utbm

université de technologie
Belfort-Montbéliard

LO52

INFORMATIQUE MOBILE ET COMMUNICATIONS COURTES PORTEES

Rapport TPS

COMPTE RENDU DES TPS : TP1, TP2, TP3 et TP4

Rédigé par :

- NOUASSI Zechirine
- MANOU Abbaltu Auguste

Responsable de l'unité de valeur :

- M. BRISSET Fabien

Année académique 2020 / 2021

TABLE DE MATIERES

TABLE D'ILLUSTRATIONS

INTRODUCTION

Android est un système d'exploitation mobile crée par google et qui équipe la majorité des smartphones actuels. D'où son utilisation dans notre unité de valeur LO52 (Informatique mobile et communication courtes portées) que nous suivons dans le cadre de notre cursus d'ingénierie informatique à l'UTBM (Université de Technologie de Belfort Montbéliard).

Pour ce semestre d'automne 2020, les objectifs sont les suivants : la mise en place de l'environnement de développement ensuite, la définition du modèle de données pour une application Android puis, la manipulation du kernel linux, la création d'un device Android enfin, le développement de l'application et l'utilisation du JNI. Ce document retrace les différentes étapes suivies pour la réalisation de ces objectifs.

PARTIE I : MISE EN PLACE DE L'ENVIRONNEMENT DE DEVELOPPEMENT

Le but de cette partie est de nous familiariser avec le gestionnaire de code source Git et l'environnement de développement Android Studio. Pour cela nous avons évolué sous plusieurs étapes. Il sera donc question par la suite de développer toutes ces étapes qui nous ont permis de réaliser cette première partie.

I.1) TELECHARGEMENT, INSTALLATION ET CONFIGURATION DE GIT

Git étant déjà installé sur nos ordinateurs, ils nous restaient qu'à configurer git. Pour ce faire nous avons modifier le fichier de configuration de Git pour y faire figurer notre email UTBM et notre nom d'utilisateur au travers des commandes ci-après :

- git config --global user.name "<user_name>"
- git config --global user.email "<email>"

Clonage du dépôt https://github.com/gxfab/LO52_A2020

Après la configuration de Git, nous avons procédé au clonage du dépôt du TP à travers la commande :

- git clone **https://github.com/gxfab/LO52_A2020**

Cette commande crée un répertoire du même nom que le dépôt distant, initialise un répertoire .git à l'intérieur et récupère toutes les données de ce dépôt.

Création d'une branche

Le dépôt cloné, nous avons créé une branche pour notre groupe afin d'éviter les conflits lors des push des uns et des autres. Nous avons nommé notre branche **ManouAbbaltu_NouassiZechirine** (combinaison de noms des membres de l'équipe). Une fois créé nous nous sommes positionnés dessus à travers la commande :

- git checkout -b ManouAbbaltu_NouassiZechirine

Nous avons exécutés les commandes suivantes afin de mettre à jour le dépôt distant en uploadand nos changements :

- git add . (pour indexer tout le contenu du répertoire courant)
- git commit -m ''Initial commit''
- git push origin ManouAbbaltu_NouassiZechirine

I.2) CREATION DE NOTRE PREMIERE APPLICATION HELLOWORLD

Cette première application a pour but de constituer deux vues que nous allons appeler activité. La première activité doit afficher un bouton, au clic de ce bouton nous devons naviguer vers une autre activité affichant le label *Hello World*.

Pour le faire nous avons créé un projet sur Android Studio et nous l'avons sauvegardé dans le dossier TP1 du dépôt. A l'aide de l'assistant de création de projet, nous avons pu créer un projet avec déjà une activité vide appelé *Empty Activity*. Le langage de programmation choisi était *Kotlin* car nous avons déjà une bonne base en JAVA mais pas en KOTLIN et donc profiter pour apprendre davantage sur ce langage.

A l'ouverture du projet nous avons déjà le *MainActivity* avec ces composants *MainActivity.kt* (fichier kotlin) et *activity_main.xml* (fichier pour le layout). Dans le fichier *activity_main.xml* nous avons ajouté un bouton et un label.

