Rapport TPs LO52

Table des matières

Rapport TP1 (sujet moodle) 2
I. Paramétrage git 2
II. Clonage de dépôt et création de la branche 2
III. Création du projet “HelloWorld” sous Android Studio 2
Rapport TP2 (sujet moodle) 4
I. Configuration du noyau hikey-linaro 4
II. Identification de la configuration par défaut 4
III. Différences entre les configurations 4
IV. Première compilation pour Android 64bits 4
V. Modification de la configuration 4
1. Activation du NFC et d’un driver spécifique à un émulateur matériel 5
2. Activation de l’option permettant de mettre un logo/image au boot 5
3. Désactivation du MTP Gadget et activation de l’USB LED 5
4. Mettre en dur l’option TI SOC Drivers 5
VI. Deuxième compilation pour Android 64bits 5
VII. Comparaison entre les images et configurations 6
Rapport TP3 (sujet moodle) 7
I. Merger la branche SnakeTeacher dans la vôtre afin de récupérer le répertoire Rsc-libusb 7
II. Implémentation de la libusb 7
III. Implémentation d’un nouveau produit Android 9
Rapport TP4 10
I. Création de l’application NDK 10
II. Capture du layout de la MainActivity 11

Rapport TP1 (sujet moodle)

I. Paramétrage git

Nous avons commencé le tp par la configuration du nom et de l'email de l'utilisateur en local. Les commandes que nous avons utilisées sont :

- git config –global user.name "Prenom Nom"
- git config –global user.email "email"

II. Clonage de dépôt et création de la branche

Après avoir ouvert le terminal, et nous être placé dans le dossier voulu, nous avons cloné le repository des tps en effectuant :

- git clone https://github.com/gxfab/LO52_A2020.git

Puis nous avons procédé à la création et au placement de la branche sur laquelle nous travaillons :

- git checkout -b "FelixDayet_MehdiFracso"

III. Création du projet "HelloWorld" sous Android Studio

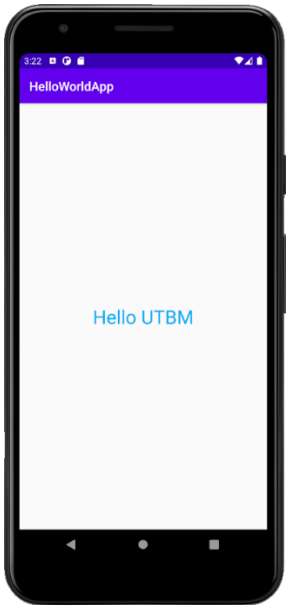
Nous avons tout d'abord procédé à la création de notre projet sous Android 4.1, et nous avons travaillé sur le template Empty Activity.

En suivant le tutoriel, nous avons premièrement modifié le Layout du Main Activity en lui ajoutant un bouton et un texte d'accueil (voir capture ci-dessous).

Il nous a ensuite fallu créer une nouvelle activité qui serait affichée lors de l'appui du bouton par l'utilisateur. On a ajouté un Textview sur la layout de cette activité, permettant d'afficher « hello UTBM » et nous avons modifié quelques-unes de ses propriétés (couleur, taille).

Par la suite nous avons ajouté une méthode dans notre Main Activity, permettant l'appel de notre seconde activité lors de l'appui du bouton à travers un intent.

On a donc en résultat final la capture ci-dessous qui s'affiche lors de l'appui sur le bouton de la Main Activity.



Rapport TP2 (sujet moodle)

I. Configuration du noyau hikey-linaro

Pour importer les sources nécessaires, on a utilisé les commandes suivantes :

mkdir android-kernel && cd android-kernel
repo init -u https://android.googlesource.com/kernel/manifest -b hikey-linaro-android-4.19
repo sync -j 8

II. Identification de la configuration par défaut

On peut trouver cette configuration en suivant ce chemin :
hikey-linaro/arch/{plateforme}/configs/defconfig

C'est cette configuration qui correspond à celle par défaut :
DEFCONFIG=hikey960_defconfig

III. Différences entre les configurations

Pour afficher les différences entre les configurations, nous avons utilisé la commande suivante :
Git diff build.config hikey-linaro/arch/{plateforme}/configs/defconfig

Nous avons exécuté cette commande en étant sur le dossier ./android-kernel.

IV. Première compilation pour Android 64bits

On utilise la commande "make -j8" qui nous permet de faire cette première compilation en utilisant les threads.

V. Modification de la configuration

Pour effectuer les différentes modifications, nous avons utilisé l'outil graphique Qt. Ainsi, nous avons utilisé la commande :

make xconfig

Une interface graphique s'ouvre, et on procède ainsi à la modification.

- Compatibilité pour la carte ARMv8 Versatile uniquement
-> Device Drivers-> Hardware Monitoring support

On coche l'activation.

