TP 1 : Compilation du noyau

1ère Partie. Création de la chaine de compilation croisée (cross-compilation)

Pour créer cette chaine de compilation, nous allons utiliser « buildroot ».
Afin de créer un système léger, nous utiliserons la bibliothèque C ***uclibc*** *au lieu de* ***glibc****, d'où l'utilisation de buildroot.*

Cet outil permet aussi de créer entièrement un système embarqué (compilation du noyau..), mais nous nous limiterons à la chaine de cross-compilation de façon à découvrir les différentes étapes.

REMARQUE PREALABLE : si les outils de téléchargement **wget**, **git** ... ne fonctionnent pas correctement du fait du proxy, se référer à l'**ANNEXE « configuration des outils »**

Liens :
- bibliothèque uclibc : https://uclibc.org/
- buildroot : https://buildroot.org/

Q1. Récupérer l'archive de buildroot.

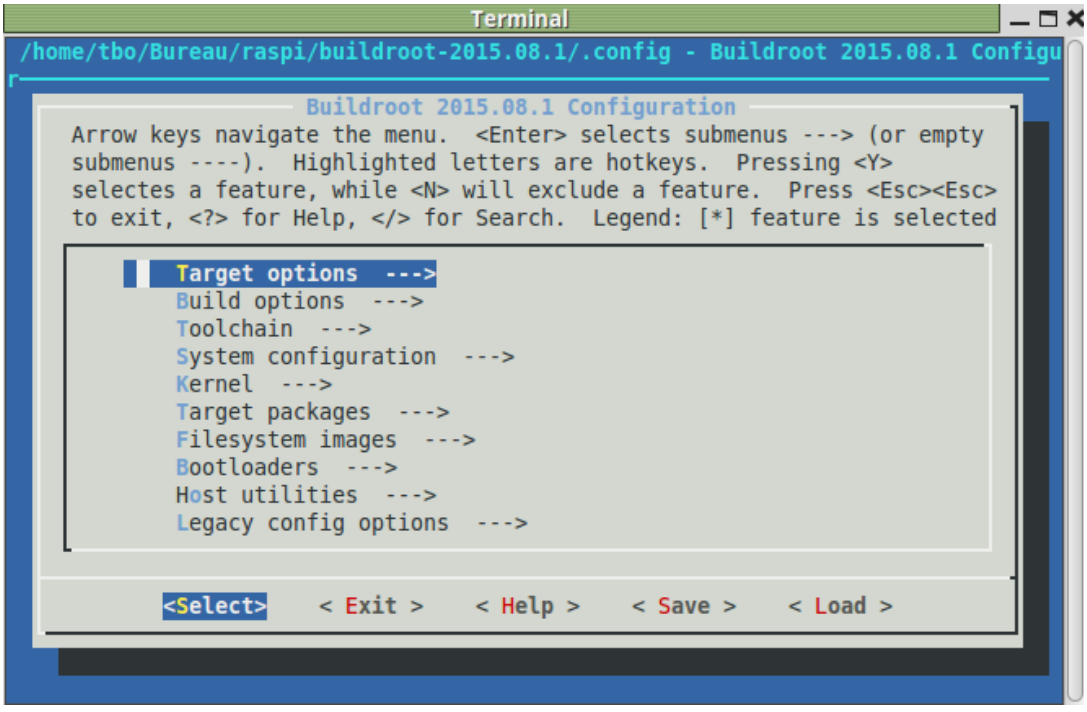
- Depuis un terminal, créer un répertoire dans /home/"user" appelé « **raspi_emb ».**

Dorénavant, nous travaillerons dans ce répertoire

- Télécharger la dernière version de buildroot : (**wget buildroot-«date».tar.bz2**).
- Décompresser le fichier (à l'aide de la commande **tar jxvf …**)
- entrer dans le répertoire (**cd buildroot…**)

