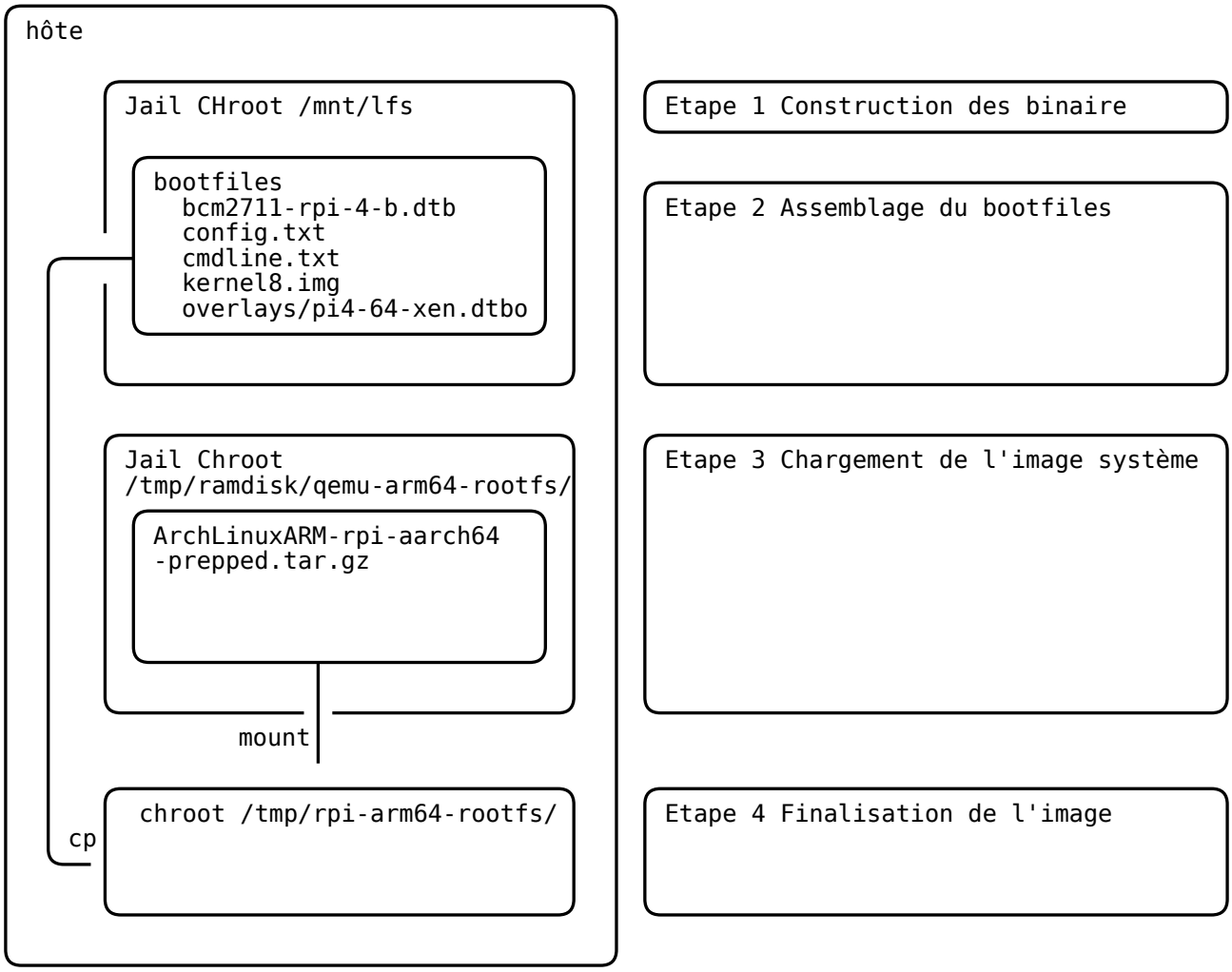
Construction d'une Image XEN archlinux

Table of Contents

Cet article présente la création d'une image Archlinux (ARM32 ou ARM64) avec un noyau xen.

Il s'inspire de https://github.com/dornerworks/xen-rpi4-builder qui propose un ensemble de scripts permettant de produire une image Ubuntu avec XEN pour Raspberry Pi4.

Les étapes de construction de l'image sont résumées dans le cookbook rpi-xen-dom-0 et le schéma suivant:



Il ne faut pas se mélanger dans les chroot, chacun a normalement son propre prompt!

