Logo

Description automatically generated

UV : LO52 – Rapport

Travaux Pratiques

A2020

Étudiant : Gédéon AGOTSI

TP 1 : mise en place de l’environnement de développement

1. Mise en place de l’environnement de développement

L’objectif est de faire une prise en main de l’outil Git, configurer et mettre en place l’environnement de développement.

* 1. Installation et configuration de git
* Installation

Nous avons installé git sur notre machine en suivant les instructions, depuis le site de téléchargement de git : https://git-scm.com/downloads.

* Configuration

L’installation faite, nous avons procédé à la configuration de git avec nos informations d’identification sur github :

Text

Description automatically generated

Ensuite nous avons clone le dépôt git du projet sur github afin d’avoir une copie du code source sur notre machine.

Graphical user interface, application

Description automatically generated

Pour le suivi de notre travail, nous avons créé une branche git spécifique :

Graphical user interface, text, application, chat or text message

Description automatically generated

Pour se positionner sur notre branche :

Graphical user interface, text, application, chat or text message

Description automatically generated

Après avoir expliqué l’organisation de travail dans le fichier TP1.txt, on met à jour notre dépôt distant sur github.

Text

Description automatically generated

Git étant configuré et l’accès au répertoire de travail étant fait, nous passons à l’étape de développement de l’application android.

* 1. Développement application android

L’objectif de cette partie est de pouvoir installé et prendre en main l’environnement de travail pour le développement android et ensuite développer une application “Hello world ” de navigation entre différents activity.

* Installation

L’installation d’Android studio (IDE de développement android) est faite en suivant le guide officiel : <https://developer.android.com/studio/install>

* Réalisation du projet

Le projet consiste à créer deux activity android et ensuite passer de l’une à l’autre quand on clique sur un bouton. Nous développerons en JAVA.

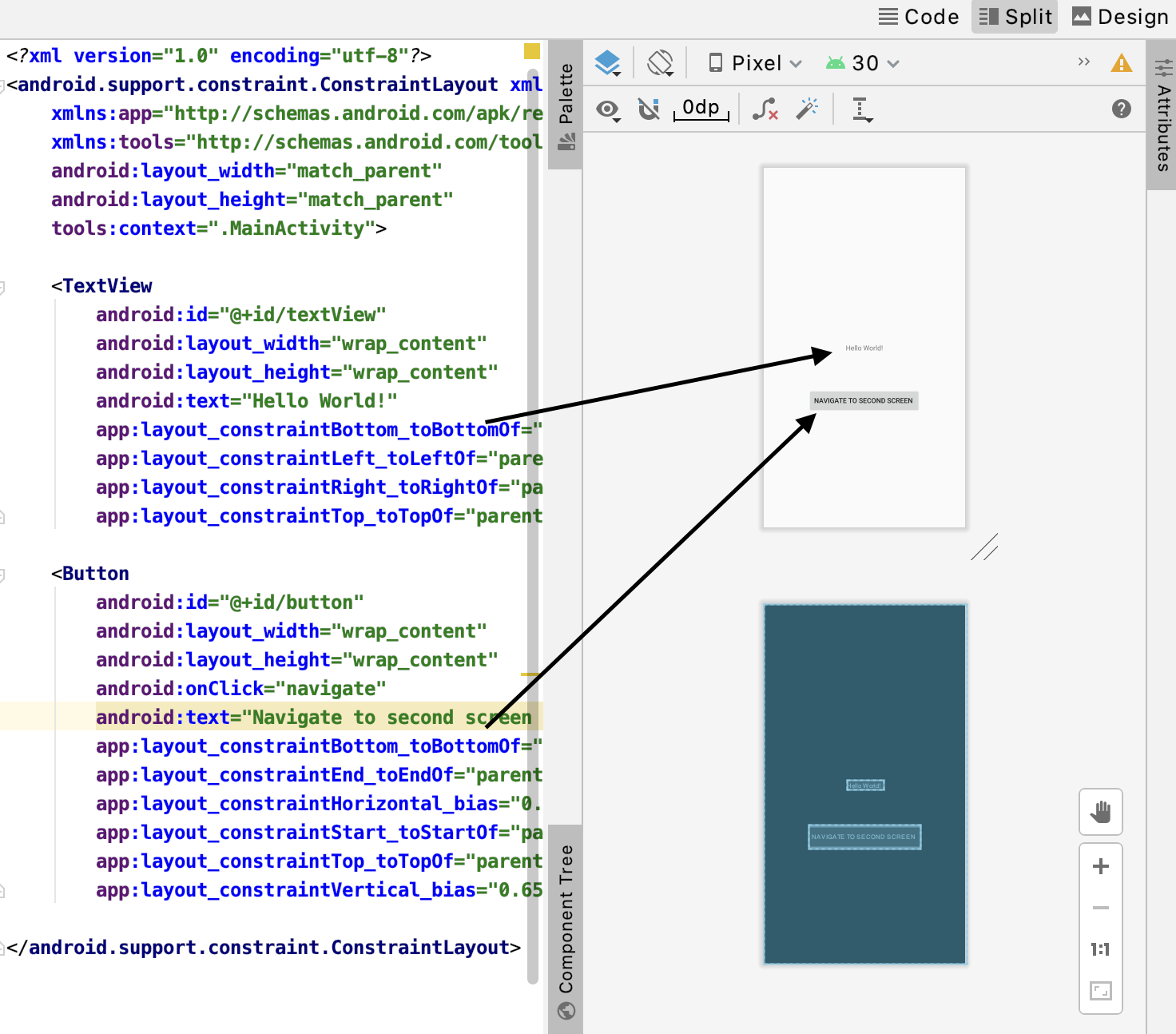
Après avoir créé un “projet vide” avec l’assistant d’android studio, nous avons créé l’activity principale et son layout associé :

