Pour le projet :

- vendorsetup : avec les 3 possibilité, mettre le nom de notre projet

- cleanSpecl 🡪 garder le même

- Boardconfig 🡪

- androidproducts 🡪 pointe vers le makefile

- arrow.mk 🡪 hériter de la pandaboard, garder la meme ligne (≠ arrow)

Dans device :

**Dans external => plein d’exemples d’Android.mk :**

LO52\_projet/Sources/linaro-ics\_tp/external/tcdump/Android.mk (LOCAL\_C\_INCLUDES => définit où se trouvent les .h, local\_module\_tags => mettre optional)

LO52\_projet/Sources/linaro-ics\_tp/external/openssl/Android .mk => exemple d’android.mk chapeau

source/ android\_toolchain => sera celle qu’on utilisera

**Commande à taper dans « Pied\_Piper » avant de faire « make » :**

time make TARGET\_NO\_HARDWAREGFX=1 KERNEL\_CONFIG=android\_omap4plus\_defconfig TARGET\_PRODUCT=Pied\_piper TARGET\_SIMULATOR=false TARGET\_TOOLS\_PREFIX=/home/lo52tp/LO52\_projet/Sources/android-toolchain-eabi/bin/arm-linux-androideabi- boottarball systemtarball userdatatarball

5 Partitions de « out » : recovery, system, cache, userdata, boot

à faire :

fond d’écran par défaut 🡪 overlay

boot animation 🡪 product copy file

- lier la boot animation au device dans ficher .mk,

- pour le fond d’écran il y en a 3 pour 3 formats d’écran ≠ : les remplacer avec la même définition.

La libusb se trouve dans /home/lo52tp/lo52\_Projet/Sources/ressources\_projet

./configure

make → compiler

• écrire ensuite l’android.mk

• faire la JNI

Avancement du projet

**Sommaire**

1 Commandes 3

2 Informations 4

3 Pour commencer 4

4 Boot animation & Wallpaper 4

4.1 Modifier la boot animation & le wallpaper 4

4.2 Mise à jour du makefile du produit 4

Principe des makefiles : 4

Makefile du produit 4

5 Libusb 5

5.1 Création de la libusb 5

5.2 Création du Makefile 5

6 Compilation du projet 5

7 Composant mlbin => executable 6

7.1 Utilité 6

7.2 Android.mk 6

7.3 mlbin.exe 7

8 mlbin => JNI 7

9 Test du projet sur la Pandaboard 7

# Commandes

**ls** : liste des fichiers

**ls -la** : liste des fichiers avec les infos

**mkdir** : crée un répertoire

**rmdir** : supprime un répertoire vide

**rm nom\_fichier**: supprime le fichier

**touch** : crée un fichiers

**cd** : bouger dans les fichiers

**pwd**: voir le répertoire où on se trouve

tar xfvj nom\_fichier\_compressé.bz2 : décompresser le fichier

./configuration : créer le makefile quand on a tout plein de trucs

**git init** : à taper dans le répertoire que l’on veut transformer en répertoire git

**git add nom\_du\_fichier** : ajoute le fichier au projet git

**git status** : montre la liste des fichiers du répertoire git qui n’ont pas encore été commit

**git commit -m « description de la modification »**

**git show** : montre tous les projets du répertoire git, leurs informations [git show début de l’identifiant du projet]

**mv <ancien nom> <new nom>**: renommer un fichier

**nano <nom\_fichier>**: modifier le contenu d’un fichier (Q pour sortir)

# Informations

**github**:

id : [margaux.barillot@gmail.com](mailto:margaux.barillot@gmail.com)

password : 6…...1…

# Pour commencer

* Création d’un dossier portant le nom du produit « Pied\_Piper » dans /home/lo52tp/LO52\_projet/Sources/linaro-ics-tp/device/utbm/
* Création d’un git repository dans « Pied\_Piper »

