Rapport de TP LO52

Bartuccio Antoine & Le Chevanton Yann

Vendredi 13 Janvier 2019

Table des matières

Mise en place de l'environnement de développement **2**
- Configuration du dépôt . . . 2
- Création du projet . . . 2

Création d'un device Android **2**
- Récupérer la LibUsb . . . 2
- Implémentation de la LibUsb . . . 3
- Création du Android.mk . . . 3
- Correction de la macro TIMESPEC_TO_TIMEVAL . . . 3
- Correction des erreurs de link . . . 3
- Implémentation d'un nouveau produit Android . . . 3

Utilisation de la JNI **4**
- Création du projet . . . 4
- Création d'une application simple . . . 5

Mise en place de l'environnement de développement

Configuration du dépôt

```
# Création du dépôt
cd <dossier pour le projet>
git clone https://github.com/gxfab/LO52_A2019
cd LO52_A2019
# On ne met pas --global à ces commandes pour n'affecter que ce répertoire
git config user.name "Bartuccio Antoine"
git config user.email "antoine.bartuccio@utbm.fr"
# Création et basculement sur la nouvelle branche fail
git checkout -b fail
# Création du dossier du TP1 et du README
mkdir -p TP1/LO52_2019_fail
touch TP1/LO52_2019_fail/README.md
```

Création du projet

Cette étape reviens à suivre l'assistant de création de projet d'Android studio. On crée le projet dans le dossier LO52_2019_failz crée précédemment.

Très important : pour la version minimale d'Android à choisir pour le projet, il vaut mieux privilégier Nougat. Pendant la réalisation du TP, nous avions sélectionné ICS et c'était impossible de build et exécuter.

Création d'un device Android

Récupérer la LibUsb

```
# On récupère les dernière sources
git pull
# On effectue le merge
git merge SnakeTeacher
# Alternativement, il est possible de se rebase
git rebase SnakeTeacher
```

Implémentation de la LibUsb

Création du Android.mk

```
# Rsc-libusb/libusb-1.0.3/libusb/Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES := core.c descriptor.c io.c sync.c os/linux_usbfs.c
LOCAL_C_INCLUDES += $(LOCAL_PATH) $(LOCAL_PATH)/os/
LOCAL_C_FLAGS += -pthread
LOCAL_MODULE := libusb
LOCAL_MODULE_TAGS := optional

include $(BUILD_SHARED_LIBRARY)
```

Correction de la macro TIMESPEC_TO_TIMEVAL

```
// Rsc-libusb/libusb-1.0.3/libusb/io.c

/* Début du fichier */

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

/* Reste du fichier */
```

Correction des erreurs de link

Le but est de corriger l'erreur *build/tools/apriori/prelinkmap.c(137): library 'libusb.so' not in prelink map*

Ajouter dans ./build/core/prelink-linux-arm.map

```
libqcamera.so       0xA9400000
libusb.so           0xA8000000
```

Implémentation d'un nouveau produit Android

Pour ajouter le nouveau produit Android, il est nécessaire de remplir les fichiers suivant :

```
# Rsc-libusb/device/utbm/lo52_fail/vendorsetup.sh
# On définie les cibles pour lunch en user, engineering et userdebug
add_lunch_combo lo52_fail-eng
add_lunch_combo lo52_fail-user
add_lunch_combo lo52_fail-userdebug

# Rsc-libusb/device/utbm/lo52_fail/BoardConfig.mk
# On hérite du produit hikey de Linaro pour le matériel
include device/linaro/hikey.mk/BoardConfig.mk

# Rsc-libusb/device/utbm/lo52_fail/AndroidProducts.mk
# On défini le Makefile à utiliser pour ce produit Android
PRODUCT_MAKEFILES := $(LOCAL_DIR)/lo52_fail.mk

# Rsc-libusb/device/utbm/lo52_fail/lo52_fail.mk
# On hérite du produit hikey de Linaro pour le makefile
$(call inherit-product, device/linaro/hikey.mk)

# On a joute la libusb
PRODUCT_PACKAGES += libusb
# On ajoute les propriétés demandées dans le projet
PRODUCT_PROPERTY_OVERRIDES += ro.hw=lo52 \
                              net.dns1=8.8.8.8 \
                              net.dns2=4.4.4.4
# On définie le dossier où mettre les fichiers d'overlay
PRODUCT_PACKAGE_OVERLAYS := device/utbm/lo52_fail/overlay

# On choisis les noms du produit
PRODUCT_NAME := lo52_fail
PRODUCT_DEVICE := lo52_fail
PRODUCT_BRAND := lo52_fail
PRODUCT_MODEL := lo52_fail

include $(call all-subdir-makefiles)
```

Enfin, on surcharge le fichier *sym_keyboard_delete.png* en le plaçant dans *Rsc-libusb/device/utbm/lo52_fail/overlay/sample/SoftKeyboard/res/drawable-mdpi*.

Utilisation de la JNI

Création du projet

Cette étape reviens à suivre l'assistant de création de projet d'Android studio pour un projet de type *Native C++*.

Toutes les dépendances sont automatiquement téléchargées par Android studio. C'est très pratique.

Création d'une application simple

Une application utilisant la JNI se compose, comme une application classique, d'activités Android Java (ou Kotlin dans ce cas).

Ici, notre activité se nomme *MainActivity* et se situe dans *MainActivity.kt*. Elle utilise un layout xml crée comme pour une application Android classique.

Le code cpp se situe dans le fichier *native-lib.cpp* qui se situe dans *app/src/main/cpp*.

```cpp
#include <iostream>
#include <jni.h>
#include <random>
#include <string>

float get_random(int min, int max) {
  static std::default_random_engine e;
  static std::uniform_real_distribution<> dis(min, max); // rage 0 - 1
  return dis(e);
}

// Exemple d'une des fonctions demandées, ici, la fonction READ
extern "C" JNIEXPORT jstring JNICALL
Java_com_e_testjni_MainActivity_READ(
        JNIEnv* env,
        jobject obj) {
    auto a = int(get_random(-1, 11));
    std::string label = std::to_string(a * a);
    return env->NewStringUTF(label.c_str());
}
```

Le lien avec l'activité se fait directement dans celle-ci. On déclare les fonctions comme des méthodes externes de la classe de l'activité et on défini une fonction init dans l'objet compagnon de celle-ci.

```kotlin
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        /* Gestion des autres boutons */
        ...
```

```
        /* Utilisation de la fonction READ déclarée en JNI */
        read.setOnClickListener({
            readLabelOut.text = READ()
        })
    }

    /* Déclaration des fonctions présentes dans la lib native c++ */
    external fun READ(): String
    external fun WRITE(): String
    external fun ToGerman(text: String): String

    companion object {

        /* Chargement de la librairie */
        init {
            System.loadLibrary("native-lib")
        }
    }
}
```