03/01/2021

# Rapport de TP -LO52

Automne 2020-2021

Ndeye Marie SALL & Mouhamed DIAGNE
UTBM

# Table des matières

# TP1 : Mise en place de l'environnement de développement

## I. Prérequis

Pour réaliser ce TP nous avons besoin d'installer :

- Android studio qui est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntellIJ IDEA et utilise le moteur de production gradle. Il peut être téléchargé sous les systèmes d'exploitation Windows, macOS, Chrome OS et Linux.

- Un SDK (Software Development Kit) qui permet de créer des applications sur la plateforme Android.

- La configuration du proxy :
  - HTTP
  - Hostname proxy.utbm.fr
  - Port 3128

## II. Objectif

L'objectif de ce TP était de créer une première application Android à l'aide d'Android Studio. Celle-ci devait contenir deux activités. La première activité redirige vers la seconde lors de l'appuie sur un bouton.

## III. TP Réalisé

Le but de ce TP est de pouvoir charger une page qui affiche Hello World lorsque l'on clique sur le bouton de la page d'accueil.

- **MainActivity.java**

```java
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = (Button) findViewById(R.id.btn);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                viewHelloWordActivity();
            }
        });
    }

    public void viewHelloWordActivity() {
        Intent intent = new Intent(this, HelloWordActivity.class);
        startActivity(intent);
    }
}
```

Dans la classe MainActivity, la page activity_main sera chargée. Ensuite nous avons créé une fonction setOnClickListener qui pemet de détecter un clic sur un bouton.

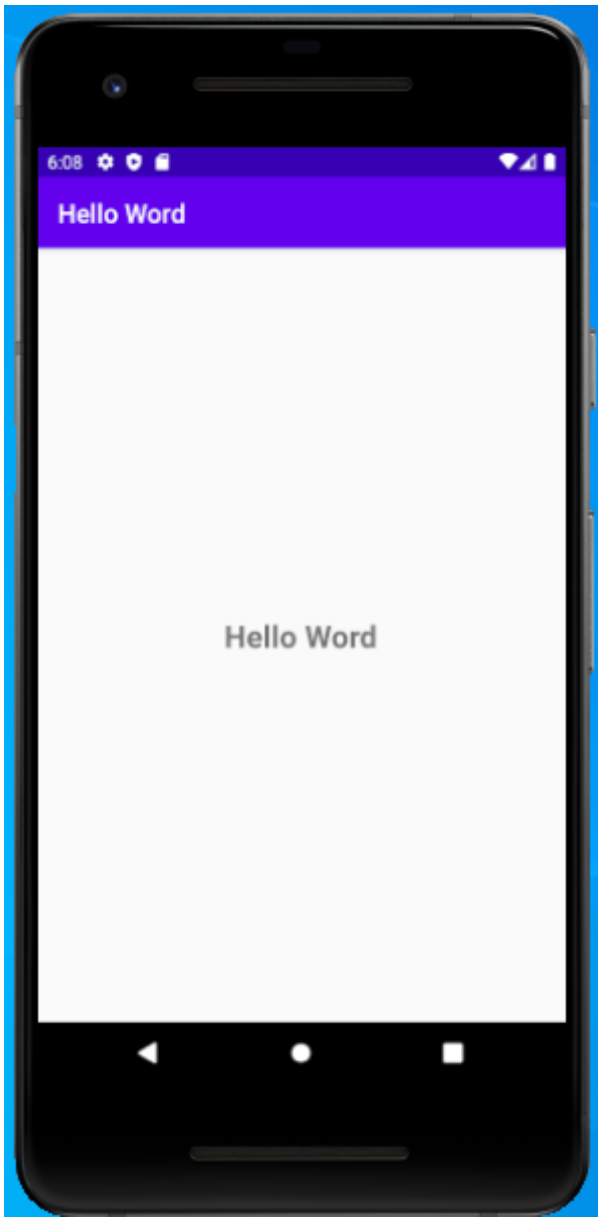
Après un clic sur le bouton ***Go to next page*** la fonction viewHelloWordActivity() sera appelée. Cette méthode permet de charger HelloWordActivity.java qui affiche à son tour Hello Word.

- **Design activity_main.xml**


- **HelloWordActivity.java**

```java
package com.nsallmdiagn01.helloword;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class HelloWordActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_word);
    }
}
```

- **Design du fichier activity_hello_word.xml**




## IV. Commandes utilisées

**Git clone** est utilisé pour cloner un repository git distant dans un répertoire local. Par défaut la commande recrée le répertoire contenant le dossier. git et y télécharge le contenu du repository.

**Git branch** permet de lister les branches que nous avons, de créer une nouvelle branche en locale, supprimer et renommer des branches.

**Git checkout** permet de naviguer entre les branches crées par git branch. Lorsqu'on utilise cette commande, les fichiers présents dans le répertoire pour correspondre à la version stockée dans cette branche et git enregistrera tous les nouveaux commits de la branche.

**Git pull** : est en fait la combinaison de deux autres commandes, git fetch suivi de git merge. Lors de la première étape, git pull exécute une commande git fetch étendue à la branche locale vers laquelle HEAD pointe. Une fois le contenu téléchargé git pull entre en workflow de merge.

**Git push** est utilisée pour charger le contenu d'un dépôt local vers un dépôt distant. Le push vous permet de transférer les commits de votre dépôt local vers un dépôt distant.

# TP2 : Manipulation d'un kernel Linux

## I. Environnement

Pour réaliser ce TP, nous avons tout d'abord cloné le dépôt git **https://android.googlesource.com/**. Ensuite, nous avons récupéré le dépôt git en faisant un git clone https ://github.com/gxfab/LO52_A2020. Dans ce dossier, il y'avait un dossier TP2 que nous avons récupéré et c'est dans ce dernier se trouve un fichier repo.

