TP Android

Compte-rendu de l'ensemble des TPs réalisés dans le cadre de l'UV LO52

Abdelhamid KACIMI
LO52 – A20

Établissement :
Université de Technologie de Belfort-Montbéliard
Professeur encadrant :
Pierre ROMET & Fabien BRISSET
Responsable UV :
Fabien BRISSET

Table des matières

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

But du TP

Installation et mise en place de l'environnement de travail avec la mise en place d'une branche de travail dans Git, l'installation et le développement de la première application Android avec Android Studio.

Mise en place de Git

La mise en place de Git commença par son installation, la récupération d'un clone du dépôt en local et la modification du fichier de configuration. Une branche a aussi été créé pour y déposer mes différents travaux.

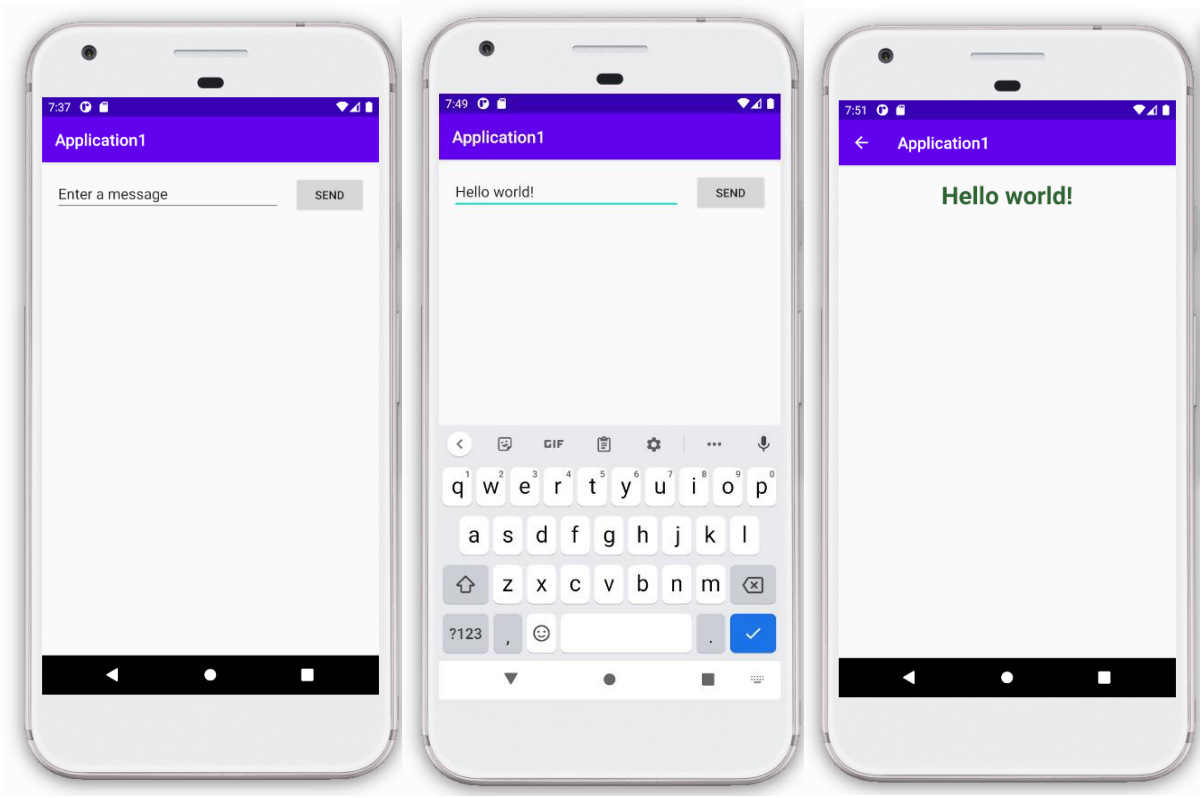
git clone https://github.com/gxfab/LO52_A2020
git branch KacimiAbdelhamid
git checkout KacimiAbdelhamid
git push -u

Installation et familiarisation avec Android Studio

L'installation d'Android a été réalisée dans ma machine personnelle. L'installation d'un environnement de travail avec les machines de l'UTBM a été très laborieuse. L'installation a précédé le développement de la première application avec l'aide du guide que nous pouvons retrouver dans la documentation en ligne.