Etape 1: Construction des binaires

Cette section décrit la compilation dans un jail chroot `/mnt/lfs` des binaires:

- bcm2711-rpi-4-b.dtb
- pi4-xx-xen.dtbo
- kernel
- xen
- xen-tools
- modules

Entrer dans un jail chroot de build

Système hôte

```
export LFS=/mnt/lfs

sudo mount -v --bind /dev $LFS/dev

sudo mount -vt devpts devpts $LFS/dev/pts
sudo mount -vt tmpfs shm $LFS/dev/shm
sudo mount -vt proc proc $LFS/proc
sudo mount -vt sysfs sysfs $LFS/sys

sudo mkdir -pv $LFS/build
sudo mount -v --bind ./build $LFS/build

$ sudo chroot $LFS /usr/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /bin/bash --login
```

Jail chroot /tmp/lfs

Préparation de l'environnement de construction

```
export WRKDIR=$(pwd)/
export ARM64_TOOLCHAIN_WRKDIR=$(pwd)/
export ARM64_TOOLCHAIN_SCRIPTDIR=$(cd $(dirname ${BASH_SOURCE}) && pwd)/
```

Installation des paquets requis

```
sudo apt install device-tree-compiler tftpd-hpa flex bison qemu-utils kpartx
git curl qemu-user-static binfmt-support parted bc libncurses5-dev libssl-
dev pkg-config python acpica-tools wget gettext
```

```
sudo apt install ccache
```

Chargement de la Toolchain arm64

```
export ARM64_TOOLCHAIN_VERSION="9.2-2019.12"
export ARM64_TOOLCHAIN_FILENAME=gcc-arm-${ARM64_TOOLCHAIN_VERSION}-x86_64-
aarch64-none-linux-gnu

sudo ln -s ../../bin/ccache /usr/lib/ccache/aarch64-none-linux-gnu-gcc
sudo ln -s ../../bin/ccache /usr/lib/ccache/aarch64-none-linux-gnu-g++

wget
```

```
https://developer.arm.com/-/media/Files/downloads/gnu-a/${ARM64_TOOLCHAIN_VE
RSION}/binrel/${ARM64_TOOLCHAIN_FILENAME}.tar.xz
tar -xf ${ARM64_TOOLCHAIN_FILENAME}.tar.xz
cd ${ARM64_TOOLCHAIN_WRKDIR}
PATH=/usr/lib/ccache:${ARM64_TOOLCHAIN_SCRIPTDIR}${ARM64_TOOLCHAIN_FILENAME}
/bin:$(echo ${PATH} | sed 's|/usr/lib/ccache:||g')
```

Chargement de la Toolchain arm32

```
export ARM32_TOOLCHAIN_VERSION="9.2-2019.12"
export ARM32_TOOLCHAIN_FILENAME=gcc-arm-${ARM32_TOOLCHAIN_VERSION}-x86_64-
arm-none-linux-gnueabihf
export ARM32_TOOLCHAIN_WRKDIR=$(pwd)/
export ARM32_TOOLCHAIN_SCRIPTDIR=$(cd $(dirname ${BASH_SOURCE}) && pwd)/

sudo ln -s ../../bin/ccache /usr/lib/ccache/arm-none-linux-gnueabihf-gcc
sudo ln -s ../../bin/ccache /usr/lib/ccache/arm-none-linux-gnueabihf-g++

cd ${ARM32_TOOLCHAIN_SCRIPTDIR}
wget
https://developer.arm.com/-/media/Files/downloads/gnu-a/${ARM32_TOOLCHAIN_VE
RSION}/binrel/${ARM32_TOOLCHAIN_FILENAME}.tar.xz
tar -xf ${ARM32_TOOLCHAIN_FILENAME}.tar.xz
cd ${ARM32_TOOLCHAIN_WRKDIR}

PATH=/usr/lib/ccache:${ARM32_TOOLCHAIN_SCRIPTDIR}${ARM32_TOOLCHAIN_FILENAME}
/bin:$(echo ${PATH} | sed 's|/usr/lib/ccache:||g')
```

Chargement des firmware

```
export XEN_ADDR=0x00200000
export DTBFILE=bcm2711-rpi-4-b.dtb
if [ "${BUILD_ARCH}" == "arm64" ]; then
    DTBXENO=pi4-64-xen
else
    source ${SCRIPTDIR}toolchain-arm-linux-gnueabihf.sh
    DTBXENO=pi4-32-xen
fi
```

Cloner les sources

```
if [ ! -d firmware ]; then
    mkdir -p firmware/boot
    cd firmware/boot
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/fixup4.da
t
```

```
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/fixup4cd.
dat
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/fixup4db.
dat
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/fixup4x.d
at
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/start4.el
f
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/start4cd.
elf
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/start4db.
elf
    wget
https://raw.githubusercontent.com/raspberrypi/firmware/master/boot/start4x.e
lf
    cd ${WRKDIR}
fi
```

Récupération des sources

Cloner le dépôt xen:

```
if [ ! -d xen ]; then
    git clone --depth=1 --branch RELEASE-4.15.1
git://xenbits.xen.org/xen.git
fi
```

