 sur le git : https://github.com/mirror/busybox
Puis dans le répertoire, j'effectue les commande :
> make clean
> make defconfig
> make menuconfig

Puis je désactive les réglage comme dans l'énoncé :

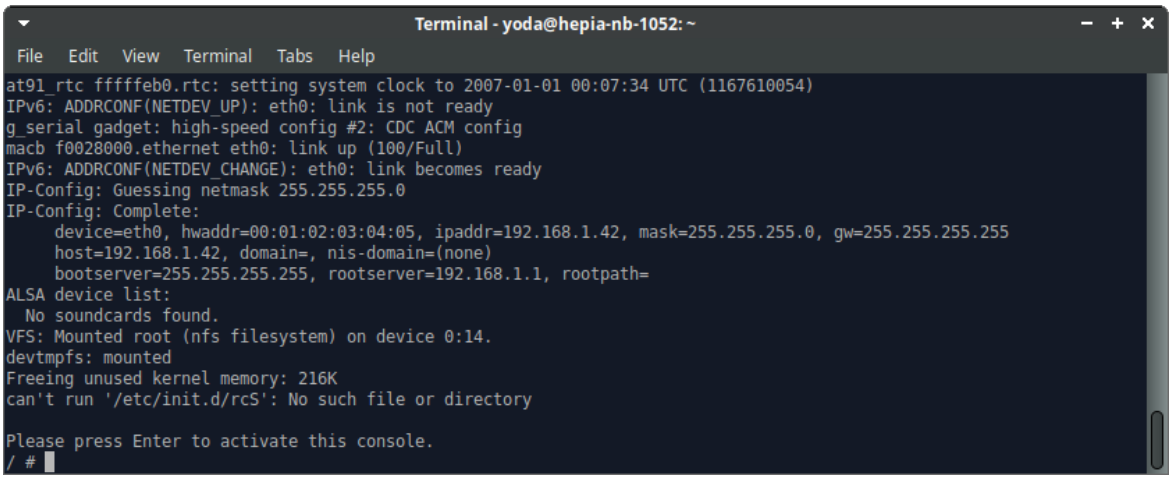
- Dans "Settings" :
  - Enable compatibility for full-blown desktop systems
  - History saving
  - Support infiniband HW
- Dans "Print Utilites" : toutes les options
- Dans "Mail Utilites" : toutes les options
- Dans "Runit Utilites" : toutes les options
- Dans "Shells" : hush

Dans "Shells", assurez-vous aussi que "Choose which shell is aliased to 'sh' name" et "Choose which shell is aliased to 'bash' name" pointent tous deux sur le shell "ash".

Mettre la compilation en static dans Settings → Build static binary
Puis je compile > make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-ARCH=arm

Une fois la compilation terminé, je refais un menuconfig pour changer la destination :
> make menuconfig
puis dans Settings → Installation Options, je rentre ../nfsroot
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-ARCH=arm
> make install CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm

Maintenant je reboot ma carte. Et ça fonctionne :



Q4 : Que se produit-il si vous exécutez la commande ps ?
→ un message d'erreur apparaît : **ps : can't open /proc : no such file or directory**

Je crée un dossier proc, et un dossier sys avec le terminal de la carte:
> mkdir proc
> mkdir sys
Ensuite je monte les dossier sur le système :
> mount -t proc name /proc
> mount sysfs -t name /sys

Maintenant si j'essaie la commande ps, des processus s'affiche.

Configuration du système et démarrage :
Je crée un dossier etc :
> mkdir etc
> cd etc
> touch inittab
> mkdir init.d
> cd init.d
> touch rcS

Puis sur le PC, j'ouvre le fichier : rcS et je met ce texte à l'intérieur.
#! /bin/sh

/bin/mount -t proc proc /proc
/bin/mount -t sysfs sys /sys

Et dans le fichier inittab :
::sysinit:/etc/init.d/rcS
ttyS0::askfirst:/bin/sh

je donne les permission d'exécution à rcS (a partir de la carte):
> cd etc
> cd init.d
> chmod +x rcS

Puis je fais un reboot pour vérifier que tout fonctionne toujours:
> reboot

puis je test le system :
> sleep 10
CTRL+Z → le processus s'arrete
> fg → le processus reprend

Utilisation des librairies partagées :
Je crée un fichier test.c qui affiche «Hello World»
puis je le compile avec :
> arm-buildroot-linux-uclibcgnueabihf-gcc hello.c -o hello
Je déplace l'exécutable a la racine tinysystem/nfsroot

Q5 : Que se passe-t-il ?
→ Le programme affiche une erreur : bin/sh: ./hello not found

Q6 : Pouvez-vous l'expliquer ?
→ Car les liens sur les librairie ne sont pas correct

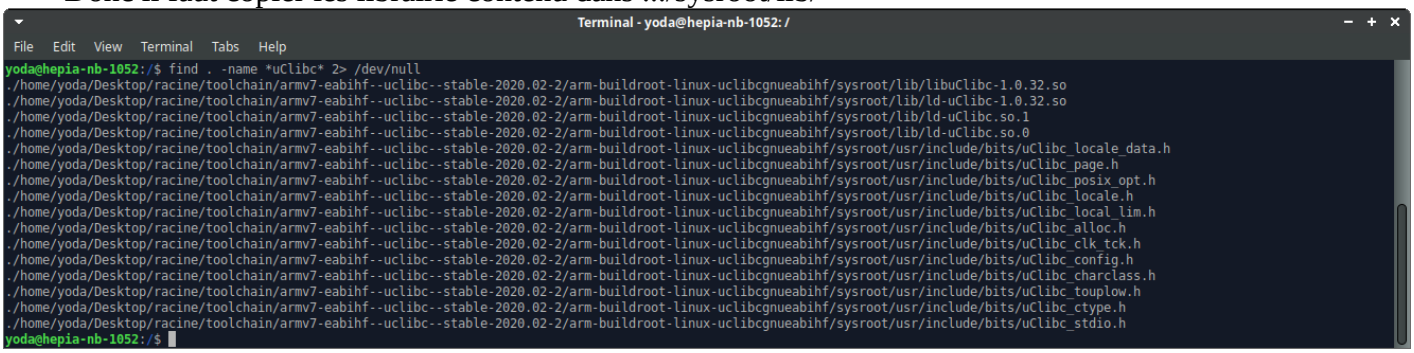
Pour faire fonctionner le programme je compile en static :
> arm-buildroot-linux-uclibcgnueabihf-gcc -static hello.c -o hello

Q7 : Avez-vous maintenant une meilleure idée de pourquoi l'exécution précédente de votre programme a échoué ?
→ Pareil que la réponse Q6, mais j'ajoute que lors de la compilation de busybox, on as specifier qu'on voulais des lien static

Pour déterminer quelle librairie mon exécutable a besoin, je le recompile sans l'option static
> arm-buildroot-linux-uclibcgnueabihf-gcc hello.c -o hello
Puis j'utilise readelf pour afficher les dépendances :
> arm-buildroot-linux-uclibcgnueabihf-readelf -a hello
Afin d'affiner la recherche, j'utilise grep :
> arm-buildroot-linux-uclibcgnueabihf-readelf -a hello | grep .so

Q8 : Déterminez alors les librairies dynamiques dont dépend votre programme, quelles sont-elles ?
→ J'ai besoin de la librairie /lib/ld-uClibc.so

Pour trouver l'emplacement de la librairie j'ai utiliser cette commande :
> find . -name *uClibc* 2> /devnull
Donc il faut copier les librairie contenu dans .../sysroot/lib/



Q9 : Dans quel(s) répertoire(s) avez-vous rajouté les librairies ?
→ Je crée un dossier /lib dans mon linux embarqué, et les copie dedans
> cp -a . ~/desktop/racine/tinysystem/nfsroot/lib

La taille de exécutable busybox (linuxrc) fait : 1'136'428 Bytes
Je retourne dans la configuration de busybox :
> make menuconfig
Mais cette fois je change la configuration pour utiliser des librairie partagé :
Je décoche la case dans : → Settings → Build Options → Build BusyBox as static binary

Enfin je réinstalle le système :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm
> make install CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm

Le nouvel exécutable de busybox fait 928'376 Bytes
Q10 : Comparez alors la taille de l'exécutable avec la version compilée statiquement, que remarquez-vous ?
→ Le nouvelle exécutable est plus petit, ce qui est normal car les liens ne sont pas static

Maintenant si je recompile mon hello.c sans l'option -static je peux l'exécuter sur mon système embarqué.

Déploiement d'un mini serveur web :
Je télécharge l'archive du site web a cette adresse :
https://gitedu.hesge.ch/flg_courses/embedded_linux_pub/-/blob/master/resources/www.tar.xz
Puis je le décompresse et le colle dans nfsroot.

Pour démarrer le serveur sur la carte, il faut exécute la commande :
> httpd -p 192.168.1.42:80 -h /www

Maintenant j'ouvre Firefox sur le pc, et accède à l'adresse 192.168.1.42 → La page web s'affiche.

