Initram: LAB 1 Base Image BusyBox

Objet	construire un iniram avec busybox
Niveau requis	débutant, avisé
Débutant, à savoir	
Suivi	:DONE:

BusyBox regroupe des versions minuscules de nombreux utilitaires UNIX courants en un seul petit exécutable, qui remplace la plupart des utilitaires qu'on trouve habituellement dans GNU `fileutils`, `shellutils`, etc. Cependant, les options incluses fournissent les fonctionnalités attendues et se comportent de manière très similaire à leurs homologues GNU: **BusyBox** fournit un environnement assez complet pour tout système de petite taille ou intégré.

BusyBox a été conçu dans un souci d'optimisation de la taille et de ressources limitées, et est extrêmement modulaire, permettant d'inclure ou d'exclure facilement des commandes (ou fonctionnalités) lors de la compilation, ce qui facilite la personnalisation des systèmes intégrés.

Il exite également des utilitaires minuscules dont la fonctionnalité n'est pas fournie par busybox.

- **SSH Dropbear**: à la fois un serveur ssh et un client ssh. Il n'a pas de dépendances externes (par exemple, il ne dépend pas d'OpenSSL, mais utilise une copie intégrée de LibTomCrypt).
- **SMTP ssmtp**: est un agent de transfert de courrier extrêmement simple.
- **ntp ntpclient**: est un minuscule client ntp. BusyBox à partir de 1.16.x a également acquis l’applet ntpd.

Ce document décrit la construction de la base, en utilisant: un noyau Linux, BusyBox, Dropbear et Syslinux pour booter le système. On utilisera une archive initramfs décompressée en RAM par le kernel au démarrage du système. On va créer la box, pour qu'elle tienne dans un système de fichier racine de 3 à 4 Mb, en utilisant 'strip' sur les bibliothèques et les binaires, pour gagner de la place.

Construction de l'image de base BusyBox

Afin de construire l'environnement minimal ne contenant qu'un shell **BusyBox** et accessible via le serveur SSH de **Dropbear**, on devra utiliser une prison chroot. Une prison chroot est un environnement isolé fonctionnant au-dessus d'un autre. Le nom lui-même dit en réalité ce qu'il est, il exécute des processus sous une racine modifiée (un sous-répertoire de la racine d'origine).

Pré requis

```
sudo apt-get install curl
sudo apt-get install libncurses5-dev
sudo apt install qemu-system-x86
```

Télécharger et extraire BusyBox

```
export CLFS="/var/lfs/x86-busybox"
mkdir -pv ${CLFS}
cd /var/lfs/
curl https://busybox.net/downloads/busybox-1.31.0.tar.bz2 | tar xjf -
curl http://matt.ucc.asn.au/dropbear/dropbear-2019.78.tar.bz2 | tar xjf -
```

Construire BusyBox

Entrer dans le répertoire

```
$ cd /var/lfs/busybox-1.31.0
```

Utiliser la configuration par défaut et la modifier en fonction des besoins.

```
$ make defconfig
$ make menuconfig
```

Pour compiler busybox pour une autre architecture (ARM pa exemple) définir les oprions **ARCH** et **CROSS_COMPILE** ainsi :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-  defconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Aller dans `BusyBox Settings –> Build Options` et activer l'option `Build BusyBox as a static binary (no shared libs)`.

Pour modifier le répertoire d'installation Dans le même menu, le menu `Installation Options (make install behavior) –> (./_install) BusyBox installation prefix`

Cette option signifie que le système de fichiers racine sera installé dans le répertoire `_install`. On peut le changer si on le souhaite. Laisser les choses telles qu'elles sont.

