LO52 - A20 - Rapport

Informatique mobile et communications courtes portées

Bastien Gaujour
Enzo Palermo

Responsable d'UV :
Fabien Brisset

utbm
université de technologie
Belfort-Montbéliard

Sommaire

TP 1 : Mise en place de l'environnement de développement **2**
Installation de Git 2
Cahier des charges 2
Rendu de l'application 3

TP 2 : Manipulation d'un Kernel Linux **5**
Installation et configuration 5
Script InstallRepo.sh 7
Script RecupBranch.sh 7
Script TP2CommandsScript.sh 7

TP 3 : Création d'un device Android **8**
Implémentation de la libusb 8
Implémentation nouveau produit 9

TP 4 : Utilisation de JNI **10**
Installation du NDK 10
Les boutons directionnels 10
Les boutons READ/WRITE 10

TP 1 : Mise en place de l'environnement de développement

1) Installation de Git

Nous avons commencé par installer Git puis nous avons cloner le repo avec la commande suivante :
"git clone https://github.com/gxfab/LO52_A2020"
Suite à cela nous nous sommes positionnés sur notre branche
"git branch GaujourBastien_PalermoEnzo" pour créer la branche en local
"git checkout GaujourBastien_PalermoEnzo" pour se positionner sur la branche
"git push -u" pour envoyer la branche sur le serveur distant

Suite à cela nous avons installé Android Studio et nous avons choisi Kotlin comme langage de programmation. Kotlin est une amélioration du java qui lui corrige beaucoup de ses défauts. Son utilisation nous convenant parfaitement nous a donc logiquement conduit à l'utiliser.

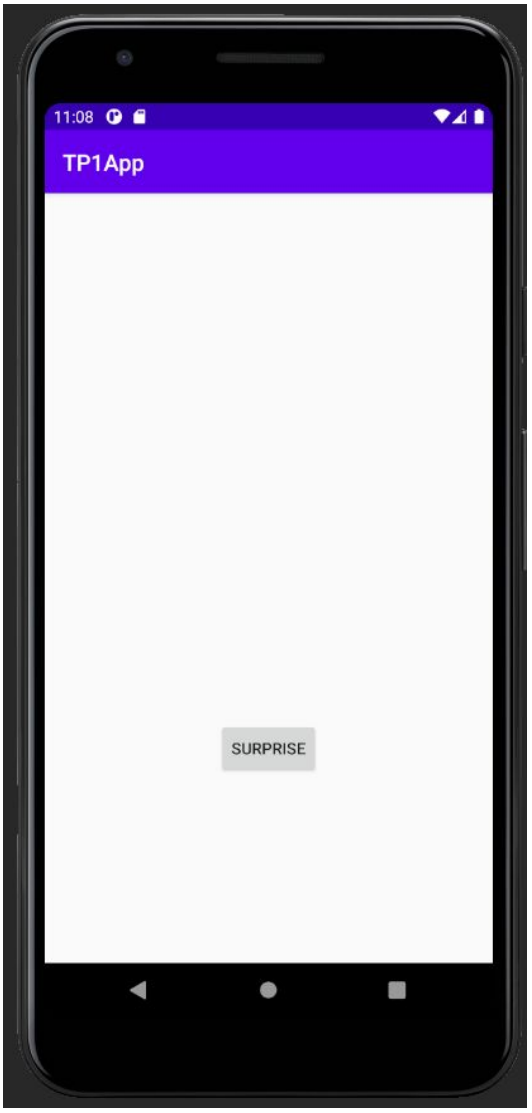
2) Cahier des charges

Avec Android Studio nous devons développer notre première application. Son fonctionnement est assez simple et nous initie à l'utilisation d'Android Studio. Voici les différents éléments composant alors notre application :
Un bouton qui au clic ouvre un nouveau panel avec "hello world"
Nous aurons donc 2 activités, une avec le bouton, une avec le texte "hello world". Le clic sur le bouton lance la 2ème activité.

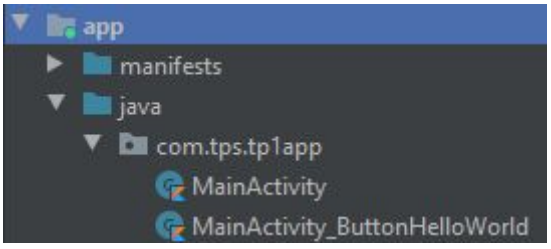
3) Rendu de l’application

Voici donc ce que nous obtenons.