Q11 : Donnez le contenu des 3 patches permettant de patcher les 3 scripts ci-dessus (cpuinfo, uptime et reboot).

cpuinfo :
En dessous de la ligne echo «Your embedded device uses the ...»

il faut ajouter : cat /proc/cpuinfo

uptime :
En dessous de la ligne echo «Your embedded device uses the ...»
il faut ajouter : uptime

Reboot :
En dessous de la ligne echo «System reboot initiated»
il faut ajouter : reboot

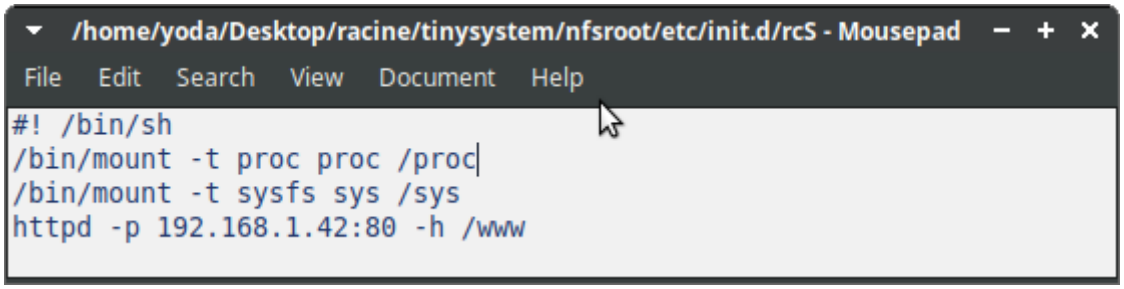
Compilation de upload.c :
> arm-buildroot-linux-uclibcgnueabihf-gcc upload.c -o upload

Pour ajouter le lien Processes qui listera tous les process en cours :

dans index, j'ajoute la ligne : <a href=''/cgi/bin/processes''>Processes<\a><\li>
Et crée un ficher processes dans cgi/bin/ et y mettre ce code à l'intérieur :

Pour démarrer le serveur au démarrage du système, j'ajoute une ligne dans etc/init.d/rcS
> httpd -p 192.168.1.42:80 -h /www

<u>Q12 : Donnez le contenu final de votre fichier rcS.</u>



TP5

Pour commencer, il faut changer les paramètre du kernel pour que l'USB soit inclus sous forme de module, pour faire cela, je vais dans les paramètre du kernel :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm menuconfig
Puis dans Device Drivers > USB support
Puis je met des M a tous les endroit qui était cocher par une étoile *
Ensuite je recompile le kernel avec :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm -j4
Ensuite avec la commande :
> find drivers -name *.ko

```
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$ find drivers -name *.ko
drivers/misc/eeprom/eeprom_93cx6.ko
drivers/hid/usbhid/usbhid.ko
drivers/usb/class/cdc-acm.ko
drivers/usb/storage/usb-storage.ko
drivers/usb/host/ohci-hcd.ko
drivers/usb/host/ehci-hcd.ko
drivers/usb/core/usbcore.ko
drivers/usb/serial/usbserial.ko
drivers/usb/serial/ftdi_sio.ko
drivers/usb/serial/pl2303.ko
drivers/net/wireless/mwifiex/mwifiex_sdio.ko
drivers/net/wireless/mwifiex/mwifiex_usb.ko
drivers/net/wireless/mwifiex/mwifiex.ko
drivers/net/wireless/rt2x00/rt2800usb.ko
drivers/net/wireless/rt2x00/rt2x00usb.ko
drivers/net/wireless/rt2x00/rt2500usb.ko
drivers/net/wireless/rt2x00/rt2800lib.ko
drivers/net/wireless/rt2x00/rt2x00lib.ko
drivers/net/wireless/rt2x00/rt73usb.ko
drivers/net/wireless/libertas_tf/libertas_tf_usb.ko
drivers/net/wireless/libertas_tf/libertas_tf.ko
drivers/net/wireless/realtek/rtl818x/rtl8187/rtl8187.ko
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$
```

La compilation a bien crée des module pour l'USB mais ainsi que pour le wifi, comme nous n'en avons pas besoin , les les enlève avec menuconfig dans Device Drivers → Network Device support → Wireless Lan , puis je recompile (je fais un clean avant).
Je crée un dossier sur la carte dans /lib/module/4.4.238

[Q1] : Quels modules avez-vous déployés sur votre système embarqués ?
> les modules USB

```
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$ find drivers -name *.ko
drivers/misc/eeprom/eeprom_93cx6.ko
drivers/hid/usbhid/usbhid.ko
drivers/usb/class/cdc-acm.ko
drivers/usb/storage/usb-storage.ko
drivers/usb/host/ohci-hcd.ko
drivers/usb/host/ehci-hcd.ko
drivers/usb/core/usbcore.ko
drivers/usb/serial/usbserial.ko
drivers/usb/serial/ftdi_sio.ko
drivers/usb/serial/pl2303.ko
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$
```

Puis pour deployer les module il faut definir la variable d'environnement :
> INSTALL_MOD_PATH=/home/yoda/Desktop
La ligne de comande a utilisé est :
> make modules_install CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm INSTALL_MOD_PATH=/home/yoda/Desktop -j4

ce qui retourne :

```
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$ ma
  INSTALL crypto/af_alg.ko
  INSTALL crypto/algif_hash.ko
  INSTALL crypto/algif_skcipher.ko
  INSTALL crypto/echainiv.ko
  INSTALL drivers/hid/usbhid/usbhid.ko
  INSTALL drivers/misc/eeprom/eeprom_93cx6.ko
  INSTALL drivers/usb/class/cdc-acm.ko
  INSTALL drivers/usb/core/usbcore.ko
  INSTALL drivers/usb/host/ehci-hcd.ko
  INSTALL drivers/usb/host/ohci-hcd.ko
  INSTALL drivers/usb/serial/ftdi_sio.ko
  INSTALL drivers/usb/serial/pl2303.ko
  INSTALL drivers/usb/serial/usbserial.ko
  INSTALL drivers/usb/storage/usb-storage.ko
  INSTALL lib/crc-ccitt.ko
  INSTALL lib/crc-itu-t.ko
  DEPMOD  4.4.238
yoda@hepia-nb-1052:~/Desktop/racine/kernel/linux-4.4.238$
```

Ensuite on copie le dossier lib crée sur le bureau dans le lib de la carte
> cp -a ~/Desktop/lib/modules/4.4.238 /home/yoda/Desktop/racine/tinysystem/nfsroot/lib/modules/4.4.238
Et je met la nouvelle Zimage sur le serveur tftp afin de mettre a jour le kernel
Je reboot la carte, appuie sur une touche pour avoir accès a U-boot, ensuite j'utilise les script qu'on avais mis en place pour mettre a jour le kernel.
> run update_kernel

Préparation de la clé USB :
Tout d'abord il faut connaître la partition de la clé USB, pour faire cela, je la branche, puis je fais la commande :
> sudo fdisk -l

```
Disk /dev/sda: 29.23 GiB, 31376707072 bytes, 61282631 sectors
Disk model: Extreme
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
yoda@hepia-nb-1052:/$
```

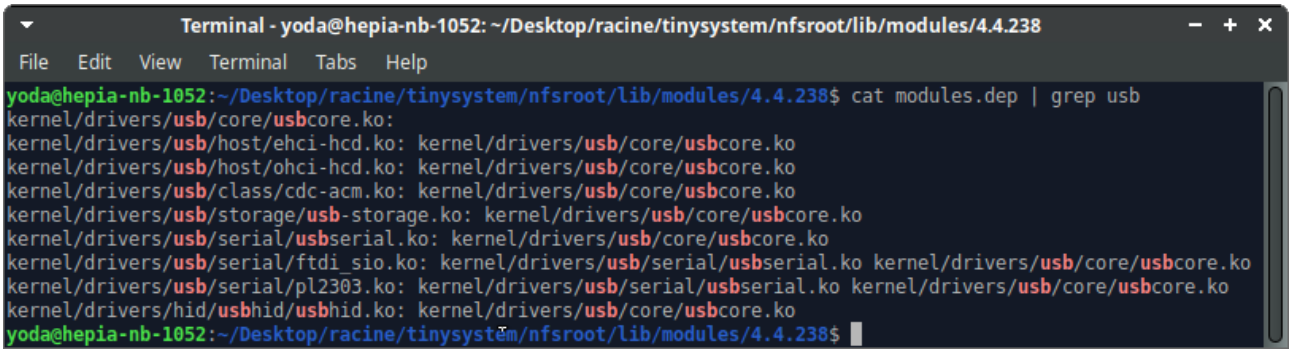
La clé USB est sur : /dev/sda

Ensuite je dump le fichier disk_ext4.img sur la clé USB :
> sudo dd if=/home/yoda/Desktop/racine/disk_ext4.img of=/dev/sda

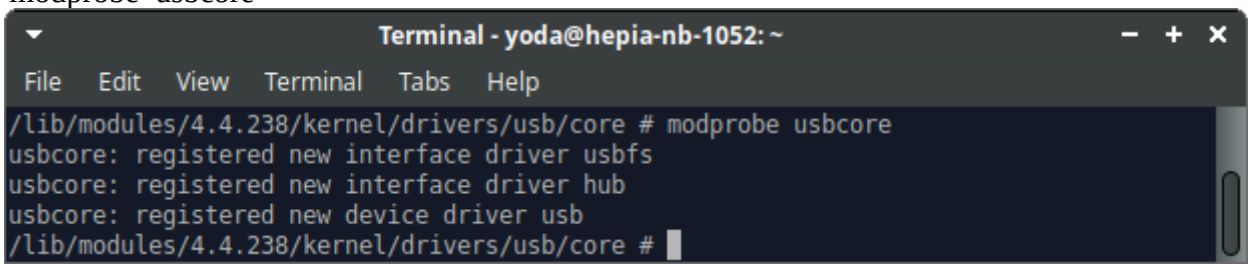
[Q2] : Quelle est la taille du système de fichiers se trouvant dans l'image et comment l'avez-vous déterminée ?
> la taille du système de fichier est de : 31MB, j'ai réutiliser la commande fdisk -l

```
Disk /dev/sda: 29.23 GiB, 31376707072 bytes, 61282631 sectors
Disk model: Extreme
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x736cbb89

Device     Boot Start   End Sectors Size Id Type
/dev/sda1        2048 65535   63488  31M 83 Linux
```

Pour connaître les module a chargé, il faut utilisé la commande dans lib/modules/4.4.238 :
> cat modules.dep | grep usb

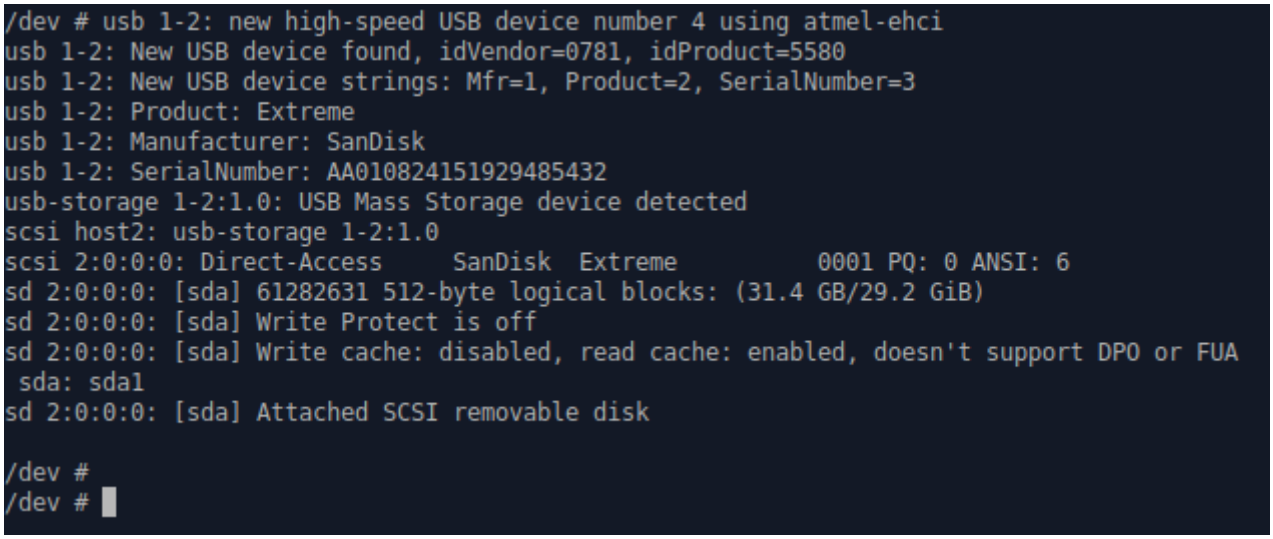


[Q3] : Quels modules avez-vous chargés afin que votre kernel puisse détecter votre clé USB ?
Il faut chargée le module usbcore.ko avec la commande :
> modprobe usbcore