## II. Configuration

Pour la configuration, nous avons créé un dossier Android-kernel où nous allons télécharger les sources nécessaires à la configuration d'un noyau hikey-linaro. Ci-dessous les commandes utilisées :

- ***mkdir android-kernel && cd android-kernel***
- ***repo init -u https://android.googlesource.com/kernel/manifest -b hikey-linaro-android-4.14***

```
root@mamy-VirtualBox:/home/mamy/Telechargements/LO52_A2020/TP2/android-kernel# r
epo init -u https://android.googlesource.com/kernel/manifest -b hikey-linaro-and
roid-4.14
warning: Python 3 support is currently experimental. YMMV.
Please use Python 2.6 - 2.7 instead.

... A new version of repo (2.8) is available.
... You should upgrade soon:
    cp /home/mamy/Téléchargements/.repo/repo/repo /usr/bin/repo


Your identity is: mamy19 <nd.marie709@hotmail.com>
If you want to change this, please re-run 'repo init' with --config-name

Testing colorized output (for 'repo diff', 'repo status'):
  black     red       green     yellow    blue      magenta   cyan      white
  bold      dim       ul        reverse
Enable color display in this user account (y/N)? y

repo has been initialized in /home/mamy/Téléchargements
If this is not the directory in which you want to initialize repo, please run:
   rm -r /home/mamy/Téléchargements/.repo
and try again.
```

- ***repo sync***

Puis, on passe à la compilation du noyau avec la commande :

***Buil/build.sh***

Le binaire du noyau, les modules et l'image correspondante se trouvent dans le répertoire out/ android-hikey-linaro /dist.

```
  INSTALL include/linux/nfsd (5 files)
  INSTALL include/linux/raid (2 files)
  INSTALL include/linux/sched (1 file)
  INSTALL include/linux/spi (1 file)
  INSTALL include/linux/sunrpc (1 file)
  INSTALL include/linux/tc_act (15 files)
  INSTALL include/linux/tc_ematch (4 files)
  INSTALL include/linux/usb (14 files)
  INSTALL include/linux/wimax (1 file)
  INSTALL include/asm (35 files)
make : on quitte le répertoire « /home/mamy/Téléchargements/LO52_A2020/TP2/andro
id-kernel/out/android-hikey-linaro-4.14/hikey-linaro »
 Copying kernel UAPI headers to /home/mamy/Téléchargements/LO52_A2020/TP2/androi
d-kernel/out/android-hikey-linaro-4.14/dist/kernel-uapi-headers.tar.gz
============================================================
 Copying kernel headers to /home/mamy/Téléchargements/LO52_A2020/TP2/android-ker
nel/out/android-hikey-linaro-4.14/dist/kernel-headers.tar.gz
/home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel/hikey-linaro /home/mamy
/Téléchargements/LO52_A2020/TP2/android-kernel
/home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel
============================================================
 Files copied to /home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel/out/an
droid-hikey-linaro-4.14/dist
root@mamy-VirtualBox:/home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel#
```

## III. Configuration par défaut

La configuration par défaut relative à un noyau hikey-linaro pour une carte ranchu64 est ranchu64_defconfig et se trouve dans le répertoire : /home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel/hikey-linaro

Dans notre cas, nous allons faire une compilation car la toolchain est compilée sur la même machine que l'hôte et doit générer des binaires. Et pour le faire nous devons configurer deux variables qui sont ARCH et CROSS_COMPILE. Ces variables permettent de spécifier notre cible. On peut configurer ces deux variables de trois manières :

- Forcer leur valeur dans le makefile principal
- Les définir en variables d'environnements
- Les passer à la ligne de compilation

Nous avons choisi de les définir en variables d'environnement pour pouvoir forcer leur initialisation à l'ouverture de chaque terminal.

Pour cela, nous ouvrons un terminal et exécutons la commande permettant de charger la configuration par défaut d'une carte ranchu64:

**make ARCH = arm64 ranchu64_defconfig**

```
root@mamy-VirtualBox:/home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel/hi
key-linaro# make ARCH=arm64 ranchu64_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

Ensuite pour modifier la toolchain de compilation nous avons tapé la commande suivante :

Make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu

```
  CC      net/sunrpc/rpc_pipe.o
  CC      net/sunrpc/svc_xprt.o
  CC      net/sunrpc/xprtmultipath.o
  CC      net/sunrpc/stats.o
  CC      net/sunrpc/sysctl.o
  AR      net/sunrpc/sunrpc.o
  AR      net/sunrpc/built-in.o
  AR      net/built-in.o
  GEN     .version
  CHK     include/generated/compile.h
  UPD     include/generated/compile.h
  CC      init/version.o
  AR      init/built-in.o
  AR      built-in.o
  LD      vmlinux.o
  MODPOST vmlinux.o
  KSYM    .tmp_kallsyms1.o
  KSYM    .tmp_kallsyms2.o
  LD      vmlinux
  SORTEX  vmlinux
  SYSMAP  System.map
  OBJCOPY arch/arm64/boot/Image
  GZIP    arch/arm64/boot/Image.gz
root@mamy-VirtualBox:/home/mamy/Téléchargements/LO52_A2020/TP2/android-ke
```

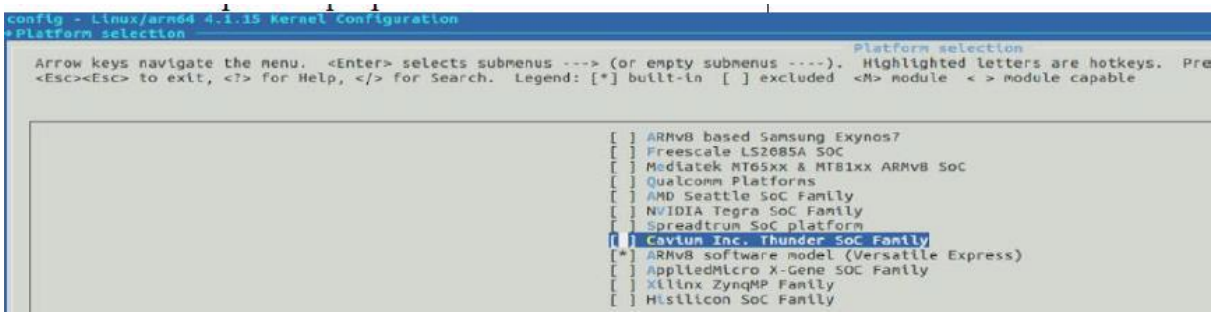
Cette compilation crée le fichier exécutable Image dans le dossier : /home/mamy/Téléchargements/LO52_A2020/TP2/android-kernel/hikey-linaro/arch/arm64/boot

## IV. Configuration avancée du noyau

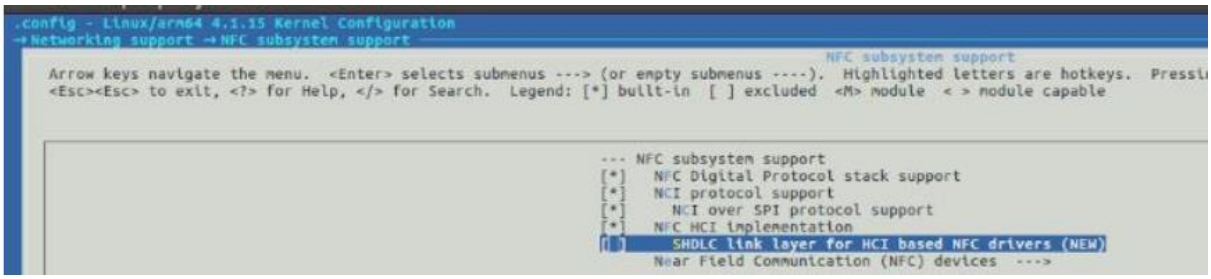
Pour la configuration personnalisée de la carte hikey, nous avons utilisé la commande Menuconfig en invite de commande.

La configuration de ladite carte devrait respecter les conditions suivantes :

- Ne la rendre compatible que pour la carte ARMv8 versatile




- Activer toutes les options du NFC et le driver relatif à un émulateur matériel



# TP3 : Création d'un device Android

## I. Environnement

Nous allons travailler dans le répertoire TP4 que nous avons récupéré et ce dernier contient deux sous dossiers Device et libusb-1.0.3.

## II. Objectif

L'objectif de cette séance est la création d'un decive Android. Nous allons ainsi pouvoir créer des makefiles qui permettent l'intégration de nouveaux composants au sein d'un device Android.

Le composant en question est « libusb » et le device Android devra hériter de nos travails réalisés au TP précédent réalisé sur « hickey-linaro ».

## III. Intégration de la libusb

### 1. Description

Dans le répertoire libusb-1.0.3 se trouve un fichier Android.mk dont le contenu est le suivant :