L’application s’ouvre sur cette page où se trouve un bouton cliquable avec marqué “Surprise”



Une fois que l'on a appuyé sur le bouton "Surprise" une nouvelle page s'ouvre avec marqué "Hello World !" dessus.



Voici la structure de l'application

MainActivity.kt = l'activité qui s'ouvre au lancement de l'application
Activity_main.xml = interface de MainActivity.kt
AndroidManifest.xml = décrit les caractéristiques de l'application et défini ses composants

TP 2 : Manipulation d'un Kernel Linux

1) Installation et configuration

Nous avons commencé par télécharger la branche hikey-linaro-android-4.19

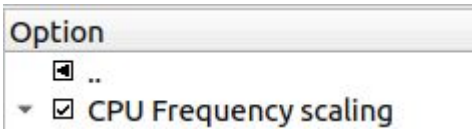
Les étapes jusqu'au chargement de la config ranchu par défaut sont décrites dans les commentaires du script **TP2CommandsScript.sh**

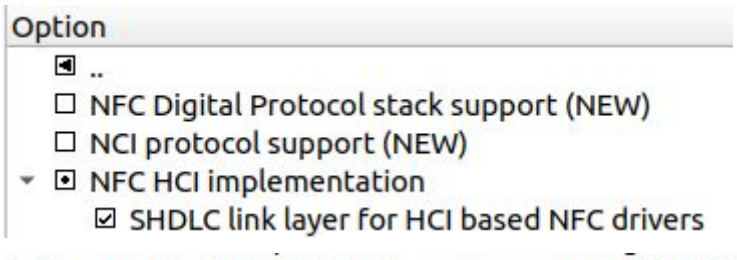
On observe ensuite les différences entre les configurations **hikey_defconfig** et la config chargée pour une carte ranchu avec la commande **"diff arch/arm64/configs/ranchu_defconfig arch/arm64/configs/hikey_defconfig"**
On remarque l'activation d'options liées à **Goldfish** après le chargement de la config **board-ranchu**.

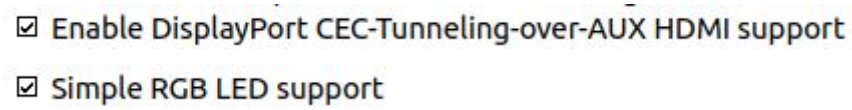
```
295d294
< CONFIG_KEYBOARD_GOLDFISH_EVENTS=y
314d312
< CONFIG_GOLDFISH_TTY=y
340d337
< CONFIG_POWER_RESET_SYSCON_POWEROFF=y
342d338
< CONFIG_BATTERY_GOLDFISH=y
366d361
< CONFIG_FB_GOLDFISH=y
488d482
< CONFIG_VIRTIO_MMIO_CMDLINE_DEVICES=y
496,498d489
< CONFIG_GOLDFISH_AUDIO=y
< CONFIG_GOLDFISH=y
< CONFIG_GOLDFISH_PIPE=y
528a520
> CONFIG_SDCARD_FS=y
541a534
> CONFIG_SECURITY_PERF_EVENTS_RESTRICT=y
550a544
> CONFIG_CRYPTO_SHA512=y
```