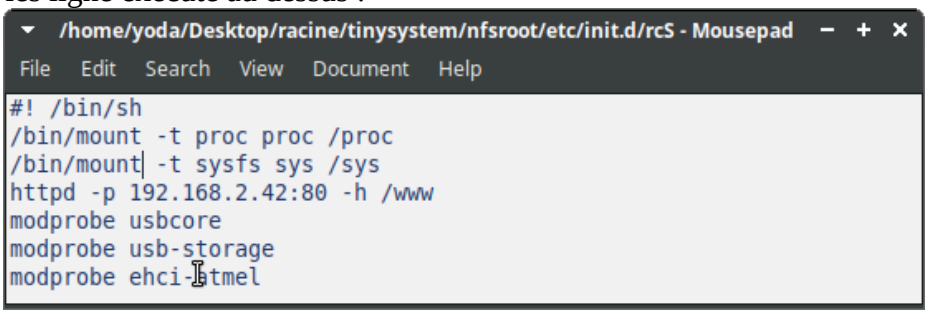


> modprobe usb-storage
> modprobe ehci-atmel

Maintenant si je branche la clé USB, elle est detecté par le kernel :



Pour que les module soit chargé automatiquement au démarrage, j'ajoute à etc/init.d/rcS
les ligne exécuté au dessus :



Pour monter la clé manuellement, je crée un dossier media, puis un dossier sda1 :
> mkdir media
> mkdir sda1
Ensuite, pour monter la clé, j'utilise la commande :
> mount /dev/sda1 -t ext4 -r /media/sda1/

```
/ # mount /dev/sda1 -t ext4 -r /media/sda1/
EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
/ # ls
bin      etc      lib      media    sbin     usr
dev      hello    linuxrc  proc     sys      www
/ # cd media
/media # cd sda1/
/media/sda1 # ls
25011113087_3b6b0c3d33_o.jpg  38761179922_832b1ac777_o.jpg
30460715436_61f507f81c_o.jpg  39561351474_e0a72219c8_o.jpg
32157084494_9c9b67a55d_o.jpg  FUJI3858-2.jpg
32178534444_7274dc0d9e_o.jpg  FUJI3892-3.jpg
36039691873_df77eaf249_o.jpg  FUJI4014-2.jpg
36419625564_53c1b736fb_o.jpg  FUJI4822-2.jpg
36419629184_f5a1aec2b9_o.jpg  FUJI5090-2.jpg
36443288663_75409837bb_o.jpg  FUJI5147-2.jpg
36443290273_2eef6cb613_o.jpg  FUJI5247-2.jpg
37067280976_9930e7ca6d_o.jpg  FUJI5367-2.jpg
37085463132_6c7139ecc4_o.jpg  FUJI5393-2.jpg
37114915041_3a308851db_o.jpg  FUJI5651-2.jpg
37264612392_0d621f1d36_o.jpg  lost+found
37905595855_7d42fdb424_o.jpg
/media/sda1 #
```

La clé est bien monté, et je peux voire ce qu'il y a dedans.

Mise en place de HotPlug :
Pour monter la clé automatiquement, il faut commencer par redirigée les éventement du hotplug (uevent), a mdev, pour faire cela on remplace le contenue du fichier hotplug avec '/sbin/mdev'
Pour faire cela, il faut utiliser la commande :
> echo /sbin/mdev > /proc/sys/kernel/hotplug

Maintenant afin que ce soit effectué a chaque démarrage, je l'ajoute au fichier rcS, ainsi que la ligne mdev -s qui permet de donner tous les périphérique qui se sont connecté pendant le démarrage.

/home/yoda/Desktop/racine/tinysystem/nfsroot/etc/init.d/rcS

File Edit Search View Document Help

```
#! /bin/sh
/bin/mount -t proc proc /proc
/bin/mount -t sysfs sys /sys
httpd -p 192.168.2.42:80 -h /www
modprobe usbcore
modprobe usb-storage
modprobe ehci-atmel
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

Ensuite je crée le fichier de configuration de mdev :
> touch /etc/mdev.conf
Puis j'y ajoute la ligne suivante : **sd[a-z][0-9]+ 1000:1000 660 */etc/KeyMount.sh***
cette ligne signifie que les device seront nomme avec sda0,sda1, sda2, … , sdb0 , … ,sdz9
Ensuite la partie 1000:1000 correspond a uid:gid sur la carte les deux groupe sont 1000…
ensuite il faut mettre les permission : 660 dans notre cas.
Pour finir les asterix indique que l'on veut exécuter une commande lorsque la clés est branché.

Je crée un fichier KeyMount.sh dans etc, et y met le script suivant :
[Q4] : Donnez le contenu du ou des scripts que vous avez implémentés pour réaliser le travail demandé.

/home/yoda/Desktop/racine/tinysystem/nfsroot/etc/KeyMoun

File Edit Search View Document Help

```
#!/bin/bash

#echo "DEBUG" > /dev/console
if [ $ACTION == "add" ]
then
        mkdir /media/$MDEV
        mount /dev/$MDEV -t ext4 /media/$MDEV
        cp /media/$MDEV/*.jpg /www/upload/files/
elif [ $ACTION == "remove" ]
then
        rm /www/upload/files/*.jpg
        umount /media/$MDEV
        rmdir /media/$MDEV
fi
```

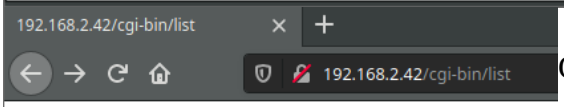
Puis j'effectue un chmod +x afin de lui donner les droit d'exécution.
> chmod +x /etc/KeyMount.sh

[Q5] : Donnez le contenu final de votre fichier rcS.

/home/yoda/Desktop/racine/tinysystem/nfsroot/etc/init.d/rcS

File Edit Search View Document Help

```
#! /bin/sh
/bin/mount -t proc proc /proc
/bin/mount -t sysfs sys /sys
httpd -p 192.168.2.42:80 -h /www
modprobe usbcore
modprobe usb-storage
modprobe ehci-atmel
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```



Ça fonctionne, les fichier apparaisse sur la page web.

File collection

25011113087_3b6b0c3d33_o.jpg

30460715436_61f507f81c_o.jpg

32157084494_9c9b67a55d_o.jpg

32178534444_7274dc0d9e_o.jpg

36039691873_df77eaf249_o.jpg

36419625564_53c1b736fb_o.jpg

TP6

Objectif :
L'objectif de ce travail pratique est de mettre en place un système embarqué Linux basé sur un support de stockage de type bloc. Vous transitionnerez donc d'un système de fichiers racine basé sur NFS vers un contenu stocké sur un support MMC (carte flash SD). De manière similaire, le kernel et device tree ne seront plus stockés dans la flash NAND RAW, mais sur la carte SD.

[Q1] : depuis le shell de votre système embarqué, comment pouvez-vous vous assurez que le kernel est capable de gérer les systèmes de fichiers squashfs, f2fs et ext4 ?

Sur la carte, je lance la commande :
> cat /proc/filesystems

```
/ # cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   tmpfs
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   sockfs
nodev   pipefs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext2
        ext4
        vfat
nodev   nfs
nodev   ubifs
/ #
```

On peut voir qu'il manque les système de fichier squashfs, et f2fs.
Mais le système de fichier ext4 est déjà pris en charge.

Donc il faut recompiler le kernel en ajoutant les nouveau systeme de fichier, pour cela je fais la commande :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm menuconfig
puis je vais dans : File systems > Miscellaneous filesystems > SquashFS 4.0
Et dans : File systems > F2FS filesystem support
Puis je met a jour le kernel avec la procédure habituel (run update_kernel)

```
/ # cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   tmpfs
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   sockfs
nodev   pipefs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext2
        ext4
        squashfs
        vfat
nodev   nfs
        f2fs
nodev   ubifs
/ #
```

squashfs, et f2fs sont maintenant disponible…

Je branche la carte SD avec son adaptateur sur le port USB du PC.

[Q2] : quel est le nom du périphérique correspondant à votre carte SD détecté par le kernel Linux ? Pensez à inspectez les logs du système.

Pour voir les log du kernel, j'utilise la commande :

> cat /var/log/kern.log

```
Dec  5 09:48:11 hepia-nb-1052 kernel: [  665.891245] usb 1-4: new high-speed USB device number 4 using xhci_hcd
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.056422] usb 1-4: New USB device found, idVendor=05e3, idProduct=0745, bcdDevice= 9.03
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.056427] usb 1-4: New USB device strings: Mfr=0, Product=1, SerialNumber=2
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.056430] usb 1-4: Product: USB Storage
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.056432] usb 1-4: SerialNumber: 000000000903
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.083414] usb-storage 1-4:1.0: USB Mass Storage device detected
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.083494] scsi host1: usb-storage 1-4:1.0
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.083562] usbcore: registered new interface driver usb-storage
Dec  5 09:48:11 hepia-nb-1052 kernel: [  666.084979] usbcore: registered new interface driver uas
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.106454] scsi 1:0:0:0: Direct-Access     Generic  STORAGE DEVICE   0903 PQ: 0 ANSI: 6
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.107129] sd 1:0:0:0: Attached scsi generic sg0 type 0
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.356141] sd 1:0:0:0: [sda] 15564800 512-byte logical blocks: (7.97 GB/7.42 GiB)
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.358586] sd 1:0:0:0: [sda] Write Protect is off
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.358592] sd 1:0:0:0: [sda] Mode Sense: 21 00 00 00
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.360066] sd 1:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
Dec  5 09:48:12 hepia-nb-1052 kernel: [  667.388407] sd 1:0:0:0: [sda] Attached SCSI removable disk
```

Il semblerait que le nom du peripherique soit sda, pour vérifier cela, j'utilise :

> sudo fdisk -l

```
Disk /dev/sda: 7.43 GiB, 7969177600 bytes, 15564800 sectors
Disk model: STORAGE DEVICE
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
yoda@hepia-nb-1052:~/Desktop$
```

Ce qui me confirme que la carte SD est monté sur */dev/sda*

[Q3] : quels sont les différents périphériques de type "block" détectés sur votre système de développement ?