Graphical user interface, text, application, email

Description automatically generated

Code source MainActivity

Il est associé à l’activity un fichier xml permettant de construire l’interface : on remarque la présence d’un texte « Hello Word » et d’un bouton intitulé : navigation to second screen.



Code source activity\_main.xml

Ensuite nous, allons créer la seconde activity : HelloActivity , voici ci-dessous le code source de HelloActivity.java et son fichier xml activity\_hello.xml décrivant l’interface

Graphical user interface, text, application, email

Description automatically generated

Code source HelloActivity

Nous avons déposé le composant textView dans le layout lié à HelloActivity affichant : Hello Word

A picture containing chart

Description automatically generated

Code source activity\_hello.xml

Le fichier string.xml contient la valeur des textes affichés sur nos composant, nous permettant de la modification de nos intitulés :

Text

Description automatically generated

Pour pouvoir passer du Main Activity vers le Hello Activity, nous avons créé la méthode navigation qui lors d’un réagira lors d’un évènement onClick du bouton spécifié dans l’activity\_main.xml

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Code source méthode navigate et button de navigation

* 1. Démo

Nous avons pu obtenu le résultat suivant :

Graphical user interface, application

Description automatically generated`Chart

Description automatically generated

Demo TP1

TP 2 : manipulation d’un Kernel Linux

1. Préparation de l’environnement et configuration du noyau

L’objectif de ce TP est de configurer un noyau Android avec des modifications et générer par la suite une image.

* 1. Préparation de l’environnement

Afin de pouvoir récupérer le code source du noyau hikey-linaro, nous avons installé l’indispensable outil, repo :

sudo apt-get install repo

* 1. Récupération du code source

Repo étant installé, nous pouvons l’utiliser pour récupérer le code source du noyau hikey-linaro dans le dossier android-kernel, préalablement créé.

mkdir ~/android-kernel

cd ~/android-kernel

repo init -u https://android.googlesource.com/kernel/manifest -b hikey-linaro-android-4.19

repo sync

* 1. Spécification de l’architecture du système

Pour pourvoir configurer le noyau récupéré, nous avons besoin de spécifier l’architecture système utilisé, dans notre cadre, c’est l’arm64.

export ARCH=arm64

* 1. Sauvegarde de la configuration par défaut

Étant donné, que nous allons charger la configuration relative pour une carte ranchu64, nous allons dans un premier temps charger la configuration par défaut du noyau hikey-linaro, qui se trouve dans le dossier hikey-linaro et la sauvegarder dans notre dossier config, sous le nom de « defconfig\_default »

cd ~/android-kernel/hikey-linaro

make defconfig

make savedefconfig

mkdir ~/config

cp defconfig ~/config/defconfig\_default

* 1. Chargement de la configuration pour la carte ranchu64

La configuration par défaut étant sauvegardée, dans un second temps, nous avons cherché l’emplacement du fichier config de la carte ranchu64, qui se trouve dans le dossier *arch/mips/configs/generic*. Nous copions ce fichier dans le dossier *arch/arm64/configs* afin de le charger.