Cloner les sources du noyau (par exemple pour le noyau 5.10.y

```
if [ ! -d linux ]; then
    git clone --depth 1 --branch rpi-5.10.y
https://github.com/raspberrypi/linux.git linux
    cd linux
    git am ${SCRIPTDIR}patches/linux/*.patch
    cd ${WRKDIR}
fi
```

<codedoc toggle “le patch 0001-Add-Xen-overlay-for-the-Pi-4.patch peut être récupéré ici”

>

F

rom: Corey Minyard cminyard@mvista.com Date: Wed, 17 Nov 2021 12:41:32 -0500 Subject: [PATCH] Add Xen overlay for the Pi 4

Signed-off-by: Corey Minyard <cminyard@mvista.com>

arch/arm/boot/dts/overlays/Makefile | 4 +- .../boot/dts/overlays/pi4-32-xen-overlay.dts | 21 ++++++++++ .../boot/dts/overlays/pi4-64-xen-overlay.dts | 39 ++++++++++++++++++++ 3 files changed, 63 insertions(+), 1 deletion(-) create mode 100644 arch/arm/boot/dts/overlays/pi4-32-xen-overlay.dts create mode 100644 arch/arm/boot/dts/overlays/pi4-64-xen-overlay.dts

diff –git a/arch/arm/boot/dts/overlays/Makefile b/arch/arm/boot/dts/overlays/Makefile index a6b0d9ea0385..4cfe5885eca1 100644 — a/arch/arm/boot/dts/overlays/Makefile +++ b/arch/arm/boot/dts/overlays/Makefile @@ -244,7 +244,9 @@ dtbo-$(CONFIG*ARCH*BCM2835) += \ w1-gpio-pullup.dtbo \ w5500.dtbo \ wittypi.dtbo \ - wm8960-soundcard.dtbo + wm8960-soundcard.dtbo \ + pi4-32-xen.dtbo \ + pi4-64-xen.dtbo

targets += dtbs dtbs*install targets += $(dtbo-y) diff –git a/arch/arm/boot/dts/overlays/pi4-32-xen-overlay.dts b/arch/arm/boot/dts/overlays/pi4-32-xen-overlay.dts new file mode 100644 index 000000000000..47afa6ac24b7 — /dev/null +++ b/arch/arm/boot/dts/overlays/pi4-32-xen-overlay.dts @@ -0,0 +1,21 @@ + Xen configuration for Pi 4 +/dts-v1/; +/plugin/; + +/ { + compatible = “brcm,bcm2711”; + + fragment@0 { + target-path = “/chosen”; + overlay { + #address-cells = <0x1>; + #size-cells = <0x1>; + xen,xen-bootargs = “console=dtuart dtuart=/soc/serial@7e215040 syncconsole dom0mem=512M dom0maxvcpus=1 bootscrub=0”; + + dom0 { + compatible = “xen,linux-zimage”, “xen,multiboot-module”; + reg = <0x00400000 0x01800000>; + }; + }; + }; +}; diff –git a/arch/arm/boot/dts/overlays/pi4-64-xen-overlay.dts b/arch/arm/boot/dts/overlays/pi4-64-xen-overlay.dts new file mode 100644 index 000000000000..6c499a831db6 — /dev/null +++ b/arch/arm/boot/dts/overlays/pi4-64-xen-overlay.dts @@ -0,0 +1,39 @@ + Xen configuration for Pi 4 +/dts-v1/; +/plugin/; + +/ { + compatible = “brcm,bcm2711”; + + fragment@0 { + target-path = “/chosen”; + overlay { + #address-cells = <0x1>; + #size-cells = <0x1>; + xen,xen-bootargs = “console=dtuart dtuart=/soc/serial@7e215040 sync*console dom0*mem=512M bootscrub=0”; + + dom0 { + compatible = “xen,linux-zimage”, “xen,multiboot-module”; + reg = <0x00480000 0x01780000>; + }; + }; + }; + + /* Introduce a dummy device node to make Xen map the framebuffer */ + fragment@1 { + target-path = “/”; + overlay { + *fb*bus { + compatible = “simple-bus”; + ranges; + #address-cells = <2>; + #size-cells = <1>; + fb_mem { + compatible = “dummy”; + status = “okay”; + reg = <0x0 0x3b400000 0x04c00000>; + }; + }; + }; + };

+};