Pour afficher les peripherique de type block, il suffit d'utiliser la commande :

> lsblk

```
yoda@hepia-nb-1052:~/Desktop$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0 231.1M  1 loop /snap/atom/264
loop1         7:1    0  97.8M  1 loop /snap/core/10185
loop2         7:2    0 231.1M  1 loop /snap/atom/265
loop3         7:3    0  55.3M  1 loop /snap/core18/1885
loop4         7:4    0  97.9M  1 loop /snap/core/10444
loop5         7:5    0  55.4M  1 loop /snap/core18/1932
loop6         7:6    0 290.4M  1 loop /snap/vlc/1700
sda           8:0    1   7.4G  0 disk
nvme0n1     259:0    0   477G  0 disk
├─nvme0n1p1 259:1    0   512M  0 part /boot/efi
└─nvme0n1p2 259:2    0 476.4G  0 part /
yoda@hepia-nb-1052:~/Desktop$
```

Pour m'assuré que /*dev*/sda n'est pas monté, j'essaie de le demonté :

> umount /dev/sda

```
yoda@hepia-nb-1052:~/Desktop$ umount /dev/sda
umount: /dev/sda: not mounted.
yoda@hepia-nb-1052:~/Desktop$
```

Maintenant, il faut effacé les 1024 premier byte de la carte sd avec l'outil DD, en utilisant comme source /dev/urandom :

Voici la commande que j'ai utilisé :

> sudo dd if=/dev/urandom of=/dev/sda bs=1024 count=1

```
yoda@hepia-nb-1052:~/Desktop$ sudo dd if=/dev/urandom of=/dev/sda bs=1024 count=1
[sudo] password for yoda:
1+0 records in
1+0 records out
1024 bytes (1.0 kB, 1.0 KiB) copied, 0.120035 s, 8.5 kB/s
yoda@hepia-nb-1052:~/Desktop$
```

j’utilise la commande > sudo fdisk /dev/sda
j’entre o, puis w, afin de crée la partition DOS :

```
Command (m for help): o

Created a new DOS disklabel with disk identifier 0xbbc3fccf.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Puis je relance fdisk, et je crée toutes les partition utile pour mettre le kernel sur la carte SD :
Voici les commande que j’effectue pour la première partition :

```
yoda@hepia-nb-1052:~/Desktop$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-15564799, default 2048): default
Value out of range.
First sector (2048-15564799, default 2048): 2048
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-15564799, default 15564799): +16MB

Created a new partition 1 of type 'Linux' and of size 15 MiB.

Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden or  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi ea  Rufus alignment
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         eb  BeOS fs
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ee  GPT
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fb  VMware VMFS
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fc  VMware VMKCORE
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fd  Linux raid auto
1c  Hidden W95 FAT3 75  PC/IX           bc  Acronis FAT32 L fe  LANstep
1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot    ff  BBT
Hex code (type L to list all codes): 6
Changed type of partition 'Linux' to 'FAT16'.

Command (m for help): c
DOS Compatibility flag is set (DEPRECATED!)

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Ensuite pour la deuxième partition :

```
yoda@hepia-nb-1052:~/Desktop$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (32768-15564799, default 32768): 32678
Value out of range.
First sector (32768-15564799, default 32768): 32768
Last sector, +/-sectors or +/-size{K,M,G,T,P} (32768-15564799, default 15564799): +200MB

Created a new partition 2 of type 'Linux' and of size 191 MiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

yoda@hepia-nb-1052:~/Desktop$
```

Et la troisième partition :

```
yoda@hepia-nb-1052:~/Desktop$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type
   p   primary (2 primary, 0 extended, 2 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (3,4, default 3): 3
First sector (423936-15564799, default 423936): 423936
Last sector, +/-sectors or +/-size{K,M,G,T,P} (423936-15564799, default 15564799): +32MB

Created a new partition 3 of type 'Linux' and of size 31 MiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

yoda@hepia-nb-1052:~/Desktop$
```

Boot du kernel depuis la carte SD :

Je commence par crée un systeme de fichier sur la premiere partition avec la commande :
> mkfs.fat -n KERNEL /dev/sda1
la partition KERNEL s'affiche dans le File Manager, donc je copie le kernel et le DTB a l'intérieur.
Puis j'éjecte la partition KERNEL, et débranche la carte SD du PC pour la branché sur le système embarqué.
La carte est monté sur l'interface mmc :

```
=> mmcinfo
Device: mci
Manufacturer ID: 27
OEM: 5048
Name: SD8GB
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.4 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
=> fatinfo mmc 0:1
Interface:  MMC
  Device 0: Vendor: Man 000027 Snr ee330300 Rev: 7.12 Prod: SD8GB0
            Type: Removable Hard Disk
            Capacity: 7600.0 MB = 7.4 GB (15564800 x 512)
Filesystem: FAT16 "KERNEL     "
=>
```

Je crée un script dans u-boot nommée load_mmc_kernel_dtb, qui permet de chargé en RAM le kernel, ainsi que le dtb a partir de la carte SD :
> setenv load_mmc_kernel_dtb 'fatload mmc 0:1 0x22000000 at91-sama5d3_xplained.dtb ;fatload mmc 0:1 0x21000000 zImage'

```
load_mmc_kernel_dtb=fatload mmc 0:1 0x22000000 at91-sama5d3_xplained.dtb;fatload mmc 0:1 0x21000000 zImage
```

Puis dans le script bootcmd, je remplace le script qui chargais le kernel, et le dtb de la nand, par load_mmc_kernel_dtb

```
bootcmd=run load_nand_kernel_dtb; bootz 0x21000000 - 0x22000000;
```

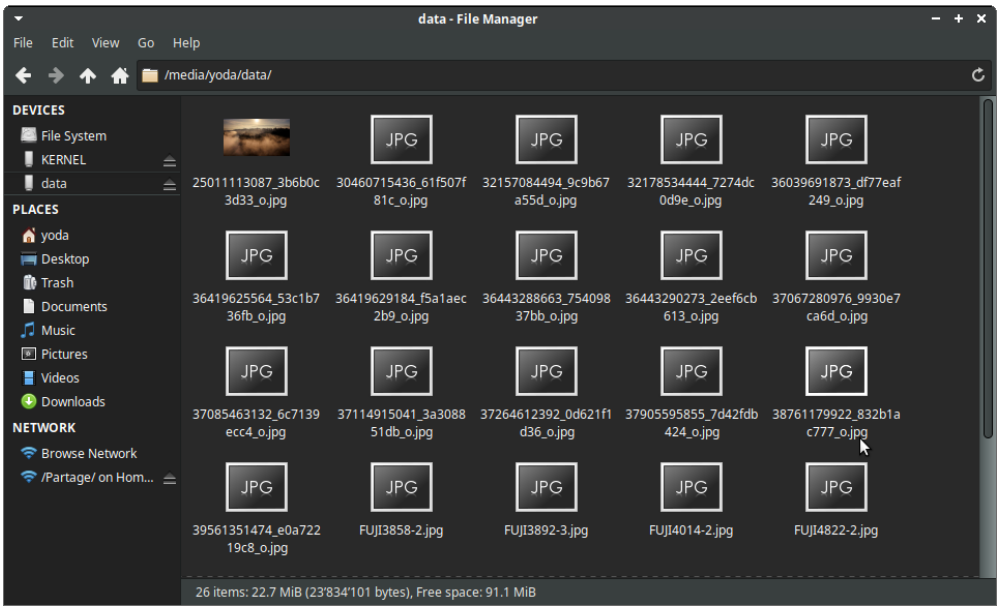
```
bootcmd=run load_mmc_kernel_dtb; bootz 0x21000000 - 0x22000000;
```

je fais un saveenv, et j'essaie de rebooter.
Ça fonctionne, le kernel, et le dtb sont lu sur la carte SD avant de décompressé le kernel, et de démarrée.

[Q4] : serait-il plus judicieux d'utiliser un système de fichiers journalisé en lieu et place du système de fichiers FAT utilisé ci-dessus ? Justifiez.
??? A répondre plus tard…

Maintenant le but vas être que le fichier uploadé sur le serveur soit stocké sur une des partition de la carte SD.
Pour faire cela, je commence par formater la deuxieme partition (celle de 200MB) en f2fs avec la commande :
> sudo mkfs.f2fs -l data /dev/sda2
Je débranche la carte SD du PC, et la reconnecte a la carte.
Maintenant, le but est que les photo soient stocké sur la carte SD, donc ce qu'on peux faire, est de monter la partition data, a l'emplacement de /www/upload/files/
J'utilise cette commande :
> mount /dev/mmcblk0p2 -t f2fs /www/upload/files/
j'ajoute aussi cette ligne dans init.d/rcs
[Q5] : comment avez-vous vérifié que les fichiers sont bien écrits sur la carte SD ?
je supprime la ligne rm /www/upload/files/*.jpg de KeyMount.sh, temporairement, pour voir si les fichier sont bien copié sur la carte SD, meme apres avoir coupé tout le systeme.

Puis j'essaie de brancher la clé USB, pour voir si les fichier apparaisse sur le site web.
Je retire la clé, et remarque que les fichier sont toujours sur le serveur, c'est parfait, maintenant, si j'éteins le système embarqué, et connecte la carte SD au PC, les fichier sont sur la partition data, donc ça fonctionne.



