

LO52 – Rapport de TPs

Emilien MONCAN

Grégoire DUVAUCHELLE

I. Partie 1

La première partie que nous devons réaliser était de cloner la branche de notre dépôt git ainsi que de créer une application Android. Cette partie a donc été réalisée simplement et rapidement.

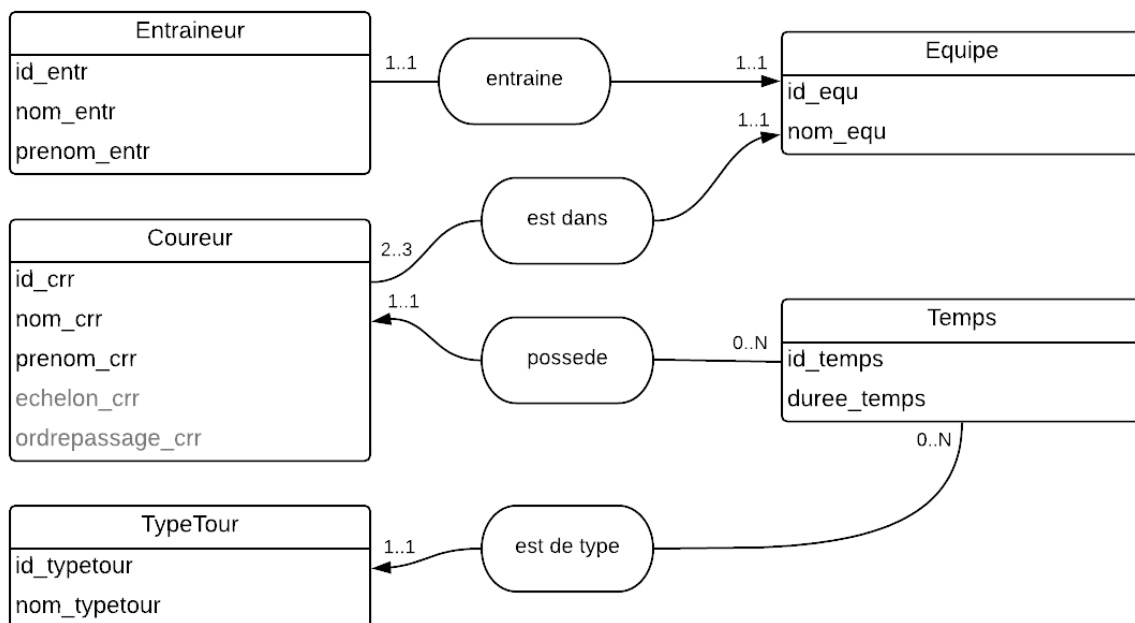
II. Partie 2

Une équipe est possédée par un et un seul entraîneur. Un entraîneur n'entraîne qu'une seule équipe. Chaque équipe peut posséder de 2 à 3 coureurs.

Les équipes sont formées automatiquement par l'application car il faut respecter la différence maximale de 5 points entre la somme des échelons de chaque membre de l'équipe. L'ordre de passage, contenu dans Coureur, est attribué en même temps que la création des équipes (attribué de manière aléatoire par l'application).

Chaque coureur possède 0 ou N temps. Chaque temps est enregistré avec un type de tour. Le type de tour correspond à : sprint, pit-stop, ateliers confondus... Il est plus simple de faire comme ceci car si un type de tour vient à être modifié, cela permet simplement de ne modifier qu'un seul enregistrement. De plus il est donc simple de pouvoir ajouter un nouveau type de tour. Les temps moyens et meilleurs temps de chaque joueurs / équipes ne sont pas stockés mais seront calculés automatiquement lors de "l'affichage". Ce calcul "au dernier moment" permet de gagner de la place niveau stockage et évite un nombre de requêtes beaucoup trop important pour un simple ajout de temps.

