RPi: PiCast

Table of Contents

Raspberry est un ordinateur monocarte, compact, puissant et peu coûteux, disposant d'un processeur Cortex-A7 quadricœur fonctionnant à 900 MHz et de 1 Go de RAM, d'une sortie HDMI, etc.

Principalement utilisé avec Java, C et les applications de traitement, qui fonctionnent à bas niveau et sont capables de contrôler les ressources matérielles, mais pour exécuter une application Web, on dépend d'un système d'exploitation qui limite ces ressources. Pour obtenir quelque chose de plus performant et tirer parti de certaines autres ressources, telles que le GPU, il faut créer une distribution personnalisée.

Choix de la distribution Linux

Habituellement, on installe **Raspbian** et on utilise simplement les applications fournies avec le système d'exploitation, sans trop de tracas, mais sans les animations et les performances:

- La mémoire est partagée entre le système et le GPU. On a 1 Go, mais il est partagé entre ces deux composants.
- Les applications ne peuvent pas utiliser nativement l'accélération matérielle du GPU pour traiter les animations.

Un hack connu consiste à forcer l'utilisation du GPU, mais pour y parvenir, il faut créer une distribution

Linux propre et abandonner tout ce qui n'est pas essentiel pour garder la RAM disponible pour l'application.

Construction avec Buildroot

buildroot est un outil simple, efficace et facile à utiliser qui génère des systèmes Linux embarqués par compilation croisée.

Préparation de buildroot

Avant de commencer, il faut s'assurer que que tous les packages sont à jour en exécutant les deux commandes suivantes.

```
sudo apt update
sudo apt full-upgrade
```

Ensuite, appliquer la configuration de base pour RPI Model 2 :

```
make rpi2_qt5webkit_defconfig
```

Sélection des packages

Ou pour accéder au menu Buildroot et voir les autres bibliothèques disponibles, utiliser :

```
make menuconfig
```

Choisir toutes les paquets nécessaires:

- ffmpeg (Audio and video applications)
- mpv (Audio and video applications)
- git (Development tools)
- fbv (Graphic libraries and applications (graphic/text) → Graphic libraries)
- tk (Graphic libraries and applications → Graphic libraries)
- rpi-userland (Hardware handling)
- libatomic (Hardware handling → Firmware)
- libwebsockets (Libraries → Networking)
- fdk-aac (Libraries→Audio/Sound)
- libcec (Libraries → Hardware handling)
- libplatform (Libraries → Other)
- protobuf (Libraries → Other)
- unclutter-xfixes (Libraries → Graphics)
- xdotool (Graphic libraries and applications → X applications)
- qt5 (Graphic libraries and applications → Other GUIs)
- sdl2 (Libraries→ Graphic libraries)

- python (Interpreter languages and scripting)
- python3 (Interpreter languages and scripting)
- python-certifi (Interpreter languages and scripting → External python modules)
- python-pillow (Interpreter languages and scripting → External python modules)
- python-requests (Interpreter languages and scripting → External python modules)
- python-six (Interpreter languages and scripting → External python modules)
- python-urllib3 (Interpreter languages and scripting → External python modules)
- python-websocket-client (Interpreter languages and scripting → External python modules)
- python-jinja2 (Interpreter languages and scripting → External python modules)
- python-jellyfin-apiclient (Custom packages)
- python-mpv-jsonipc (Custom packages)
- python-mpv (Custom packages)
- python-proxy-tools (Custom packages)
- python-pywebview (Custom packages)