Q2. Configurer buildroot

*Buildroot, comme le noyau et d'autres outils peut être configuré par un système de menu. Le résultat de la configuration se trouve dans le fichier « **.config** ». Pour accéder à ce système de menu, il est nécessaire d'installer le paquet* ***libncurses5-dev***



Depuis le répertoire **buildroot-<date>** :

- installer le paquet libncurses5-dev (« **apt-get install libncurses5-dev** »)
- télécharger le fichier de configuration **config-buildroot-tbo**
- remplacer le fichier actuel .config par votre fichier de configuration (**cp .config-tbo .config**)
- Pour accéder à la configuration, utiliser la commande **make menuconfig** (ou si nécessaire (**sudo make menuconfig**),
- dans le menu « **Target options** → » , vérifier que :
  - la cible est bien un processeur ARM (**Target Architecture**),
  - le processeur correspond bien à la carte Raspberry B+ (PRECISEZ le type de processeur)
  (voir https://en.wikipedia.org/wiki/Raspberry_Pi#Specifications)

- dans le menu « **Build options** », vérifier que la chaîne de compilation sera bien installée dans le répertoire **/usr/local/cross-rpi**)

Q3. Compilation

- compiler la chaîne de cross-compilation à l'aide de la commande **sudo make** (nous utilisons sudo pour avoir accès au répertoire **/usr/local/** du système)

Nous disposons maintenant d'une chaîne de compilation croisée dans le répertoire
/usr/local/cross-rpi

2ème partie. Le noyau linux.

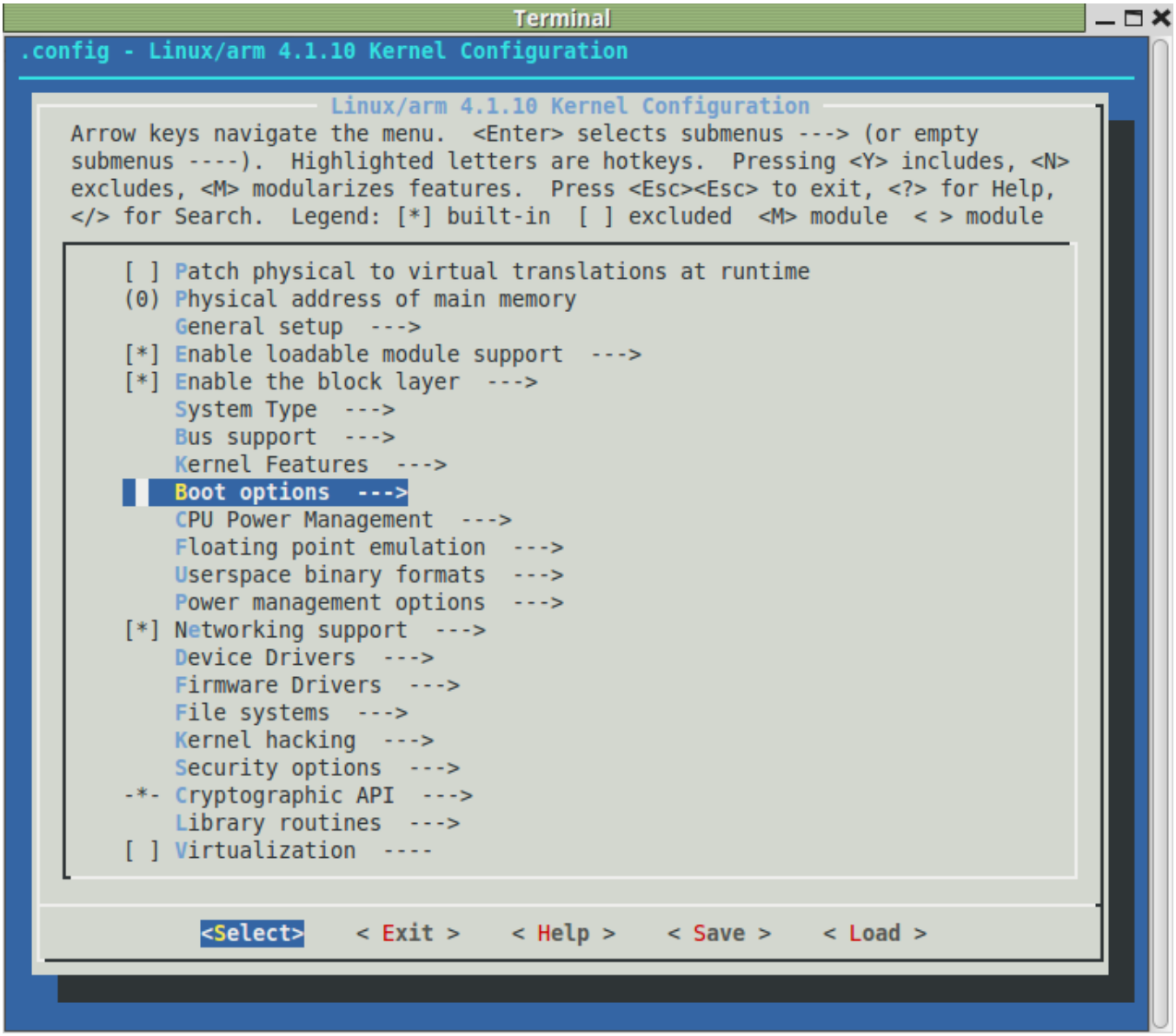
*Le noyau standard (http://www.kernel.org) n'intègre pas par défaut le support Raspberry. Plutôt que de récupérer le noyau standard et lui appliquer des patchs, nous allons récupérer le noyau maintenu par la **Raspberry Pi Foundation**.*