# Boot animation & Wallpaper

## Modifier la boot animation & le wallpaper

* Copie du dossier contenant bootanimation.zip dans « Pied\_Piper »
* Pied\_Piper/Media/bootanimation.zip)
* Modification de la bootanimation → trouver une bootanimation en ligne et la remplacer
* Copie du dossier contenant les wallpapers dans « Pied\_Piper » => 1 wallpaper par résolution d’écran
* Pied\_piper/overlay/frameworks/base/core/res/res/
* drawable-large-nodpi/default\_wallpaper.jpg
* drawable-nodpi/ default\_wallpaper.jpg
* drawable-xlarge-nodepi/ default\_wallpaper.jpg
* Remplacement des fonds d’écran avec ceux qu’on veut en respectant la résolution. **/!\ garder le nom default\_wallpaper sinon l’overlay ne reconnaît pas que ça doit remplacer le wallpaper par défaut.**

## Mise à jour du makefile du produit

### Principe des makefiles :

Android.mk → makefile du composant, 1 par composant

Pied\_piper.mk → makefile du produit

### Makefile du produit

Pour lier la bootanimation et les fonds d’écran au device il faut mettre à jour Pied\_Piper/Pied\_Piper.mk :

$(call inherit-product, device/linaro/pandaboard/pandaboard.mk)  
  
PRODUCT\_COPY\_FILES +=     device/utbm/Pied\_piper/media/bootanimation.zip:system/media/bootanimation.zip  
DEVICE\_PACKAGE\_OVERLAYS := device/utbm/Pied\_piper/overlay  
  
PRODUCT\_NAME:= Pied\_piper  
PRODUCT\_DEVICE:= Pied\_piper  
PRODUCT\_BRAND:= Android  
PRODUCT\_MODEL:= Android

# Libusb

## Création de la libusb

* Copie de la Libusb depuis /home/lo52tp/lo52\_Projet/Sources/ressources\_projet vers « Pied\_Piper ».
* Pied\_Piper/libusbx-1.0.9
  + Dézipper le fichier,
  + Dans libusbx-1.0.9/libusb et libusbx-1.0.9/libusb /os ne garder que les .c et .h.

## Création du Makefile

Créer le fichier Pied\_Piper/libusbx-1.0.9/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 os/openbsd\_usb.c  
LOCAL\_MODULE := libusb-lib   
LOCAL\_C\_INCLUDES +=     $(LOCAL\_PATH)/include\  
            $(LOCAL\_PATH)/os  
LOCAL\_MODULE\_TAGS := optional  
include $(BUILD\_SHARED\_LIBRARY)

# Compilation du projet

Vérifier que toutes les dépendances sont présentes, et ensuite configurer et écrire un fichier Makefile qui contiendra les ordres de compilation :

* Dans « linaro\_ics\_tp» faire :

> ./configure

* Avant compilation :

> mkdir –p out/target/product/Pied\_piper/obj/lib

> mkdir –p out/target/product/Pied\_piper/system/bin

> cp out/target/product/arrow/obj/lib/crt

> cp out/target/product/arrow/obj/lib/crt\* out /target/product/Pied\_piper/obj/lib

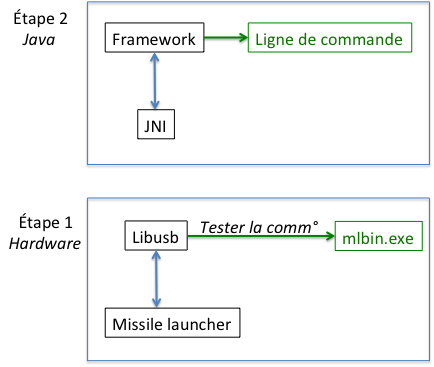
Compiler tout le projet avec nos paramètres :

* Dans « linaro\_ics\_tp» faire :

> time make TARGET\_NO\_HARDWAREGFX=1 KERNEL\_CONFIG=android\_omap4plus\_defconfig TARGET\_PRODUCT=Pied\_piper TARGET\_SIMULATOR=false TARGET\_TOOLS\_PREFIX=/home/lo52tp/LO52\_projet/Sources/android-toolchain-eabi/bin/arm-linux-androideabi- boottarball systemtarball userdatatarball

# Composant mlbin => exécutable

## Utilité



Créer un dossier mlbin dans « Pied\_Piper » (mlbin → missile laucher binary »)

Prendre le fichier mlbin.c envoyé par mail et le mettre dans Pied\_piper/mlbin

Créer le Android.mk.