Pour facilité la sélection des packages dans Buildroot, on peut ajouter un package virtuel dont les fonctionnalités sont fournies par un ou plusieurs packages, appelés « fournisseurs »:

```
mkdir /data/buildroot/package/jellyfin-mpvcast
cat > /data/buildroot/package/jellyfin-mpvcast/Config.in<< 'EOF'
config BR2_PACKAGE_LXQT_DESKTOP
bool "jellyfin-mpvcast"
        select BR2_PACKAGE_GIT
        select BR2_PACKAGE_FBV
        select BR2_PACKAGE_TK
        select BR2_PACKAGE_RPI_USERLAND
        select BR2_PACKAGE_LIBATOMIC
        select BR2_PACKAGE_LIBWEBSOCKETS
        select BR2_PACKAGE_FDK_AAC
        select BR2_PACKAGE_LIBCEC
        select BR2_PACKAGE_LIBPLATFORM
        select BR2_PACKAGE_PROTOBUF
        select BR2_PACKAGE_QT5
        select BR2_PACKAGE_SDL2
        select BR2_PACKAGE_PYTHON
        select BR2_PACKAGE_PYTHON3
        select BR2_PACKAGE_PYTHON_CERTIFI
        select BR2_PACKAGE_PYTHON_PILLOW
        select BR2_PACKAGE_PYTHON_REQUESTS
        select BR2_PACKAGE_PYTHON_SIX
        select BR2_PACKAGE_PYTHON_URLLIB3
        select BR2_PACKAGE_PYTHON_WEBSOCKET_CLIENT
        select BR2_PACKAGE_PYTHON_JINJA2
        select BR2_PACKAGE_PYTHON_JELLYFIN_APICLIENT
        select BR2_PACKAGE_PYTHON_MPV_JSONIPC
        select BR2_PACKAGE_PYTHON_MPV
        select BR2_PACKAGE_PYTHON_PROXY_TOOLS
        select BR2_PACKAGE_PYTHON_PYWEBVIEW
```

```
        select BR2_PACKAGE_FFMPEG
        select BR2_PACKAGE_MPV
        SELECT BR2_PACKAGE_JELLYFIN_MEDIA_PLAYER
        SELECT BR2_PACKAGE_JELLYFIN_MPV_SHIM
        SELECT BR2_PACKAGE_UNCLUTTER_XFIXES
        SELECT BR2_PACKAGE_XDOTOOL
        help
          Jellyfin MPV Cast device
EOF

sed -i '/menu "Custom packages"/a source "package/lxqt-
desktop/Config.in"' /data/buildroot/package/Config.in
```

Pour sélectionner tous les packages il suffira de sélectionner le package **Jellyfin-mpvcast** dans **Custom packages**.

Construction du système

Sauvegarder la configuration et exécuter :

```
make
```

Le processus prend un certain temps.

Installation sur la carte SD

On a maintenant une image prête à être graver sur la carte SD. Lorsque la carte est prête, on peut démarrer le RPI et se connecter au système via SSH (Login : root, Mot de passe : root).

```
ssh root@192.168.1.100 # Remplacer par l'adresse IP de RPI !
```

La distribution qui en résulte est vraiment propre. On n'y trouvera pas d'éléments Debian classiques, comme apt-get. Certaines fonctionnalités dont on peut avoir besoin doivent être installées manuellement.

Compilation à partir des sources

Exécuter la commande suivante pour installer tous les packages requis.

```
sudo apt install -y autoconf make automake build-essential gperf yasm
gnutls-dev texi2html pkg-config libv4l-dev checkinstall libtool libtool-bin
libxslt1-dev libre2-9 libminizip1 libharfbuzz-dev libfreetype6-dev
```

```
libfontconfig1-dev libx11-dev libcec-dev libxrandr-dev libvdpau-dev libva-dev mesa-common-dev libegl1-mesa-dev libasound2-dev libpulse-dev libbluray-dev libdvdread-dev libcdio-paranoia-dev libsmbclient-dev libcdio-cdda-dev libjpeg-dev libluajit-5.1-dev libuchardet-dev zlib1g-dev libfribidi-dev git libgnutls28-dev libgl1-mesa-dev libgles2-mesa-dev libsdl2-dev libmp3lame-dev libass-dev libgpac-dev libx264-dev cmake python3 python3-pip help2man mercurial python git mpv libmpv-dev wget g++ qtwebengine5-dev qtquickcontrols2-5-dev libqt5x11extras5-dev libcec-dev qml-module-qtquick-controls qml-module-qtwebengine qml-module-qtwebchannel qtbase5-private-dev curl unzip device-tree-compiler libssl-dev bison flex
```

Il y a un grand nombre de packages à installer, ce processus peut donc prendre un certain temps.