Q4. Téléchargement du noyau
- se placer dans le répertoire : **/home/« user »/raspi_emb**
- Télécharger le noyau : (**git clone https://github.com/raspberrypi/linux linux-raspi**)

*Nous allons compiler un noyau avec modules. Cependant, il faudra vérifier que les drivers nécessaires au démarrage soient compilé en « dur » dans le noyau. Les modules non nécessaires au démarrage seront appelés par l'outil **modprobe** si besoin.*
*Nous utiliserons une partition formatée en **ext4** pour stocker le système*

Q5. Configuration du noyau.

- entrer dans le répertoire **linux-raspi**
- télécharger le fichier **config-linux-tbo**,
- remplacer le fichier de configuration **.config** par le fichier précédemment téléchargé.
- lancer la configuration du noyau en précisant que nous allons utiliser une architecture ARM :
  make ARCH=arm menuconfig



- dans la **configuration** :
    - vérifier si la compilation des drivers sous forme de module est activée ?
    - quel est le système de fichier choisi par défaut (=qui n'est pas compilé en module) ?
    - vérifier si le noyau utilise bien le format de binaires ELF ?
    - modifier la version du noyau (General Setup → local version) en mettant vos initiales.

Sur un système embarqué, nous n'avons généralement pas de sortie vidéo/claviers. Nous allons donc demander au noyau d'envoyer les messages sur le port série ***console=ttyAMA0,115200n8*** *(port série ttyAMA0, à 115200 bauds, sans bit de parité, par paquets de données 8 bit),*

*La partition racine sera la 2ème de la carte sd (****root=/dev/mmcblk0p2****) et sera une partition de type ext4 (****rootfstype=ext4****)*

*Le système doit attendre (indéfiniment si nécessaire) que la partition soit prête au démarrage (****rootwait****)*

- vérifier si la configuration des paramètres de démarrage du noyau est correcte dans « **Boot options** → »
- sortie de la configuration (**<Exit>**) et enregistrer si nécessaire.