make clean

cp arch/mips/configs/generic/board-ranchu.config arch/arm64/configs/

make board-ranchu.config

make savedefconfig

Après avoir chargé et sauvegardé le fichier config relative à la carte ranchu64 avec les commandes ci-hauts, nous le sauvegardons dans le dossier config sous le nom de « defconfig\_ranchu64 »

* 1. Différences entre les deux configurations (par défaut et ranchu64)

cp defconfig ~/config/defconfig\_ranchu64

Pour observer la différence entre le fichier de configuration par défaut (defconfig\_default) du noyau et du fichier de configuration relative à la carte ranchu64 (defconfig\_ranchu64), nous avons utilisé l’outil de comparaison de l’IDE Visual studio code.

Nous avons enregistré une vidéo que vous pouvez visualiser sur ce lien (<https://youtu.be/drKN3LU02dU>)   qui montre les différences que nous avons pu observer.

1. Modification de la configuration du noyau

Il nous a été demandé de modifier certaines configurations du noyau comme :

* Compatibilité pour la carte ARMv8 Versatile mais aussi Qualcomm et Realtek
* Activation du NFC et du protocole NFC HC
* Activation de l’option Frequency Scaling de la CPU
* Activation du support de l’HDMI CEC et activation du support LED

Pour le faire, nous avons utilisé l’outil graphique xconfig, nous permettant en un premier temps, de cocher les configurations à ajouter et de décocher les options superflues comme :

* Plan 9 Resource Sharing
* Carte son
* Bluetooth

make xconfig

Après avoir fait les modifications on ferme la fenêtre de l’outil, on enregistre les modifications et on le copie vers notre dossier config sous le nom de « ranchu64\_optimise »

make savedefconfig

cp defconfig ~/config/defconfig\_ranchu64\_optimise

1. Bonus : images du noyau

Pour le bonus, nous avons créé les images respectives des noyaux par défaut et optimisé.

Pour pouvoir le faire, nous avons téléchargé la toolchain de compilation du noyau et l’avons exporté comme une variable globale du système.

mkdir ~/toolchain

cd ~/toolchain

wget https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86\_64\_aarch64-linux-gnu.tar.xz

tar -xf gcc-linaro-7.5.0-2019.12-x86\_64\_aarch64-linux-gnu.tar.xz

Avec la toolchain, nous pouvons ainsi générer le noyau.

make -j8

Les images se trouvent dans le dossier arch/arm64/boot/

Nous avons observé la différence de taille pour ces deux images.

Nous avons créé un fichier script contenant nos scripts et un fichier TP2.txt contenant toutes nos commandes.

Nous avons envoyé les configurations, les images, le script et le fichier TP2.txt sur notre dépôt git.

TP 3 : Création d’un device Android

L’objectif de ce TP est d’intégrer de nouveaux composants et produits à Android

1. Implémentation de la libusb

Les fichiers sources et headers pour la compilation d’une libusb sur Android que nous avons identifié sont les suivantes :

* Fichiers sources :
* descriptor.c
* core.c
* io.c
* os/darwin\_usb.c
* os/linux\_usbfs.c
* sync.c
* Fichiers headers
* os/darwin\_usb.h
* os/linux\_usbfs.h
* libusb.h
* libusbi.h
  1. Écriture du fichier Android.mk

Nous avons identifié deux fichiers android.mk dans le dossier libusb-1.0.3.

Le premier fichier android.mk qui se trouve directement dans le dossier exemple est préalablement rempli et contient la portion de code suivante.

LOCAL\_PATH := $(call my-dir)

subdirs := $(addprefix $(LOCAL\_PATH)/,$(addsuffix /Android.mk, \

libusb \

))

include $(subdirs)

Étant donné, que ce fichier sert d’exemple, nous avons essayé de comprendre les lignes de codes en se référant au cours.

La première ligne

LOCAL\_PATH := $(call my-dir)

permet de spécifier la variable L*OCAL\_PATH* qui indique le répertoire des fichiers sources à utiliser en spécifiant le répertoire courant avec la macro *my-dir* .