Une fois le processus d'installation terminé, On peut utiliser le gestionnaire de packages Python 3 appelé « pip » pour installer une version plus récente de **Meson** et **Ninja** que celle disponible via le référentiel de packages.

```
sudo pip3 install meson ninja
```

Compilation de FFMPEG

Créer un répertoire pour contenir les sources

```
mkdir ~/ffmpeg_sources
```

Télécharger et compiler **libfdk-aac** (décodeur et encodeur aac):

```
cd ~/ffmpeg_sources
wget -O fdk-aac.zip https://github.com/mstorsjo/fdk-aac/zipball/master
unzip fdk-aac.zip
cd mstorsjo-fdk-aac*
autoreconf -fiv
./configure --prefix="$HOME/ffmpeg_build" --disable-shared
sudo make
sudo make install
```

Télécharger et compiler **ffmpeg**:

```
cd ~/ffmpeg_sources
wget http://ffmpeg.org/releases/ffmpeg-snapshot.tar.bz2
tar xjvf ffmpeg-snapshot.tar.bz2
cd ffmpeg
PKG_CONFIG_PATH="$HOME/ffmpeg_build/lib/pkgconfig"
export PKG_CONFIG_PATH
./configure --prefix="$HOME/ffmpeg_build" --extra-cflags="-I$HOME/ffmpeg_build/include" \
```

```
   --extra-ldflags="-L$HOME/ffmpeg_build/lib" --bindir="$HOME/bin" --extra-
libs="-ldl" --enable-gpl \
   --enable-libass --enable-libfdk-aac --enable-libmp3lame --disable-
ffserver --disable-ffplay\
   --enable-libx264 --enable-nonfree
sudo make
sudo make install
```

À ce stade, on doit trouver ffmpeg et ffprobe dans ~/bin.

Compilation de mpv accéléré matériellement

Pour utiliser la lecture vidéo accélérée par le matériel de mpv sur le Raspberry Pi il faut construire deux packages basés sur les packages officiels, modifiés

- **ffmpeg** avec `--enable-mmal` pour la prise en charge de

l'accélération matérielle MMAL

- **mpv** avec `--enable-rpi` pour la prise en charge de Raspberry Pi

Il faut d'abord cloner la branche principale **MPV** sur le Raspberry Pi en exécutant la commande suivante.

```
clone git https://github.com/mpv-player/mpv-build.git
```

Ensuite, il faut passer au répertoire nouvellement cloné.

```
cd mpv-build
```

Avant de compiler **mpv**, il faut apporter quelques modifications à ses options de configuration.

On peut le faire en ajoutant deux lignes au fichier `mpv_options`.

Exécuter les deux commandes suivantes pour ajuster les options de compilation.

```
echo --enable-libmpv-shared > mpv_options
echo --disable-cplayer >> mpv_options
echo --extra-libs="-latomic" >> ffmpeg_options
echo --enable-rpi >> mpv_options
echo --enable-mmal >> ffmpeg_options
```

Bien que **ffmpeg** mette plus de temps à construire, la prise en charge de MMAL est requise pour l'accélération matérielle dans mpv.

La première commande ajoute une option lui indiquant de compiler les bibliothèques **MPV** partagées.

La deuxième commande désactive l'interface de ligne de commande pour **MPV** car on n'en a pas

besoin pour Plex Media Player.

La dernière chose que il faut faire est de dire aux scripts de build d'utiliser les versions finales de **MPV** et **libplacebo**.

On peut y parvenir en exécutant les commandes suivantes.

```
./use-mpv-release
./use-libplacebo-release
```

On peut égalemùent choisir d'utiliser la version finale de **ffmpeg** en exécutant la commande suivante:

```
./use-ffmpeg-release
```

Démarrer le processus de compilation en utilisant la commande suivante dans le terminal.

```
BUILDSYSTEM=waf ./rebuild -j$(nproc)
```

En utilisant **$(nproc)**, on utilisera autant de tâches que l'on peut. Pour le Raspberry Pi 4, cela représentera quatre tâches actives.

Ce processus peut prendre un temps considérable. Il doit cloner et compiler **MPV** et **libplacebo** sur le Pi.

Une fois le processus de compilation terminé, On peut maintenant installer les bibliothèques sur le système d'exploitation du Raspberry Pi.