Q6. Compiler le noyau

Pour compiler le noyau, nous allons utiliser la chaîne de cross-compilation préparée dans la 1ère partie de façon à obtenir un noyau exécutable par la carte Raspberry Pi et son processeur ARM.
Le préfixe de la chaine de cross-compilation est ***/usr/local/cross-rpi/usr/bin/arm-linux-***

- compiler le noyau : **make ARCH=arm CROSS_COMPILE=/usr/local/cross-rpi/usr/bin/arm-linux-**
- en fin de compilation, vérifier la présence du noyau qui se nomme zImage (« z » car compressé avec la zlib). Ce dernier se trouve dans le répertoir **arch/arm/boot/**

```
tbo@VirtualMINT ~/Bureau/raspi/linux_raspi $ ls -l arch/arm/boot/zImage
-rwxr-xr-x 1 tbo tbo 4060616 déc.   5 12:25 arch/arm/boot/zImage
```

3ème Partie. Préparer la carte SD

Il est nécessaire de créer 2 partitions sur la carte SD :
- une partition (de ***128 Mo****) en* ***vfat*** *qui contiendra le bootloader et l'image du noyau,*
- une partition (le ***reste*** *de la carte) en* ***ext4*** *qui contiendra le système racine (/root)*

Pour créer les partitions, nous allons utiliser l'outil ***fdisk****. Lorsque cet outil est lancé, nous avions accés à l'aide via la commande 'm'*

Q7. Supprimer les partitions existantes

- Insérer la carte SD dans le lecteur
- à l'aide de la commande **dmesg**, vérifier quel est le lecteur associé à la carte (si il n'y a qu'un disque en SATA, cela devrait être **/dev/sdb**),
- dans le terminal, supprimer les partitions de la carte SD avec fdisk (sudo fdisk /dev/sdX, X suivant le périphérique associé à la carte).

Q8. Créer les 2 partitions

- créer une 1ère partition de 128Mo en **fat32**. Il faut aussi la rendre « bootable »,
- créer une 2de partition qui occupera le reste de la carte en **ext4**
- formater la 1ère partition en la nommant **Boot** (voir « man mkfs.vfat » + volume name).
- formater la 2de partition en la nommant **Root** (voir « man mkfs.ext4 » + volume Label).

5ème partie. Installer le bootloader et le noyau sur la carte

Le système raspberry Pi utilise un bootloader particulier qui va être installé dans la partition **/Boot** de la carte SD.

Q9. Installer le bootloader
- se placer dans le répertoire **/home/« user »/raspi_emb**
- télécharger le bootloader : **git clone https://github.com/raspberrypi/firmware**
Nous obtenons les fichiers suivants dans le répertoire firmware/boot :

```
tbo@VirtualMINT ~/Bureau/raspi $ ls -l firmware/boot
total 19792
-rw-r--r-- 1 tbo tbo    9846 déc.   7 18:18 bcm2708-rpi-b.dtb
-rw-r--r-- 1 tbo tbo   10125 déc.   7 18:18 bcm2708-rpi-b-plus.dtb
-rw-r--r-- 1 tbo tbo    9850 déc.   7 18:18 bcm2708-rpi-cm.dtb
-rw-r--r-- 1 tbo tbo   11113 déc.   7 18:18 bcm2709-rpi-2-b.dtb
-rw-r--r-- 1 tbo tbo   17900 déc.   7 18:18 bootcode.bin
-rw-r--r-- 1 tbo tbo   18693 déc.   7 18:18 COPYING.linux
-rw-r--r-- 1 tbo tbo    2473 déc.   7 18:18 fixup_cd.dat
-rw-r--r-- 1 tbo tbo    6449 déc.   7 18:18 fixup.dat
-rw-r--r-- 1 tbo tbo    9687 déc.   7 18:18 fixup_db.dat
-rw-r--r-- 1 tbo tbo    9691 déc.   7 18:18 fixup_x.dat
-rw-r--r-- 1 tbo tbo 4035396 déc.   7 18:18 kernel7.img
-rw-r--r-- 1 tbo tbo 4059512 déc.   7 18:18 kernel.img
-rw-r--r-- 1 tbo tbo    1494 déc.   7 18:18 LICENCE.broadcom
drwxr-xr-x 2 tbo tbo    4096 déc.   7 18:18 overlays
-rw-r--r-- 1 tbo tbo  604792 déc.   7 18:18 start_cd.elf
-rw-r--r-- 1 tbo tbo 4866440 déc.   7 18:18 start_db.elf
-rw-r--r-- 1 tbo tbo 2731192 déc.   7 18:18 start.elf
-rw-r--r-- 1 tbo tbo 3820296 déc.   7 18:18 start_x.elf
```