2.34.0 </codedoc>

Construction du noyau arm64

En l'absence d'un fichier `${WRKDIR}linux/.build-arm64/.config` on utilise kernel/configs/xen.config fragment

```
cd ${WRKDIR}linux
if [ "${BUILD_ARCH}" == "arm64" ]; then
    if [ ! -s ${WRKDIR}linux/.build-arm64/.config ]; then
        # utilize kernel/configs/xen.config fragment
        make O=.build-arm64 ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu-
bcm2711_defconfig xen.config
    fi
    make O=.build-arm64 ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- -j
$(nproc) broadcom/${DTBFILE}
    make O=.build-arm64 ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- -j
$(nproc) overlays/${DTBXENO}.dtbo
    if [ ! -s ${WRKDIR}linux/.build-arm64/arch/arm64/boot/Image ]; then
        echo "Building kernel. This takes a while. To monitor progress, open
a new terminal and use \"tail -f buildoutput.log\""
        make O=.build-arm64 ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu-
-j $(nproc) > ${WRKDIR}buildoutput.log 2> ${WRKDIR}buildoutput2.log
    fi
elif [ "${BUILD_ARCH}" == "armhf" ]; then
    if [ ! -s ${WRKDIR}linux/.build-arm32/.config ]; then
        # utilize kernel/configs/xen.config fragment
        make O=.build-arm32 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf-
bcm2711_defconfig xen.config
    fi
    make O=.build-arm32 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- -j
$(nproc) ${DTBFILE}
    make O=.build-arm32 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- -j
$(nproc) overlays/${DTBXENO}.dtbo
    if [ ! -s ${WRKDIR}linux/.build-arm32/arch/arm/boot/zImage ]; then
        echo "Building kernel. This takes a while. To monitor progress, open
a new terminal and use \"tail -f buildoutput.log\""
        make O=.build-arm32 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf-
-j $(nproc) zImage modules dtbs > ${WRKDIR}buildoutput.log 2>
${WRKDIR}buildoutput2.log
    fi
fi
cd ${WRKDIR}
```

Construction du noyau Xen

```
if [ ! -s ${WRKDIR}xen/xen/xen ]; then
    cd ${WRKDIR}xen
    if [ ! -s xen/.config ]; then
        cat > xen/arch/arm/configs/rpi4_defconfig << EOF
CONFIG_DEBUG=y
CONFIG_SCHED_ARINC653=y
CONFIG_EARLY_UART_CHOICE_8250=y
CONFIG_EARLY_UART_BASE_ADDRESS=0xfe215040
CONFIG_EARLY_UART_8250_REG_SHIFT=2
EOF
```

```
        make -C xen XEN_TARGET_ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-
gnu- rpi4_defconfig
    fi
    make XEN_TARGET_ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- dist-
xen -j $(nproc)
    cd ${WRKDIR}
fi
```

Construction des outils Xen

Cette section présente la construction et l'activation les outils XEN suivants:

1. **xen-qemu-dom0-disk-backend**: Démarrage de QEMU en tant que backend de disque
2. **xen-init-dom0.service xen-init-dom0**: initialiser la configuration Dom0 (nœuds xenstore, stub de configuration JSON)
3. **xenconsoled**: gère la journalisation à partir des consoles invitées et de l'hyperviseur
4. **xendomains**: démarre et arrête les invités au démarrage et à l'arrêt
5. **xen-watchdog**: exécute le démon de surveillance xen

Retourner dans le répertoire des sources xen:

```
cd ${WRKDIR}xen
```

Déclarer les préfixes utilisés pour la construction

```
if [ "${BUILD_ARCH}" == "arm64" ]; then
    LIB_PREFIX=aarch64-linux-gnu
    CROSS_PREFIX=aarch64-none-linux-gnu
    XEN_ARCH=arm64
elif [ "${BUILD_ARCH}" == "armhf" ]; then
    LIB_PREFIX=arm-linux-gnueabihf
    CROSS_PREFIX=arm-none-linux-gnueabihf
    XEN_ARCH=arm32
fi
```

Modifier les liens symboliques de la bibliothèque partagée en liens relatifs au lieu de liens absolus afin qu'ils fonctionnent bien avec la compilation croisée:

```
sudo chroot ${MNTROOTFS} symlinks -c /usr/lib/${LIB_PREFIX}/
```

Indiquer au compilateur natif dans quel système il inclut les répertoires qu'il recherche.

```
SYSINCDIRS=$(echo bash -c "echo | gcc -E -Wp,-v -o /dev/null - 2>&1" | grep
"^ " | sed "s|^ /| -isystem${MNTROOTFS}|"))
SYSINCDIRSCXX=$(echo bash -c "echo | g++ -x c++ -E -Wp,-v -o /dev/null -
2>&1" | grep "^ " | sed "s|^ /| -isystem${MNTROOTFS}|"))
```

```
CC="${CROSS_PREFIX}-gcc --sysroot=${MNTROOTFS} -nostdinc ${SYSINCDIRS} -
B${MNTROOTFS}lib/${LIB_PREFIX} -B${MNTROOTFS}usr/lib/${LIB_PREFIX}"
CXX="${CROSS_PREFIX}-g++ --sysroot=${MNTROOTFS} -nostdinc ${SYSINCDIRSCXX} -
B${MNTROOTFS}lib/${LIB_PREFIX} -B${MNTROOTFS}usr/lib/${LIB_PREFIX}"
LDFLAGS="-Wl,-rpath-link=${MNTROOTFS}lib/${LIB_PREFIX} -Wl,-rpath-
link=${MNTROOTFS}usr/lib/${LIB_PREFIX}"
```