Lancer l'installation

```
$ make install
```

Ou pour une architecture ARM

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- install
```

Construire le système de fichiers racine.

Si la compilation se termine correctement, le système de fichiers racine se trouvera dans le répertoire _install.

Construire la structure de répertoire de rootfs

```
mkdir -pv ${CLFS}/{rootfs,_src}
cd ${CLFS}/rootfs
mkdir -pv
{bin,dev,sbin,etc,proc,sys/kernel/debug,usr/{bin,sbin,lib},lib,lib64,mnt/roo
t,root}
cp -av /var/lfs/busybox-1.31.0/\_install/* ${CLFS}/rootfs
sudo cp -av /dev/{null,console,tty,sda1,urandom} ${CLFS}/rootfs/dev/
```

dropbear-0.50 - Client et serveur SSH légé

Dropbear est un petit client et serveur sécurisé, supportant le protocole SSH 2. Dropbear est compatible avec openSSH, et utilise `/var/lfs/.ssh/authorized_keys` pour la gestion des clés public. Dropbear fournit aussi une version de scp, qu'il faut compiler avec 'manke scp':

Installation de dropbear

```
cd ${CLFS}/_src
wget https://matt.ucc.asn.au/dropbear/releases/dropbear-2019.77.tar.bz2
tar xjf dropbear-2019.77.tar.bz2
cd dropbear-2019.77
./configure --prefix=/
make
make scp
make DESTDIR=$PWD/_pkg install
strip -v scp
strip -v _pkg/bin/*
strip -v _pkg/sbin/*
```

Installer le client, et des outils associés dans `/bin`, et le server dans `/sbin`:

```
cp scp ${CLFS}/rootfs/bin
cp -a _pkg//bin/* ${CLFS}/rootfs/bin
cp -a _pkg/sbin/* ${CLFS}/rootfs/sbin
```

Faire un ldd sur drobear pour déterminer la liste des bibliothèques

```
ldd _pkg/sbin/dropbear
```

```
    libutil.so.1 => /lib64/libutil.so.1 (0x00007fce9a035000)
    libz.so.1 => /lib64/libz.so.1 (0x00007fce99e1f000)
    libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007fce99be7000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fce9981a000)
    libfreebl3.so => /lib64/libfreebl3.so (0x00007fce99617000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fce9a487000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007fce99412000)
```

Copier les bibliothèques dans rootfs/lib :

```
cp -avLn
/lib64/{libutil.so.1,libz.so.1,libcrypt.so.1,libc.so.6,libfreebl3.so,ld-
linux-x86-64.so.2,libdl.so.2} ${CLFS}/rootfs/lib64
```

Configuration de Dropbear

Les fichiers de configuration utilisateur sont dans `/var/lfs/.ssh`, contenant `authorized_keys` et `known_hosts`. Les répertoire `/var/lfs/.ssh` et le fichier `known_hosts`, sont créés automatiquement la première fois que l'utilisateur lance dbclient. Les fichiers de configuration système du server Dropbear sont par défaut dans `/etc/dropbear`:

```
mkdir ${CLFS}/rootfs/etc/dropbear
```

Pour que le serveur fonctionnent, il faut générer les clés sécurisées avant de démarrer le serveur.

Passer en chroot dans rootfs (`/dev` doit être monté)

```
cd ${CLFS}/rootfs
/sbin/chroot . /bin/ash
```

```
dropbearkey -t rsa -f /etc/dropbear/dropbear_rsa_host_key
dropbearkey -t dss -f /etc/dropbear/dropbear_dss_host_key
```

On peut démarrer le serveur SSH avec la commande :

```
/etc/init.d/dropbear start
```

Configuration de la Box

Créer les fichiers nécessaires dans `/etc`, il faut commencer par créer quelques fichiers utiles au fonctionnement de base du sytème:

Réseau

Création des fichiers de base utilisés pour configurer le réseau:

```
echo "127.0.0.1      localhost" > etc/hosts
echo "localnet    127.0.0.1" > etc/networks
echo "minix86" > etc/hostname
echo "order hosts,bind" > etc/host.conf
echo "multi on" >> etc/host.conf
```

/etc/nsswitch.conf

Fichier de configuration utilisé pour la résolution des noms:

```
cat >> etc/nsswitch.conf << EOF
# /etc/nsswitch.conf: GNU Name Service Switch config.
#

passwd:     files
group:      files
shadow:     files

hosts:      files dns
networks:   files
EOF
```

/etc/securetty

Le fichier /etc/securetty, liste les terminaux sur lesquels root peut se connecter:

```
cat >> etc/securetty << EOF
# /etc/securetty: List of terminals on which root is allowed to login.
#
console

# For people with serial port consoles
ttyS0

# Standard consoles
tty1
tty2
tty3
tty4
```

```
tty5
tty6
tty7
EOF
```

/etc/shells

Le fichier /etc/shells, liste les shells de connection valides. Ce fichier est entre autre utilisé par le serveur SSH dropbear:

```
cat >> etc/shells << EOF
# /etc/shells: valid login shells.
/bin/sh
/bin/ash
/bin/hush
EOF
```

/etc/issue et /etc/motd

Création des fichiers /etc/issue affiché à la fin du boot, et du message du jour affiché après le login:

```
echo "MiniX86 GNU/Linux 1.0 Kernel \r \l" > etc/issue
echo "" >> etc/issue
cat >> etc/motd << EOF
 (°-  { Get documentation in: /usr/share/doc.
 //\    Use: 'less' or 'more' to read files, 'su' to be root. }
 v_/_

SliTaz is distributed in the hope that it will be useful, but
with ABSOLUTELY NO WARRANTY.
EOF
```

/etc/busybox.conf

Ce fichier est le fichiers de configuration de BusyBox, il permet de configurer, entre autre des droits sur les applications Busybox :

```
cat >> etc/busybox.conf << EOF
# /etc/busybox.conf: SliTaz GNU/linux Busybox configuration.
#

[SUID]
# Allow command to be run by anyone.
```

```
su = ssx root.root
passwd = ssx root.root
loadkmap = ssx root.root
mount = ssx root.root
reboot = ssx root.root
halt = ssx root.root
EOF
```

Pour plus de sécurité, changer les permission sur ce fichier:

```
chmod 600 etc/busybox.conf
```

/etc/inittab

Fichier de configuration d'init minimal. Il permet d'avoir d'avoir une console root sans passer par le login, et une console activable sur tty2.

```
cat >> etc/inittab << EOF
# /etc/inittab: init configuration for SliTaz GNU/Linux.

::sysinit:/etc/init.d/rcS
::respawn:-/bin/sh
tty2::askfirst:-/bin/sh
::ctrlaltdel:/bin/umount -a -r
::ctrlaltdel:/sbin/reboot
EOF
```

/etc/profile

Ce fichier est lu lors de chaque login, et affecte tous les utilisateurs. Il faut utiliser le fichier `/var/lfs/.profile` pour la config propre à chaque user.

```
cat >> etc/profile << EOF
# /etc/profile: system-wide .profile file for the Bourne shells

PATH="/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"
LD_LIBRARY_PATH="/usr/lib:/lib"

if [ "`id -u`" -eq 0 ]; then
  PS1='\e[1m\u@\h:\w\#\e[m '
else
  PS1='\e[1m\u@\h:\w\$\e[m '
fi

DISPLAY=:99.0
```

```
export PATH LD_LIBRARY_PATH PS1 DISPLAY ignoreeof
umask 022
EOF
```

Utilisateurs, groupes, et mots de passes

Création des fichiers de configuration de l'utilisateur, du groupes, du mot de passe root dans: `etc/{passwd,shadow,group,gshadow}`, et ajustage des permissions:

```
echo "root:x:0:0:root:/root:/bin/sh" > etc/passwd
echo "root::13525:0:99999:7:::" > etc/shadow
echo "root:x:0:" > etc/group
echo "root:*::" > etc/gshadow
chmod 640 etc/shadow
chmod 640 etc/gshadow
```

On peut ajouter d'autres utilisateurs. On peut aussi configuer un mot de passe pour le super-utilisateur root, avec la commande passwd. Pour ajouter un utilisateur existant dans un groupe existant, éditer les fichiers `/etc/group` et `/etc/gshadow`, car l'applet `adduser` fourni avec busybox n'offre pas toutes les options fournies par le programme original.

/etc/fstab et /etc/mtab

Liste les systèmes de fchiers à monter:

```
cat >> etc/fstab << EOF
# /etc/fstab: information about static file system.
#
proc            /proc        proc    defaults          0       0
sysfs           /sys         sysfs   defaults          0       0
devpts          /dev/pts     devpts  defaults          0       0
tmpfs           /dev/shm     tmpfs   defaults          0       0
EOF
```

Le fichier `/etc/mtab` est utilisé entre autre par `mkfs*`, il liste les partitions montées. Il a besoin de `/proc`, car c'est un lien sur `/proc/mounts`:

```
ln -s /proc/mounts /etc/mtab
```

Clavier

Créer un fichier `.kmap` spécifique au clavier grâce à la commande `dumpkmap` fournie avec BusyBox. Pour créer un fichier .kmap (changez fr_FR en fonction de la configuration):

```
/sbin/chroot . /bin/ash
```

Dans le chroot créer le fichier .kmap

```
mkdir -pv usr/share/kmap
/bin/busybox dumpkmap > usr/share/kmap/fr_FR.kmap
```

Une fois ceci fait on peut charger automatiquement le clavier avec `loadkmap` dans un script tel que etc/ini.d/rcS, par exemple.

/etc/init.d/rcS

Pour finir il faut créer le script d'initialisation `/etc/init.d/rcS` pour monter les systèmes de fichiers, et lancer quelques commandes.

Il faut changer la valeur de la variable **KMAP=** pour que le bon clavier soit chargé:

```
mkdir etc/init.d
cat >> etc/init.d/rcS << EOF
#! /bin/sh
# /etc/init.d/rcS: rcS initial script.
#

KMAP=fr_FR

echo "Processing /etc/init.d/rcS... "

/bin/mount proc
/bin/mount -a
/bin/hostname -F /etc/hostname
/sbin/ifconfig lo 127.0.0.1 up
/sbin/loadkmap < /usr/share/kmap/$KMAP.kmap
EOF

chmod +x etc/init.d/rcS
```

sortir du chroot

```
exit
```

Installation du script udhcpc

Udhcpc est un client DHCP stable et rapide, fourni avec Busybox, mais ayant un développemnt indépandent. On peut utiliser `default.script` de l'archive BusyBox. Ce script se met dans `/usr/share/udhcpc/default.script`, mais cela peut être modifié en ligne de commande. Sur

SliTaz le client est lancé au boot par le script /etc/init.d/network.sh via le fichier de configuration /etc/network.conf:

```
mkdir -pv usr/share/udhcpc
cp /var/lfs/busybox-1.31.0/examples/udhcp/simple.script \
   usr/share/udhcpc/default.script
chmod +x usr/share/udhcpc/default.script
```

Génération de l'initramfs

L'initramfs est une archive cpio du système générée depuis la racine, elle est décompressée en RAM par le noyau Linux lors du démarrage (boot), pour créer le système de fichiers en mémoire vive. Pour générer une archive initramfs, dans le répertoire racine du système de fichiers (rootfs), on fait une recherche avec find, et on utilise des pipes |. Ensuite on créer une archive cpio gzipée avec gzip, que l'on place dans le répertoire de travail.

L'initramfs se nomme rootfs.gz, c'est le nom du système racine, mais avec l'extension .gz. Si on veut changer le nom, il faudrat le spécifier dans le fichier `isolinux.cfg`.