- monter la partition Boot de la carte SD dans le répertoire /**media/Boot**,
- copier les fichiers : **start.elf**, **bootcode.bin** dans la partition **/media/Boot**
- un autre fichier est nécessaire : « **cmdline.txt** » qui permet de donner les options de démarrage au noyau. Comme nous avons déjà réglé les bon pramètres lors de la compilation, il suffit de créer un fichier quasi-vide (avec juste un espace dedans) : **echo ' ' > /media/Boot/cmdline.txt**

ATTENTION : Le noyau compilé s'appelle **zImage** mais il devra être renommé **kernel.img** lorsqu'il sera copié dans la partition **/Boot**

Q10. Installer le noyau
- copier le noyau compilé dans la 2ème partie sur la partition **/media/Boot**.
- démonter la partition **/media/Boot**

6ème partie. Tester le noyau.

*Dans un premier temps, nous allons brancher un écran sur la prise HDMI. Cependant, les systèmes embarqués ne disposent pas toujours de sortie vidéo. Par conséquent, nous utiliserons par la suite le port série (**/dev/tyAMA0**) pour afficher les messages du système Raspberry Pi.*

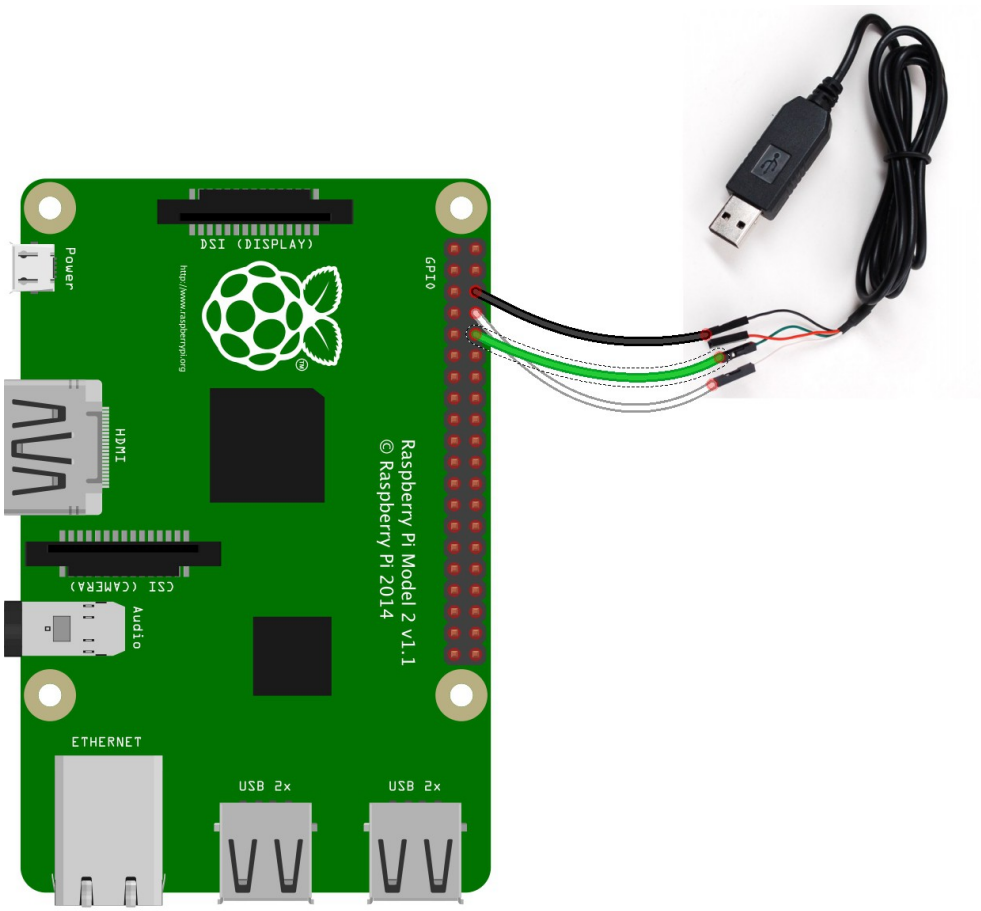
Q11. Tester le système avec sortie vidéo sur écran
- placer la carte SD dans le raspberry Pi, brancher un écran sur la sortie HDMI
- brancher la carte et observer l'information donnée lors du kernel panic
- expliquer pourquoi ce 'kernel panic' apparaît ?