```
LOCAL_PATH := $(call my-dir)
ubdirs := $(addprefix $(LOCAL_PATH)/,$(addsuffix /Android.mk, \
libusb \
))
include $(subdirs)
```

Indiquons le contenu de ce fichier :

- **LOCAL_PATH : = $(call my-dir)** : Cette variable indique l'emplacement des fichiers sources dans l'arborescence de développement. Ici, la macro my-dir, fournie par le système de construction, renvoie le chemin du répertoire courant (libusb-1.0.3).
- **Addprefix $(LOCAL_PATH)** : permet un prefix au nom de base **$(addsuffix /Android.mk, \ libusb \):permet d'ajouter n suffixe (/Android.mk) à \ libusb **

- **include $(subdirs):** permet d'ajouter ces sous-dossiers.

## 2. Rédaction du fichier Android.mk du répertoire libusb-1.0.3/libusb

Dans ce dossier libusb1.0.3/libusb se trouve un fichier Android.mk. Nous allons présenter ci-dessous son contenu en suivant les règles :

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

# Sources
LOCAL_SRC_FILES:= \
                  core.c \
                  descriptor.c \
                  io.c \
                  sync.c \
                  os/linux_usbfs.c \
                  os/threads_posix.c


# Includes
LOCAL_C_INCLUDES += \
                $(LOCAL_PATH)/ \
                $(LOCAL_PATH)/os/

LOCAL_CFLAGS += $(TOOL_CFLAGS) -DTIMESPEC_TO_TIMEVAL -pthread
LOCAL_LDFLAGS := $(TOOL_LDFLAGS) -pthread

# Output
LOCAL_MODULE := libusb
LOCAL_MODULE_TAGS := optional
LOCAL_PRELINK_MODULE := false
INCLUDE $(BUILD_SHARED_LIBRARY)
```

## 3. Gestion des erreurs après compilation

Lors de la première compilation, nous allons avoir un message d'erreur indiquant que la macro TIMESPEC_TO_TIMEVAL n'est pas définie. Et pour résoudre cela, nous avons trouvé un include sur LOCAL_CFLAGS.

- **$(TOOL_CFLAGS) -DTIMESPEC_TO_TIMEVAL -pthread**

Une autre erreur apparait alors : «build/tools/apriori/prelinkmap.c(137) : library "libusb.so" not in pre-link map» parce que Android conserve la trace de toutes les bibliothèques partagées du système de fichiers racine (RFS). De ce fait, nous pouvons ajouter la bibliothèque à la carte pré-trouvée dans ./build/core/prelink-linux-arm.map ou en définissant la paramètre « LOCAL_PRELINK_MODULE : =false » dans le fichier Android.mk

- **LOCAL_PRELINK_MODULE := false**

### 4. Rédaction du fichier Android.mk pour compiler l'utilitaire lssub :

Ensuite si nous souhaitons compiler l'exécutable lssub qui va nous permettre de tester notre lssub, il faut créer un fichier Android.mk pour pouvoir compiler cet utilitaire. Pour cela, nous avons créé un dossier lssub à la racine et dans le dossier nous avons créé le fichier android.mk. Le contenu de ce fichier est ci-dessous :

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
LOCAL_SRC_FILES:= lsusb.c
LOCAL_MODULE := lsusb
LOCAL_C_INCLUDES += external/libusb-1.0.3/
LOCAL_SHARED_LIBRARIES := libc libusb
include $(BUILD_EXECUTABLE)
```

## IV. Définition d'un nouveau produit

### 1. Création de l'arborescence

Supposons que nous voulons créer notre produit dont le vendeur sera utbm et le non du produit sera lo52. Il héritera de la carte hikey. Pour cela nous devons créer l'arborescence Device/utbm/lo52 qui contient 5 fichiers :

- AndroidProduct.mk
- BoardConfig.mk
- Vendorsetup.sh
- CleanSpec.mk
- lo52.mk

## 2. Remplissage des fichiers créés

Dans cette partie nous allons présenter le contenu des fichiers et dossier (overlay) créés.

- VendorSetup.sh : on supposera que seuls les builds eng et user sont souhaités