Configurer la source des build:

```
PKG_CONFIG_LIBDIR=${MNTROOTFS}usr/lib/${LIB_PREFIX}/pkgconfig:${MNTROOTFS}us
r/share/pkgconfig \
PKG_CONFIG_SYSROOT_DIR=${MNTROOTFS} \
LDFLAGS="${LDFLAGS}" \
PYTHON=${MNTROOTFS}/usr/bin/python3 \
./configure \
    --with-system-qemu=/usr/bin/qemu-system-i386 \
    --enable-systemd \
    --disable-xen \
    --enable-tools \
    --disable-docs \
    --disable-stubdom \
    --disable-golang \
    --prefix=/usr \
    --with-xenstored=xenstored \
    --build=x86_64-linux-gnu \
    --host=${CROSS_PREFIX} \
    CC="${CC}" \
    CXX="${CXX}"
```

Construire les binaires dist-tools:

```
PKG_CONFIG_LIBDIR=${MNTROOTFS}usr/lib/${LIB_PREFIX}/pkgconfig:${MNTROOTFS}us
r/share/pkgconfig \
PKG_CONFIG_SYSROOT_DIR=${MNTROOTFS} \
LDFLAGS="${LDFLAGS}" \
make dist-tools \
    CROSS_COMPILE=${CROSS_PREFIX}- XEN_TARGET_ARCH=${XEN_ARCH} \
    CC="${CC}" \
    CXX="${CXX}" \
    -j $(nproc)
```

Construire les binaires install-tools:

```
sudo --preserve-env PATH=${PATH} \
PKG_CONFIG_LIBDIR=${MNTROOTFS}usr/lib/${LIB_PREFIX}/pkgconfig:${MNTROOTFS}us
r/share/pkgconfig \
PKG_CONFIG_SYSROOT_DIR=${MNTROOTFS} \
LDFLAGS="${LDFLAGS}" \
```

```
make install-tools \
    CROSS_COMPILE=${CROSS_PREFIX}- XEN_TARGET_ARCH=${XEN_ARCH} \
    CC="${CC}" \
    CXX="${CXX}" \
    DESTDIR=${MNTROOTFS}
```

Compilation des modules

```
cd ${WRKDIR}linux
if [ "${BUILD_ARCH}" == "arm64" ]; then
    sudo --preserve-env PATH=${PATH} make O=.build-arm64 ARCH=arm64
CROSS_COMPILE=aarch64-none-linux-gnu- INSTALL_MOD_PATH=${MNTROOTFS}
modules_install > ${WRKDIR}modules_install.log
elif [ "${BUILD_ARCH}" == "armhf" ]; then
    sudo --preserve-env PATH=${PATH} make O=.build-arm32 ARCH=arm
CROSS_COMPILE=arm-none-linux-gnueabihf- INSTALL_MOD_PATH=${MNTROOTFS}
modules_install > ${WRKDIR}modules_install.log
fi

cd ${WRKDIR}
```

Etape 2: Assemblage du bootfiles

Cette section décrit l'asssemblage du bootfiles qui contindra:

- bootfiles/config.txt
- bootfiles/cmdline.txt
- bootfiles/kernel8.img
- booutfiles/bcm2711-rpi-4-b.dtb
- bootfiles/overlays/pi4-xx-xen.dtbo (pi4-32-xen.dtbo pour armhf, pi4-64-xen.dtbo pour aarch64)

Création du répertoire boot

Créer le répertoire nommé bootfiles:

```
cd ${WRKDIR}

if [ ! -d bootfiles ]; then
    mkdir bootfiles
fi

cp ${WRKDIR}firmware/boot/fixup4*.dat ${WRKDIR}firmware/boot/start4*.elf
bootfiles/

mkdir -p bootfiles/overlays
```

Charger dedans les binaires dtb et dtbo:

```
if [ "${BUILD_ARCH}" == "arm64" ]; then
    cp ${WRKDIR}linux/.build-arm64/arch/arm64/boot/dts/broadcom/${DTBFILE}
bootfiles/
    cp ${WRKDIR}linux/.build-
arm64/arch/arm64/boot/dts/overlays/${DTBXENO}.dtbo bootfiles/overlays
elif [ "${BUILD_ARCH}" == "armhf" ]; then
    cp ${WRKDIR}linux/.build-arm32/arch/arm/boot/dts/${DTBFILE} bootfiles/
    cp ${WRKDIR}linux/.build-
arm32/arch/arm/boot/dts/overlays/${DTBXENO}.dtbo bootfiles/overlays
fi
```