1. Activation du NFC et d'un driver spécifique à un émulateur matériel

-Activation en cochant la case correspondante.
Networking support -> NFC subsystem support -> NFC digital stack support (option)
Pour le driver spécifique, il faudra le choisir sous l'onglet :
Networking support -> NFC subsystem support -> NFC digital stack support (sous onglet) -> Driver de carte physique NFC à sélectionner

2. Activation de l'option permettant de mettre un logo/image au boot

-Activation en cochant la case correspondante.
-> Device Drivers -> Graphics support -> Bootup logo (LOGO=y)

3. Désactivation du MTP Gadget et activation de l'USB LED

Pour la désactivation du MTP Gadget, nous avons cherché dans tous les onglets, en particulier avec les chemins :
Device Drivers ->USB support ->USB Gadget Support
Device Drivers -> Multimedia support -> Media USB Adapters
Cependant, il semblerait que l'option soit absente, donc par défaut elle ne pourra pas être activée.
Device Drivers -> LED Support -> LED class support

4. Mettre en dur l'option TI SOC Drivers

Device Drivers -> SOC -> TI SOC drivers support

VI. Deuxième compilation pour Android 64bits

On utilise de nouveau la commande "make" pour faire la compilation mais en y ajoutant l'architecture spécifique ARMv8 a travers l'option make ARCH=arm64.
export ARCH=arm64
Export CROSSCOMPILE=/home/felix/Desktop/android-kernel-customized/hikey-linaro/toolchain/aarch64-linux-gnu
make -j10

On sélectionne ensuite les options “yes” pour l’armv8.

VII. Comparaison entre les images et configurations

En comparant les deux images on obtient le résultat suivant :
Taille de l’image avec la configuration par défaut : 11.1MB
Taile de l’image avec la configuration personnalisée : 8.6MB

Pour la différence entre entre les deux configuration, on utilise la commande diff .config .config.old. On obtient ce résultat :

```
felix@felix-VirtualBox:~/Desktop/android-kernel-customized/hikey-linaro$ diff .config .config.old
3c3
< # Linux/x86 4.19.152 Kernel Configuration
---
> # Linux/arm64 4.19.152 Kernel Configuration
26,29d25
< CONFIG_HAVE_KERNEL_BZIP2=y
< CONFIG_HAVE_KERNEL_LZMA=y
< CONFIG_HAVE_KERNEL_XZ=y
< CONFIG_HAVE_KERNEL_LZO=y
31,36c27,28
< CONFIG_KERNEL_GZIP=y
< # CONFIG_KERNEL_BZIP2 is not set
< # CONFIG_KERNEL_LZMA is not set
< # CONFIG_KERNEL_XZ is not set
< # CONFIG_KERNEL_LZO is not set
< # CONFIG_KERNEL_LZ4 is not set
---
> # CONFIG_KERNEL_GZIP is not set
> CONFIG_KERNEL_LZ4=y
55a48
> CONFIG_GENERIC_IRQ_SHOW_LEVEL=y
57d49
< CONFIG_GENERIC_PENDING_IRQ=y
58a51
> CONFIG_HARDIRQS_SW_RESEND=y
64,65c57
< CONFIG_GENERIC_IRQ_MATRIX_ALLOCATOR=y
< CONFIG_GENERIC_IRQ_RESERVATION_MODE=y
---
> CONFIG_HANDLE_DOMAIN_IRQ=y
69c61
< CONFIG_CLOCKSOURCE_WATCHDOG=y
---
```

Rapport TP3 (sujet moodle)

I. Merger la branche SnakeTeacher dans la vôtre afin de récupérer le répertoire Rsc-libusb

Tout d'abord, on se place dans notre branche avec la commande :

$ git checkout FelixDayet_MehdiFracso

Nous n'avons pas eu le besoin de réaliser un Git merge SnakeTeacher, car les deux dossiers étaient déjà présents.

On trouve deux répertoires :

- Libusb-1.0.3 : pour l'implémentation du composant libusb
- Device : pour l'ajout du produit

II. Implémentation de la libusb

1. Déterminez les fichiers sources et les headers pour la compilation d'une libusb sur Android

En analysant les différents fichiers, on trouve les fichiers sources (extension .c) et headers (extension .h) sous le dossier libusb :

-core.c
-descriptor.c
-io.c
-libusb.h
-libusbi.h
-sync.c
-os/linux_usbfs.c
-os/linux_usbfs.h

2. Ecrire le fichier Android.mk

Pour pourvoir écrire ce fichier, nous avons dû visiter la documentation le concernant en visitant le lien suivant :
https://android.googlesource.com/platform/ndk/+/4e159d95ebf23b5f72bb707b0cb1518ef96b3d03/docs/ANDROID-MK.TXT
Et en étudiant cette dernière, on a pu relever des points importants dont on peut citer :

- LOCAL_PATH : indique le chemin des fichiers sources dans la branche de développement
- include $(CLEAR_VARS): reinitialise toute les autres variables sauf le LOCAL_PATH

- LOCAL_MODULE : contient le nom du module compilé
- LOCAL_SRC_FILES : Liste les noms des fichiers sources à compiler par le système de build, ces fichiers seront assemblés en un module.
- include $(BUILD_SHARED_LIBRARY): Fait appel à une macro qui collecte tout le paramétrage défini par les variables définies avant et que leurs noms commencent avec un LOCAL_*.