```
add_lunch_combo full_lo52-eng
add_lunch_combo full_lo52-user
```

- BoardConfig.mk : le produit hérite de la carte hikey

```
include device/linaro/hikey/BoardConfig.mk
```

- AndroidProduct.mk

```
PRODUCT_MAKEFILES := $(LOCAL_DIR)/lo52.mk
```

- CleansSpec.mk

```
$(call add-clean-step, rm -f $(PRODUCT_OUT)/system/build.prop)
$(call add-clean-step, rm -f $(PRODUCT_OUT)/system/build.prop)
```

- lo52.mk

```
$(call inherit_product, device/linaro/hikey/hikey.mk)
DEVICE_PACKAGE_OVERLAYS := device/utbm/lo52/overlay
PRODUCT_NAME := lo52
PRODUCT_DEVICE:= lo52
PRODUCT_BRAND:= Android
PRODUCT_MODEL:= Android
include $ (call-subdir-makefiles)
```

## 3. Personnalisation du produit

Dans cette partie, nous allons ajouter quelques particularités à notre produit :

- Surcharger le fichier sym_keyboard_delete.png présent dans le projet Framework. Pour cela il faut créer l'arborescence overlay\frameworks\base\core\

- Personnaliser les propriétés suivantes : ro.hw avec la valeur lo52 et net.dns1 avec le DNS (8.8.8.8 )et net.dns2 avec le DNS (4.4.4.4). Pour cela, il faut modifier notre device.mk (lo52.mk) et ajouter le PRODUCT_PROPERTY_OVERRIDES.

```
$(call inherit_product, device/linaro/hikey/hikey.mk)

PRODUCT_PACKAGES += libsub_lib
DEVICE_PACKAGE_OVERLAYS := device/utbm/lo52/overlay
PRODUCT_PROPERTY_OVERRIDES += \
    ro.hw = lo52\
    net.dns1 = 8.8.8.8\
    net.dn2 = 4.4.4.4
PRODUCT_NAME := lo52
PRODUCT_DEVICE:= lo52
PRODUCT_BRAND:= Android
PRODUCT_MODEL:= Android


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

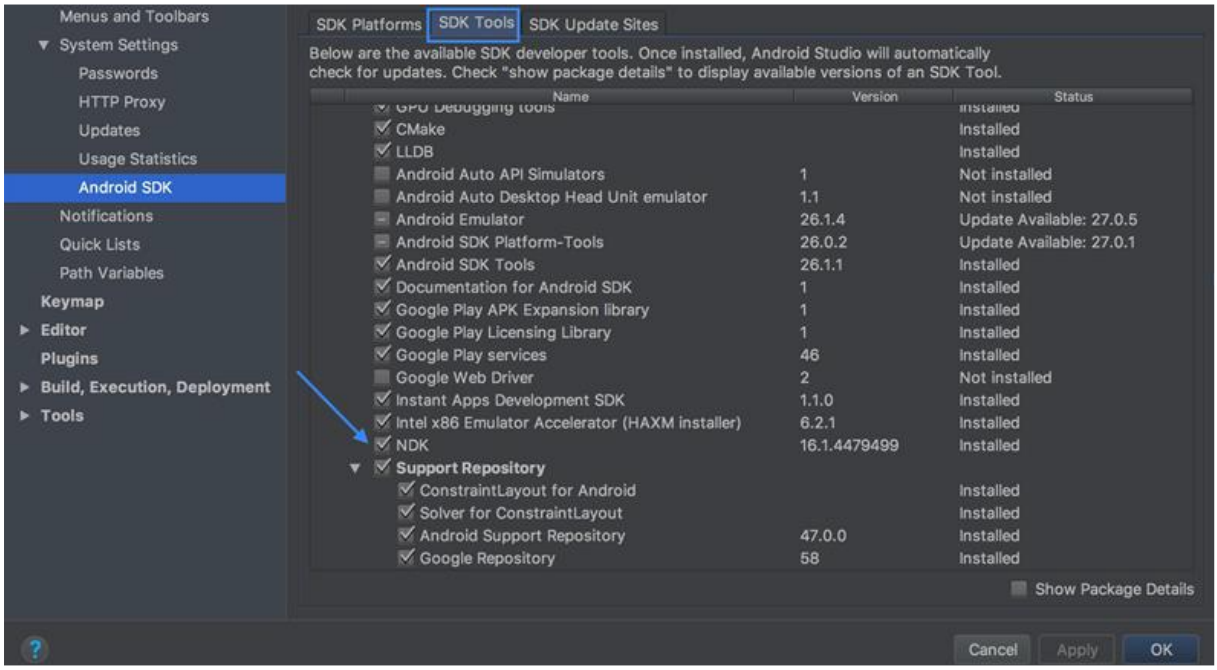
- Pour pouvoir inclure le composant, il faut utiliser la directive PRODUCT_PACKAGES permettant d'inclure le composant précédent. Et cette directive prendra la valeur du LOCAL_MODULE du fichier Androrid.mk précédemment.

```
PRODUCT_PACKAGES += libsub_lib
```