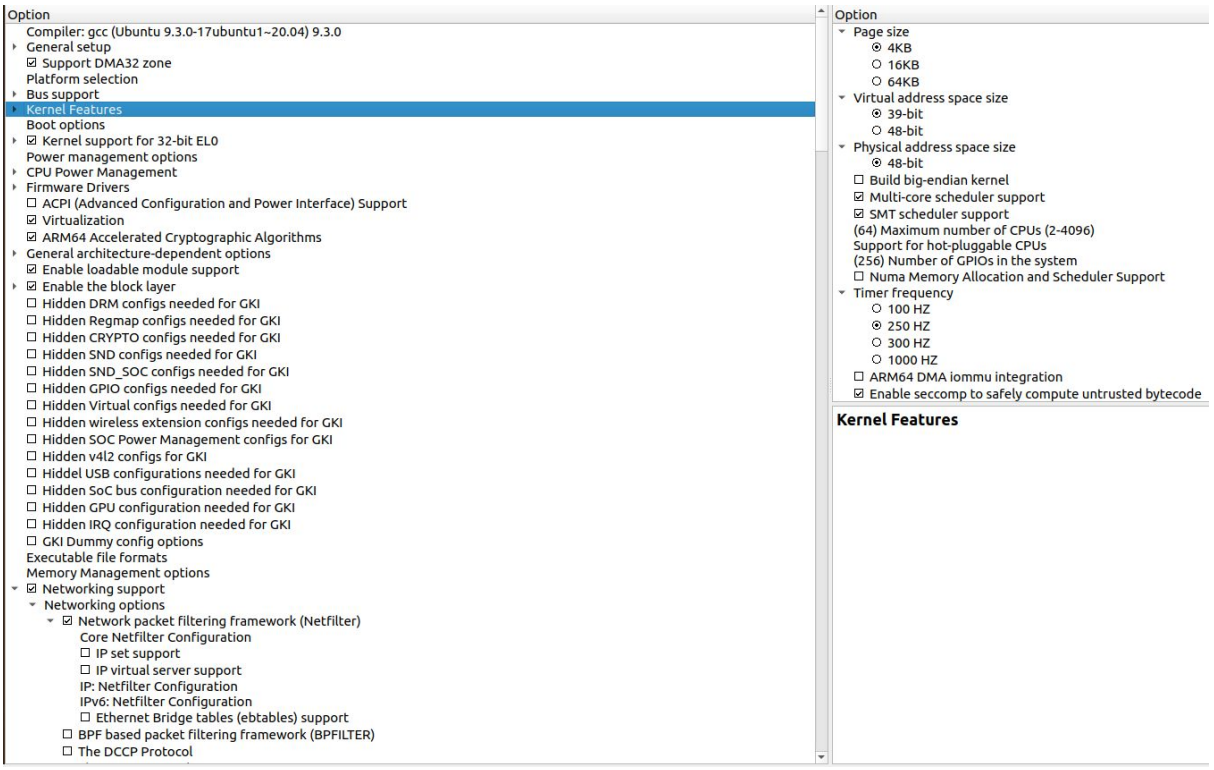
Après installation des **qtdeclarative5-dev**, on utilise "**make xconfig**" pour modifier la configuration précédemment créée. Cette commande permet de faciliter le processus grâce à une GUI sur QT.

Nous avons donc **activé les 4 options demandées** et sauvegardé la nouvelle configuration appelée "**custom**".









Nous n'avons pas su déterminer quelles options pouvaient être qualifiées de superflues, l'objectif de la config n'étant pas clairement défini. De plus, il y en a beaucoup, ce qui complique la tâche.

2) Script InstallRepo.sh

On utilise le **shell bash** car c'est le plus répandu sur linux.

Ce script **crée un répertoire bin**, l'ajoute à la variable **PATH**, l'enregistre dans le fichier .**bashrc** pour une modification permanente, et télécharge le fichier **repo** dans ce répertoire. On donne enfin les droits d'exécution à tous les utilisateurs au fichier repo avec **chmod**.

Note : il faut penser à donner les droits d'exécution au script au préalable avec la commande suivante (depuis le répertoire du script):
"chmod +x InstallRepo.sh"

3) Script RecupBranch.sh

Script intégré au script TP2CommandsScript.sh

4) Script TP2CommandsScript.sh

Ce script **rassemble donc toutes les commandes du TP2**, de la création du dossier android-kernel jusqu'à la création de la nouvelle config par défaut.

TP 3 : Création d'un device Android

L'objectif de ce TP est d'intégrer un composant (libusb du dossier libusb1.0.3) à un produit Android (kernel du TP2, dossier device).

1) Implémentation de la libusb

Les fichiers sources et headers pour la compilation d'une libusb sur Android sont (présents dans le dossier libusb):

----Sources:
core.c
descriptor.c
io.c
os/darwin_usb.c et os/linux_usbfs.c
sync.c

----Headers:
libusb.h
libusbi.h
os/darwin_usb.h et os/linux_usbfs.h

Nous avons ensuite du écrire un makefile Android.mk relatif à l'implémentation de la libusb.
Il y a deux fichiers Android.mk dans le dossier libusb-1.0.3 :
-Le premier qui va servir à **appeler les Makefiles** des sous-dossiers
-Le second, au chemin **libusb-1.0.3/libusb** qui **définit les sources et les headers** du composant que l'on va implémenter.
Celui que nous avons écrit est donc le second.

