CHICHERY Delphine LA1
DUCOULOMBIER Martin

CR LPE RPI

Introduction

Ce document présente une analyse détaillée d'un script Bash destiné à configurer et monter une carte SD pour un JoyPi, un kit éducatif basé sur Raspberry Pi. Le script automatise la création de partitions, le formatage, et l'installation d'un environnement minimaliste. Ce compte rendu est structuré pour fournir une compréhension complète du script, de ses étapes, et de ses implications techniques.

Plan

Analyse du contenu

Présentation générale du script / réponses aux questions

Le script Bash est conçu pour automatiser la configuration et le montage d'une carte SD pour un JoyPi. Il vérifie les permissions, crée des partitions, les formate, et installe un environnement de base en utilisant BusyBox.

Partie 1 : Prise en main

Quel est le système de fichiers utilisé par les deux partitions créées ?
Les deux partitions créées utilisent le système de fichiers FAT32 pour la première partition (boot) et EXT4 pour la seconde partition (système de fichiers racine).

Quelle est l'architecture matérielle du Raspberry Pi ?
L'architecture matérielle du Raspberry Pi est une ARMv6.

Comment fonctionne le chargeur de démarrage ?
Lorsque le Raspberry Pi démarre, le processeur commence par exécuter le code de démarrage stocké dans une mémoire ROM interne, qui est bootcode. Bootcode est responsable de l'initialisation de certains composants matériels et de la configuration initiale du système, c'est son bootloader.

Une des tâches importantes de bootcode est de localiser et de charger le firmware du GPU (Graphical Processing Unit) sur la carte SD ou sur un périphérique de stockage externe. Ce firmware du GPU contient les informations nécessaires pour initialiser et contrôler les différents composants matériels du Raspberry Pi, y compris le framebuffer pour l'affichage graphique.

Après avoir chargé le firmware du GPU, bootcode charge ensuite le noyau Linux à partir de la partition de démarrage de la carte SD. Une fois que le noyau est chargé en mémoire, le contrôle est transféré au noyau Linux, qui prend la main pour initialiser le reste du système.

Est-il possible de proposer un choix de plusieurs distributions au démarrage ?
Oui, il est possible de proposer un choix de plusieurs distributions au démarrage en modifiant les fichiers de configuration.

Quelle est la différence entre les fichiers start.elf, start_cd.elf et start_x.elf ?
Les fichiers start.elf, start_cd.elf et start_x.elf sont des fichiers de démarrage qui permettent de charger le noyau et les modules de démarrage. start.elf est le fichier standard, start_cd.elf est une version allégée pour les configurations avec moins de mémoire, et start_x.elf inclut des codecs supplémentaires.

Partie 2 : Compilation du compilateur

Quelle est la différence entre les affichages obtenus entre le compilateur de l'hôte et le compilateur croisé lorsqu'on utilise l'option "-v" ?

Les affichages obtenus entre le compilateur de l'hôte et le compilateur croisé lorsqu'on utilise l'option "-v" diffèrent au niveau de la target. Le compilateur croisé affiche comme target “--target=arm-linux-gnueabihf”, alors que gcc par exemple affiche “--target=x86_64-linux-gnu”.

Comment connaître les chemins des répertoires où chercher les fichiers d'en-têtes et les bibliothèques standards pour la cible ?

On peut connaître les chemins des répertoires où chercher les fichiers d'en-têtes et les bibliothèques standards pour la cible en utilisant l'option "-v" du compilateur croisé. Il est également possible d’ajouter certain chemin personnalisé à l'aide des options au moment de la compilation

Partie 3 : Noyau et démarrage

Comment recompiler le noyau du Raspberry Pi ?

On peut recompiler le noyau du Raspberry Pi en récupérant les sources du noyau de la distribution officielle, en reconfigurant le noyau pour ne pas produire de pilotes sous forme de modules, en supprimant quelques pilotes inutiles et en donnant un nom à notre version.

Comment booter le Raspberry Pi avec le nouveau noyau ?

On peut booter le Raspberry Pi avec le nouveau noyau en reformattant la carte SD et en utilisant une partition racine en ext2.

Étapes principales du script

1) Vérification des permissions root

```
if [ "$EUID" -ne 0 ]; then
    echo "Please run as root"
    exit
fi
```

Cette étape vérifie si le script est exécuté avec les droits superutilisateur (root). Si ce n'est pas le cas, le script affiche un message et s'arrête. Cela est crucial pour éviter des problèmes de permissions lors de l'exécution des commandes suivantes.

2) Chargement des variables d'environnement

```
source config.source
envPath=$(pwd)
```

Le script charge les variables d'environnement depuis un fichier de configuration (config.source) et initialise une variable envPath avec le chemin courant. Cela permet de centraliser la configuration et de faciliter les modifications ultérieures.