Afin de mieux gérer nos valeurs, dans le fichier de ressource *String.xml* nous avons ajouté plusieurs valeurs :

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="app_message">Hello World</string>
    <string name="welcome_message">Bienvenue dans notre Application</string>
    <string name="button_message">Cliquer ici</string>
</resources>
```

Figure 1: Configuration fichier String.xml

Nous avons également appliqué des contraintes au bouton pour qu'il soit au milieu et le label pour qu'il placé comme un titre.

Ensuite nous avons créé une nouvelle activité avec ces composants bien sûr *HelloWordActivity.kt* et *activity_hello_word.xml* afin qu'il affiche le texte « Hello World »

Une fois les deux activités misent en place, nous avons donc créer la méthode, *afficher* dans le fichier *MainActivity.kt* et lier cette méthode au layout *activity_main.xml* en tant que propriété « OnClick ». Ci-après le code qui nous a permis de réaliser ceci.

```kotlin
package com.lo52.tp1.helloworld

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View

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

    fun afficher(view : View){
        val intent = Intent(this, HelloWordActivity::class.java)
        startActivity(intent)
    }
}
```

Figure 2 : Méthode afficher de l'activité MainActivity

```xml
<Button
    android:id="@+id/button"
    android:layout_width="174dp"
    android:layout_height="66dp"
    android:onClick="afficher"
    android:text="@string/button_message"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Figure 3: Propriété OnClick affectée au bouton

Cette méthode permet de passer de l'activité *MainActivity* à *HelloWordActivity* grâce à la méthode *startActivity* tout ceci une fois le bouton cliqué.

Nous avons pu tester l'application grâce à notre Android. Nous avons pu constater qu'au click du bouton, il affiche bien la deuxième activité.