```
find . -print | cpio -o -H newc | gzip -9 > ../rootfs.gz
```

On doit obtenir un fichier `rootfs.gz` d'environ 1 à 2 Mb dans le répertoire de travail.

Pour une nouvelle image, lors de modif dans rootfs, il suffit de copier la nouvelle archive `rootfs.gz` dans `rootcd/boot`, et de créer une nouvelle image ISO avec **genisoimage**.

Pour vérifier l'initramfs, faire :

```
ls -hl ${CLFS}/rootfs.gz
```

On doit voir quelque chose comme:

```
-rw-r--r--. 1 root root 2.8M Sep 13 09:43 /var/lfs/x86-busybox/rootfs.gz
```

Pour vérifier le contenu, faire:

```
mkdir /checkarchive
cd /checkarchive
zcat ${CLFS}/rootfs.gz | cpio -idmv --no-absolute-filenames
```

Ensuite, pour recréer à partir de ce répertoire, faire:

```
cd /checkarchive
find . | cpio -H newc -o > ../checked-initramfs.cpio
```

```
cd ..
cat checked-initramfs.cpio | gzip > ${CLFS}/checked-rootfs.gz
```

Création du noyau linux

Télécharger et extraire le noyau Linux

```
mkdir -pv ${CLFS}/vmlinuz
cd ${CLFS}/vmlinuz
curl https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.10.6.tar.xz | tar
xJf -
```

Configurer le noyau

Configurer le noyau avec la configuration minimale

```
cd linux-4.10.6
make O=${CLFS}/vmlinuz/linux-x86-allnoconfig allnoconfig
```

Chosir les options de configuration

Les options décrites ci-dessous permettent de construire un NUTSHELL busybox pouvant être utilisé dans Docker.

```
cd /linux-4.10.6
make O=mkdir -pv ${CLFS}/vmlinuz/linux-x86-allnoconfig nconfig
```

`Nconfig` est menu pseudo-graphique basé sur `ncurses`

```
*] 64-bit kernel

-> General setup
  -> Configure standard kernel features
[*] Enable support for printk

-> General setup
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

```
-> Executable file formats / Emulations
[*] Kernel support for ELF binaries
[*] Kernel support for scripts starting with #!

-> Device Drivers
  -> Character devices
[*] Enable TTY

-> Device Drivers
  -> Character devices
    -> Serial drivers
[*] 8250/16550 and compatible serial support
[*]   Console on 8250/16550 and compatible serial port

-> File systems
  -> Pseudo filesystems
[*] /proc file system support
[*] sysfs file system support

-> Kernel hacking
    -> Compile-time checks and compiler options
[*] Debug filesystem

-> Kernel hacking
[*] Early printk
```

`Debug filesystem` n'est pas nécessaire mais l'inclure permet de disposer d'une interface de débogage ce qui est un bon moyen de commencer à travailler avec le noyau. `Early printk` n'est pas non plus nécessaires, mais cela aide à voir le noyau démarrer.

Généralement nécéssaire:

clé	Commentaire
NAMESPACES {NET,PID,IPC,UTS}_NS	
CGROUPS CGROUP_CPUACCT CGROUP_DEVICE CGROUP_FREEZER CGROUP_SCHED CPUSETS MEMCG	Les installations par défaut de docker utilisent toujours cgroupfs et la plupart, pour modifier les valeurs par défaut utiliser `--exec-opt native.cgroupdriver=systemd`
KEYS	
VETH BRIDGE BRIDGE_NETFILTER	
NF_NAT_IPV4 IP_NF_FILTER IP_NF_TARGET_MASQUERADE	
NETFILTER_XT_MATCH_{ADDRTYPE,CONNTRACK,IPVS}	
IP_NF_NAT NF_NAT NF_NAT_NEEDED	
POSIX_MQUEUE	required for bind-mounting /dev/mqueue into containers

clé	Commentaire
DEVPTS_MULTIPLE_INSTANCES	pour les kernel < 4.8
CONFIG_SECURITY_APPARMOR CONFIG_DEFAULT_SECURITY=“apparmor” CONFIG_SECURITY_APPARMOR_BOOTPARAM_VALUE=1	AppArmor et apparmor_parser doivent être installés

