Rapport LO52

Luc Heydel, Guillaume Muller,

Aurélien Saunier, Victor Sonza

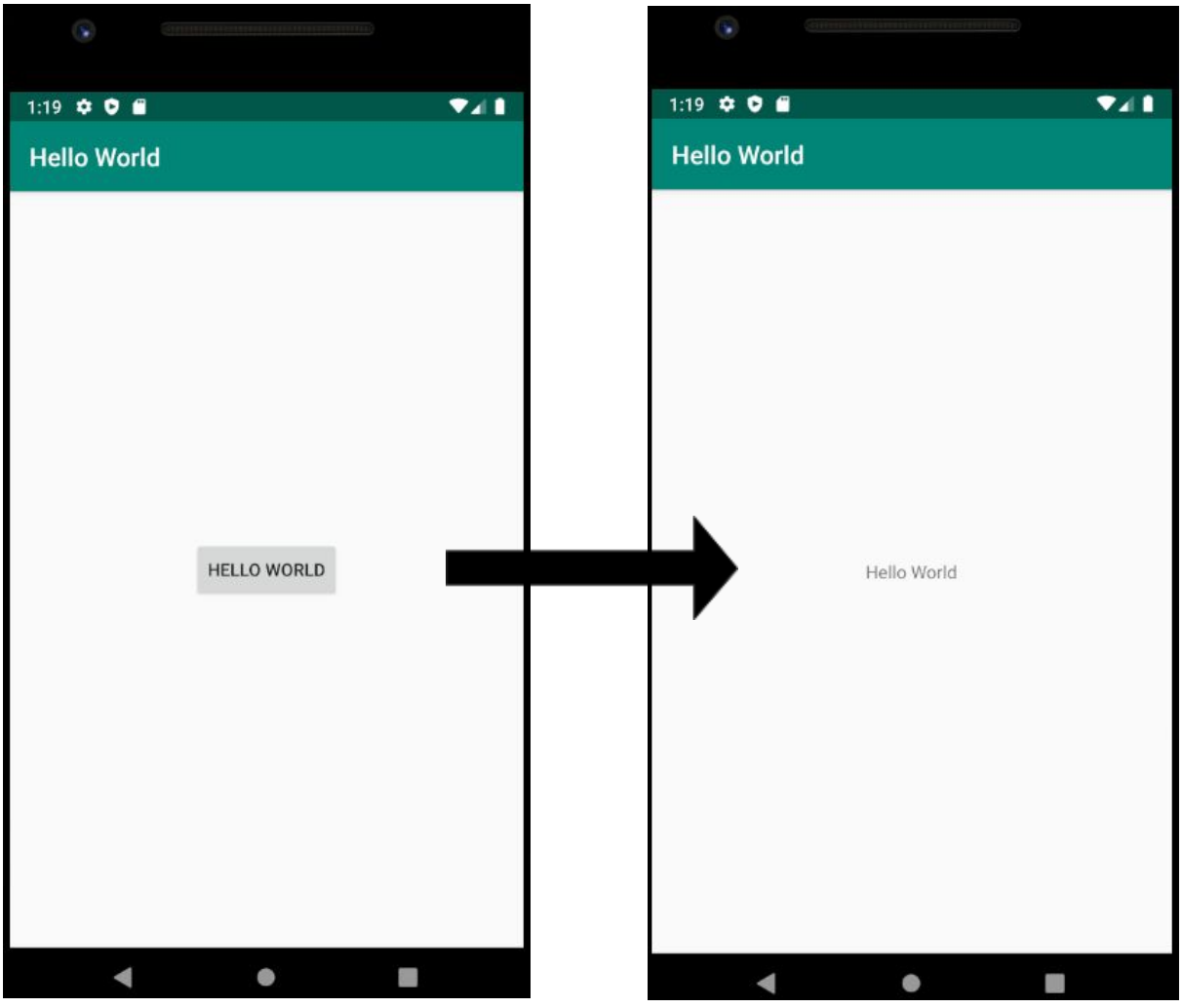
1. TP 1 : Mise en place de l'environnement de développement

a. Mise en place

Commencer par installer Git si ce n'est pas fait.
Cloner le dépôt : https ://github.com/gxfab/LO52_A2019.
Modifier le fichier de configuration de Git pour y faire figurer notre adresse email UTBM et notre nom d'utilisateur.
Créer une branche portant le nom de notre groupe de projet et se positionner dessus.
Créer un répertoire LO52_2019_SandraJaiFroid dans lequel nous déposerons toutes nos sources ainsi qu'un README.txt expliquant notre organisation.
Installation de l'environnement de développement Android Studio.

b. Création d'une application helloWorld

Cette application se constitue d'une première Activité liée à une vue ne possédant qu'un seul bouton HelloWorld en son centre. Sur clic du bouton, on est redirigé vers une seconde Activité liée à une vue comprise d'un textView contenant HelloWorld en son centre.