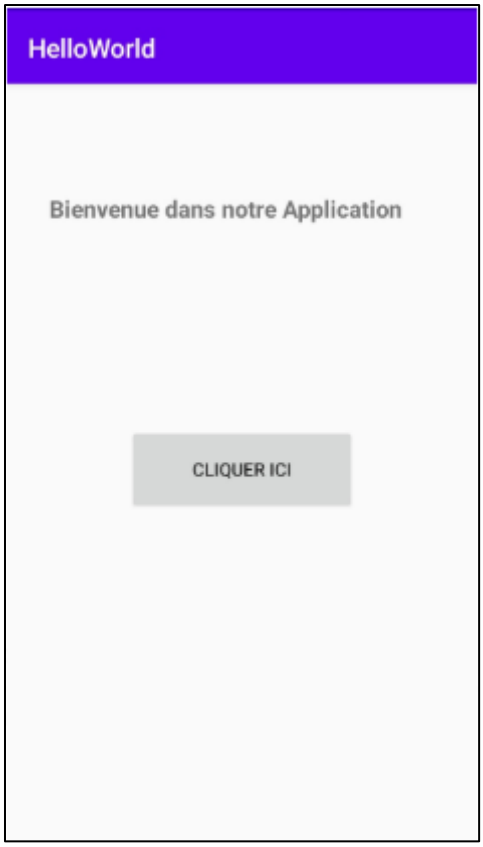

Figure 5: Activité 1

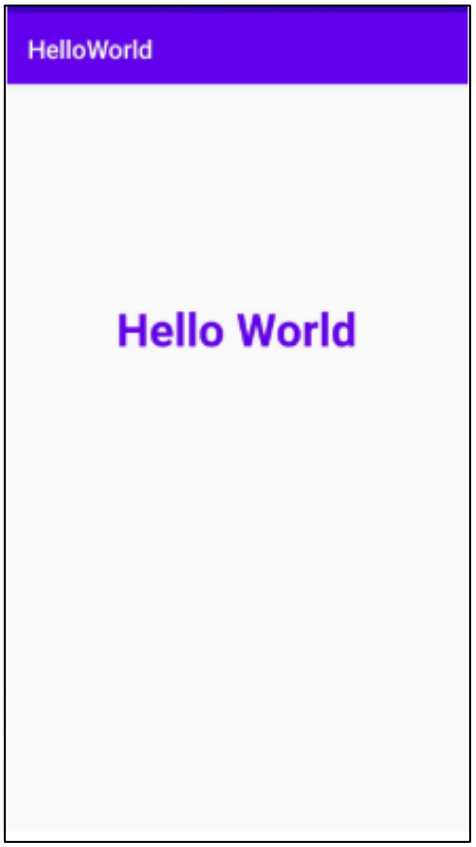

Figure 4 : Activité 2

PARTIE II: MANIPULATION D'UN KERNEL LINUX

I.1) PREPARATION D'UN ENVIRONNEMENT PERMETTANT DE CONFIGURER UN NOYAU HIKEY-LINARO

Pour ce faire, nous avons suivi les recommandations indiquées sur https://source.android.com/setup/build/building-kernels . Nous sommes passés par les étapes suivantes :

Installation de l'outils repo

Repo est un outil aidant à gérer plusieurs dépôts Git. Il automatise certaines parties des flux de développement.

- $ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
- $ chmod a+x ~/bin/repo

Création du dossier contenant les fichiers du noyau :

Nous avons créé le dossier à travers la commande suivante.

- $ mkdir android-kernel && cd android-kernel

Initialisation de repo et téléchargement dans le répertoire courant avec la branche *hikeylinaro*

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

Téléchargement des outils de compilation dans un dossier :

Ainsi nous créerons un dossier toolchain dans lequel nous décompresserons le fichier gcc-linaro-7.5.0-2019.12 téléchargé.

- $ 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 aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz

Définition de l'architecture CPU et spécification du chemin (jusqu'au préfixe des outils de compilation) :

Cette partie sera essentielle pour la compilation du noyau.

- $ export CROSS_COMPILE=~/toolchain/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
- $ export ARCH=arm64

I.2) CONFIGURATION ET COMPILATION (ETAPE BONUS)

Dans cette partie nous compilons notre noyau en la spécifiant les configurations correspondantes.

- **Configuration par défaut relative à un noyau ranchu64 :** cette configuration se retrouve dans le dossier *arch/mips/configs/generic/board-ranchu.config*
- **Chargement de la configuration par défaut :** Pour ce faire nous avons copié le fichier *board-ranchu.config* vers le dossier *arch/arm64/configs* et charger le fichier avec la commande suivante. Bien avant de charger cette configuration, nous avons effectué une sauvegarde de la configuration existante (vers le dossier configuration) à l'aide de la commande ***make savedefconfig***.

  $ make board-ranchu.config
- **Différence entre les deux configurations :** Nous avons utilisé l'outil ***kdiff3*** pour afficher la comparaison des deux configurations dans une interface graphique à travers la commande suivante.

  $ kdiff3 defconfig ../../configurations/defconfig_first_version

  A l'aide de kdiff3, nous avons exporté la comparaison dans le fichier difference.pdf dans le dossier configurations.