3) Demande du nom du disque et vérification de son existence

Le script liste les disques disponibles (lsblk), demande à l'utilisateur de spécifier le disque à utiliser, et vérifie si le disque existe. Si l'utilisateur ne spécifie pas de disque, sda est utilisé par défaut. Ensuite, il vérifie l'existence du disque.

```
lsblk
echo -e "\nEnter the name of the disk you want to use (default: sda) > "
read disk
if [ "$disk" = "" ]; then
    disk="sda"
fi

if [ ! -e /dev/$disk ]; then
    echo "The disk you entered doesn't exist. Exiting..."
    exit 1
fi

echo "The disk you want to use is $disk. Are you sure? (Y/n) > "
read answer

if [[ "$answer" =~ [nN] ]]; then
    echo "Exiting..."
    exit 1
fi

echo "Starting creation of files system image on /dev/$disk."
```

L'utilisateur doit confirmer son choix de disque. Si la réponse commence par "n" ou "N", le script s'arrête.

4) Création et montage des partitions

Le script crée deux partitions sur le disque spécifié : une pour le boot (/mnt/rpi_boot) et une pour le système de fichiers principal (/mnt/rpi_root). Les partitions sont ensuite

formatées et montées. Cette étape est essentielle pour préparer le disque à recevoir les fichiers système et les données utilisateur.

Avant de reconfigurer les partitions, il est nécessaire de démonter les partitions existantes pour éviter tout conflit :

```
umount /dev/${disk}1
umount /dev/${disk}2
```

Ces commandes démontent les partitions /dev/sda1 et /dev/sda2 de la carte SD.

```
# Create folders rpi_boot and rpi_root in /mnt
mkdir -p /mnt/rpi_boot
mkdir -p /mnt/rpi_root

# Unmount the SD card
umount /dev/${disk}1
umount /dev/${disk}2

# Format the key and create partitions
reset="o\n"
createPart1="n\np\n\n\n+100M\n"
createPart2="n\np\n\n\n+200M\n"
changeType1="t\n1\nc\n"
bootable1="a\n1\n"
write="w\n"

echo -e $reset$createPart1$createPart2$changeType1$bootable1$write |
fdisk /dev/${disk}

# Format partitions
/sbin/mkfs.vfat -n rpi_boot /dev/${disk}1
/sbin/mkfs.ext4 -L rpi_root /dev/${disk}2

# Mount partitions
mount /dev/${disk}1 /mnt/rpi_boot
mount /dev/${disk}2 /mnt/rpi_root
```

Nous créons les répertoires nécessaires pour monter les partitions de la carte SD. L'option -p de mkdir permet de créer tous les sous répertoire et de ne pas afficher d'erreur si les répertoires existent déjà.

Ensuite, les commandes automatisent la création et la configuration des partitions via fdisk. Les étapes sont les suivantes :

- Réinitialisation de la table de partitions (o)
- Création de la première partition (n, p, +100M)
- Création de la deuxième partition (n, p, +200M)
- Modification du type de la première partition en FAT32 (t, 1, c)
- Marquage de la première partition comme amorçable (a, 1)
- Écriture des changements et sortie (w)

Une fois les partitions créées, on copie les fichier de boot dans le dossier /mnt/rpi_boot

5) Installation de BusyBox

BusyBox est un logiciel qui fournit des versions minimales de nombreux utilitaires Unix courants dans un seul exécutable. Il est souvent utilisé dans les systèmes embarqués en raison de sa faible empreinte mémoire. Cette étape configure et installe BusyBox dans l'environnement cible pour offrir une suite d'outils de base nécessaires à l'exécution des scripts et commandes Unix.

```
# Create necessary directories in rpi_root
mkdir -p
rootRPI/{bin,dev,etc,home,lib,mnt,proc,root,sbin,sys,tmp,usr,var}

# Install busybox
cd srcRPI

if [ ! -d busybox ]; then
    echo "Downloading ..."
    tar -xvf busybox.tar.bz2
    mv busybox-1.36.1 busybox

fi
cd busybox
cp ../config/busybox.config .config
make -j$(nproc)
echo "installing busybox..."
make CROSS_COMPILE=$PATH_CC/arm-linux-gnueabihf-
CONFIG_PREFIX=$envPath/rootRPI install
cd $envPath
```

6) Configuration du boot

Le script configure le processus de démarrage en créant les fichiers nécessaires (inittab, rcS) et en configurant le clavier avec une disposition azerty. Cette étape assure que le

système démarre correctement et que l'utilisateur dispose d'un environnement de terminal configuré selon ses préférences linguistiques.

```
# Configure boot
echo "Configuring boot..."
mkdir -p rootRPI/etc/init.d/
touch rootRPI/etc/inittab
touch rootRPI/etc/init.d/rcS

# copy inittab from config
cp srcRPI/config/inittab rootRPI/etc/inittab

# Edit rcS
cp srcRPI/config/rcS rootRPI/etc/init.d/rcS
chmod +x rootRPI/etc/init.d/rcS

# Create azerty.kmap
if [ ! -f srcRPI/config/azerty.kmap ]; then
    curl -o srcRPI/config/azerty.kmap
http://www.blaess.fr/christophe/files/glmf/rpi-scratch-02/azerty.kmap
fi
cp srcRPI/config/azerty.kmap rootRPI/etc/
```