Dans la deuxième ligne, on trouve les fonctions addprefix et addsuffix que nous avons cherché à comprendre en se référant à la documentation de make ici (<http://www.gnu.org/software/make/manual/make.html>).

Selon la documentation de make, addprefix est :

The argument *names* is regarded as a series of names, separated by whitespace; *prefix* is used as a unit. The value of *prefix* is prepended to the front of each individual name and the resulting larger names are concatenated with single spaces between them. For example,

$(addprefix src/,foo bar)

produces the result ‘src/foo src/bar’.

Et addsuffix est:

The argument names is regarded as a series of names, separated by whitespace; suffix is used as a unit. The value of suffix is appended to the end of each individual name and the resulting larger names are concatenated with single spaces between them. For example,

$(addsuffix .c,foo bar)

produces the result ‘foo.c bar.c’.

Dans notre contexte, cette ligne de code permet donc, en un premier temps avec la fonction addsuffix, d’ajouter comme suffix les fichiers du dossier libusb et de la racine du libusb-1.0.3 au ficher Android.mk, ensuite, avec la fonction addprefix,on ajoute la variable LOCAL\_PATH comme prefix des fichiers obtenus avec addsuffix.

On aura donc par exemple comme resultat de subdirs (libusb-1.0.3/file Android.mk, …).

Et par la suite on include la variable subdirs contenant les fichiers, (nous avons dû corriger la variable subdirs qui était écris ubdirs).

* Écriture du fichier android.mk

En se référant à la documentation d’android.mk ici

(<https://developer.android.com/ndk/guides/android_mk>) , nous avons écrit le contenu du fichier android.mk.

#Cette variable indique l'emplacement des fichiers sources

#dans l'arbre de développement.

#Ici, la fonction macro my-dir, fournie par le système

#de compilation, renvoie le chemin du répertoire courant

#(le répertoire contenant le fichier Android.mk lui-même).

LOCAL\_PATH := $(call my-dir)

#La variable CLEAR\_VARS pointe vers un Makefile GNU

#spécial qui efface de nombreuses variables LOCAL\_XXX,

#telles que LOCAL\_MODULE, LOCAL\_SRC\_FILES

#et LOCAL\_STATIC\_LIBRARIES

include $(CLEAR\_VARS)

#Definir le module à build ici libusb

LOCAL\_MODULE := libusb-1.0.3

#Enumérer les fichiers sources à compiler

LOCAL\_SRC\_FILES := core.c descriptor.c io.c sync.c os/darwin\_usb.c os/darwin\_usbfs.c

#La variable BUILD\_SHARED\_LIBRARY pointe vers un script

#GNU Makefile qui collecte toutes les informations que

#vous avez définies dans les variables LOCAL\_XXX depuis

#faut construire et comment le faire.

include $(BUILD\_SHARED\_LIBRARY)

* 1. Erreur de compilation
* TIMESPEC\_TO\_TIMEVAL

La macro TIMESPEC\_TO\_TIMEVAL n’étant pas été définie au préalable, nous avons eu une erreur. Afin de résoudre cette erreur nous avons ajouté la portion de code ci-dessous dans le fichier **io.c** qui permet de définir la macro

#define TIMESPEC\_TO\_TIMEVAL(tv, ts) \

do \

{ \

(tv)->tv\_sec = (ts)->tv\_sec; \

(tv)->tv\_usec = (ts)->tv\_nsec / 1000; \

} while (0)

* Seconde erreur

A la compilation de libusb, on a une seconde erreur : **build/tools/apriori/prelinkmap.c(137) : la bibliothèque 'libusb.so' not in prelinkmap** . Ceci est dû au fait qu'Android garde une trace de toutes les bibliothèques partagées du système de fichiers racine (RFS). Nous devons donc ajouter la bibliothèque à la carte prelinkmap qui se trouve dans le fichier ./build/core/prelink-linux-arm.map.

Pour le faire, nous avons d'ajouter libusb.so sous libqcamera.so.

libqcamera.so 0xA9400000

libusb.so 0xA8000000

1. Implémentation d’un nouveau produit android

2.1. Définition du produit

En respectant les contraintes du cahier des charges, le nom du produit est : lo52\_AgotsiGedeon et hérite du produit hikey de Linaro.