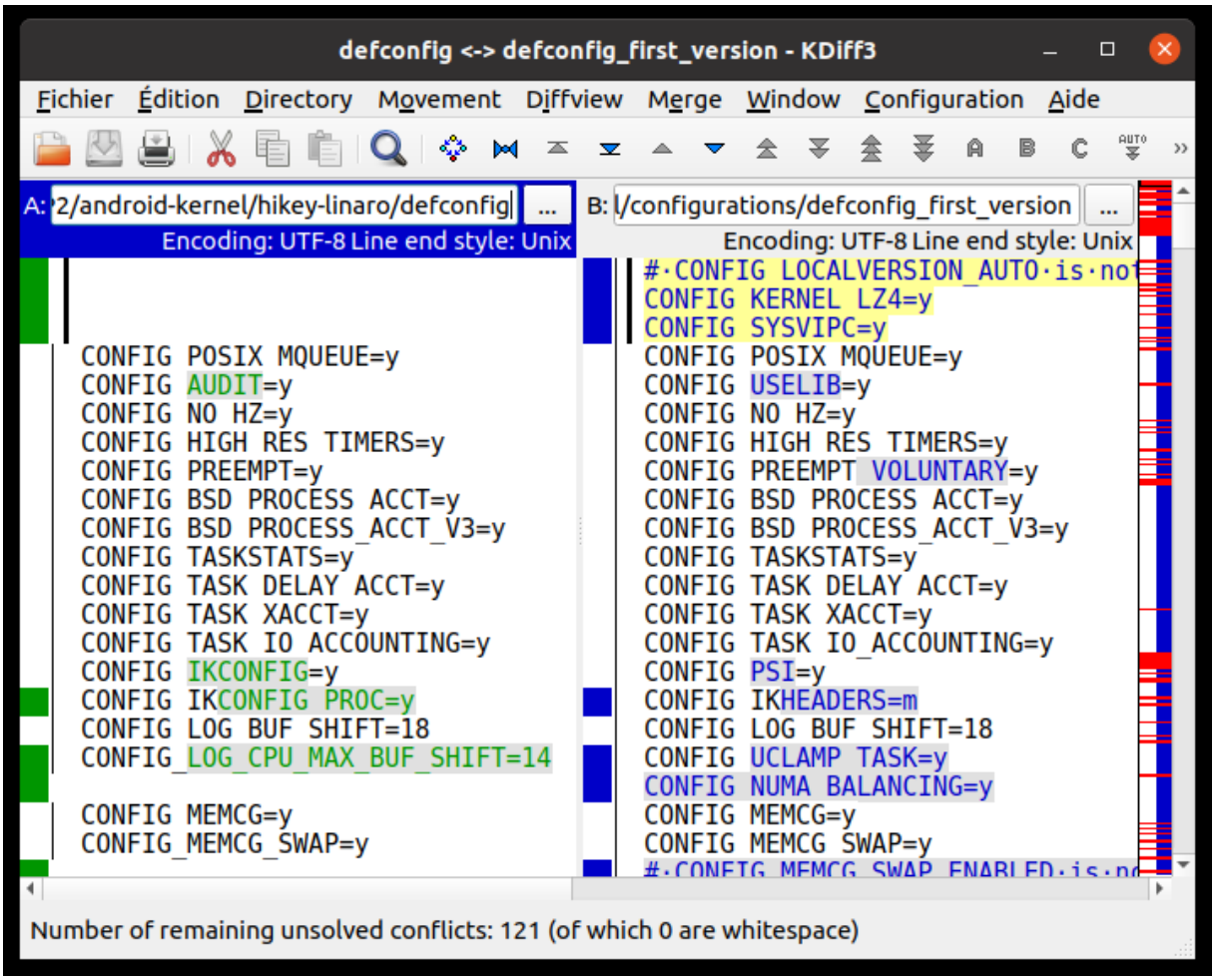

Figure 6: Interface de kdiff3

- **Script d'automatisation des commandes** : Nous avons créé un fichier script.sh dans lequel nous mettons la liste des commandes tapées précédemment afin d'automatiser les commandes précédentes. Ce fichier peut être exécuté à l'aide de la commande suivante :

  $ sh script.sh

- **Modification et Génération de la configuration par défaut :** Nous avons modifier la configuration par défaut à l'aide de la commande **make xconfig.** Cette commande exécute un utilitaire qui nous présente graphiquement. En ce qui concerne la désactivation des options superflues, nous n'avons pas pu détecter quelle option pouvait être considérée comme superflue et nous avons donc désactivé les options suivantes :
  - Bluetooth system support
  - SPI support