7) Copie des bibliothèques

Les bibliothèques nécessaires sont copiées dans le répertoire rootRPI/lib pour garantir que tous les liens et dépendances soient satisfaits sur la carte SD. Cela permet aux programmes compilés pour l'architecture ARM de s'exécuter correctement sur le Raspberry Pi.

```
# Copy libraries
echo "Copying libraries..."
cp -r $PATH_CC/../arm-linux-gnueabihf/libc/lib/arm-linux-gnueabihf/*
rootRPI/lib
```

8) Création d'un nouvel utilisateur

Le script crée un nouvel utilisateur avec les informations fournies par l'utilisateur (nom d'utilisateur et mot de passe), configure son répertoire personnel et ajuste les fichiers de configuration pour le nouvel utilisateur. Cela permet de personnaliser l'environnement utilisateur et de sécuriser l'accès au système.

```
# # Create a new user
# Demande des informations de l'utilisateur
echo -e "\e[36m"
read -p "Username: " username
read -p "Password: " password
echo -e "\e[0m"

# # Création des utilisateurs

# Création de root sans mot de passe
echo "root:x:0:0:root:/root:/bin/sh" >> rootRPI/etc/passwd
echo "root::18657:0:99999:7:::" >> rootRPI/etc/shadow

# Création de l'utilisateur
mkdir -p rootRPI/home/$username
password=$(openssl passwd -1 -salt xyz $password)
echo "$username:x:1003:1000::/home/$username:/bin/sh" >>
rootRPI/etc/passwd
echo "$username:$password:18657:0:99999:7:::" >> rootRPI/etc/shadow

# Ajout de l'utilisateur au groupe users
echo "users:x:1000:" >> /mnt/lemb/etc/group*

# Copie du fichier .profile
cp srcRPI/config/.profile rootRPI/root/.profile
cp srcRPI/config/.profile rootRPI/home/$username/.profile
```

- L'utilisateur est invité à entrer un nom d'utilisateur et un mot de passe.
- Le mot de passe est chiffré avec openssl passwd.
- L'utilisateur et ses informations sont ajoutés aux fichiers passwd et shadow.
- Un répertoire home est créé pour le nouvel utilisateur.

9) Installation de ncurses

Ncurses est une bibliothèque de programmation qui fournit une API pour la gestion des interfaces textuelles dans un terminal. Elle est utilisée par de nombreux programmes pour la gestion de l'affichage textuel. Cette étape configure, compile et installe ncurses pour permettre aux applications textuelles de fonctionner correctement.

```
# # Install ncurses
echo "Downloading and installing ncurses..."
cd srcRPI
if [ ! -d ncurses ]; then
    # Extract ncurses
```

```
    tar -xvf ncurses-stable.tar.gz
    mv ncurses-6.3 ncurses
fi
cd ncurses

# Configure ncurses with specified options
export CC=$CCC
export CXX=$PATH_CC/arm-linux-gnueabihf-c++
./configure --with-shared --prefix=$envPath/rootRPI
--host=x86_64-build_unknown-linux-gnu --target=arm-linux-gnueabihf
--disable-stripping

# Compile ncurses using available processors
make -j$(nproc)

# Install ncurses
make install

# Creation test file
cd srcRPI/ncursesTestFiles
export PREFIX="$envPath/rootRPI"
make -j$(nproc)
make install

cd $envPath
```

export CC=$CCC : Définit la variable d'environnement CC à la valeur de CCC, qui est le compilateur C croisé (par exemple, arm-linux-gnueabihf-gcc).
export CXX=$PATH_CC/arm-linux-gnueabihf-c++ : Définit la variable d'environnement CXX à la valeur du chemin vers le compilateur C++ croisé.

10) Installation de WiringPI

Ce script sert à installer la bibliothèque WiringPi sur notre système. WiringPi est une bibliothèque logicielle utilisée pour accéder aux broches GPIO du Raspberry Pi et contrôler divers périphériques connectés à ces broches, tels que des LED, des boutons, des capteurs, etc.

```
# # Install WiringPi
echo "Installing WiringPi..."

# Download WiringPi
cd srcRPI
if [ ! -d wiringPi ]; then
```

```
    # Extract WiringPi
    tar -xvf wiringPi.tar.gz
    mv wiringPi-36fb7f1 wiringPi
fi

# Install WiringPi
cd wiringPi/wiringPi
make clean
export DESTDIR="$envPath/rootRPI"
export PREFIX=""
export CC=$CCC
make -j$(nproc) V=1
make install

# Install WiringPiDev
cd ../devLib
make clean
export DESTDIR="$envPath/rootRPI"
export PREFIX=""
export CC="$CCC -I$DESTDIR/include"
make -j$(nproc) V=1
make install

# Install GPIO
cd ../gpio
make clean
export DESTDIR="$envPath/rootRPI"
export PREFIX=""
export CC="$CCC"
make -j$(nproc) V=1
make install

# Clean up
unset DESTDIR
unset PREFIX
unset CC

cd $envPath


rm rootRPI/lib/libwiringPi.so
rm rootRPI/lib/libwiringPiDev.so

cd rootRPI/lib
ln -sr libwiringPi.so.2.50 libwiringPi.so
ln -sr libwiringPiDev.so.2.50 libwiringPiDev.so
```

```
cd $envPath
# # Install WiringPi Test
echo "Installing WiringPi Test..."
cd srcRPI/wiringPiTestFiles
export PREFIX="$envPath/rootRPI"
make -j$(nproc)
make install

cd $envPath
```