Optional Features:

clé	Commentaire
USER_NS bad	dans RHEL7/CentOS si User namespaces disabled 'user_namespace.enable=0 ; add 'user_namespace.enable=1' to boot command line
SECCOMP	
CGROUP_PIDS	
MEMCG_SWAP MEMCG_SWAP_ENABLED	/sys/fs/cgroup/memory/memory.memsw.limit_in_bytes ne doit pas être défini
LEGACY_VSYSCALL_NATIVE	bad dangereux, fournit une cible contournant ASLR avec des gadgets ROP
LEGACY_VSYSCALL_EMULATE	good
LEGACY_VSYSCALL_NONE	bad Les ontainers utilisant eglibc ⇐ 2.13 ne fonctionneront pas. Cela désactivera ASLR pour le VDSO, ce qui peut aider à exploiter des vulnérabilités de sécurité.
MEMCG_KMEM	pour les kernel < 4.5
RESOURCE_COUNTERS	pour les kernel < 3.18
NETPRIO_CGROUP	
CGROUP_NET_PRIO	
BLK_CGROUP BLK_DEV_THROTTLING IOSCHED_CFQ CFQ_GROUP_IOSCHED	
CGROUP_PERF	
CGROUP_HUGETLB	
NET_CLS_CGROUP $netprio	
CFS_BANDWIDTH FAIR_GROUP_SCHED RT_GROUP_SCHED	
IP_NF_TARGET_REDIRECT	
IP_VS	
IP_VS_NFCT	
IP_VS_PROTO_TCP	
IP_VS_PROTO_UDP	
IP_VS_RR	
EXT4_USE_FOR_EXT2	
EXT3_FS EXT3_FS_XATTR EXT3_FS_POSIX_ACL EXT3_FS_SECURITY	pour utiliser EXT3 comme support activer
EXT4_FS EXT4_FS_POSIX_ACL EXT4_FS_SECURITY	pour utiliser EXT3 ou EXT4 comme support

Network Drivers

clé	Commentaire
VXLAN BRIDGE_VLAN_FILTERING	
CRYPTO CRYPTO_AEAD CRYPTO_GCM CRYPTO_SEQIV CRYPTO_GHASH XFRM XFRM_USER XFRM_ALGO INET_ESP INET_XFRM_MODE_TRANSPORT	Optionnel (for encrypted networks)
IPVLAN	ipvlan
MACVLAN DUMMY	Macvlan
NF_NAT_FTP NF_CONNTRACK_FTP NF_NAT_TFTP NF_CONNTRACK_TFTP	ftp,tftp client in container

Storage Drivers

clé	Commentaire
AUFS_FS	certains noyaux incluent des correctifs AUFS mais pas l'indicateur AUFS_FS
BTRFS_FS BTRFS_FS_POSIX_ACL	btrfs
BLK_DEV_DM DM_THIN_PROVISIONING	devicemapper
OVERLAY_FS	overlay

Pour `zfs` il faut s'assurer d' avoir `/dev/zfs` et les ommandes `zfs` et `zpool` opérationnelles.

Contruire le noyau Linux

```
cd /linux-4.10.6
make O=${CLFS}/linux-x86-allnoconfig -j2
```

À la fin du journal, on doit voir:

```
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Setup is 15804 bytes (padded to 15872 bytes).
System is 550 kB
CRC 57d75ca8
Kernel: arch/x86/boot/bzImage is ready  (#1)
make[1]: Leaving directory '/var/lfs/x86-busybox/linux-x86-allnoconfig'
```

Construction de l'image CD

Les étapes suivantes vont permettre de créer la racine du cd-rom bootable.