## Android.mk

Contenu de l’android.mk :

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

include $(CLEAR\_VARS)

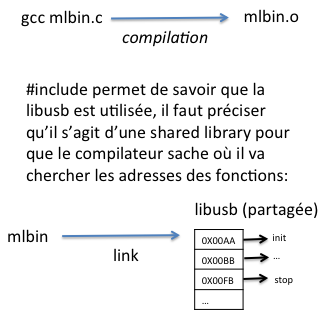
LOCAL\_SHARED\_LIBRARIES := libusb-lib

LOCAL\_SRC\_FILES := mlbin.c

LOCAL\_MODULE := mlbin

LOCAL\_C\_INCLUDES += $(LOCAL\_PATH)/../libusbx-1.0.9/libusb/include\ $(LOCAL\_PATH)/../libusbx-1.0.9/libusb/os

LOCAL\_MODULE\_TAGS := optional include $(BUILD\_EXECUTABLE)



=> Présence de LOCAL\_SHARED\_LIBRARIES := libusb-lib

## mlbin.exe

# mlbin => JNI

Créer mlbin\_jni.c et écrire les prototypes fonctions de mlbin.c sous la forme JNI EXPORT (cf. session 9), sauf pour le ‘main()’ :

mlb\_init\_usb JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbInitUsb(JNIEnv\*env, Jobject this)

mlbin\_free\_usb JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbFreeUsb(JNIEnv\*env, Jobject this)

mlbin\_fire JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinFire(JNIEnv\*env, Jobject this)

mlbin\_move\_down JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinMoveDown(JNIEnv\*env, Jobject this)

mlbin\_move\_left JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinMoveLeft(JNIEnv\*env, Jobject this)

mlbin\_move\_right JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinMoveRight(JNIEnv\*env, Jobject this)

mlbin\_move\_up JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinMoveUp(JNIEnv\*env, Jobject this)

mlbin\_stop JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbStop(JNIEnv\*env, Jobject this)

mlbin\_count\_devices JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinCountDevices(JNIEnv\*env, Jobject this)

À la suite de « mlbin\_jni.c » :

* Copier le code de mlbin.c à la suite des prototypes de fonctions
* Remplacer tous les noms de fonctions par les prototypes écrits au dessus, à partir du mot ‘’JNIEXPORT ‘’. Ex :

JNIEXPORT jint JNICALL Java\_com\_android\_PiedPiper\_InitUsb\_mlbinFire(JNIEnv\*env, Jobject this)
{

unsigned char data[] = {0x5f, ML\_ACTION\_FIRE, 0xe0, 0xff, 0xfe};

libusb\_control\_transfer(devh, 0x21, 0x09, 0, 0, data, 5, 300);

printf("Fire!\n");

return 0;

}

* Supprimer la fonction ‘main()’
* Supprimer tous les prototypes écrits en haut du fichier.
* Ajouter #include "jni.h" pour que lors de la compilation la structure ‘’JNIEXPORT’’ des prototypes soit identifiée.

Modifier l’android.mk pour compiler mlbin\_jni.c comme une JNI. Pour cela, faire comme si on ajoutait un composant, à la suite de android.mk  ajouter:

include $(CLEAR\_VARS)
LOCAL\_SHARED\_LIBRARIES := libusb-lib\

libutils\

libcutils

LOCAL\_SRC\_FILES := mlbin\_jni.c

LOCAL\_MODULE := libmlbin\_jni

LOCAL\_C\_INCLUDES += $(LOCAL\_PATH)/../libusbx-1.0.9/libusb/include\

$(LOCAL\_PATH)/../libusbx-1.0.9/libusb/os\

$(JNI\_H\_INCLUDE)

LOCAL\_MODULE\_TAGS := optional

include $(BUILD\_SHARED\_LIBRARY)

## Extension de framework

(à partir du TD9)

Optionnel : créer un dossier Java et tout mettre ce qui concerne le framework dedans. On fera un ‘gros’ android.mk pour tout.

# Test du projet sur la Pandaboard

Pour qu’il prenne en compte le fond d’écran :

Dans linarro-ics-tp/out/host/linx-x86/bin

> ./adb devices

> ./adb shell

> # rm /data/system/wallpaper\_info.xml

# reboot

Extension de Framework

Faire l’application