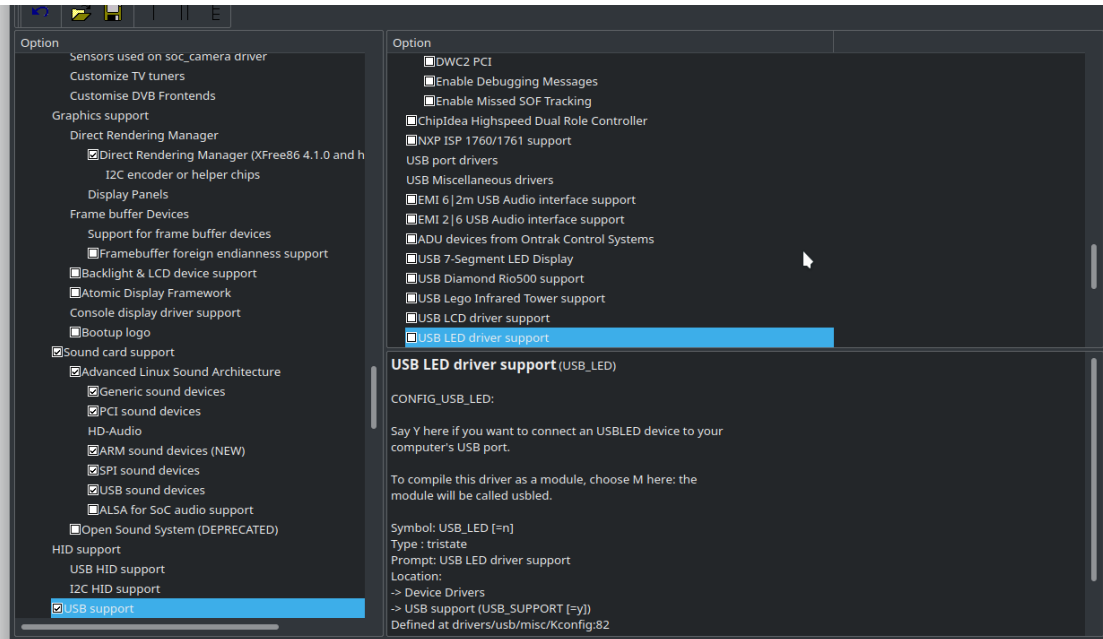
Figure 1 : Schéma de la base de données



III. Partie 3

```
1.  # Clonage du dépôt contenant les sources qui nous intéressent
2.  git clone https://android.googlesource.com/kernel/hikey-linaro
3.
4.  # Création de la branche et déplacement dans celle-ci
5.  git checkout -b android-hikey-linaro-4.1 origin/android-hikey-linaro-4.1
6.
7.  # Création des deux variables d'environnement les plus importantes pour la compilation
8.  export ARCH=arm64
9.  export CROSS_COMPILE=/usr/bin/aarch64-linux-gnu-
10.
11. # Chargement de la configuration
12. make hikey_defconfig
13.
14. # Compilation (oui, il lui offre la totalité de notre processeur parce qu'on est des fous)
15. # Cette compilation met un certain temps à se réaliser
16. make -j8
17.
18. # ----- Modification de la configuration -----
19. # Utilisation de kconfig
20. # Recherche des différentes configurations à changer
21.
22. # Compatibilité pour la carte ARMv8 Versatile uniquement
23. CONFIG_ARCH_VERSATILE=y
24.
25. # Activation du NFC et d'un driver spécifique à un émulateur matériel
26. CONFIG_NFC=y
27. CONFIG_NFC_SIM=y
28.
29. # Activation de l'option permettant de mettre un logo/image au boot
30. CONFIG_LOGO=y
31.
32. # Désactivation de MTP Gadget et activation de l'USB LED
33. # Pour pouvoir désactive MTP Gadget, activation préalable de USB functions configurable through configs
34. CONFIG_USB_CONFIGFS=y
35. # Désactivation de MTP Gadget
36. CONFIG_USB_CONFIGFS_F_MTP=n
37. # Activation de l'USB LED
38. CONFIG_USB_LED=y
39.
40. # Activation en dur de l'option TI SOC Drivers
41. CONFIG_SOC_TI=y
42.
43. # Recompilation de l'image et comparaison des deux images
44.
45. # La différence entre les 2 configuration fait 3 400 lignes pour seulement 7 changements
46. # Les images font exactement le même poids à 4 Ko près
47.
48. # Génération de la configuration par défaut pour cette nouvelle configuration
49. make savedefconfig
```

Figure 2 : Interface de configuration Kconfig



IV. Partie 4

Les fichiers à inclure lors de la compilation sont :

- Les fichiers source (core.c ...)
- Le header libusb.h
- Les fichiers :
 - o os/linux_usbfs.c
 - o os/linux_usbfs.h

En effet, Android étant basé sur linux, les fichiers dont le nom commence par Darwin sont pour MacOS. Nous écrivons ensuite le Android.mk comme appris en TD :

1.	#Android.mkforlibusb.
2.	
3.	commonSources:=\
4.	core.c\
5.	descriptor.c\
6.	io.c\
7.	libusb.h\
8.	sync.c\
9.	linux_usbfs.c\
10.	linux_usbfs.h
11.	
12.	LOCAL_PATH:= \$(call my_dir)
13.	
14.	include \$(CLEAR_VARS)
15.	
16.	LOCAL_SRC_FILES:= \
17.	\$(commonSources)
18.	
19.	LOCAL_C_INCLUDES:=\
20.	os/
21.	