Créer le fichier `bootfiles/cmdline.txt`:

```
cat > bootfiles/cmdline.txt <<EOF
console=hvc0 clk_ignore_unused root=/dev/mmcblk0p2 rootwait
EOF
```

Créer le fichier `bootfiles/config.txt` (L'image de démarrage doit être nommée kernel8.img pour que le First Stage Bootloader -FSBL- la charge en mode 64 bits et Xen doit être placé à l'adresse 2M, cf. https://www.raspberrypi.org/documentation/configuration/config-txt/boot.md):

```
cat > bootfiles/config.txt <<EOF
kernel=kernel8.img
arm_64bit=1
kernel_address=${XEN_ADDR}
dtoverlay=${DTBXENO}
enable_gic=1

#disable_overscan=1

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

[pi4]
max_framebuffers=2

[all]

enable_jtag_gpio=1
enable_uart=1
init_uart_baud=115200
EOF
```

Chargement de l'image du kernel

Création d'un fichier image vide de 26MiB

```
dd if=/dev/zero of=bootfiles/kernel8.img bs=1024 count=26624
```

Chargement du noyau xen dedans (en supposant que le fichier soit de moins de 2MiB):

```
dd if=${WRKDIR}xen/xen/xen of=bootfiles/kernel8.img bs=1024 conv=notrunc
```

Charger le Kernel Linux dedans (en supposant que la taille du kernel Linux est inférieure à 23,5 Mio l'image est décalée de 2,5 Mio depuis le début du fichier):

```
if [ "${BUILD_ARCH}" == "arm64" ]; then
    dd if=${WRKDIR}linux/.build-arm64/arch/arm64/boot/Image
of=bootfiles/kernel8.img bs=1024 seek=2560 conv=notrunc
elif [ "${BUILD_ARCH}" == "armhf" ]; then
    dd if=${WRKDIR}linux/.build-arm32/arch/arm/boot/zImage
of=bootfiles/kernel8.img bs=1024 seek=2048 conv=notrunc
fi
```

```
if [ -d /media/${USER}/boot/ ]; then
    cp -r bootfiles/* /media/${USER}/boot/
    sync
fi
```

Sortir du jail chroot de build

```
exit
sudo umount $LFS/sys
sudo umount $LFS/proc
sudo umount $LFS/dev/shm
sudo umount $LFS/dev/pts
sudo umount $LFS/dev
sudo umount $LFS/tmp
```

Système hôte

Etape 2: Préparation du système

Cete section ne couvre que le nécessaire pour avoir un système bootable:

1. Le chargement du système de base est opéré directement dans le système hôte
2. La configurtaion est opérée dans un jail chrooté diretctement dans la nouvelle structure de fichier.

Téléchargement du système de base

```
curl -OLf
http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-aarch64-latest.tar.gz
```

Il faut créer un dossier où on va décompresser le système de base. On va utiliser une variable pour garder le chemin:

```
export MNTRAMDISK=/tmp/ramdisk/
export MNTROOTFS=${MNTRAMDISK}qemu-arm64-rootfs/

sudo mkdir -p ${MNTRAMDISK}
sudo mount -t tmpfs -o size=3g tmpfs ${MNTRAMDISK}
```

Chargement du système de fichier

Extraire tous les fichiers de l'archive tar dans l'image disque montée:

```
sudo mkdir -p ${MNTROOTFS}
sudo tar -C ${MNTROOTFS} -xf ArchLinuxARM-rpi-aarch64-latest.tar.gz
```

Entrer dans un jail chroot système

Monter les systèmes de fichiers spéciaux dans l'image:

```
sudo mkdir -p ${MNTROOTFS}
sudo mount -o bind /proc ${MNTROOTFS}proc
sudo mount -o bind /dev ${MNTROOTFS}dev
sudo mount -o bind /dev/pts ${MNTROOTFS}dev/pts
sudo mount -o bind /sys ${MNTROOTFS}sys
sudo mount -o bind /tmp ${MNTROOTFS}tmp
```

La configuration se fait dans un nouveau jail chrooté qui devrait normalement avoir comme prompt :

```
# chroot "${MNTROOTFS}" /bin/bash

(installation) / #
```

Jail Chroot /mnt/ramdisk/qemu-arm64-rootfs/

Configuration du système de fichier

`/etc/resolv.conf` est requis pour la connectivité Internet dans chroot. Il sera écrasé par dhcp, alors il ne faut pas trop s'y attarder:

```
echo "nameserver 8.8.8.8" > /etc/resolv.conf
echo "nameserver 2001:4860:4860::8888" >> /etc/resolv.conf
```

Renseigner les partitions composant le nouveau système dans le `/etc/fstab`, exemple :

```
tmpfs       /tmp    tmpfs    nodev,nosuid    0    0

UUID=[…] swap swap defaults 0 0
UUID=[…] / ext4 defaults 0 1
UUID=[…] /home ext4 defaults 0 1
```

On peut obtenir les UUID en lançant blkid.