11) Installation de Frame buffer

- Installation des librairies graphiques

Le script installe trois bibliothèques graphiques essentielles : jpeg, zlib, et libpng. Ces bibliothèques sont nécessaires pour la manipulation et l'affichage des images :

```
# # Install fbv and graphical libraries

# Install jpeg
echo "Installing jpeg..."
cd srcRPI
if [ ! -d jpeg ]; then
    # Extract jpeg
    tar -xvf jpeg.tar.gz
fi
cd jpeg
./configure --prefix=$envPath/rootRPI --host=arm-linux-gnueabihf CC=$CCC
CFALGS="-I$envPath/rootRPI/include" LDFLAGS="-L$envPath/rootRPI/lib"
make -j$(nproc)
make install CC=$CCC CFALGS="-I$envPath/rootRPI/include"
LDFLAGS="-L$envPath/rootRPI/lib"

cd ..

# Install zlib
echo "Installing zlib..."
if [ ! -d zlib ]; then
    # Extract zlib
    tar -xvf zlib.tar.gz
fi
cd zlib
```

```
./configure --prefix=$envPath/rootRPI
make -j$(nproc) CC=$CCC CFALGS="-I$envPath/rootRPI/include"
LDFLAGS="-L$envPath/rootRPI/lib" LDSHARED="${CCC} -shared
-Wl,-soname,libz.so.1,--version-script,zlib.map"
make install CC=$CCC CFALGS="-I$envPath/rootRPI/include"
LDFLAGS="-L$envPath/rootRPI/lib"

cd ..

# Install libpng
echo "Installing libpng..."
if [ ! -d libpng ]; then
    # Extract libpng
    tar -xvf libpng.tar.gz
fi
cd libpng
./configure --prefix=$envPath/rootRPI
--with-zlib-prefix=$envPath/rootRPI
--host=x86_64-build_unknown-linux-gnu --build=arm-linux-gnueabihf
CC=$CCC CFLAGS="-I$envPath/rootRPI/include"
LDFLAGS="-L$envPath/rootRPI/lib"
make -j$(nproc) INCLUDES="-I$envPath/rootRPI/include"
make install INCLUDES="-I$envPath/rootRPI/include"

cd ..
```

- Installation de fbv

Le framebuffer est une abstraction qui permet d'afficher des images directement sur l'écran du Raspberry Pi sans nécessiter de serveur X. L'installation du logiciel fbv (Framebuffer Image Viewer) permet de visualiser des images directement à partir de la ligne de commande.

```
# Install fbv
echo "Installing fbv..."
if [ ! -d fbv ]; then
    # Extract fbv
    tar -xvf fbv.tar.gz
fi
cd fbv
./configure --prefix=$envPath/rootRPI --without-bmp CC=$CCC
--libs="-I$envPath/rootRPI/include/libpng16 -I$envPath/rootRPI/include
-L$envPath/rootRPI/lib -lpng -lz -ljpeg -lm"
make -j$(nproc) CC=$CCC CFLAGS="-I$envPath/rootRPI/include/libpng16
```

```
-I$envPath/rootRPI/include" LDFLAGS="-L$envPath/rootRPI/lib -lpng -lz
-ljpeg -lm"
make install

cd $envPath
```

Vérification et Extraction : Le script commence par vérifier si le répertoire fbv existe. Si ce n'est pas le cas, il extrait l'archive fbv.tar.gz.

Configuration : La commande ./configure configure le build de fbv pour être installé dans le chemin spécifié par --prefix. L'option --without-bmp spécifie que le support pour le format BMP ne sera pas inclus. Cela est fait pour économiser de l'espace et améliorer les performances.

Chemins d'Inclusion et Bibliothèques : Les options --libs et CFLAGS spécifient les chemins d'inclusion et les bibliothèques nécessaires pour la compilation (libpng, zlib, jpeg, m pour mathématiques).
Compilation : make -j$(nproc) compile le programme en utilisant tous les processeurs disponibles, accélérant ainsi le processus de compilation.
Installation : make install installe fbv dans le répertoire de destination spécifié.

- Copie des images

Les images nécessaires pour les tests ou l'utilisation ultérieure sont copiées dans le répertoire home de l'utilisateur créé.

```
# Copy images
cp srcRPI/images/* rootRPI/home/$username/
```

Cette commande copie toutes les images du répertoire srcRPI/images vers le répertoire home de l'utilisateur spécifié. Cela permet d'avoir des ressources disponibles pour les tests ou les démonstrations.