22.	LOCAL_MODULE:= libusb
23.	
24.	LOCAL_MODULE_TAGS:= optional
25.	
26.	#correction de l'erreur 1
27.	LOCAL_C_FLAGS+= -pthread -DTIMESPEC_TO_TIMEVAL
28.	
29.	LOCAL_LD_FLAGS:= -lstdc++ -lc
30.	
31.	#correction de l'erreur 2
32.	LOCAL_PRELINK_MODULE:= false
33.	
34.	include \$(BUILD_SHARED_LIBRARY)

Il faut changer le LOCAL_MODULE avec le nom que nous souhaitons lui donner, ici ce sera « libusb ». Nous complétons ensuite les C flags et LD flags avec le contenu du Makefile original (-pthread, -lstdc++ et -lc).

Pour corriger la première erreur, il faut ajouter aux C flags le paramètre « -DTIMESPEC_TO_TIMEVAL » qui permettra de créer la macro dans le code C. La deuxième erreur se corrige simplement en ajoutant la ligne « LOCAL_PRELINK_MODULE := false »

Implémentation du nouveau produit Android :

lo52_silentpangolin.mk :

1.	<i># Héritage du produit hikey de Linaro</i>
2.	\$(call inherit-product, device/linaro/hikey/hickey.mk)
3.	
4.	<i># Ajout de la libusb aux packages du produit</i>
5.	PRODUCT_PACKAGES += libusb
6.	
7.	<i># Personnalisation des propriétés</i>
8.	PRODUCT_PROPERTY_OVERRIDES := ro.hw=lo52 net.dns1=8.8.8.8 net.dns2=4.4.4.4
9.	
10.	<i># Surcharge des fichiers</i>
11.	DEVICE_PACKAGE_OVERLAYS:= \$(LOCAL_DIR)/overlay
12.	
13.	<i># Nom du produit</i>
14.	PRODUCT_NAME:= lo52_silentpangolin
15.	PRODUCT_BRAND:= lo52_silentpangolin
16.	PRODUCT_MODEL:= lo52_silentpangolin
17.	PRODUCT_DEVICE:= lo52_silentpangolin

AndroidProducts.mk :

1.	PRODUCT_MAKEFILES:= \$(LOCAL_DIR)/lo52_silentpangolin.mk
----	--

BoardConfig.mk :

1.	<i># Héritage du produit hikey de Linaro</i>
2.	include device/linaro/hikey/hikey/BoardConfig.mk

CleanSpec.mk :

- | | |
|----|--|
| 1. | <code>\$(call add-clean-step, rm-f \$(PRODUCT_OUT)/system/build.prop)</code> |
| 2. | <code>\$(call add-clean-step, rm-f \$(PRODUCT_OUT)/system/build.prop)</code> |

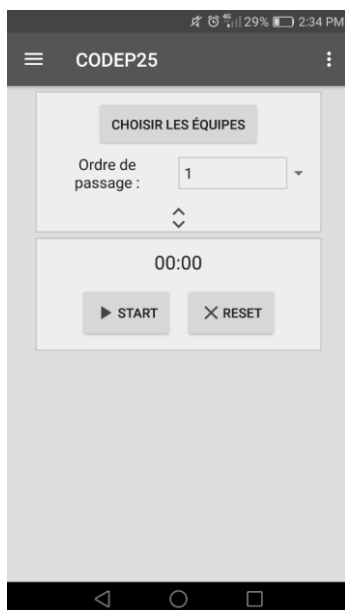
vendorsetup.sh :

- | | |
|----|--|
| 1. | <code>add_lunch_combo lo52_silentpangolin-eng</code> |
| 2. | <code>add_lunch_combo lo52_silentpangolin-user</code> |
| 3. | <code>add_lunch_combo lo52_silentpangolin-userdebug</code> |

V. Partie 5

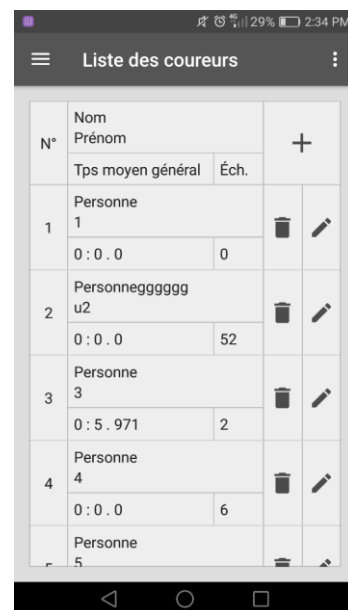
MainActivity

Le bouton « Choisir les équipes » permet de choisir les équipes qui vont participer à la course. L'ordre de passage permet simplement de choisir le coureur de l'équipe qui va passer. Le bouton « Start » permet de lancer le chronomètre et le bouton « Reset » offre la possibilité de recommencer la course.