# TP4 : Utilisation de JNI

## I. Environnement

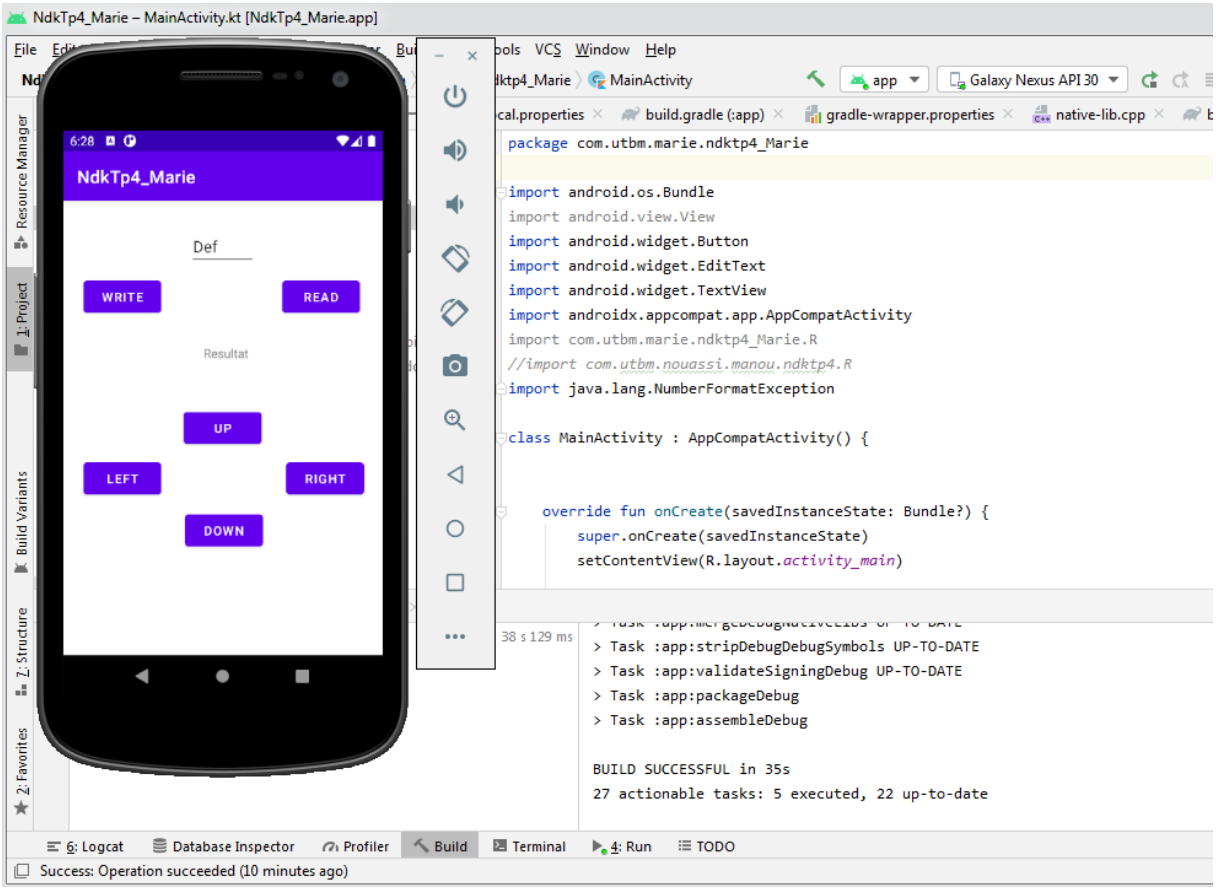
L'utilisation de JNI nécessite d'installer plusieurs composants. Dans les préférences Android Studio on cherche Android SDK, ensuite on va sur SDK Tools et on coche NDK pour l'installer puis CMake qui est un outil de compilation ainsi que LLDB qui permet à Android Studio de débugger la partie native du code.




## II. Objectif

L'objectif de cette séance était d'installer l'ensemble des composants nécessaires pour le NDK Android afin de créer une application de type NDK.