Pour y parvenir, il suffit d'exécuter la commande ci-dessous.

```
sudo BUILDSYSTEM=waf ./install
```

Enfin, exécuter la commande suivante afin que le système d'exploitation sache qu'il existe de nouvelles bibliothèques auxquelles il doit se lier.

```
sudo ldconfig
```

Construction de jellyfin_media_player

jellyfin media player est un client de bureau utilisant **jellyfin-web** avec lecteur **MPV** intégré. Les médias sont lus dans la même fenêtre en utilisant l'interface **jellyfin-web**. Il prend en charge le flux audio.

Pour télécharger la dernière version stable, récupérer la dernière balise de version sur la page des

dernières versions et ajouter ce qui suit à la commande pull pendant la phase de construction pour JMP `--branch $VERSIONTAG --single-branch`

Example:

```
git clone https://github.com/jellyfin/jellyfin-media-player.git --branch
v1.9.1 --single-branch
```

Compiler et installer **jellyfin-media-player**

```
mkdir ~/jmp; cd ~/jmp
git clone https://github.com/mpv-player/mpv-build.git
cd mpv-build
echo -Dlibmpv=true > mpv_options
echo -Dpipewire=disabled >> mpv_options # hopefully temporary
./rebuild -j4
sudo ./install
sudo ln -s /usr/local/lib/x86_64-linux-gnu/libmpv.so /usr/local/lib/x86_64-
linux-gnu/libmpv.so.1
sudo ln -sf /usr/local/lib/x86_64-linux-gnu/libmpv.so
/usr/local/lib/libmpv.so.2
sudo ldconfig
cd ~/jmp/
git clone https://github.com/jellyfin/jellyfin-media-player.git
cd jellyfin-media-player
./download_webclient.sh
cd build
cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=/usr/local/ ..
make -j4
sudo make install
rm -rf ~/jmp/
```

Pour installer via pip, il faut installer **mpv**.

```
sudo pip3 install --upgrade jellyfin-mpv-shim python-mpv
```

Pour bénéficier des fonctionnalités de l'interface graphique et de la barre d'état système, installer également **pystray** et **tkinter**:

```
sudo pip3 install pystray
sudo apt install python3-tk
```

Pour prendre en charge la mise en miroir de l'affichage, installer les dépendances de mise en miroir :

```
sudo apt install python3-jinja2 python3-webview
# -- OR --
sudo pip3 install jellyfin-mpv-shim[mirror]
```

```
sudo apt install gir1.2-webkit2-4.0
```

Configuration

Fichiers de boot

Placer le contenu suivant dans `/boot/cmdline.txt`. Cela empêchera le journal de démarrage d'apparaître:

```
dwc_otg.fiq_fix_enable=1 sdhci-bcm2708.sync_after_dma=0
dwc_otg.lpm_enable=0 vt.global_cursor_default=0 console=tty3
root=/dev/mmcblk0p2 rootwait loglevel=3 quiet
```

Dans le fichier `/boot/config.txt`, il faut au minimum définir les propriétés utilisées par ce projet :

- **gpu_mem_1024=512** pour définit la mémoire GPU à 512M
- **hdmi_group=1** pour forcer le mode CEA destinés à la télévision
- **hdmi_mode=16**: pour utiliser le Full HD 60p
- **hdmi_force_hotplug=1**: force la sortie HDMI

```
disable_splash=1
disable_overscan=1
boot_delay=0
arm_freq=1000
gpu_freq=500
over_voltage=6
avoid_warnings=1
force_turbo=0
gpu_mem_256=128
gpu_mem_512=256
gpu_mem_1024=512
kernel=zImage
hdmi_group=1
hdmi_mode=16
hdmi_force_hotplug=1
```

Démarrage automatique

Un point intéressant sur l'utilisation du RPI est de construire un système dans lequel l'interaction humaine est rarement nécessaire. Il peut démarrer et se mettre à jour sans aucune interaction. Les scripts peuvent aider dans ce processus

L'exemple suivant utilise git pour mettre à jour un dépôt local:

```
cat >> /etc/init.d/S80init <<EOF
```

```
#!/bin/sh
#
# Start processing
#

wget -q --spider http://google.com
if [ $? -eq 0 ]; then
     cd /home/default/yourdirectory
     git pull --rebase
fi
EOF
```