PlayerActivity

Cette activité affiche les noms, prénoms, échelons et le temps moyen général de chaque coureur. Il est possible de supprimer un coureur, d'en ajouter un nouveau ou encore d'en modifier un existant.



TeamActivity

Cette activité affiche les équipes ainsi que les coureurs qui la composent.

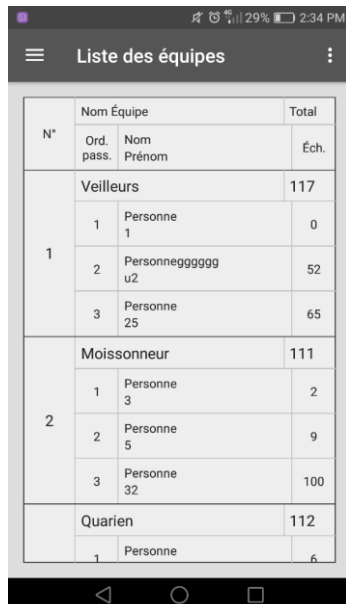


Tableau de la liste des équipes :

N°	Nom Équipe	Total
1	Veilleurs	117
	1 Personne	0
	2 Personnegggggg u2	52
	3 Personne 25	65
2	Moissonneur	111
	1 Personne 3	2
	2 Personne 5	9
	3 Personne 32	100
	Quarien	112
	1 Personne	6

RankingPlayerActivity

Cette activité affiche les différents temps d'un coureur selon 4 catégories : sprint, fractionné, pit-stop et général (temps cumulé d'une course).



Classement Individuel

Type de tour : Sprint

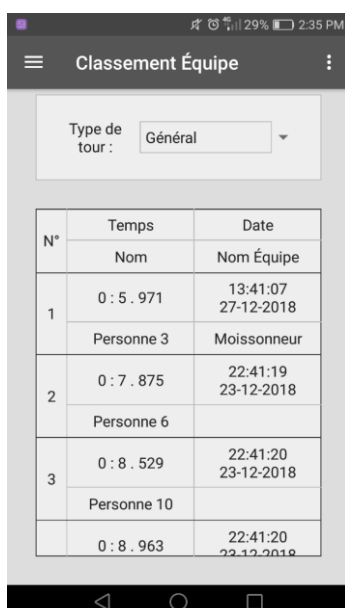
Coureur : Personne 3

Temps moyen : 0 : 1 . 129

1	0 : 0 . 984	13:41:07 27-12-2018
2	0 : 1 . 274	13:41:03 27-12-2018

RankingTeamActivity

Cette activité affiche les temps qu'une équipe réalise (tous les coureurs confondus). Il est possible de voir les temps pour les différentes catégories : sprint, fractionné, pit-stop et général. Il est aussi possible de voir classement des temps moyens de chaque équipes selon les 4 catégories précédentes.



Classement Équipe

Type de tour : Général

N°	Temps	Date
1	0 : 5 . 971	13:41:07 27-12-2018
	Personne 3	Moissonneur
2	0 : 7 . 875	22:41:19 23-12-2018
	Personne 6	
3	0 : 8 . 529	22:41:20 23-12-2018
	Personne 10	
	0 : 8 . 963	22:41:20 23-12-2018

VI. Partie 6

Dans cette application de type NDK, nous créons 6 boutons : « READ », « WRITE », « UP », « LEFT », « DOWN » et « RIGHT ». Nous y ajoutons un simple TextView. Lorsque nous cliquons sur un bouton, la fonction native associée au bouton retourne la valeur à afficher sur le TextView.

Pour les boutons « READ » et « WRITE », les fonctions natives « read » et « write » sont appelée avec en paramètre un entier aléatoire entre 0 et 10. La fonction « read » retourne le carré du nombre passé en paramètre et la fonction « write » renvoie le cube de l'entier fourni en paramètre. Le texte du TextView est alors remplacé par « READ : » suivi du retour de la fonction native « read » pour le bouton « READ » et « WRITE : » suivi du retour de la fonction elle aussi native « write » pour le bouton « WRITE »

Les 4 derniers boutons, la même fonction native est appelée. Il s'agit de la fonction « translate » qui prend en paramètre une chaîne de caractères. Lors de l'appui sur les 4 boutons, ceux-ci appellent la fonction native « translate » en donnant en paramètre le texte affiché sur le bouton. La fonction réalise alors un cas parmi et retourne la direction correspondante en Allemand. La chaîne de caractère retournée est alors affichée dans le TextView.