Voici le fichier android.mk que nous avons écris :

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := libusb
LOCAL_SRC_FILES := core.c descriptor.c io.c sync.c os/linux_usbfs.c os/linux_usbfs.h
include $(BUILD_SHARED_LIBRARY)
```

Pour la compilation, nous avons utlisé la commande suivante en lui donnant les droits nécessaires :
$ sudo chmod 0755 configure
$ sudo ./configure
$ sudo make

3. Erreur sur la macro TIMESPEC_TO_TIMEVAL

Cette erreur est due à une absence de définition dans le fichier io.c, dans lke repertoire lisbub. Pour la corriger : on ajoute les lignes :

```
# define TIMESPEC_TO_TIMEVAL(tv, ts) {
      (tv)->tv_sec = (ts)->tv_sec;
      (tv)->tv_usec = (ts)->tv_nsec / 1000;
}
#endif
```

Source :https://code.woboq.org/userspace/glibc/time/sys/time.h.html
Lignes 42-46

4. Erreur : build/tools/apriori/prelinkmap.c(137) : library "libusb.so" not in prelink map

On corrige cette erreur en ajoutant l'adresse de la librairie libusb dans le fichier build/tools/apriori/prelinkmap.

```
# extended system libraries

libusb.so 0xAE600000
```

III. Implémentation d'un nouveau produit Android

Produit : Lo52_FelixDayet_MehdiFracso
Propriété personnalisées :
ro.hw=lo52 net.dns1 = 8.8.8.8 net.dns2 = 4.4.4.4

Voici le fichier que nous avons créé:

$(call inherit-product, device/linaro/hikey/hikey-common.mk)

PRODUCT_PACKAGES += libusb

PRODUCT_PROPERTY_OVERRIDES := /
ro.hw=lo52 net.dns1 = 8.8.8.8 net.dns2 = 4.4.4.4

DEVICE_PACKAGE_OVERLAYS := device/utbm/Lo52_FelixDayet_MehdiFracso/overlay
PRODUCT_NAME := Lo52_FelixDayet_MehdiFracso
PRODUCT_BRAND := Lo52_FelixDayet_MehdiFracso
PRODUCT_DEVICE := Lo52_FelixDayet_MehdiFracso
PRODUCT_MODEL := Lo52_FelixDayet_MehdiFracso

Lien utilisé :
https://android.googlesource.com/device/linaro/hikey/+/refs/heads/master/hikey.mk

Rapport TP4

Nous avons tout d'abord installé l'ensemble des composants nécessaire pour le NDK android. Tout d'abord avec l'activation du support pour NDK ainsi que l'acceptation des licences nécessaires pour autoriser le téléchargement des différentes dépendances requises pour le développement.
Pour accepter ces licences, nous avons utilisé l'outil sdkmanager (android_sdk/tools/bin/).

I. Création de l'application NDK

Nous avons créé un nouveau projet de type native C++. Par la suite nous sommes parti sur le layout, afin d'ajouter les boutons nécessaires de type "Button" (up, down, left, right, read, write) ainsi que le champ de saisie de type "EditText" et des zones d'affichage de texte de type "TextView" (read, write). Puis nous avons attribué à chacun d'entre eux leurs contraintes respectives, et avons procédé à une modification de leur id.