On créer l’arborescence device/lo52\_AgotsiGedeon en rajoutant les fichiers suivants pour la définition de notre produit :

* Android.mk
* AndroidProducts.mk
* BoardConfig.mk
* CleanSpec.mk
* lo52\_AgotsiGedeon.mk
* vendorsetup.mk

Nous avons récupéré les codes sources du device hikey linaro depuis github en faisant git clone <https://android.googlesource.com/device/linaro/hikey> , tout en se positionnant dans le dossier dans le dossier device.

En suivant la documentation d’android ( <https://source.android.com/setup/develop/new-device> ) pour la création d’un nouveau produit. Voici ci-dessous le contenu de nos fichiers.

* Fichier AndroidProducts.mk

PRODUCT\_MAKEFILES := $(LOCAL\_DIR)/lo52\_AgotsiGedeon.mk

Cette portion de code permet de préciser où se trouve ,le fichier makefile du produit avec la variable PRODUCT\_MAKEFILES.

* Fichier BoardConfig.mk

include device/hikey/hikey/BoardConfig.mk

Cette portion de code permet de respecter la contrainte d’héritage du produit hikey de linaro.

* Fichier CleanSpec.mk

$(call add-clean-step, rm -f $(PRODUCT\_OUT)/system/build.prop)

* Fichier lo52\_AgotsiGedeon.mk

Il contient les propriétés de définition du produit.

PRODUCT\_NAME := lo52\_AgotsiGedeon

PRODUCT\_DEVICE := lo52\_AgotsiGedeon

PRODUCT\_BRAND := UTBM

PRODUCT\_MODEL := LO52

PRODUCT\_MANUFACTURER := UTBM

Ensuite pour pouvoir définir l’héritage du produit hikey Linaro, on ajoute la ligne :

# Inherit from the common Open Source product configuration

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

2.2. Personnalisation du produit

Nous personnalisons les propriétés ro.hw à lo52, net.dns1 à 8.8.8.8 et net.dns2 à 4.4.4.4 en rajoutant dans le fichier lo52\_AgotsiGedeon la variable :

PRODUC\_PROPERTY\_OVERRIDES

PRODUCT\_PROPERTY\_OVERRIDES := \

ro.hw=lo52 \

net.dns1=8.8.8.8 \

net.dns2=4.4.4.4

* Ajout de la libusb aux packages de ce produit dans le fichier lo52\_AgotsiGedeon :

PRODUCT\_PACKAGES += \

Libusb-1.0.3

* Surcharge du fichier sym\_keyboard\_delete.png

Nous avons trouvé le fichier sym\_keyboard\_delete.png dans le projet git clone <https://android.googlesource.com/platform/frameworks/base> , après avoir cloner le projet, nous récupérons l’image, sym\_keyboard\_delete.png ,en fonction de la taille de l’écran dans le dossier lo52\_AgotsiGedeon\overlay\frameworks\base\core\res\res.On trouvera donc les fichiers suivants :

* drawable-en-hdpi/sym\_keyboard\_delete.png
* drawable-en-ldpi/sym\_keyboard\_delete.png
* drawable-en-mdpi/sym\_keyboard\_delete.png
* drawable-hdpi/sym\_keyboard\_delete.png
* drawable-mdpi/sym\_keyboard\_delete.png
* drawable-xhdpi/sym\_keyboard\_delete.png
* drawable-xxhdpi/sym\_keyboard\_delete.png

A la fin on aura comme contenu du fichier lo52\_AgotsiGedeon :

# Inherit from the common Open Source product configuration

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

PRODUCT\_PROPERTY\_OVERRIDES := \

ro.hw = lo52 \

net.dns1 = 8.8.8.8 \

net.dns2 = 4.4.4.4

DEVICE\_PACKAGE\_OVERLAYS := device/utbm/lo52\_AgotsiGedeon/overlay

#libusb

PRODUCT\_PACKAGES += libusb-1.0.3

#Product definition

PRODUCT\_NAME := lo52\_AgotsiGedeon

PRODUCT\_DEVICE := lo52\_AgotsiGedeon

PRODUCT\_BRAND := UTBM

PRODUCT\_MODEL := LO52

PRODUCT\_MANUFACTURER := UTBM

* Fichier vendorstetup.sh

add\_lunch\_combo lo52\_AgotsiGedeon-user

add\_lunch\_combo lo52\_AgotsiGedeon-userdebug

TP 4 : Utilisation de JNI