Ce changement d'Activité se fait via le code suivant :

```
Intent intent = new Intent(MainActivity.this, HelloWorldTxt.class);
startActivity(intent);
```

2. TP 3 : Création d'un device Android

Les fichiers sources et les headers pour la compilation d'une libusb sur Android sont :

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

On a cherché à quoi correspond LOCAL_PATH. Il s'agit de "/external/libusb/".

Pour corriger l'erreur sur la macro TIMESPEC_TO_TIMEVAL, il faut rajouter dans io.c les lignes suivantes :

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

Concernant l'erreur dans "build/tools/apriori/prelinkmap.c" (libusb.so not in pre-link map), il faut rajouter la ligne suivante dans l'Android.mk :

```
LOCAL_SHARED_LIBRARIES := libc libusb
```

Pour nommer le produit, le faire hériter de hikey, et le personnaliser, il faut ajouter les lignes suivantes dans le product.mk :

```
$(call inherit-product, device/Linaro/hikey.mk)
# Overrides
PRODUCT_NAME := lo52_SandraJaiFroid
PRODUCT_DEVICE := lo52_SandraJaiFroid
PRODUCT_PROPERTY_OVERRIDES := \
    ro.hw=lo_52 \
    net.dns1=8.8.8.8 \
    net.dns2=4.4.4.4
```

Le fichier sym_keyboard_delete.png se trouve dans le répertoire res/drawable-mdpi/
Pour le surcharger, on emploie la notion d'overlay :

- Créer un dossier overlay (nous l'avons mis dans le répertoire device)
- Dans ce dossier, créer une arborescence mirroir (identique) du dépôt du fichier à surcharger
- Y placer le fichier avec le même nom que celui à surcharger
- Ajouter dans le product.mk la ligne `PRODUCT_PACKAGE_OVERLAYS := overlay`

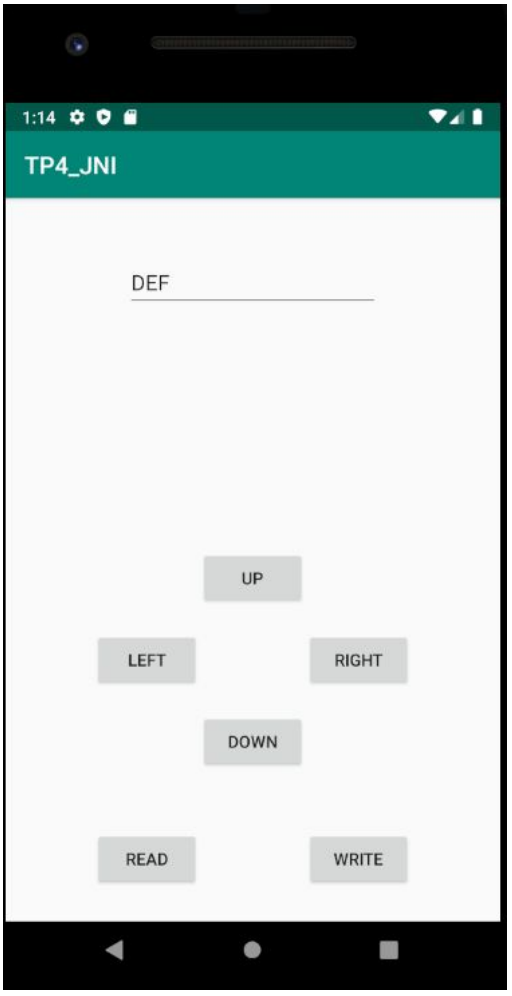
Enfin, pour ajouter libusb aux packages du produit, il faut ajouter la ligne suivante dans le product.mk :

```
PRODUCT_PACKAGES += libusb
```

3. TP 4 : Utilisation de JNI

a. Interface

L'interface est composée de 6 boutons et d'une zone de texte. La valeur par défaut de la zone de texte est "DEF"



b. Fonctionnalités

- Les boutons **Up**, **Down**, **Left** et **Right** font appel à des fonctions natives qui leur retournent une traduction en Allemand. Ces traductions sont affichées dans la zone de texte.

- Le bouton **Read** génère un nombre aléatoire entre 0 et 10. Ce nombre est alors passé en argument à une fonction native qui renvoie son carré.

- Le bouton **Write** génère un nombre aléatoire entre 0 et 10. Ce nombre est alors passé en argument à une fonction native qui renvoie son cube.

c. Fonctions Natives

Les fonctions natives sont des fonctions écrites en C++. Elles peuvent êtres appelées par une application Java.
Dans notre cas nous avons la classe Java **MainActivity** qui fait appel à des fonctions natives qui se trouve dans **native-lib.cpp**. Ces fonctions sont déclarées dans la classe Java comme fonctions natives :

```
public native String stringFromJNI();
public native int squareFromJNI(int a);
public native int cubedFromJNI(int a);
public native String obenFromJNI();
public native String niederFromJNI();
public native String rechtFromJNI();
public native String linksFromJNI();
```

Les fonction natives dans **native-lib.cpp** sont définies de la façon suivante :

```
extern "C" JNIEXPORT jstring JNICALL
Java_com_utbm_tp4_1jni_MainActivity_stringFromJNI(
        JNIEnv *env,
        jobject /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF(hello.c_str());
}
```