## III. Interface graphique de l'application

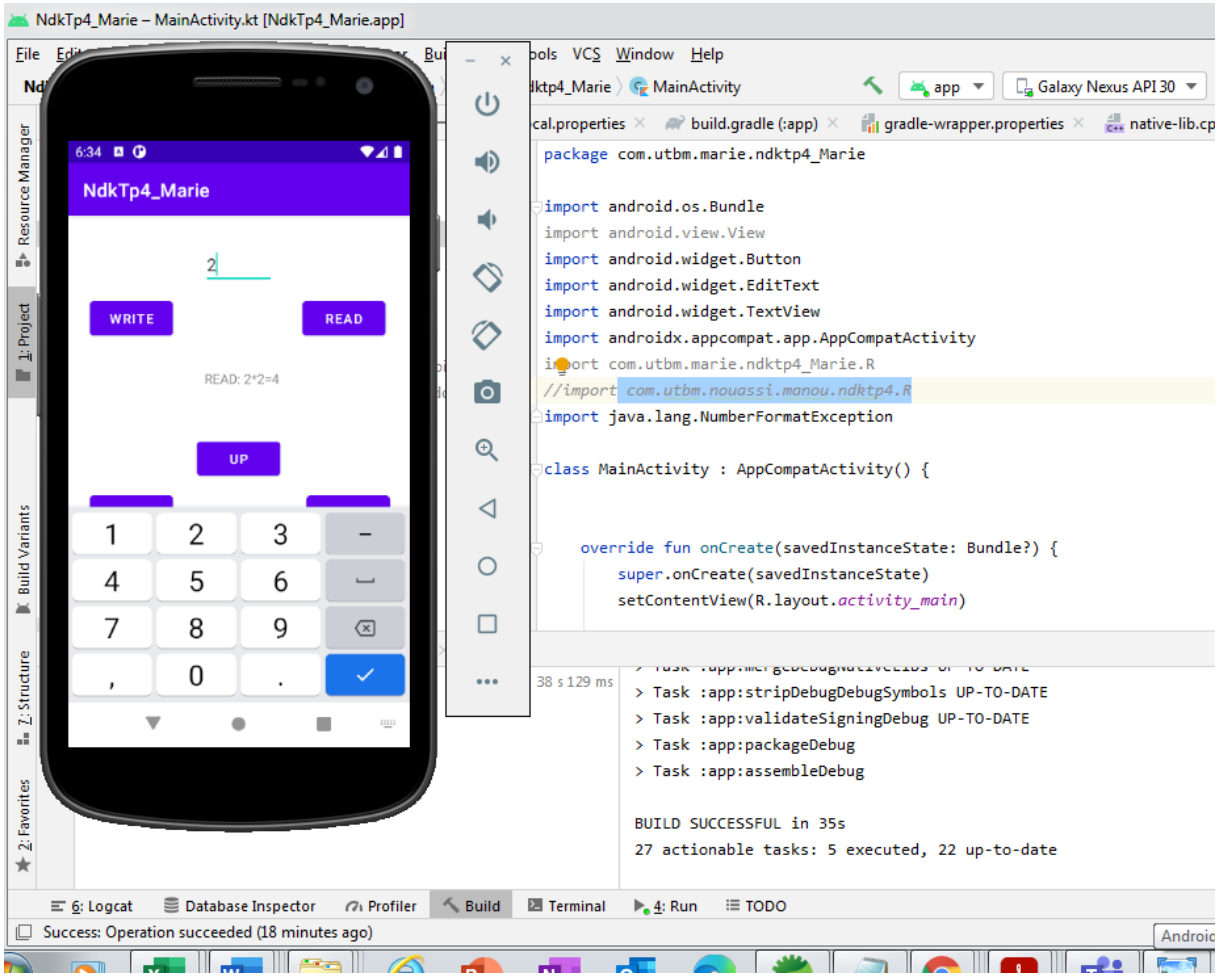



## 1. Fonctions JNI pour le bouton Read

La fonction Read appelle une fonction native qui retourne la valeur du champ de saisie. Comme valeur d'entrée on a la valeur du champ de saisie sous format jint et on envoie la valeur de retour de la fonction qui affichera dans le label a*a (a est le paramètre qui est passé à la fonction).

```
46  extern "C" JNIEXPORT jstring JNICALL
47  Java_com_utbm_marie_ndktp4_1Marie_MainActivity_read(JNIEnv *env, jobject thiz, jint a) {
48      std::stringstream ss;
49
50      if(a < 0 || a > 10){
51          ss << "ERREUR: le nombre saisi doit être compris entre 0 et 10";
52      }
53      else{
54          ss << "READ: " << a << "*" << a << "=" << a*a;
55      }
56
57      return env->NewStringUTF(ss.str().c_str());
58  }
```

Résultat :

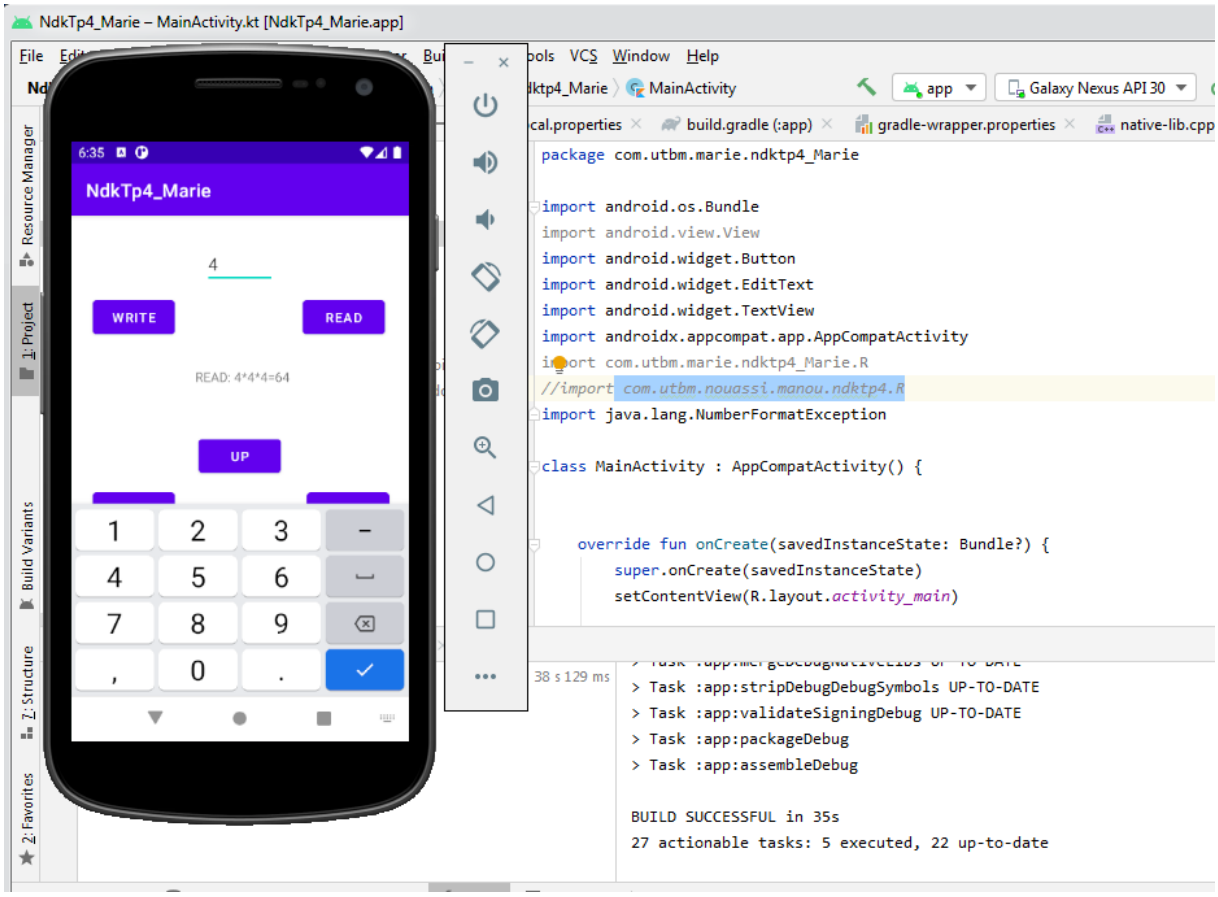



## 2. Fonction JNI pour le bouton Write

La fonction write effectue la même opération que la fonction read. La différence est que la fonction read envoie le carré d'un entier alors que write envoie la valeur de retour de la fonction qui affichera dans le label a*a*a (a valeur comprise entre 0 et10) c'est-à-dire le cube d'un entier.

```
28
29  extern "C" JNIEXPORT jstring JNICALL
30  Java_com_utbm_marie_ndktp4_1Marie_MainActivity_write(JNIEnv *env, jobject, jint a) {
31      std::stringstream ss;
32      if(a < 0 || a > 10){
33          ss << "ERREUR: le nombre saisi doit être compris entre 0 et 10";
34      }
35      else{
36          ss << "READ: " << a << "*" << a  << "*" << a << "=" << a*a*a;
37      }
38
39      return env->NewStringUTF(ss.str().c_str());
40  }
41
```

Résultat :