Je remet la ligne dans KeyMount.sh

[Q6] : y-aurait-il un désavantage à utiliser un système de fichiers de type FAT ou ext2 en lieue et place du système de fichiers utilisé ici ?
??? A répondre plus tard…

Afin de modifier l'emplacement ou sont sauvegardé les log de l'upload du serveur, je vais dans /www/cgi-bin/upload.cfg, et je modifie la ligne qui contient LogFile avec cette ligne :
→ LogFile = /var/log/upload.log
et je crée le dossier */var/log/* ainsi que le fichier upload.log
> mkdir var
> cd var
> mkdir log
> cd log
> touch upload.log

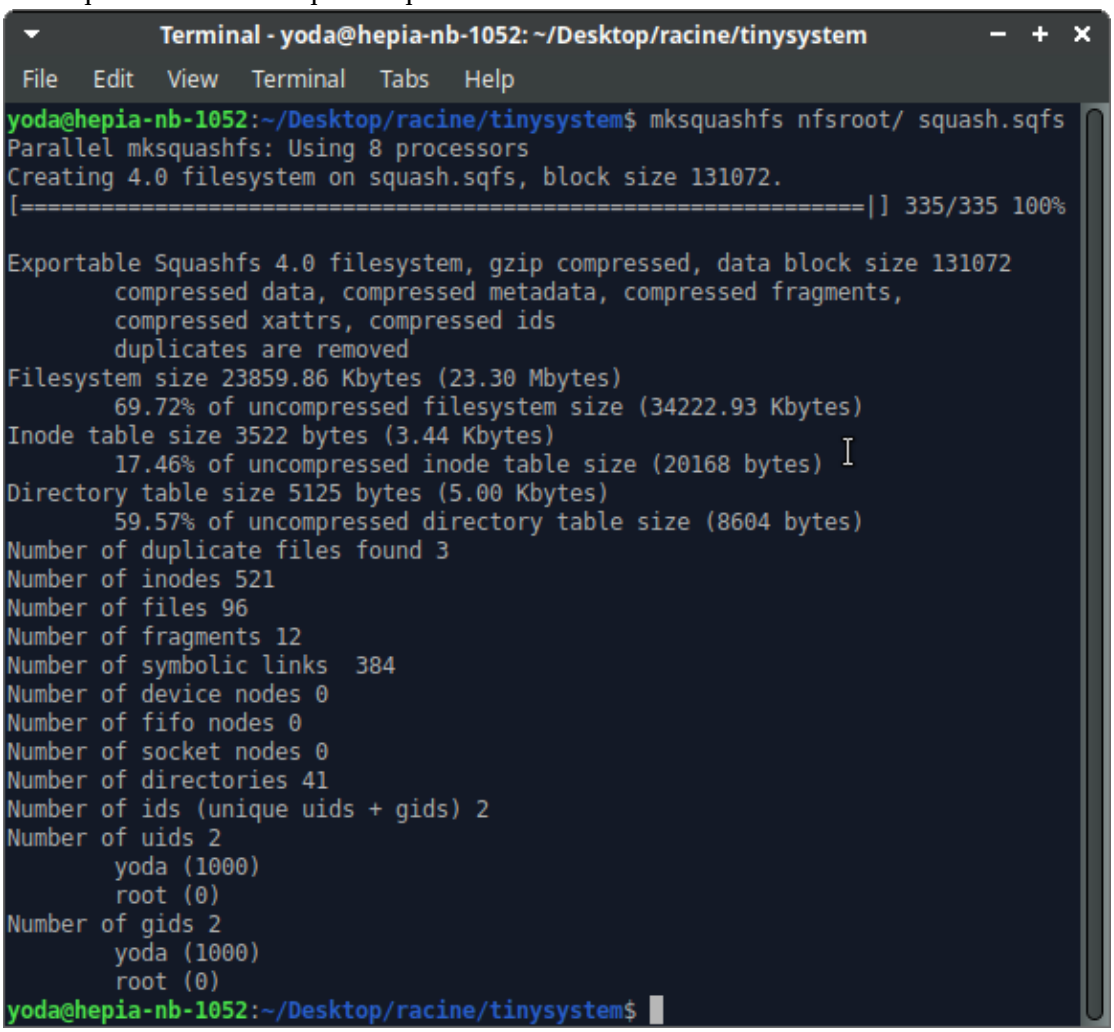
Puis j'ajoute cette ligne a etc/init.d/rcS :
> mount -t tmpfs varlog /var/log -o size=16M

Enfin j'effectue un reboot :
> reboot

[Q7] : si aucune taille n'est spécifiée, quelle sera la taille mémoire maximum utilisée par tmpfs ?

```
TMPFS_SIZE
          Maximum size for all tmpfs filesystems if no specific size is
          provided. The default is 20%VM (20% of virtual memory, including
          swap space). If no value is provided here, the kernel default (50%
          RAM) will be used. Note that the "%VM" suffix may be used in this
          and all the _SIZE settings below, but may not be used in /etc/fstab
          (the absolute size is calculated by the init scripts).
```

Système de fichiers racine (rootfs) sur carte SD :
je crée une image squashfs de mon rootfs avec la commande :
> mksquashfs nfsroot/ squash.sqfs



[Q8] : quel type de compression avez-vous choisi pour squashfs et pourquoi ?
J'ai laissé les option par défaut. (gzip)

Ensuite, je copie l'image sur la derniere partition avec dd :
> sudo dd if=squash.sqfs of=/dev/sda3

Maintenant je configure U-Boot afin qu'il utilise la carte SD comme rootfs.
D'abord je commence par faire une copie du contenue de bootargs, afin de le réutilise plus tard :
> setenv bootargs_old 'root=/dev/nfs ip=192.168.2.42:::::eth0 nfsroot=192.168.2.1:/home/yoda/Desktop/racine/tinysystem/nfsroot,v3 rw'

Puis je modifie le bootargs :
> setenv bootargs root=/dev/mmcblk0p3 rootwait ip=192.168.2.42:::::eth0
Enfin je n'oublie pas de sauvegarder :
> saveenv

Enfin j'effectue un reset afin de tester que le system démarre toujours :
> reset

[Q9] : comment pouvez-vous vous assurer que le rootfs se trouve bien sur la carte SD et qu'il n'utilise plus le système de fichiers NFS ?

Je débranche le câble RJ45, et effectue un reset, pour voir si la carte démarre quand même.

```
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
Waiting up to 110 more seconds for network.
Waiting up to 100 more seconds for network.
Waiting up to 90 more seconds for network.
Waiting up to 80 more seconds for network.
Waiting up to 70 more seconds for network.
Waiting up to 60 more seconds for network.
Waiting up to 50 more seconds for network.
random: nonblocking pool is initialized
Waiting up to 40 more seconds for network.
Waiting up to 30 more seconds for network.
Waiting up to 20 more seconds for network.
Waiting up to 10 more seconds for network.
Waiting up to 0 more seconds for network.
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
     device=eth0, hwaddr=00:01:02:03:04:05, ipaddr=192.168.2.42, mask=255.255.255.0, gw=255.255.255.255
     host=192.168.2.42, domain=, nis-domain=(none)
     bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
ALSA device list:
  No soundcards found.
VFS: Mounted root (squashfs filesystem) readonly on device 179:3.
devtmpfs: mounted
Freeing unused kernel memory: 216K
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
usbcore: registered new interface driver usb-storage
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-atmel: EHCI Atmel driver
atmel-ehci 700000.ehci: EHCI Host Controller
atmel-ehci 700000.ehci: new USB bus registered, assigned bus number 1
atmel-ehci 700000.ehci: irq 54, io mem 0x00700000
atmel-ehci 700000.ehci: USB 2.0 started, EHCI 1.00
usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb1: Product: EHCI Host Controller
usb usb1: Manufacturer: Linux 4.4.238 ehci_hcd
usb usb1: SerialNumber: 700000.ehci
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 3 ports detected

Please press Enter to activate this console.
/ #
```

Le systeme démarre quand même, mais je note qu'il attend pendant 120 seconde que le réseau se connecte…

[Q10] : comment pouvez-vous être 100% sûre que votre système n'utilise plus du tout NFS et n'utilise plus du tout de flash NAND RAW ?

J'accède a U-Boot, et efface le dtb, ainsi que le kernel, puis j'essaie de demandé :
> run cleardtb
> run clearkernel

Le system embarqué démarre toujours, donc il n'utilise pas ce qui était présent dans la NAND.

TP7

Objectif :
Stocker le DTB, et le kernel dans la NAND de la carte.

[Q1] : quelle est la taille d'un erase block, d'une page, et des zones OOB pour la flash NAND de votre carte Sama5D3 ?
J'utilise la commande : nand info

```
=> nand info

Device 0: nand0, sector size 128 KiB
  Page size       2048 b
  OOB size          64 b
  Erase size    131072 b
  subpagesize     2048 b
  options     0x40000200
  bbt options 0x    8000
=>
```

La taille d'un erase block est de 131072 bit
La taille d'une page est de 2048 bit
La taille des zone OOB est de 64 bit

[Q2] : comment avez-vous déterminés ces informations de la flash de votre carte, depuis U-Boot et Linux ?
En utilisant la commande : nand info
Sur linux ces information sont accessible dans le dossier : /sys/class/mtd0/

```
/sys/class/mtd # cd mtd0
/sys/devices/soc0/ahb/60000000.nand/mtd/mtd0 # ls
bad_blocks          ecc_failures        name                subpagesize
bbt_blocks          ecc_step_size       numeraseregions     subsystem
bitflip_threshold   ecc_strength        offset              type
corrected_bits      erasesize           oobsize             uevent
dev                 flags               power               writesize
device              mtdblock0           size
/sys/devices/soc0/ahb/60000000.nand/mtd/mtd0 # cat erasesize
131072
```

Mise à jour du kernel :

Les option de MTD et de UBIFS sont déjà activé, alors je ne recompile pas le kernel.