- Mailbox Hardware Support

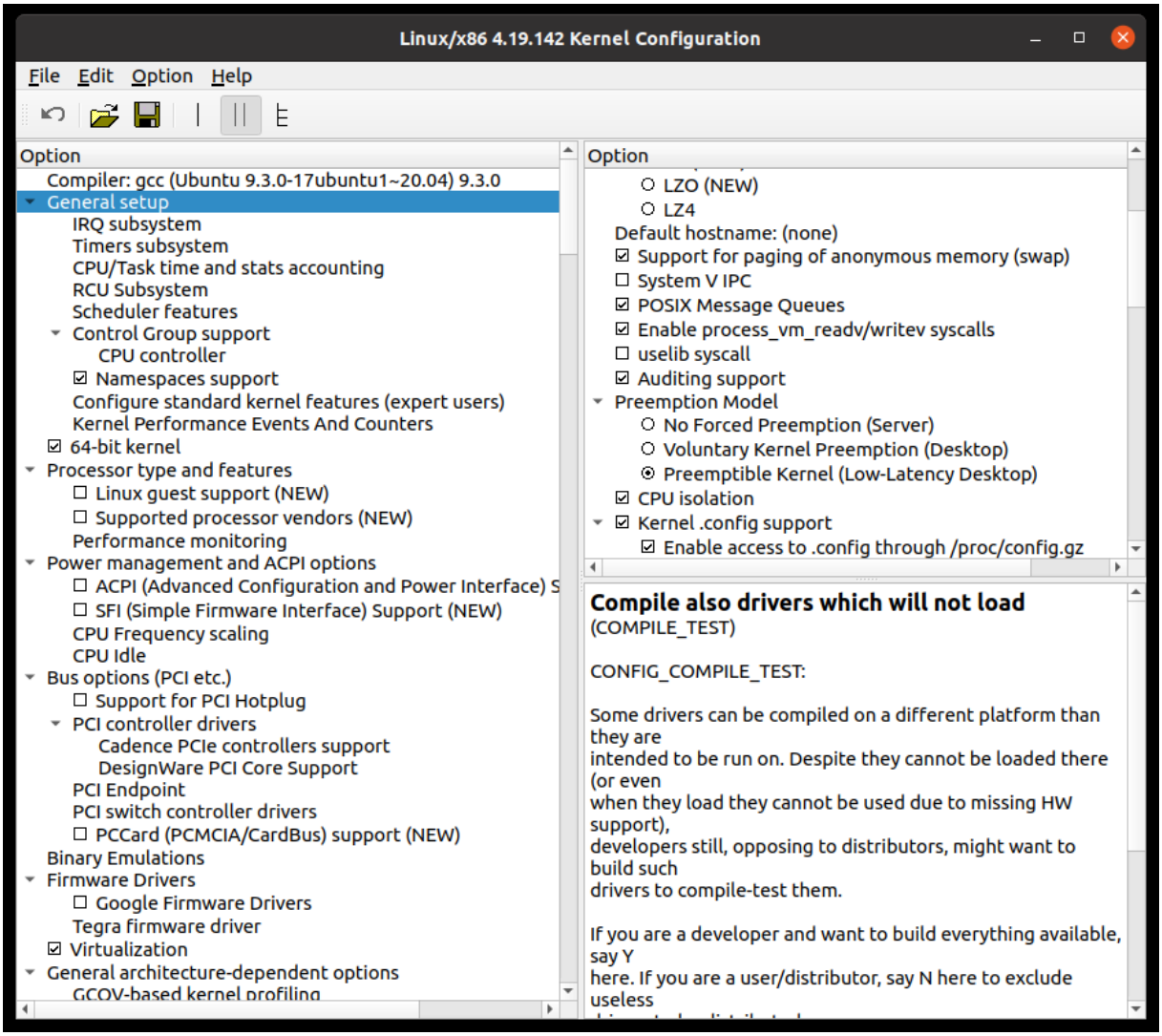

Figure 7: Interface présentée par le make xconfig

- **Compilation (Etape bonus) :** la compilation se fait avec la commande ***make.*** Afin que chaque configuration puisse être compilée, elle doit être chargée par défaut avant l'exécution de la commande ***make***.
- **Envoi par git**

- **Fichier. gitignore :** Afin de pouvoir envoyer uniquement les configurations et les scripts nous avons créé un fichier .gitignore à la racine de notre dossier(TP2) qui contient la liste des fichiers/dossier à ne pas envoyer. Ce fichier contient la ligne « **android-kernel**/ » qui permet d'envoyer tous les fichiers exceptés le dossier android-kernel.
- **Envoi :** Nous envoyons notre projet à travers les commandes suivantes :

- $ git add .
- $ git commit -m “TP2”
- $ git push