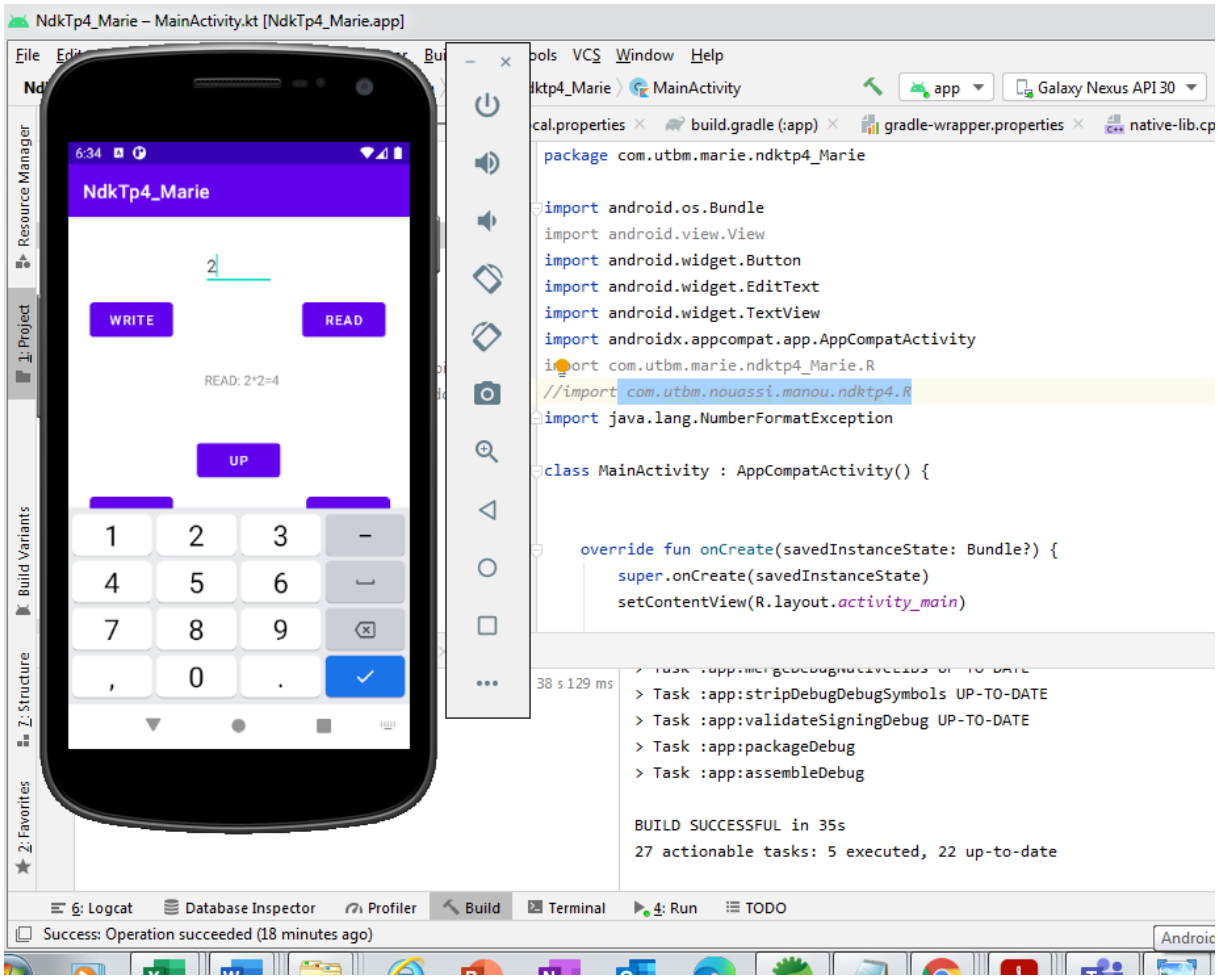



## 3. Fonction JNI

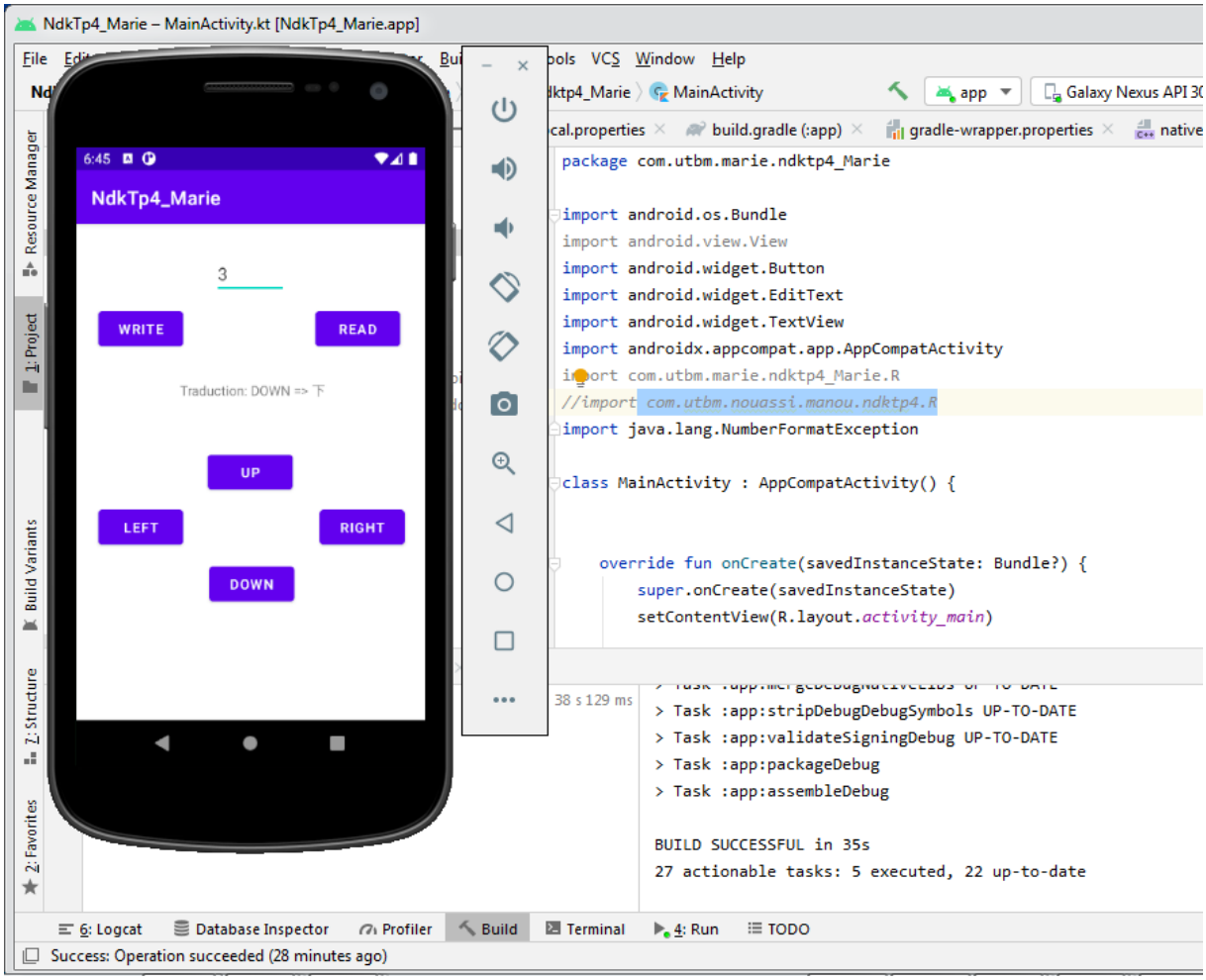
Les boutons LEFT, RIGHT, UP et DOWN appellent une fonction native qui va afficher le texte de la direction correspondante en Japonais en passant en paramètre le nom des boutons.

```cpp
#include <string>
#include <sstream>

extern "C" JNIEXPORT jstring JNICALL
Java_com_utbm_marie_ndktp4_1Marie_MainActivity_translateJaponais(JNIEnv *env, jobject, jstring direction) {
    const char *str = env->GetStringUTFChars(direction, isCopy: nullptr);
    std::string resultat;
    if(strcmp(str, "LEFT")==0) {
        resultat = "左";
    }
    else if(strcmp(str, "RIGHT")==0) {
        resultat = "右";
    }
    else if(strcmp(str, "UP")==0) {
        resultat = "を";
    }
    else if(strcmp(str, "DOWN")==0) {
        resultat = "下";
    }
    else {
        resultat = "エラー";
    }
    std::string mydirection = str;
    resultat = "Traduction: " + mydirection + " => " + resultat;
    return env->NewStringUTF(resultat.c_str());
}
```