L'exemple suivant permet de lancer l'application **jellyfin-mpv-shim**:

```
cat >> /etc/init.d/S90app <<EOF
#!/bin/sh

nohup jellyfin-mpv-shim
EOF
```

Afin d'automatiser un maximum sans devoir saisir un mot de passe sudo crée un fichier `/etc/sudoers.d/90-rpi` contenant:

```
pi ALL = NOPASSWD:/usr/bin/fbi
```

Démarrage comme un service

Créer le fichier `/lib/systemd/system/jellyfin-mpv-shim.service`:

```
cat > /lib/systemd/system/jellyfin-mpv-shim.service<<'EOF'
[Unit]
Description=Jellyfin Client
After=network.target

[Service]
Environment=DISPLAY=:0.0
Environment=XAUTHORITY=/home/pi/.Xauthority
User=root
Group=root
ExecStart=/usr/bin/xinit /usr/local/bin/jellyfin-mpv-shim
RestartSec=5
Restart=always

[Install]
WantedBy=multi-user.target
```

```
EOF
```

Démarrer le service

```
sudo systemctl start jellyfin-mpv-shim.service
sudo systemctl status jellyfin-mpv-shim.service
```

Écran de démarrage

On peut implémenter un écran de démarrage en quelques étapes simples :

- Créer une séquence PNG nommée frame*.png et la placer dans un répertoire à l'intérieur de RPI.
- Modifier /etc/init.d/S01logging et inclure le code suivant dans la première ligne :

```
# Splash initiale !
cat /dev/zero 1> /dev/fb0 2>/dev/null
fbv -i -c /home/default/bootanimations/frame*.png --delay 1
```

Configuration en mode kiosk

Le mode Kiosk consiste à dédier l'affichage à l'usage exclusif d'une application. On va configurer le lancement automatique d'un diaporama en utilisant l'affichage framebuffer (le plus léger en ressources et paquets).

Configuration de l'affichage

Vérifier l'existence d'un framebuffer

- soit directement dans sysfilesystem

```
ls /sys/class/graphics/fb0

bits_per_pixel  console  device  name   rotate  subsystem
blank           cursor   mode    pan    state   uevent
bl_curve        dev      modes   power  stride  virtual_size
```

```
cat /sys/class/graphics/fb0/name
bochs-drmdrmfb
```

```
cat /sys/class/graphics/fb0/virtual_size
1024,768
```

```
ls -l /dev/fb*
crw-rw---- 1 root video 29, 0 11 déc.  14:09 /dev/fb0
```

- soit avec **fbset**

```
sudo aptitude install fbset
```

```
fbset -i |egrep "^mode|Name"

mode "1024x768"
    Name        : bochs-drmdrmfb
```

Sur les systèmes 64 bits (ARM64/AARCH64) il est recommandé d'utiliser le pilote open source **vc4-kms-v3d**:

1. Rechercher à quel HDMI le moniteur est connecté (si aucun moniteur ne fonctionne, utiliser **HDMI-A-1**)
2. Dans `/boot/config.txt` définir `dtoverlay=vc4-kms-v3d`
3. Vérifier le paramètre approprié pour l'affichage: `cat /sys/class/drm/card0/card0-HDMI-A-1/modes`
4. Dans `/boot/config.txt` ajuster la mémoire gpu: `gpu_mem=256`
5. Dans `/boot/cmdline.txt` ajouter la ligne `video=HDMI-A-1:1024x768M@75D` (remplacer les valeurs par de résolution souhaitée avec le “D” final)

L'ancien pilote **vc4-fkms-v3d** (driver Broadcom VideoCore 4 présent dans Raspberry Pi) ne sera plus pris en charge à l'avenir. Il s'appuie fortement sur des éléments spécifiques aux ordinateurs Pi et non directement pris en charge par Linux. Le nouveau pilote **vc4-kms-v3d** introduit avec Bullseye est a privilégié, mais comme il est nouveau, il y aura des ajustements nécessaires (en particulier pour les personnes qui dépendaient des anciens pilotes et logiciels fkms).