12) Configuration du réseau

La configuration réseau est essentielle pour permettre au Raspberry Pi de se connecter à un réseau et d'obtenir une adresse IP via DHCP.

```
# # Network configuration
echo "Configuring network..."

mkdir -p rootRPI/etc/network
mkdir -p rootRPI/etc/share/udhcpc
```

```
cp srcRPI/config/udhcpc.default.script
rootRPI/etc/share/udhcpc/default.script
chmod +x rootRPI/etc/share/udhcpc/default.script
```

cp copie le script de configuration DHCP par défaut dans le répertoire approprié.
chmod +x rend le script exécutable, assurant ainsi que le Raspberry Pi peut exécuter ce script pour obtenir une adresse IP via DHCP.

13) Configuration du serveur Web

La configuration d'un serveur web de base permet de servir des fichiers HTML sur le réseau, ce qui peut être utile pour des démonstrations ou des interfaces utilisateur simples.

```
echo "Configuring web server..."

mkdir -p rootRPI/var/www/
cp srcRPI/config/index.html rootRPI/var/www/index.html

mkdir -p crée le répertoire
```

mkdir -p crée le répertoire où les fichiers du serveur web seront stockés.
cp copie une page d'accueil HTML par défaut (index.html) dans le répertoire du serveur web. Cela permet de tester le serveur web et de vérifier qu'il fonctionne correctement en accédant à cette page par un navigateur.

14) Copie des fichiers vers la partition racine de la carte SD

```
# Copy from rootRPI to /mnt/rpi_root
cp -r rootRPI/* /mnt/rpi_root
```

Enfin, tous les fichiers et répertoires de rootRPI sont copiés dans la partition racine de la carte SD, complétant ainsi la configuration.

Variables et dépendances

Variables :

- envPath : Chemin de l'environnement de travail.
- reset, createPart1, createPart2, etc. : Chaînes de commandes pour fdisk.

Dépendances :

fdisk : Utilisé pour gérer les partitions.
mkfs.vfat, mkfs.ext4 : Utilisés pour formater les partitions.
make : Utilisé pour la compilation des libraires et exécutables

Analyse technique

Structure du script

Le script est structuré de manière logique, avec des étapes séquentielles et bien définies. Chaque section est dédiée à une tâche spécifique, ce qui facilite la compréhension et la maintenance.

1. **Initialisation** : Vérification des permissions root et chargement des variables d'environnement.

2. **Préparation de la carte SD** : Démontage, partitionnement, formatage et montage.

3. **Création des partitions** : Utilise fdisk pour créer deux partitions, l'une pour le boot et l'autre pour le système de fichiers principal.

4. **Configuration du système** : Crée les répertoires nécessaires, copie les fichiers de configuration, et configure les utilisateurs et les bibliothèques.

5. **Installation de BusyBox, ncurses et WiringPI**: Compile et installe ces outils en utilisant des options spécifiques pour le compilateur croisé.

6. **Configuration du framebuffer** : Cette section configure le framebuffer, permettant ainsi l'affichage graphique sur le Raspberry Pi.

7. **Configuration du réseau** : Le script configure les paramètres réseau, y compris la connexion Wi-Fi si nécessaire, pour assurer la connectivité du Raspberry Pi.

8. **Finalisation :** Copie finale des fichiers et configurations spécifiques.

Analyse des commandes et des options utilisées

1. **Vérification des permissions root** : La vérification de EUID est une bonne pratique pour s'assurer que le script a les permissions nécessaires pour exécuter des commandes privilégiées.

2. **Partitionnement avec fdisk** : Utiliser echo -e pour envoyer des commandes à fdisk est efficace pour automatiser le partitionnement. Les commandes o, n, t, a, et w sont bien utilisées pour créer et configurer les partitions.

3. **Formatage avec mkfs** : Les commandes mkfs.vfat et mkfs.ext4 sont standard pour formater des partitions en FAT32 et ext4 respectivement.

4. **Téléchargement et installation de BusyBox** : L'utilisation de wget et make pour télécharger et compiler BusyBox est appropriée. La vérification de l'existence du répertoire avant le téléchargement évite les redondances.

Compilation croisée

Stratégies :

Cloner le dépôt Git de la fondation Raspberry Pi et compiler à partir des sources.

Compilateurs croisés disponibles :

arm-bcm2708hardfp-linux-gnueabi
arm-bcm2708-linux-gnueabi
arm-rpi-4.9.3-linux-gnueabihf
gcc-linaro-arm-linux-gnueabihf-raspbian
gcc-linaro-arm-linux-gnueabihf-raspbian-x64

Choix du compilateur 64 bits :

```
export envPath=$PWD
export
PATH_CC=$envPath/srcRPI/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gn
ueabihf-raspbian-x64/bin
export CCC=$PATH_CC/arm-linux-gnueabihf-gcc
```

Afficher la version des compilateurs :

```
gcc -v
arm-linux-gnueabihf-gcc -v
```

Tests GPIO

Cet exercice explore l'utilisation des GPIO (General Purpose Input/Output) du Raspberry Pi, ce qui est essentiel pour de nombreux projets liés à l'embarqué et à l'IoT (Internet des objets). En manipulant les GPIO à l'aide de la bibliothèque WiringPi, les

développeurs peuvent contrôler divers périphériques matériels et capteurs, ce qui ouvre un large éventail de possibilités pour la création de projets interactifs.