Renseigner le nom de la machine dans le fichier `/etc/hostname` ainsi que `/etc/hosts`.

Éditer le fichier `/etc/vconsole.conf` afin d'y spécifier la disposition du clavier que l'on souhaite utiliser.

```
cat > /etc/vconsole.conf << "EOF"
KEYMAP=fr-pc
EOF
```

Créer un lien symbolique `/etc/localtime` afin de choisir le fuseau horaire, par exemple:

```
ln -s /usr/share/zoneinfo/Europe/Paris /etc/localtime
```

Définir un mot passe pour le root :

```
passwd
```

Comme on n'a pas installer depuis un système Arch Linux, il faut configurer la liste des miroirs ainsi que pacman keyring, éditer le fichier `/etc/pacman.d/mirrorlist` et décommenter les serveurs correspondants à la région:

```
# Any
# Server = ftp://mirrors.kernel.org/archlinux/$repo/os/$arch
Server = http://mirrors.kernel.org/archlinux/$repo/os/$arch
```

Puis initialiser pacman keyring :

```
pacman-key --init
```

```
pacman-key --populate archlinux
```

Suite à l'installation, il se peut que l'on rencontre un problème avec pacman et les certificats, pour le régler et avoir un système pleinement opérationnel installer simplement le package ca-certificates-utils

```
pacman -S ca-certificates
pacman -S ca-certificates-utils
```

Mettre à jour `/etc/fstab` pour le périphérique de bloc SD différent par rapport au Raspberry Pi 3 :

```
sed -i 's/mmcblk0/mmcblk1/g' racine/etc/fstab
```

Avec ceci, on a un système que l'on peut utiliser de façon autonome.

Création de l'image prepped

```
sudo tar -czf ArchLinuxARM-rpi-aarch64-prepped.tar.gz -C ${MNTROOTFS} $(ls
${MNTROOTFS})
```

Sortir du jail chroot de build

```
exit
sudo umount ${MNTROOTFS}proc || true
sudo umount ${MNTROOTFS}dev/pts || true
sudo umount ${MNTROOTFS}dev || true
sudo umount ${MNTROOTFS}sys || true
sudo umount ${MNTROOTFS}tmp || true
sudo umount ${MNTBOOT} || true
sudo umount ${MNTROOTFS} || true
```

Système Hôte

Etape 4: Construction de l'image disque du système

Cette section décrit l'assemblage du système de fichier issu des précédentes étapes:

1. le contenu de l'image ArchLinuxARM-rpi-aarch64-prepped.tar.gz
2. le contenu du dossier bootfiles

Initialisation des variables

```

```

```
export MNTRAMDISK=/tmp/ramdisk/
export MNTROOTFS=/tmp/rpi-arm64-rootfs/
export MNTBOOT=${MNTROOTFS}boot/
export IMGFILE=${MNTRAMDISK}rpixen.img
```

Mappage des partitions de prepped

kpartx lit les tables de partitions sur le périphérique spécifié et créé des maps de périphérique sur les segments de partition détectés dans /dev/mapper/:

- **-a**: ajoute des mappages de partition
- **-s**: mode de synchronisation=attend que les partitions soient créées

```
export LOOPDEVS=$(sudo kpartx -avs ArchLinuxARM-rpi-aarch64-prepped.tar.gz |
awk '{print $3}')
export LOOPDEVBOOT=/dev/mapper/$(echo ${LOOPDEVS} | awk '{print $1}')
export LOOPDEVROOTFS=/dev/mapper/$(echo ${LOOPDEVS} | awk '{print $2}')
```

Création d'un chroot spécial

```
sudo mkdir -p ${MNTROOTFS}
if ! mount | grep ${LOOPDEVROOTFS}; then
    sudo mount ${LOOPDEVROOTFS} ${MNTROOTFS}
fi
sudo mkdir -p ${MNTBOOT}
sudo mount ${LOOPDEVBOOT} ${MNTBOOT}
sudo mount -o bind /proc ${MNTROOTFS}proc
sudo mount -o bind /dev ${MNTROOTFS}dev
sudo mount -o bind /dev/pts ${MNTROOTFS}dev/pts
sudo mount -o bind /sys ${MNTROOTFS}sys
sudo mount -o bind /tmp ${MNTROOTFS}tmp

sudo sync

sudo mkdir -p ${MNTRAMDISK}
sudo mount -t tmpfs -o size=3g tmpfs ${MNTRAMDISK}
```