Lorsque le nouveau pilote fonctionne mal, il faut utiliser: `dtoverlay=vc4-fkms-v3d`

On peut également configurer un kiosk en mode graphique, sans faire appel à un gestionnaire de session, ni à un gestionnaire de fenêtre ou de bureau. Pour cela il faut:
- Installer le serveur graphique (par exemple xorg) `apt install --without-recommends xorg xserver-xorg xserver-xorg-video-dummy xserver-xorg-input-evdev xinit xterm`
- Recopier le fichier **xinitrc**, nouveau nom **.xinitrc** dans le répertoire de l'utilisateur kiosk : `cp /etc/X11/xinit/xinitrc ~/.xinitrc`
- Vérifier que l'affichage se fait correctement: `startx /usr/bin/xterm`
- Fermer la fenêtre du terminal **(exit)**

Configuration du kiosk

On peut utiliser **mpv** comme visionneuse de diaporama, en utilisant cette commande :

```
mpv mpv --hwdec=mmal-copy --no-keepaspect --
playlist="C:\Users\classroom\Pictures\1.2 demo\playlist.m3u" --loop-
playlist=inf --fullscreen
```

On peut adapter la durée pendant laquelle l'image s'affiche en utilisant le paramètre `--image-display-duration=<secondes|inf>` dans la ligne de commande ou dans le fichier de configuration.

Le script kiosk, devrait ressembler quelque peu à cela:

```
cat > ~/kiosk.sh<<'EOF'
#!/bin/bash
xset s noblank
xset s off
xset -dpms

unclutter -idle 0.5 -root &

wget -q -O -
https://raw.githubusercontent.com/dconnolly/chromecast-backgrounds/master/RE
ADME.md | sed -n "s/\!\[\](//;s/)$//p" | xargs -l1 wget -P ~/Images/ &

ls -d ~/Images/* > ~/playlist.m3u &

#wget -q -O -
https://raw.githubusercontent.com/dconnolly/chromecast-backgrounds/master/RE
ADME.md | sed -n "s/\!\[\](//;s/)$//p" > ~/playlist.m3u &
mpv --no-keepaspect --playlist="~/playlist.m3u" --loop-playlist=inf --
fullscreen --image-display-duration=20`
EOF
```

Configuration du service

Après avoir créé ce script, il faut s'assurer que l'utilisateur dispose des privilèges d'exécution pour celui-ci.

Accorder à l'utilisateur propriétaire des privilèges d'exécution du script en exécutant la commande suivante.

```
chmod u+x ~/kiosk.sh
```

Avant de commencer, il faut déterminer quelle est la valeur d'affichage actuelle.

Cette valeur est utilisée par le système d'exploitation pour savoir sur quel écran afficher le kiosque, sans cela, le kiosque ne parviendra pas à se charger ou se chargera sur le mauvais écran.

Exécuter la commande suivante pour imprimer la valeur de la variable système **«$DISPLAY»**. Cette commande doit être exécutée directement sur le Raspberry Pi et non via SSH.

```
echo $DISPLAY
```

Pour que le kiosque Raspberry Pi démarre au démarrage, il faut créer un fichier de service (il faut modifier la ligne « Environment=DISPLAY=: », en remplaçant le « 0 » par la valeur récupérée dans **$DISPLAY**, et remplacer « pi » par le nom de l'utilisateur. Par exemple, avec le nom d'utilisateur « pimylifeup », le chemin « /home/pi/kiosk.sh » deviendrait « /home/pimylifeup/kiosk.sh »):

```
cat > /lib/systemd/system/kiosk.service<<'EOF'
[Unit]
Description=MPV Kiosk
Wants=graphical.target
After=graphical.target

[Service]
Environment=DISPLAY=:0.0
Environment=XAUTHORITY=/home/pi/.Xauthority
Type=simple
ExecStart=/bin/bash /home/pi/kiosk.sh
Restart=on-abort
User=pi
Group=pi

[Install]
WantedBy=graphical.target
EOF
```

Le service démarre le script bash `kiosk.sh` nouvellement créé. Après avoir enregistrer le fichier, exécuter la commande suivante pour activer le nouveau service :

```
sudo systemctl start kiosk.service
```

Redémarrer ou exécuter ce qui suit pour démarrer le service :

```
sudo systemctl start kiosk.service
```