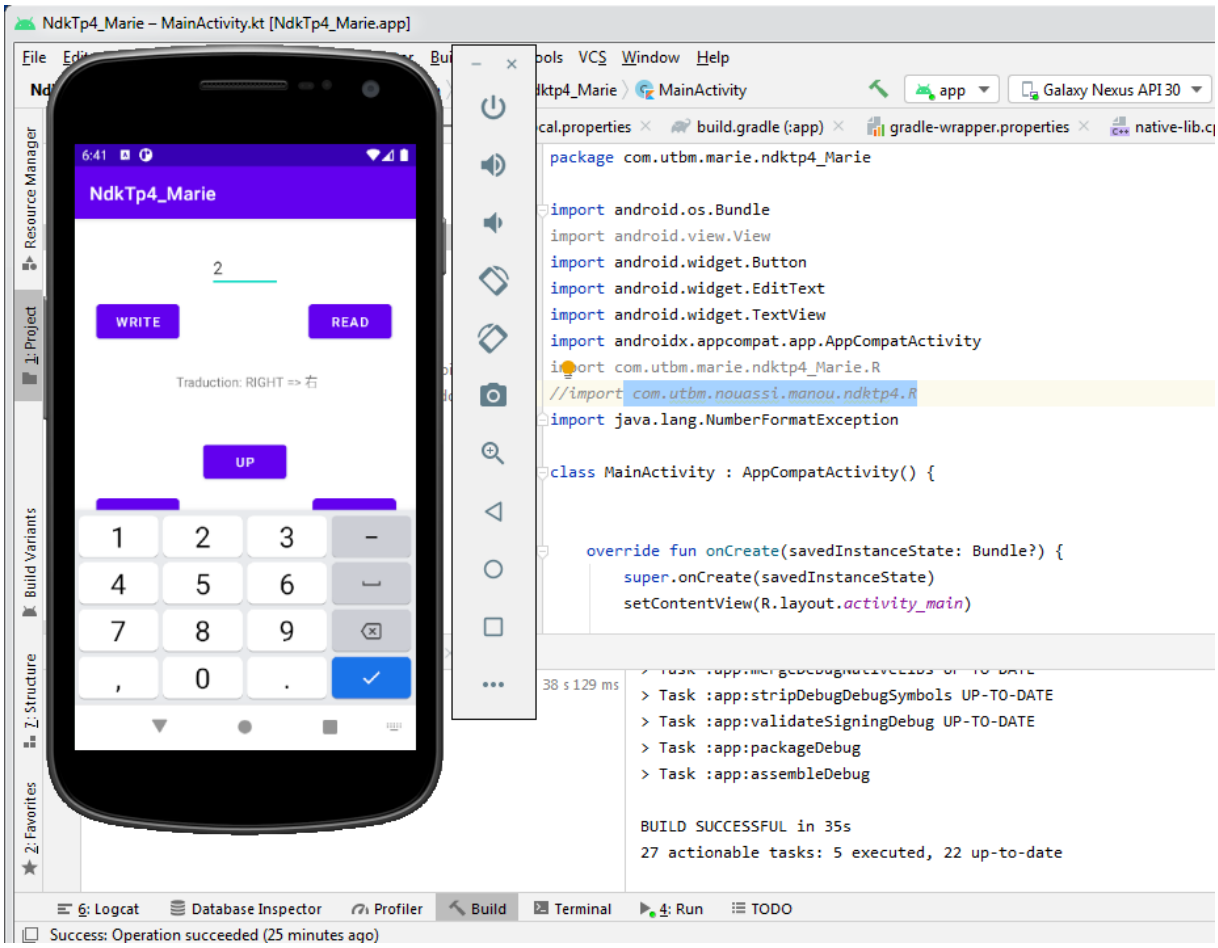
## ➢ Résultat Bouton UP



## ➢ Résultat bouton Down




## ➢ Résultat fonction Right




## ➢ Résultat fonction Left