Ci-dessous une capture du layout de la MainActivity :

II. Capture du layout de la MainActivity

Par la suite nous avons effectué les différentes étapes demandées :

- Le bouton READ appelle une fonction native avec un nombre compris entre 0 et 10 qui va retourner le texte affiché dans le label "READ : a*a", a étant le nombre passé à la fonction.
- Le bouton WRITE appelle une fonction native avec un nombre compris entre 0 et 10 2 qui va retourner le texte affiché dans le label "WRITE : a*a*a", a étant le nombre passé à la fonction
- Les boutons LEFT, RIGHT, UP et DOWN appellent une fonction native qui va afficher le texte de la direction correspondante en Japonais en passant en paramètre le nom des boutons.

Nous avons commencé par la déclaration des signatures des fonctions natives au niveau de la classe MainActivity.java :

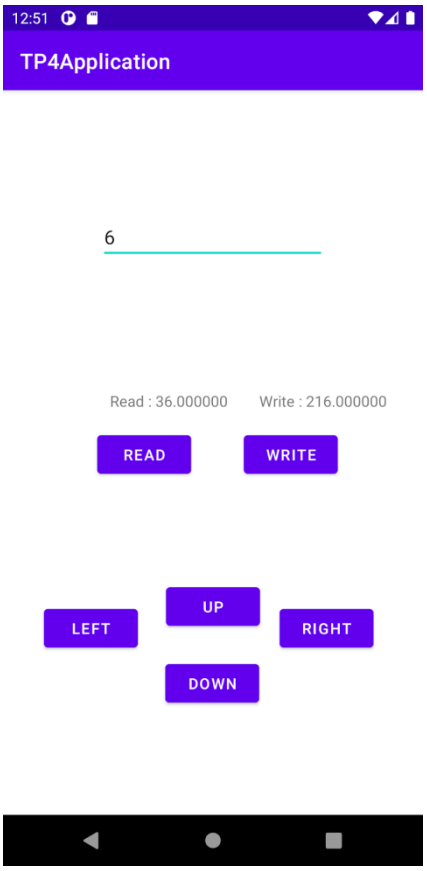
```
public native String calculateSquare(int value);
public native String calculateCube(int value);
public native String getJapaneseEquivalent(int direction);
```

Puis Android studio nous a proposé de générer les définitions de ces fonctions dans le fichier c++ (native-lib.cpp) . Nous l'avons donc appliqué et par la suite, nous nous sommes consacré à compléter ces différentes définitions avec le code c++ nécessaire.

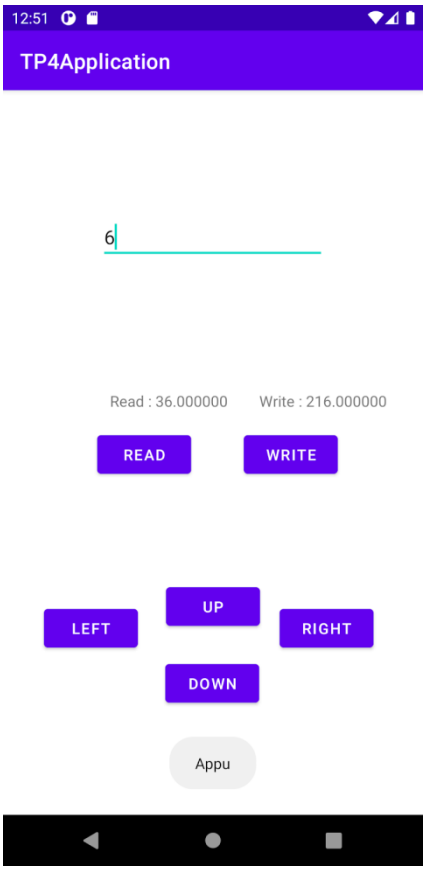
Le challenge ici a été pour nous de convertir les différents types de données échangées car pour faire les calculs en c++, il nous fallait convertir les jint en int, nous avons utilisé un cast pour cela. Pour retourner un jstring à la fin de nos fonctions qui nous donnaient un string, nous avons utilisé une fonction pour convertir le type (env->NewStringUTF(to_string(valeurEntiere).c_str())).

Pour finaliser le tp, nous avons enfin appliqué la traduction en japonais des directions souhaitées par l'utilisateur pour les afficher grâce à des toasts lors de l'appui des boutons.
Nous avons aussi affecté à chaque bouton les event handlers correspondants.

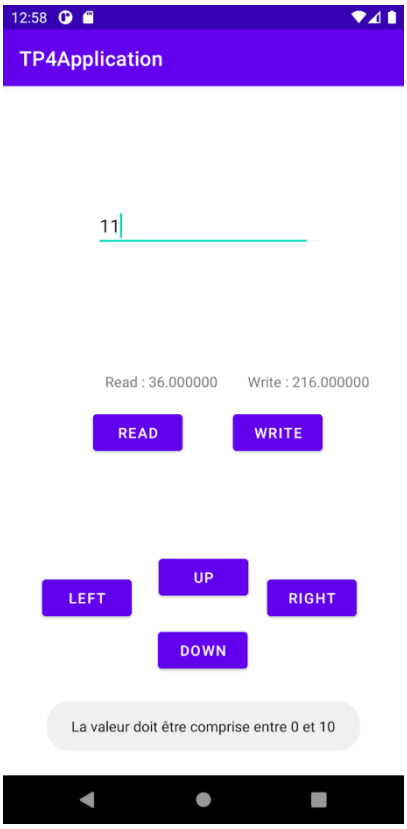
Voici différentes captures illustrant notre travail :



Appui sur les boutons read et write pour afficher le cube et le carré du nombre souhaité



Appui sur le bouton up, déclenchant l'affichage du toast Appu



Message d'erreur pour une valeur non comprise entre 0 et 10