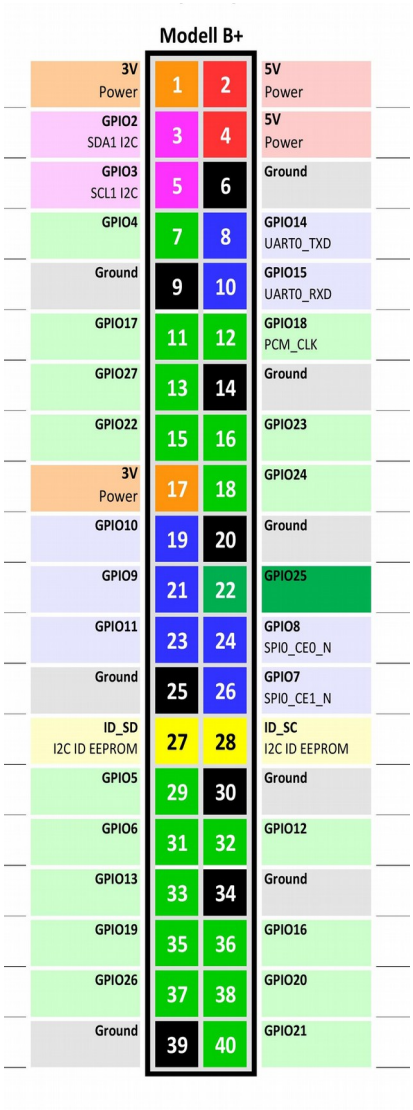
Q12. Test du système via la liaison série
- installer **cutecom** ou gtkterm qui sont des interfaces graphiques pour le terminal série,
- configurer le terminal graphique pour une liaison série :
  port série =/dev/ttyUSB0**,** 115200 bauds, sans bit de parité, par paquets de données 8 bit

ATTENTION : le **GPIO** du raspberry travaille en **3,3V**. Il ne faut surtout brancher un câble avec des sorties en **5V**

Renseignements technique sur le cable utilisé :
There are four wires: red power, black ground, white RX into USB port, and green TX out of the USB port.
The power pin provides the 5V @ 500mA direct from the USB port and the RX/TX pins are 3.3V level for interfacing with the most common 3.3V logic level chipsets.

- alimentation du raspberry débranchée : brancher le câble « USB vers série TTL » sur le GPIO (nous n'avons pas besoin de brancher le 5V rouge car nous alimentons la carte Raspberry par un adaptateur)





- brancher l'alimentation du raspberry et vérifier la communication sur le terminal série via cutecom ou gtkterm

Dans la suite des TP, nous n'utiliserons que la sortie sur terminal série pour l'affichage des messages du noyau linux.