Makefile :

Le Makefile pour compiler ce programme avec le compilateur croisé (cross-compiler) est le suivant :

```
PATH_CC?=/usr/bin
CCC?=$(PATH_CC)/gcc

PREFIX?=/usr/local

all: wiringPi
wiringPi: wiringPi_test_led_up wiringPi_test_led_wink
wiringPi_test_led_button wiringPi_test_led_toggle

# besoin de localiser les fichiers d'entete
# => option -I
# besoin de localiser les librairies non standards
# => option -L

wiringPi_test_led_up : wiringPi_test_led_up.c
    $(CCC) -L$(PREFIX)/lib -I$(PREFIX)/include $^ -o $@ -lwiringPi

wiringPi_test_led_wink : wiringPi_test_led_wink.c
    $(CCC) -L$(PREFIX)/lib -I$(PREFIX)/include $^ -o $@ -lwiringPi

wiringPi_test_led_button : wiringPi_test_led_button.c
    $(CCC) -L$(PREFIX)/lib -I$(PREFIX)/include $^ -o $@ -lwiringPi

wiringPi_test_led_toggle : wiringPi_test_led_toggle.c
    $(CCC) -L$(PREFIX)/lib -I$(PREFIX)/include $^ -o $@ -lwiringPi


install: wiringPi
    cp wiringPi_test_led_button $(PREFIX)/bin
    cp wiringPi_test_led_toggle $(PREFIX)/bin
    cp wiringPi_test_led_wink $(PREFIX)/bin
    cp wiringPi_test_led_up $(PREFIX)/bin
#   cp menu_ncurses $(PREFIX)/bin

clean:
    rm wiringPi_test_led_wink wiringPi_test_led_button
wiringPi_test_led_toggle
```

Ce Makefile utilise le compilateur croisé arm-linux-gnueabihf-gcc pour compiler le programme en utilisant la bibliothèque WiringPi.

Exercice 1 : wiringPi_test_led_button.c

Ce script en langage C contrôle une LED à l'aide d'un bouton poussoir sur un Raspberry Pi. Il utilise la bibliothèque WiringPi pour interagir avec les GPIO du Pi, permettant ainsi de contrôler les périphériques connectés. Lorsque le bouton est enfoncé, la LED s'allume, et lorsqu'il est relâché, la LED s'éteint. Le programme reste en cours d'exécution en continu, surveillant en permanence l'état du bouton pour mettre à jour l'état de la LED en conséquence. Lorsque l'utilisateur interrompt le programme en appuyant sur Ctrl+C ou qu'un signal d'interruption est reçu, le programme s'arrête proprement et éteint la LED.

```c
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#define LED_GPIO  0  // DEFAULT LED GPIO
#define BUTTON_GPIO 24  // DEFAULT BUTTON GPIO

void sigint_handler(int sig);
void cleanup();


// Light the given led when the givent button is pressed using wiringPi
int main(int argc, char *argv[]) {
      //define the signal handler
      signal(SIGINT, sigint_handler);


      int led_gpio;
      int button_gpio;
      if (argc == 2) {
      led_gpio = atoi(argv[1]);
      button_gpio = BUTTON_GPIO;
      }
      else if (argc == 3) {
      led_gpio = atoi(argv[1]);
      button_gpio = atoi(argv[2]);
      }
      else {
      led_gpio = LED_GPIO;
```

```
        button_gpio = BUTTON_GPIO;
        }
        printf ("LED GPIO: %d\n", led_gpio);
        printf ("BUTTON GPIO: %d\n", button_gpio);
        wiringPiSetup();
        pinMode(led_gpio, OUTPUT);
        pinMode(button_gpio, INPUT);
        while (1) {
        digitalWrite(led_gpio, digitalRead(button_gpio));
        }
        return 0;
}

// turn off the led when the program exits
void cleanup() {
        digitalWrite(LED_GPIO, HIGH);
}

// turn off the led when the program is interrupted
void sigint_handler(int sig) {
        cleanup();
        exit(0);
}
```

Exercice 2 : wiringPi_test_led_toggle.c

Ce script en langage C permet de faire clignoter une LED lorsqu'un bouton poussoir est enfoncé sur un Raspberry Pi. Il utilise la bibliothèque WiringPi pour contrôler les GPIO du Pi. Lorsque le bouton est enfoncé, la LED commence à clignoter à un rythme régulier. Lorsque le bouton est relâché, la LED s'arrête de clignoter. Le programme surveille en permanence l'état du bouton et met à jour l'état de la LED en conséquence. Lorsque l'utilisateur interrompt le programme en appuyant sur Ctrl+C ou qu'un signal d'interruption est reçu, le programme s'arrête proprement et éteint la LED.