[Q3] : quelles tailles de volumes UBI avez-vous choisies et pourquoi ?
Dabord je commence par afficher les taille des partition avec la commande :
> cat /proc/mtd/

```
/ # cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00020000 "at91bootstrap"
mtd1: 00080000 00020000 "bootloader"
mtd2: 000c0000 00020000 "bootloader env"
mtd3: 00080000 00020000 "device tree"
mtd4: 00600000 00020000 "kernel"
mtd5: 0f800000 00020000 "rootfs"
/ #
```

→ pour reconvertir la taille en kB, je commence par convertir 0x40000 en decimale :
0x40000 = 262144
→ ensuite je divise par 1024:
262144/1024 = 256k

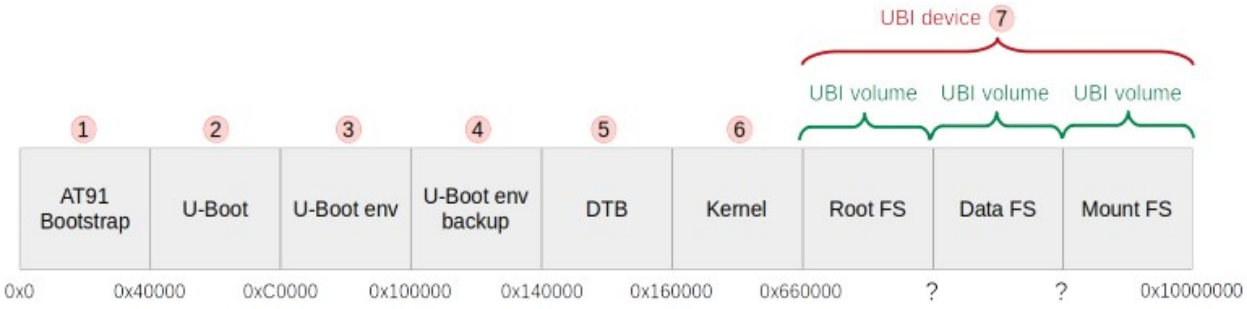
j'ai utiliser les partition donnée dans l'enoncé du TP, ensuite j'ai transformé RootFS, DataFS, et Mount FS comme une seul partition (UBI-Device), par la suite, cette partition sera decouper en 3 volumes. UBI-Device vas de 0x660000 a 0x10000000, elle a donc une taille de 261 750 784 Bytes

[Q4] : où donc sont définies les partitions MTD de la NAND de votre carte ?

Les partition sont definit par bootargs dans U-Boot.
Pour crée le partition, j'ajoute ces ligne a bootargs :
mtdparts=atmel_nand:256k(AT91Bootstrap)ro,512k(U-Boot)ro,256k(U-Boot_env),256k(U-Boot_env_backup),128k(DTB)ro,5m(Kernel)ro,−(UBI-Device)

```
/proc # cat mtd
dev:    size   erasesize  name
mtd0: 00040000 00020000 "AT91Bootstrap"
mtd1: 00080000 00020000 "U-Boot"
mtd2: 00040000 00020000 "U-Boot_env"
mtd3: 00040000 00020000 "U-Boot_env_backup"
mtd4: 00020000 00020000 "DTB"
mtd5: 00500000 00020000 "Kernel"
mtd6: 0f9a0000 00020000 "UBI-Device"
```

Les 7 partition s'affiche



j'efface la memoire avec : nand erase 0x660000 0xF9A0000
J'utilise cette commande pour crée les peripherique UBI :
> ubiattach -m 6 /dev/ubi_ctrl

```
/ # ubiattach -m 6 /dev/ubi_ctrl
ubi0: attaching mtd6
ubi0: scanning is finished
ubi0: attached mtd6 (name "UBI-Device", size 249 MiB)
ubi0: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
ubi0: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
ubi0: VID header offset: 2048 (aligned 2048), data offset: 4096
ubi0: good PEBs: 1974, bad PEBs: 23, corrupted PEBs: 0
ubi0: user volume: 0, internal volumes: 1, max. volumes count: 128
ubi0: max/mean erase counter: 2/1, WL threshold: 4096, image sequence number: 1403185545
ubi0: available PEBs: 1953, total reserved PEBs: 21, PEBs reserved for bad PEB handling: 17
ubi0: background thread "ubi_bgt0d" started, PID 636
/ #
```

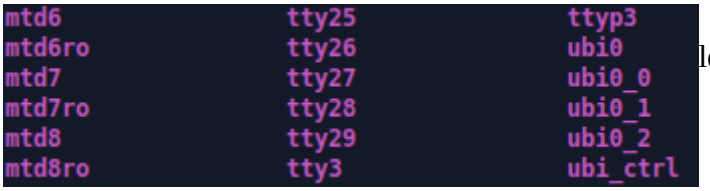
[Q5] : pour chaque type de volume UBI, indiquez son type (statique ou dynamique) et justifiez chacun de vos choix.

Chacune des partition seront dynamique afin que ce soit le FS qui gère l'intégrité des donnée.
Pour connaître la taille de partition, je commence par regarder la taille du dossier nfsroot sur ma machine. Le dossier fait 10.6M, je vais allouer 12M pour avoir de la marge.

Ensuite je ne peux pas vraiment prévoir la taille de DataFS car cela vas dépendre de la taille des donnée envoyé sur le serveur web. Je passe a la dernière partition, MountFS, l'image de la clée USB que vous nous avez donnée fait 33MB, je vais donc réservé 34MB pour avoir un peux de marge au cas ou.
Le UBI-Device fait 249MB, donc pour connaître la taille de DataFS, je fait 249-12-34=203MB

En resumé :
Root FS → 12MB dynamic
Data FS → 203MB dynamic
Mount FS → 34MB dynamic (ou le reste de memoire dispo donc -m)

Sur le système embarqué, je crée les volume avec ces lignes :
> ubimkvol -N RootFs -s 12MiB -t dynamic /dev/ubi0
> ubimkvol -N DataFS -s 203MiB -t dynamic /dev/ubi0
> ubimkvol -N MountFS -m -t dynamic /dev/ubi0



les nouveau device UBI apparaissent.

Maintenant il faut que je crée des image UBI a mettre dans ces partitions :
sur le PC j'exécute ces commande :

Pour connaître la taille de l'argument -c je fais (12* 2^20) / 126976 = 99.096 → 100
> mkfs.ubifs -m 2048 -e 126976 -c 100 -r ~/Desktop/racine/tinysystem/nfsroot imgRoot.img

Pour le dataFS, et le MountFS, je crée un répertoire vide, et l'utilise pour crée l'image UBI.
Pour DataFS : (203* 2^20) / 126976 = 1676.387 → 1677
> mkfs.ubifs -m 2048 -e 126976 -c 1677 -r ~/Desktop/racine/tinysystem/empty imgData.img

Pour Mount FS : (34* 2^20) / 126976 = 280.774→ 281
> mkfs.ubifs -m 2048 -e 126976 -c 281 -r ~/Desktop/racine/tinysystem/empty imgMount.img

Ensuite je transfert les image sur le systeme embarquée en passant par la page web.
Et je «flash» les image sur la nand :
> ubiupdatevol /dev/ubi0_0 www/upload/files/imgRoot.img
> ubiupdatevol /dev/ubi0_1 www/upload/files/imgData.img
> ubiupdatevol /dev/ubi0_2 www/upload/files/imgMount.img

Je redémarre la carte, et stop le démarrage a U-Boot afin de modifier le bootargs (le rootfs) :
je modifie bootargs :

setenv bootargs 'root=ubi0:RootFs rootfstype=ubifs ubi.mtd=6 ip=192.168.2.42:::::eth0 rootwait mtdparts=atmel_nand:256k(AT91Bootstrap)ro,512k(U-Boot)ro,256k(U-Boot_env),256k(U-Boot_env_backup),128k(DTB)ro,5M(Kernel)ro,-(UBI-Device)'

puis saveenv…

maintenant je peux essayer de démarré le système embarqué, donc j'effectue un reset :

```
UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "RootFs", R/O mode
UBIFS (ubi0:0): LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes: 2048 bytes/2048 bytes
UBIFS (ubi0:0): FS size: 11427840 bytes (10 MiB, 90 LEBs), journal size 1904640 bytes (1 MiB, 15 LEBs)
UBIFS (ubi0:0): reserved for root: 0 bytes (0 KiB)
UBIFS (ubi0:0): media format: w4/r0 (latest is w4/r0), UUID 4196E5A9-6EB5-4C2B-BC1A-F7080C6A12CE, small LPT model
VFS: Mounted root (ubifs filesystem) readonly on device 0:13.
```

le systeme a bien démarré avec le rootfs de ubi0

Il me reste a modifier le serveur web afin que les fichier soit sauvegarder sur le volume DataFS, et non sur la carte SD.
Mais comme le systeme est monté en Read-Only je ne peut pas modifier directement le fichier rcS.
Donc je vais sur le PC ou etait sauvegarder le rootFS, puis je modifie le fichier rcS :

```
#mount /dev/mmcblk0p2 -t f2fs /www/upload/files/
mount -t ubifs ubi0_1:DataFs /www/upload/files/
```

Je crée une nouvelle image rootFS.img avec la commande :
> mkfs.ubifs -m 2048 -e 126976 -c 100 -r ~/Desktop/racine/tinysystem/nfsroot imgRoot.img

Pour mettre a jour l'image je doit me mettre sur le squashfs qui etait sur la carte SD, donc je modifie le bootargs.
> editenv bootargs root=/dev/mmcblk0p3 ip=192.168.2.42:::::eth0 rootwait mtdparts=atmel_nand:256k(AT91Bootstrap)ro,512k(U-Boot)ro,256k(U-Boot_env),256k(U-Boot_env_backup),128k(DTB)ro,5M(Kernel)ro,-(UBI-Device)
> saveenv
> reset

Je redémarre en utilisant la carte SD. J'upload le fichier rootFs.img, puis je met a jour l'image sur la memoire de la carte :
> ubiattach -m 6 /dev/ubi_ctrl
> ubiupdatevol /dev/ubi0_0 /www/upload/files/imgRoot.img

Je redémarre la carte, et stop le démarrage a U-Boot afin de modifier le bootargs (le rootfs) :
je modifie bootargs :

> setenv bootargs 'root=ubi0:RootFs rootfstype=ubifs ubi.mtd=6 ip=192.168.2.42:::::eth0 rootwait mtdparts=atmel_nand:256k(AT91Bootstrap)ro,512k(U-Boot)ro,256k(U-Boot_env),256k(U-Boot_env_backup),128k(DTB)ro,5M(Kernel)ro,-(UBI-Device)'

> saveenv
> reset

ça ne fonctionne pas :

```
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 3 ports detected
UBIFS error (pid: 636): cannot open "ubi0_1:DataFs", error -22mount: mounting ubi0_1:DataFs on /www/upload/files/ failed: Invalid argument
```

Apres avoir fais des test, je me rend compte qu'il faut utiliser cette ligne :
> mount -t ubifs /dev/ubi0_1 /www/upload/files/
je modifie le fichier rcS, et refait toute la manipulation afin de mettre a jour le device rootFs

```
UBIFS (ubi0:1): background thread "ubifs_bgt0_1" started, PID 638
UBIFS (ubi0:1): recovery needed
UBIFS (ubi0:1): recovery completed
UBIFS (ubi0:1): UBIFS: mounted UBI device 0, volume 1, name "DataFS"
UBIFS (ubi0:1): LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes: 2048 bytes/2048 bytes
UBIFS (ubi0:1): FS size: 211542016 bytes (201 MiB, 1666 LEBs), journal size 9023488 bytes (8 MiB, 72 LEBs)
UBIFS (ubi0:1): reserved for root: 0 bytes (0 KiB)
UBIFS (ubi0:1): media format: w4/r0 (latest is w4/r0), UUID FCA7B75F-97D2-441F-BB45-BD33FD308265, small LPT model

Please press Enter to activate this console.
```

Ca marche,je teste d'envoyer un fichier par la page web, et il apparaît bien sur le système embarqué.
Il ne reste plus qu'as faire fonctionner la clé USB.