Création de l'image disque

Création de l'image disque dans laquelle sera installé le système de base (/mnt/ramdisk/rpixen.img):

```
qemu-img create ${IMGFILE} 2048M
/sbin/parted ${IMGFILE} --script -- mklabel msdos
MB_TO_SECTOR=2048
/sbin/parted ${IMGFILE} --script -- mkpart primary fat32 $(( 1 *
${MB_TO_SECTOR} ))s $(( 129 * ${MB_TO_SECTOR} - 1 ))s
```

```
/sbin/parted ${IMGFILE} --script -- mkpart primary ext4 $(( 129 *
${MB_TO_SECTOR} ))s -1025s

sudo mkfs.vfat ${LOOPDEVBOOT}
sudo mkfs.ext4 ${LOOPDEVROOTFS}

sudo fatlabel ${LOOPDEVBOOT} boot
sudo e2label ${LOOPDEVROOTFS} RpiUbuntu

sudo mount ${LOOPDEVROOTFS} ${MNTROOTFS}

cd ${WRKDIR}
```

Chargement du boot dans l'image

```
cd ${WRKDIR}

sudo cp -r bootfiles/* ${MNTBOOT}
```

Configuration de l'image

Comme le script de configuration des outils xen sélectionne un trop grand nombre de modules de pilote backend, on le remplace donc par une liste `/usr/lib/modules-load.d/xen.conf` plus réduite:

```
sudo bash -c "cat > ${MNTROOTFS}usr/lib/modules-load.d/xen.conf" <<EOF
xen-evtchn
xen-gntdev
xen-gntalloc
xen-blkback
xen-netback
EOF
```

Configuration de `/etc/hostname`:

```
sudo bash -c "echo ${HOSTNAME} > ${MNTROOTFS}etc/hostname"
```

Configuration de `/etc/hosts`:

```
sudo bash -c "cat > ${MNTROOTFS}etc/hosts" <<EOF
127.0.0.1   localhost
127.0.1.1   ${HOSTNAME}
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
```

```
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
EOF
```

Configuration de `/etc/fstab`:

```
sudo bash -c "cat > ${MNTROOTFS}etc/fstab" <<EOF
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p1  /boot           vfat    defaults          0       2
/dev/mmcblk0p2  /               ext4    defaults,noatime  0       1
EOF
```

configuration de `/etc/network/interfaces.d/eth0`:

```
sudo bash -c "cat > ${MNTROOTFS}etc/network/interfaces.d/eth0" <<EOF
auto eth0
iface eth0 inet manual
EOF

sudo chmod 0644 ${MNTROOTFS}etc/network/interfaces.d/eth0
```

Configuration de `/etc/network/interfaces.d/xenbr0`:

```
sudo bash -c "cat > ${MNTROOTFS}etc/network/interfaces.d/xenbr0" <<EOF
auto xenbr0
iface xenbr0 inet dhcp
    bridge_ports eth0
EOF

sudo chmod 0644 ${MNTROOTFS}etc/network/interfaces.d/xenbr0
```

Pour ne pas attendre une éternité que le réseau se mette en ligne:

```
if [ -s ${MNTROOTFS}lib/systemd/system/networking.service ]; then
    sudo sed -i -e "s/TimeoutStartSec=5min/TimeoutStartSec=15sec/"
${MNTROOTFS}lib/systemd/system/networking.service
fi
if [ -s ${MNTROOTFS}lib/systemd/system/ifup@.service ]; then
    sudo bash -c "echo \"TimeoutStopSec=15s\" >>
${MNTROOTFS}lib/systemd/system/ifup@.service"
fi
```

Configuration des comptes utilisateurs:

```
sudo chroot ${MNTROOTFS} useradd -s /bin/bash -G adm,sudo -l -m -p
```

```
${HASHED_PASSWORD} ${USERNAME}
```

Configuration des sudoers Password-less:

```
sudo chroot ${MNTROOTFS} /bin/bash -euxc "echo \"${USERNAME} ALL=(ALL)
NOPASSWD:ALL\" > /etc/sudoers.d/90-${USERNAME}-user"
```

Activer les démons:

```
sudo chroot ${MNTROOTFS} systemctl enable xen-qemu-dom0-disk-backend.service
sudo chroot ${MNTROOTFS} systemctl enable xen-init-dom0.service
sudo chroot ${MNTROOTFS} systemctl enable xenconsoled.service
sudo chroot ${MNTROOTFS} systemctl enable xendomains.service
sudo chroot ${MNTROOTFS} systemctl enable xen-watchdog.service

cd ${WRKDIR}
```

Démontage du chroot

```
sudo umount ${MNTROOTFS}proc || true
sudo umount ${MNTROOTFS}dev/pts || true
sudo umount ${MNTROOTFS}dev || true
sudo umount ${MNTROOTFS}sys || true
sudo umount ${MNTROOTFS}tmp || true
sudo umount ${MNTBOOT} || true
sudo umount ${MNTROOTFS} || true
```