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>


#define LED_GPIO  0  // DEFAULT LED GPIO
#define BUTTON_GPIO 24  // DEFAULT BUTTON GPIO
```

```c
void sigint_handler(int sig);
void cleanup();

// Wink the given led when the given button is pressed using wiringPi
int main(int argc, char *argv[]) {
      // Define the signal handler
      signal(SIGINT, sigint_handler);

      int led_gpio;
      int button_gpio;
      if (argc == 2) {
      led_gpio = atoi(argv[1]);
      button_gpio = BUTTON_GPIO;
      }
      else if (argc == 3) {
      led_gpio = atoi(argv[1]);
      button_gpio = atoi(argv[2]);
      }
      else {
      led_gpio = LED_GPIO;
      button_gpio = BUTTON_GPIO;
      }

      printf ("LED GPIO: %d\n", led_gpio);
      printf ("BUTTON GPIO: %d\n", button_gpio);

      int state = 1;
      int button_state = 0;
      int last_button_state = 0;
      wiringPiSetup();
      pinMode(led_gpio, OUTPUT);
      pinMode(button_gpio, INPUT);
      while (1) {
      button_state = digitalRead(button_gpio);
      if (button_state == LOW && last_button_state == HIGH) {
            state = !state;
      }
      digitalWrite(led_gpio, state || LOW);
      delay(100);
      digitalWrite(led_gpio, HIGH);
      delay(100);
      last_button_state = button_state;

      }
      return 0;
}
```

```
// turn off the led when the program exits
void cleanup() {
      digitalWrite(LED_GPIO, HIGH);
}

// turn off the led when the program is interrupted
void sigint_handler(int sig) {
      cleanup();
      exit(0);
}
```

Exercice 3 : wiringPi_test_led_up.c

Ce script en langage C utilise la bibliothèque WiringPi pour contrôler un GPIO et allumer une LED sur un Raspberry Pi. Lorsque le programme est exécuté, il configure le GPIO de la LED et l'allume en continu en boucle. L'utilisateur peut interrompre le programme en appuyant sur Ctrl+C ou en envoyant un signal d'interruption, ce qui éteint proprement la LED avant de quitter le programme. L'utilisateur peut également spécifier en argument en ligne de commande le numéro GPIO de la LED à contrôler. Si aucun argument n'est fourni, le programme utilise un GPIO par défaut prédéfini.

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#define LED_GPIO  0  // DEFAULT LED GPIO

void sigint_handler(int sig);
void cleanup();


// Light the given led when the program is running using wiringPi
int main(int argc, char *argv[]) {
      //define the signal handler
      signal(SIGINT, sigint_handler);

      int led_gpio;
      if (argc == 2) {
      led_gpio = atoi(argv[1]);
      }
      else {
```

```
        led_gpio = LED_GPIO;
        }
        printf("LED GPIO: %d\n", led_gpio);
        wiringPiSetup();
        pinMode(led_gpio, OUTPUT);
        while (1) {
        digitalWrite(led_gpio, LOW);
        delay(100);
        }
        return 0;
}

// turn off the led when the program exits
void cleanup() {
        digitalWrite(LED_GPIO, HIGH);
}

// turn off the led when the program is interrupted
void sigint_handler(int sig) {
        cleanup();
        exit(0);
}
```

Exercice 4 : wiringPi_test_led_wink.c

Ce script en langage C utilise la bibliothèque WiringPi pour contrôler un GPIO et allumer une LED sur un Raspberry Pi. Lorsque le programme est exécuté, il configure le GPIO de la LED et commence à clignoter la LED à intervalles réguliers. L'utilisateur peut interrompre le programme en appuyant sur Ctrl+C ou en envoyant un signal d'interruption, ce qui éteint proprement la LED avant de quitter le programme. L'utilisateur peut également spécifier en argument en ligne de commande le numéro GPIO de la LED à contrôler. Si aucun argument n'est fourni, le programme utilise un GPIO par défaut prédéfini.

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#define LED_GPIO  0  // DEFAULT LED GPIO

void sigint_handler(int sig);
void cleanup();
```

```c
// Light the given led when the program is running using wiringPi
int main(int argc, char *argv[]) {
      //define the signal handler
      signal(SIGINT, sigint_handler);

      int led_gpio;
      if (argc == 2) {
      led_gpio = atoi(argv[1]);
      }
      else {
      led_gpio = LED_GPIO;
      }
      printf("LED GPIO: %d\n", led_gpio);
      wiringPiSetup();
      pinMode(led_gpio, OUTPUT);
      while (1) {
      digitalWrite(led_gpio, LOW);
      delay(100);
      digitalWrite(led_gpio, HIGH);
      delay(100);
      }
      return 0;
}