Au debut je ne comprenait pas comment faire car pour monter la clé, car il me faut d'abord crée un dossier dans media, mais je suis en lecture seule…
Donc je me suis dit que je pourrais monter la dernière partition sur media, puis y crée les dossier qui permettent de monter la clé USB…

J'essaie de monter la derniere partition sur media avec cette commande :
> mount -t ubifs /dev/ubi0_2 /media/

Puis je branche la clé USB, pour voir si ça fonctionne.
Tout fonctionne, donc il ne me reste plus qu'as modifier une dernière fois le fichier rcS.
J'ajoute cette ligne dans le fichier rcS :

```
mount -t ubifs /dev/ubi0_1 /www/upload/files/
mount -t ubifs /dev/ubi0_2 /media/
mount -t tmpfs varlog /var/log -o size=16M
```

Je recrée un image de rootFS, et je met a jour avec les même manipulation qu'avant.

```
UBIFS (ubi0:1): FS size: 211542016 bytes (201 MiB, 1666 LEBs), journal size 9023488 bytes (8 MiB, 72 LEBs)
UBIFS (ubi0:1): reserved for root: 0 bytes (0 KiB)
UBIFS (ubi0:1): media format: w4/r0 (latest is w4/r0), UUID FCA7B75F-97D2-441F-BB45-BD33FD308265, small LPT model
UBIFS (ubi0:2): background thread "ubifs_bgt0_2" started, PID 641
UBIFS (ubi0:2): recovery needed
UBIFS (ubi0:2): recovery completed
UBIFS (ubi0:2): UBIFS: mounted UBI device 0, volume 2, name "MountFS"
UBIFS (ubi0:2): LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes: 2048 bytes/2048 bytes
UBIFS (ubi0:2): FS size: 21078016 bytes (20 MiB, 166 LEBs), journal size 4825088 bytes (4 MiB, 38 LEBs)
UBIFS (ubi0:2): reserved for root: 0 bytes (0 KiB)
UBIFS (ubi0:2): media format: w4/r0 (latest is w4/r0), UUID FA2B7176-F95A-4F13-8110-8F68C50EF8F1, small LPT model
```

Le volume 2 se monte bien dans media.
Lorsque je branche une clé USB elle se monte dans media, puis copie les fichier dans le dossier files sur la page web..

Tout fonctionne.

TP8

Objectif :
Écrire un petit driver pour contrôler la carte mylab1 avec Linux.

Configuration de U-Boot pour utiliser le serveur NFS :
Je modifie le bootargs :
> setenv bootargs ‘root=/dev/nfs nfsroot=192.168.2.1:/home/yoda/Desktop/racine/tinysystem/nfsroot,v3 rw ip=192.168.2.42:::::eth0 rootwait mtdparts=atmel_nand:256k(AT91Bootstrap)ro,512k(U-Boot)ro,256k(U-Boot_env),256k(U-Boot_env_backup),128k(DTB)ro,5M(Kernel)ro,-(UBI-Device)‘

```
hub 1-0:1.0: 3 ports detected
UBIFS error (pid: 617): cannot open "/dev/ubi0_1", error -22mount: mounting /dev/ubi0_1 on /www/upload/files/ failed: Invalid argument

UBIFS error (pid: 618): cannot open "/dev/ubi0_2", error -22mount: mounting /dev/ubi0_2 on /media/ failed: Invalid argument

Please press Enter to activate this console.
/ #
```

Ça fonctionne, mais le kernel n’est pas content a propos du montage des partition ubifs.

Connections de la carte :

PE9	--	--	--	GPIO
PE10	--	--	--	GPIO
PE11	--	--	--	GPIO
PE12	--	--	--	GPIO
PE16	--	--	--	GPIO
PE31	--	--	--	IRQ/PWML1

Je vais utiliser les pins PE10,11,12.
PE10 → CONN2.23 (Button)
PE11 → CONN2.25 (Left)
PE12 → CONN2.27 (Right)

Configuration des GPIO :
1) Export les GPIOs nécessaires (position gauche, position droite, boutton)
pin PE10 : PE10 = pin 138 (128+10)
> echo 138 > /sys/class/gpio/export
> echo 139 > /sys/class/gpio/export
> echo 140 > /sys/class/gpio/export

2) Configuration des PIN en lecture :
> echo in > /sys/class/gpio/pioE10/direction
> echo in > /sys/class/gpio/pioE11/direction
> echo in > /sys/class/gpio/pioE12/direction

3) Écriture du script qui affiche l'état des GPIO :

/home/yoda/Desktop/racine/tinysystem/nfsroot/show_GPIO.sh - !

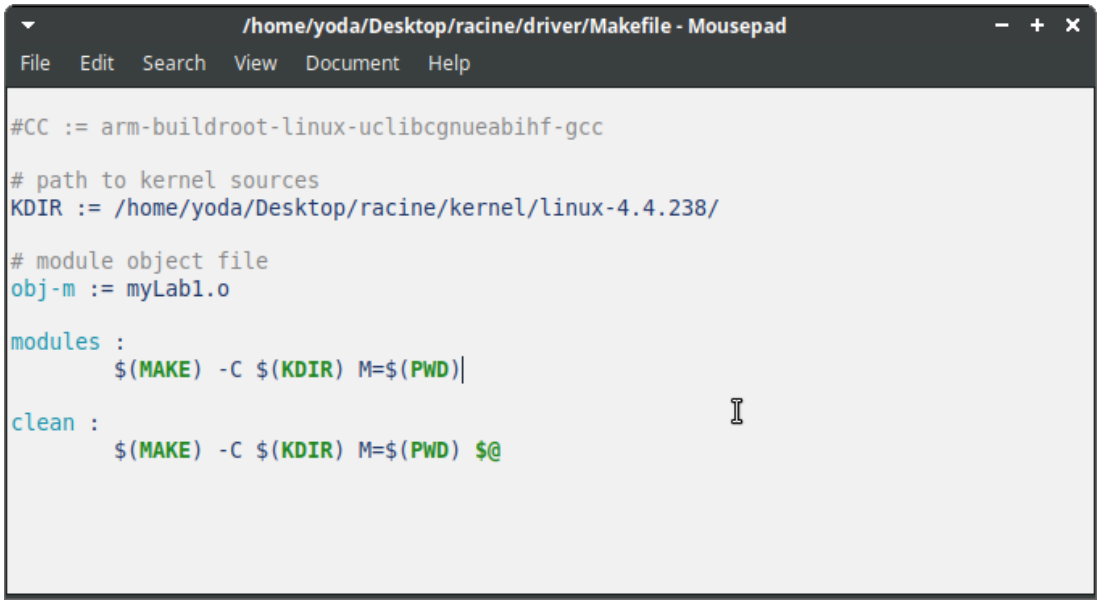
File Edit Search View Document Help

```
#! /bin/bash


#init GPIO
echo 138 > /sys/class/gpio/export
echo 139 > /sys/class/gpio/export
echo 140 > /sys/class/gpio/export
echo in > /sys/class/gpio/pioE10/direction
echo in > /sys/class/gpio/pioE11/direction
echo in > /sys/class/gpio/pioE12/direction

my_func () {
var_pe10="0"
var_pe11="0"
var_pe12="0"

echo "BUTTON | LEFT | RIGHT"
while true
do
        var_pe10=`cat /sys/class/gpio/pioE10/value`
        var_pe11=`cat /sys/class/gpio/pioE11/value`
        var_pe12=`cat /sys/class/gpio/pioE12/value`
        echo "  $var_pe10    |   $var_pe11  |  $var_pe12"
        sleep 0.1
done
}

function ctrl_c() {
        echo 138 > /sys/class/gpio/unexport
        echo 139 > /sys/class/gpio/unexport
        echo 140 > /sys/class/gpio/unexport
        echo "Unexport GPIO and quit"
        exit 0

}

trap ctrl_c INT

my_func

exit 0
```

```
/ # ./show_GPIO.sh
BUTTON | LEFT | RIGHT
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
  1    |   1  |  1
^CUnexport GPIO and quit
/ # echo $?
0
/ #
```

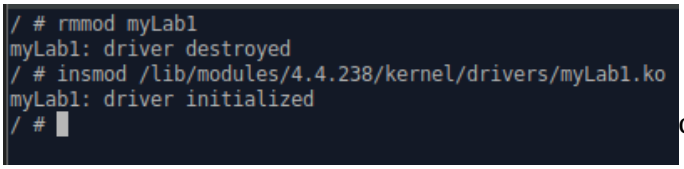
Écriture du driver :

Voici le makefile du projet.