Lors de la compilation avec make, deux erreurs se sont produites :
Pour la 1ère erreur, il faut définir la macro **TIMESPEC_TO_TIMEVAL** en rajoutant le code suivant dans le fichier io.c :

```
"#define TIMESPEC_TO_TIMEVAL(tv, ts)                     \
        do {                                            \
                (tv)->tv_sec = (ts)->tv_sec;            \
                (tv)->tv_usec = (ts)->tv_nsec / 1000;   \
        } while (0)"
```

Pour la seconde, il faut ajouter la librairie dans le fichier
"./build/core/prelink-linux-arm.map" "libusb.so"

2) Implémentation nouveau produit

Nous devons compiler avec **"build/_setup_env.sh"** depuis le dossier android-kernel (build/envsetup.sh deprecated)

Le produit s'appellera donc **lo52_GaujourBastien_PalermoEnzo**.
Nous devons ensuite ajouter les fichiers de base pour la définition d'un produit : CleanSpeck.mk, AndroidProducts.mk, BoardConfig.mk, vendorsetup.sh et lo52_GaujourBastien_PalermoEnzo.mk.

Dans le fichier lo52_GaujourBastien_PalermoEnzo.mk, on ajoute les lignes suivantes pour définir le produit :
"PRODUCT_NAME := lo52_GaujourBastien_PalermoEnzo
PRODUCT_DEVICE := ranchu
PRODUCT_BRAND := Android
PRODUCT_MODEL := Ranchu
PRODUCT_MANUFACTURER := Google"

Nous ajoutons ensuite cette ligne pour que le produit hérite du produit hikey de linaro :
"$(call inherit-product, device/linaro/hikey/hikey.mk)"

Puis pour personnaliser le produit avec les 3 options demandées :
**"PRODUCT_PROPERTY_OVERRIDES := **
**ro.hw=lo52 **
**net.dns1=8.8.8.8 **
net.dns2=4.4.4.4 "

Apres recherche sur android.googlesource, le fichier sym_keyboard_delete.png se trouve dans **"android/platform/development/master/./samples/SoftKeyboard/res/drawable-mdpi"** et **"/drawable-hdpi"**
On rajoute cette ligne pour spécifier le chemin des fichiers d'overlay
"DEVICE_PACKAGE_OVERLAYS := device/lo52_GaujourBastien_PalermoEnzo/overlay"

Enfin nous ajoutons la libusb aux packages du produit :
"PRODUCT_PACKAGES += libusb-1.0.3"

Pour ce qui est des **4 autres fichiers** .mk/.sh, nous les avons remplis en nous basant sur le cours session 5.

TP 4 : Utilisation de JNI

1) Installation du NDK

Nous avons commencé par créer une activité native C/C++
Tools -> SDK Manager > SDK Tools > cocher NDK, CMake

2) Les boutons directionnels

Le code initialise **un Listener pour chaque bouton**, qui lors du click vont **initialiser une variable String** ("direction"), et **lancer la fonction C++** avec le nom du bouton correspondant en paramètre.

La fonction C++ récupère la String en paramètre, la convertit en un format String compatible C++ (**const char***), puis compare cette String avec le texte du bouton grâce à "**strcmp()**".
Si il s'agit du bon bouton, le texte en japonais est **renvoyé sous format String** compatible kotlin, et **copié dans la variable de départ** ("direction").

Enfin, la zone de texte **prend la variable direction** pour valeur.
(Nous avions utilisé un widget editText plutôt qu'un textView au début, d'où la difficulté à éditer sa valeur)

3) Les boutons READ/WRITE

On initialise là aussi un listener par bouton, au click on vérifie que le champ de texte ne soit pas vide et on lance la fonction **callMultiplication** avec le nom du bouton et la valeur du champ editText en paramètre (Valeur convertie de **Editable** vers **int** au préalable).

La fonction C++ récupère ces paramètres et vérifie que le chiffre soit bien entre 1 et 10.
Si oui, elle vérifie de quel bouton il s'agit, réalise le calcul et renvoie le résultat dans un string.
Sinon elle renvoie un message d'erreur dans un string.