// turn off the led when the program exits
void cleanup() {
      digitalWrite(LED_GPIO, HIGH);
}

// turn off the led when the program is interrupted
void sigint_handler(int sig) {
      cleanup();
      exit(0);
}
```

Critique et approfondissement

Public cible

Le script est destiné aux utilisateurs du JoyPi, en particulier aux éducateurs, étudiants, et développeurs impliqués dans des projets éducatifs ou de développement basés sur Raspberry Pi. Un minimum de connaissances en administration Linux et en script Bash est nécessaire pour comprendre et utiliser ce script efficacement.

Points forts

- **Automatisation complète** : Le script couvre toutes les étapes nécessaires, de la préparation de la carte SD à l'installation des composants logiciels, réduisant ainsi le temps et l'effort manuel.
- **Clarté et structure** : Les étapes sont bien définies et logiquement organisées, facilitant la compréhension et la maintenance du script.
- **Utilisation de BusyBox** : L'intégration de BusyBox fournit un environnement minimaliste mais fonctionnel, adapté aux systèmes embarqués comme le JoyPi.

Sujet d'approfondissement

- **Installation et configuration de ncurses et SDL**

SDL (Simple DirectMedia Layer) est une bibliothèque utilisée pour la gestion des médias, tels que les graphiques, les sons et les entrées utilisateur.

Installation de SDL :

```
# Installation de SDL
echo "Installing SDL..."
cd srcRPI
if [ ! -d SDL ]; then
    # Extraire SDL
    tar -xvf SDL.tar.gz
fi
cd SDL
./configure --prefix=$envPath/rootRPI --host=arm-linux-gnueabihf CC=$CCC
CFLAGS="-I$envPath/rootRPI/include" LDFLAGS="-L$envPath/rootRPI/lib"
make -j$(nproc)
make install
cd ..
```

Exemple de programme avec SDL :

```
#include <SDL2/SDL.h>

int main() {
    SDL_Init(SDL_INIT_VIDEO); // Initialiser SDL
```

```
    SDL_Window* window = SDL_CreateWindow("Hello, SDL!",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 640, 480,
SDL_WINDOW_SHOWN);
    SDL_Renderer* renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);
    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255); // Couleur rouge
    SDL_RenderClear(renderer); // Effacer l'écran
    SDL_RenderPresent(renderer); // Afficher l'écran
    SDL_Delay(3000); // Attendre 3 secondes
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

Pour compiler ce programme :

```
arm-linux-gnueabihf-gcc -o hello_sdl hello_sdl.c -lSDL2
```

Dans notre projet, nous avons choisi d'intégrer SDL (Simple DirectMedia Layer) en plus de ncurses pour combiner les avantages des interfaces textuelles simples et des fonctionnalités graphiques avancées. Voici les principaux intérêts de cette approche :

- Complémentarité des Bibliothèques

ncurses : Idéale pour des interfaces en mode texte légères et rapides, parfaite pour les applications nécessitant peu de ressources.

SDL : Offre des capacités graphiques, sonores, et de gestion des périphériques, permettant des interfaces riches et interactives.

- Richesse de l'Interface Utilisateur

SDL permet d'ajouter des éléments graphiques et multimédias (images, vidéos, animations) à une application textuelle ncurses, enrichissant ainsi l'expérience utilisateur.

CONCLUSION

En conclusion, ce compte rendu a couvert divers aspects de la préparation et de la configuration d'une carte SD pour une utilisation avec un Raspberry Pi, ainsi que

l'installation et la configuration de différentes bibliothèques et outils logiciels essentiels pour le développement de projets sur Raspberry Pi.

La première partie a traité de la préparation de la carte SD, comprenant le démarrage, le partitionnement, le formatage et le montage de la carte SD. Ces étapes sont cruciales pour garantir un fonctionnement efficace et fiable du système sur le Raspberry Pi.

Ensuite, nous avons abordé l'installation de diverses bibliothèques et outils logiciels, notamment WiringPi, ncurses et BusyBox. WiringPi est utilisé pour accéder aux broches GPIO du Raspberry Pi, tandis que ncurses permet la création d'interfaces utilisateur graphiques et BusyBox fournit des commandes système de base. De plus, nous avons exploré l'utilisation de fonctionnalités plus avancées telles que le framebuffer pour la gestion d'affichage graphique, la configuration du réseau WiFi pour la connectivité sans fil, et l'intégration de SDL pour le développement d'applications graphiques plus complexes. Enfin, nous avons présenté des exemples pratiques d'utilisation des fonctionnalités du Raspberry Pi, tels que le clignotement d'une LED et la gestion d'un bouton, à l'aide de scripts Shell et de programmes C.

En résumé, ce compte rendu offre une vue d'ensemble complète des étapes nécessaires pour préparer, configurer et commencer à développer des projets sur Raspberry Pi. En suivant les instructions fournies dans ce compte rendu, les utilisateurs pourront démarrer rapidement avec leur Raspberry Pi et commencer à explorer ses nombreuses possibilités en matière de développement de projets.