L'application permet d'afficher un texte qui aura été écrit par l'utilisateur précédemment et validé par un appui sur un bouton. Le développement de cette application a été très utile et très enrichissante. En effet, cela m'a permis, dans un premier temps, de me familiariser avec Android Studio et son interface graphique. Nombreux sont les différents outils nécessaires au développement d'une application et ce TP m'a donc permis de me familiariser avec ces différents composants tels que le Layout Editor et ses widgets, les Key, les différents fichiers sur lesquelles nous pouvons travailler, le système d'appel de fonction, etc…

Résultats


7:37
Application1
Enter a message
SEND
7:49
Application1
Hello world!
SEND
7:51
Application1
Hello world!

TP 2 : Manipulation d'un Kernel Linux

But du TP

Écriture d'un script permettant d'automatiser plusieurs les commandes afin de modifier la configuration du Kernel.

Préparation de l'environnement

L'importations des sources a été réalisé avec les commandes :
mkdir android-kernel && cd android-kernel
repo init -u https://android.googlesource.com/kernel/manifest-bhikey-linaro-android-4.19
repo sync -j 8

Le lien correspond à un noyau hikey-linaro comme demandé dans le TP.

Identification de la configuration par défaut relative à un noyau

L'identification de la configuration par défaut se fait à l'aide des fichiers de configurations. Nous retrouvons cela avec la valeur qui est affecté à *DEFCONFIG=hikey960_defconfig* dans le fichier de configuration que l'on retrouve dans *hikey-linaro/arch/arm64/configs/defconfig .*

Différences de configuration

Nous pouvons obtenir la différence des configurations avec la commande *build.config hikey-linaro/arch/arm64/configs/defconfig.*

Bonus

Une première compilation de la première version du noyau a été réalisée avec *make -j8* puis une autre a eu lieu avec, cette fois-ci, la version modifiée et optimisée.
On remarque plusieurs différences dans les images générées avec la commande *diff* et une différence au niveau de la taille. La première version occupe effectivement plus de place que la version optimisée (Plus de 11Mb et moins de 9Mb).

TP 3 : Création d'un device Android

But du TP

Écriture des fichiers Makefile Android relatifs à l'intégration d'un composant et d'un produit Android.

Implémentation de la libusb

L'implémentation débute avec l'écriture du fichier Android.mk. On y écrit les lignes suivantes :

```
LOCAL_PATH:=$(callmy-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES:= core.c descriptor.c io.c sync.c os/linux_usbfs.c
LOCAL_C_INCLUDES += external/libusb-1.0.3/ external/libusb-1.0.3/libusb/
external/libusb-1.0.3/libusb/os
LOCAL_MODULE:= libusb
include $(BUILD_SHARED_LIBRARY)
```

Ces lignes permettent de définir les fichiers sources et les headers pour la compilation.

L'erreur liée à la macro TIMESPEC_TO_TIMEVAL est corrigé en la définissant dans le fichier io.c avec ses lignes:

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

La deuxième erreur est réglée en mappant la librairie libusb.so dans le fichier ./build/core/prelink-linux-arm.map

Implémentation d'un nouveau produit Android

Ici, nous devons implémenter un produit Android s'appelant lo52_KacimiAbdelhamid qui hérite du produit hikey de Linaro.

Pour le nom du produit, nous devons écrire dans le fichier mk les lignes suivantes :

```
PRODUCT_BRAND := UTBM_LO52
PRODUCT_NAME := lo52_KacimiAbdelhamid
PRODUCT_DEVICE:=
lo52_KacimiAbdelhamid
```

On ajoute aussi une ligne afin de faire hériter le nouveau produit du produit hikey de Linaro. Cette ligne est :

```
$(call inherit-product, device/Linaro/hikey.mk)
```

Par ailleurs, des propriétés doivent aussi être initialisées avec certaines valeurs. En effet, nous devons personnaliser ro.hw à lo52, net.dns1 à 8.8.8.8 et net.dns2 à 4.4.4.4. Il suffit de changer la valeur de PRODUCT_PROPERTY_OVERRIDES de cette façon :

```
PRODUCT_PROPERTY_OVERRIDES :=       \
    ro.hw=lo_52                     \
    net.dns1=8.8.8.8                \
    net.dns2=4.4.4.4
```

Une surcharge du fichier sym_keyboard_delete.png doit être réalisée. Nous la retrouvons dans android/platform/development/master/./samples/SoftKeyboard/res/drawable-mdpi.