Pour compiler, j'utilise la commande :
> make modules CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm

Pour installer le module :
> insmod /lib/modules/4.4.238/kernel/drivers/myLab1.ko



→ le driver s'installe et se désinstalle correctement

Voilà ce qu'affiche l'ouverture de /dev/myLab1

→ ici j'ai appuyé sur le bouton central

Utilisation du module depuis l'espace utilisteur :

Voici le code que j'ai écrit pour utiliser la carte myLab1

```c
int main()
{
  char buf[4];
  int count=0;
  int fd = open("/dev/myLab1", O_RDWR | O_SYNC );
  if(fd==-1){printf("ERROR Opening myLab1\n");return 1;}
  while(1)
  {
    count = read(fd, buf, 4);
    if(count<0){printf("ERROR READING myLab1\n");}
    else
    {
      if(buf[0]=='0'){printf("button\n");}
      else if(buf[1]=='0'){printf("left\n");}
      else if(buf[2]=='0'){printf("right\n");}
      else{}
    }
    usleep(10000);
  }
  return 0;
}
```

```
left
left
left
left
right
right
right
right
right
right
right
right
right
button
button
button
button
button
button
```

J'ai aussi du réécrire la fonction read() de mon driver :

```c
static ssize_t dev_read( struct file *filep, char *buffer, size_t len, loff_t *offset) {
  // TODO
  // - Read the joystick state (ie. write it to the user space buffer)
  //printk(KERN_INFO "%d %d %d\n",p10_val,p11_val,p12_val);
  //printk(KERN_INFO "myLab1: BUTTON = %d | LEFT = %d | RIGHT = %d\n",p10_val,p11_val,p12_val);
  //printk(KERN_INFO "myLab1: device read\n");
  joystick[0]=p10_val+'0';
  joystick[1]=p11_val+'0';
  joystick[2]=p12_val+'0';
  joystick[3]=0;
  return copy_to_user(buffer, joystick, 4);;
}
```

Compilation croisée de la librairie ncurses

Je commence par téléchargé ncurse ici : https://invisible-mirror.net/archives/ncurses/ncurses-6.2.tar.gz

Ensuite pour y configuré, j'exécute les commande :

> ./configure --host=arm-linux --prefix=/usr --without-progs --with-shared CC=arm-buildroot-linux-uclibcgnueabihf-gcc

> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc

> make DESTDIR=~/Desktop/tmp_ncurse/ install

Je déplace les fichier qui nous sont utile dans nfsroot :

> cp -a usr/lib /home/yoda/Desktop/racine/tinysystem/nfsroot/usr/

> cp -a usr/share /home/yoda/Desktop/racine/tinysystem/nfsroot/usr/

Compilation croisée de nInvaders :

Je le télécharge a cette adresse : https://sourceforge.net/projects/ninvaders/files/latest/download
Je modifie le makeFile :

/home/yoda/Desktop/ninvaders-0.1.1/Makefile - Mousepad

File Edit Search View Document Help

```
CC=arm-buildroot-linux-uclibcgnueabihf-gcc
CFLAGS=-O3 -Wall
LIBS=-lncurses -I/home/yoda/Desktop/tmp_ncurse/usr/include -L/home/yoda/Desktop/tmp_ncurse/usr/lib

CFILES=globals.c view.c aliens.c ufo.c player.c nInvaders.c
HFILES=globals.h view.h aliens.h ufo.h player.h nInvaders.h
OFILES=globals.o view.o aliens.o ufo.o player.o nInvaders.o
all:            nInvaders

nInvaders:      $(OFILES) $(HFILES)
                $(CC) $(LDFLAGS) -o$@ $(OFILES) $(LIBS)

.c.o:
                $(CC) -c  -I. $(CFLAGS) $(OPTIONS) $(LIBS) $<
clean:
                rm -f nInvaders $(OFILES)
```

Pour compiler le programme, j'ai du ajouté dans la variable LIBS, l'options -I et -L.
En plus de ça, j'ai ajouter $(LIBS) dans .c.o

Bugfix de nInvaders :

```
45  #define ALIENS_MAX_NUMBER_X 10
46  #define ALIENS_MAX_NUMBER_Y 5
47  #define ALIENS_MAX_MISSILES 10
48
49  // todo: move to structure
50  int lowest_ship[ALIENS_MAX_NUMBER_X];
51  int alienshotx[ALIENS_MAX_MISSILES];
52  int alienshoty[ALIENS_MAX_MISSILES];
```

Dans aliens.h

→ donc ici la taille du tableau est de 10 (0 a 9)

Dans aliens.c :

```
166     for (k=0;k<11;k++) {
167       lowest_ship[k]=-1;
168     }
```

→ mais ici l'acces au tableau vas de 0 a 10, ce qui pourrais crée une erreur

Donc je modifie le code afin de ne pas depassé la taille du tableau.

```
166     for (k=0;k<ALIENS_MAX_NUMBER_X;k++) {
167       lowest_ship[k]=-1;
168     }
```

Exécution de nInvaders :

Je transfert le jeux a la racine de nfsroot, puis je l'exécute. > ./nInvaders
le jeux s'exécute, mais est très lent.

[Q1] : pourquoi à votre avis ?

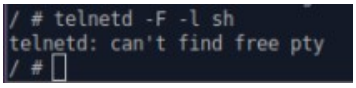
Car l'interface du jeux est transféré par l'USB a 115200 bit/s, ce qui n'est pas très rapide.

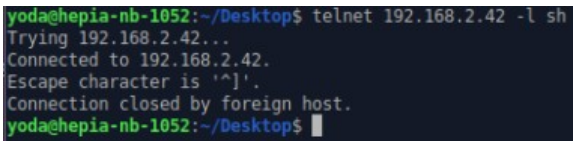
Utilisation de telnet :
Dans mon cas, telnetd est déjà activé dans le kernel. Donc j'execute la commande :
> telnetd -F -l sh

[Q2] : que réalisent les différentes options ?
-F → run in foreground
-l → demande un login a la connexion (ici le login sera sh)

Pour m'y connecté avec le PC, j'utilise cette commande :
> telnet 192.168.2.42 -l sh

Mais j'obtiens une erreur :





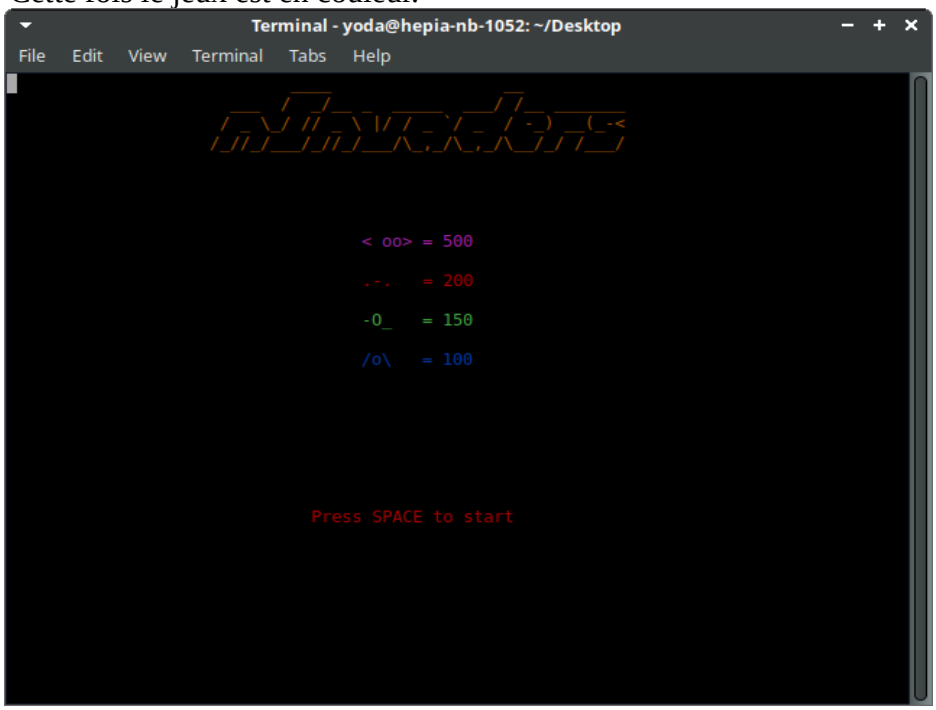
Afin que la connection fonctionne, j'execute la commande suivante :
> mount -t devpts devpts /dev/pts

J'obtiens une erreur car pts n'existe pas. Donc je le crée
> mkdir /dev/pts

J'exécute la commande de monture, cette fois je n'ai pas d'erreur.
Et je relance le serveur telnetd.
Tout fonctionne, et le jeux s'exécute de manière beaucoup plus fluide.
Il ne reste plus qu'as mettre les couleur.

A partir de la fenetre telnet j'exécute la commande :
> TERM=xterm-color

Cette fois le jeux est en couleur.



Modification de nInvaders pour utiliser le joystick

Les modification sont a faire dans la fonction readInput() qui se trouve dans nInvaders.c
Pour faire cette partie je réutilise le code que j'avais fait pour lire le driver myLab1.

Voici le code que j'ai écrit :

```
149   int fd = open("/dev/myLab1", O_RDWR | O_SYNC );
150   if(fd==-1)
151   {
152     ch = getch(); //si myLab1 n'existe pas, utilse le clavier
153   }
154   else//utilise le driver myLab1
155   {
156     count = read(fd, buf, 4);
157     button=buf[0]-'0';
158     left=buf[1]-'0';
159     right=buf[2]-'0';
160     while(!buttonPressed)
161     {
162       count = read(fd, buf, 4);
163       if(count<0){printf("ERROR READING myLab1\n");}
164       else
165       {
166         old_button=button;
167         old_left=left;
168         old_right=right;
169         button=buf[0]-'0';
170         left=buf[1]-'0';
171         right=buf[2]-'0';
172         if((old_button==1)&&(button==0)){ch=' ';buttonPressed=1;}
173         else if((old_left==1)&&(left==0)){ch=KEY_LEFT;buttonPressed=1;}
174         else if((old_right==1)&&(right==0)){ch=KEY_RIGHT;buttonPressed=1;}
175         else{ch=0;}
176       }
177       usleep(10000);
178     }
179     close(fd);
180   }
```

→ si myLab n'est pas disponible utilise le code de base (le clavier)

→ initialise la valeur des bouton

→ sauvegarde l'ancienne valeur du bouton pour détecter un flanc

→ détecte un flanc

→ évite les rebond du bouton

Création des Patch :
Je reprend le dossier Original, et refait les modification de correction de bug, ensuite j'execute la commande :

> diff -u Original/ninvaders-0.1.1/ ninvaders-0.1.1/ > correct_bug.patch

Je reprend encore un dossier original de ninvader, et fais les modification uniquement pour le joystick. Puis j'exécute la commande suivant pour crée le patch :

> diff -u Original/ninvaders-0.1.1/ ninvaders-0.1.1/ > add_joystick.patch

Ensuite pour appliquer les patch il suffit d'utiliser les commande :
> patch -p0 < ../correct_bug.patch
> patch -p0 < add_joystick.patch.patch