Nous devons finalement ajouter la libusb aux packages du projet de cette manière :

```
PRODUCT_PACKAGES += libusb
```

TP 4 : Utilisation de JNI

But du TP

Ce TP consiste à réaliser une application simple en utilisant JNI.

Les différentes implémentations

Après avoir installé de façon très simple les composants nécessaires pour le NDK Android, j'ai commencé par mettre en place les différents éléments que nous allons retrouver plus tard à l'écran. Nous avons donc les quatre boutons correspondant à chaque direction, deux bouton Read et Write comme demandé, un plain text pour insérer le nombre si nécessaire et afficher les résultats ainsi qu'un bouton switch qui permettra de savoir si les calculs prendront un nombre aléatoire en paramètre ou bien un nombre saisi par l'utilisateur.

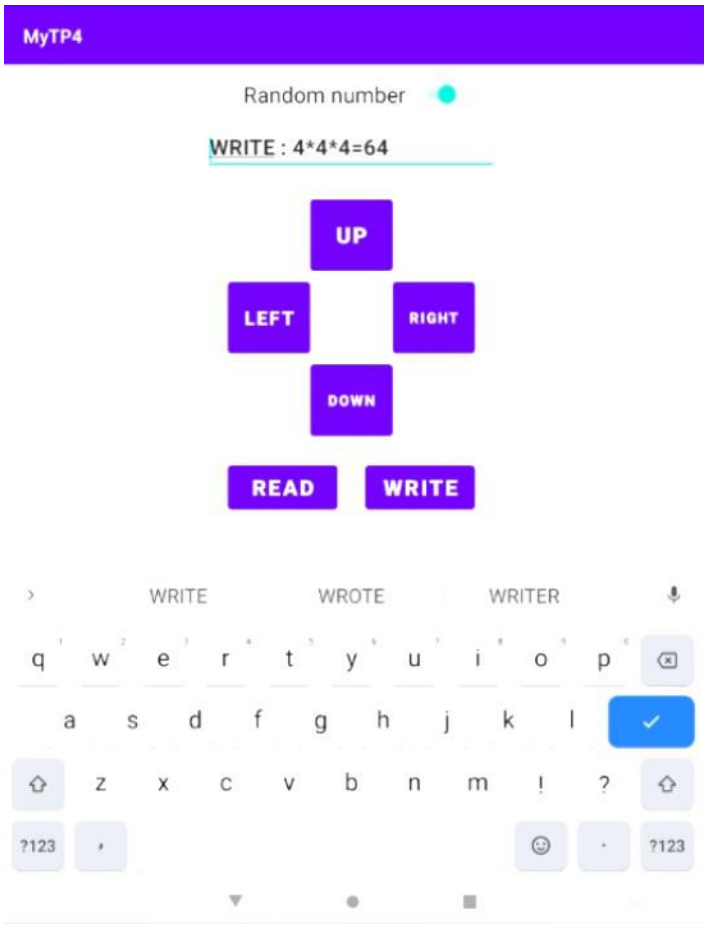
Nous retrouvons dans un premier temps la méthode directionListener qui affichera la traduction japonaise de la direction suivant le bouton qui l'a appelé. Cette méthode fait appel au JNI avec la fonction native directionInJapanese, présente dans native-lic.cpp, qui retourne la bonne
traduction.

L'autre méthode important présente ici est la méthode listener. Cette dernière génère un nombre aléatoire entre 0 et 10 si le switch est sur OFF et utilisera la valeur que l'utilisateur peut saisir. Si la saisie n'est pas un nombre supérieur à 0 et inférieur à 10, alors un message d'erreur s'affichera. Ce nombre sera ensuite traité par la fonction native calculate. Cette fonction retournera le texte demandé avec son calcul en fonction du bouton cliqué et du nombre précédemment saisi ou aléatoirement généré. Cette fois encore, la fonction est dans le fichier native-lib.cpp

Apports

Ce TP est très instructif puisqu'il m'a permis de mettre en pratique les connaissances du cours 9. Il est intéressant de voir que du code en C++ peut être utilisé avec du code Kotlin ou avec du code Java comme ici. Ce TP est essentiel pour comprendre le fonctionnement du JNI et la façon dont une fonction native doit être écrite. De plus, il permet de se familiariser davantage avec Android Studio et d'être finalement plus à l'aise pour le projet.

Interface graphique



Une simulation en vidéo est par ailleurs disponible pour une démonstration concrète du TP réalisé.