Création de la structure

Pour commencer, créer les répertoires `rootcd`, `boot`, et `isolinux` pour les fichiers déstinés au cd-rom:

```
cd ${CLFS}/x86-busybox
mkdir -p rootcd/boot/isolinux
```

En option on peut créer d'autres repértoires pour y mettre divers données, tel que des documents html ou des paquets.

Copier le kernel

Il suffit de copier le noyau préalablement compilé, dans rootcd/boot:

```
cp src/linux-2.6.20/arch/i386/boot/bzImage rootcd/boot
```

Copier l'initramfs dans rootcd/boot

Il ne faut pas oublier de générer une nouvelle archive initramfs lors de modification dans le rootfs (root file system).

```
cp rootfs.gz rootcd/boot
```

Installer le bootloader isolinux

Pour installer le bootloader isolinux, il suffit de copier `isolinux.bin` depuis l'archive des source de Syslinux:

```
cd ${CLFS}/x86-busybox/_src
tar xzf syslinux-3.35.tar.gz
cp syslinux-3.35/isolinux.bin ../rootcd/boot/isolinux
cd ${CLFS}
```

On peut aussi utiliser GRUB pour booter la box.

isolinux.cfg - Configuration d'isolinux

Voici un exemple du fichier isolinux.cfg qui devrait bien fonctionner:

```
cat >> rootcd/boot/isolinux/isolinux.cfg << EOF
display display.txt
default minix86
label minix86
    kernel /boot/bzImage
    append initrd=/boot/rootfs.gz rw root=/dev/null vga=788
implicit 0
prompt 1
timeout 80
EOF
```

On peut le modifier à colonté:

- La valeur timeout correspond au nombre de secondes à attendre avantde booter. Vous pouvez la mettre à 0, ou hôter la ligne pour démarrer instantanément, ou choisir un temps d'attente plus long tel que 10 s.
- prompt peut être mis à 0 pour désactiver le 'boot:' prompt.
- On peut ajouter encore ajouter plus de lignes pour afficher le contenu de plusieurs fichiers textes lorsque l'utilisateur appuie sur F1, F2, F3, etc.

display.txt

Un petit message de bienvenue, propulsé par isolinux:

```
cat >> rootcd/boot/isolinux/display.txt << EOF
/*                        .      .       _   _
     )))       .  .:::.        '\\-//`         \|/
    (o o)        :(o o):  .     (o o)         (o o)
ooO--(_)--Ooo-ooO--(_)--Ooo-ooO--(_)--Ooo-ooO--(_)--Ooo-

  __  __          _   _         __  __   ___      __
 |' \/ '|  ___   | \ |"|  ___   \ \/"/  ( " )    /"/_
 | |\/| | |_"_| |  \| | |_"_|   \  /   /   \   | '_ \
 | |  | |  | |  | |\  |  | |    /  \   | ( ) |  | (_) |
 |_|  |_|  | |  |_| \_|  | |   /_/\_\   \___/    \___/
  _// \\_ |___| _// \\_ |___| _// \\_       _// \\_
 (./   \.)     (_/  (_")     (_/   \_)     (__) (__)

 MiNiX86 GNU/Linux - Temporary Autonomous Zone

     <ENTER> to boot.


                                                                  */
EOF
```

Créer un image ISO avec genisoimage

```
genisoimage -R -o minix86-cooking.iso -b boot/isolinux/isolinux.bin \
   -c boot/isolinux/boot.cat -no-emul-boot -boot-load-size 4 \
   -V "MiniX86" -input-charset iso8859-1 -boot-info-table rootcd
```

Pour chaque nouvelle modification dans le système de fichier racine de la box, il faudra créer une nouvelle image ISO.

Tester l'ISO

On peut tester l'image ISO avec le logiciel d'émulation Qemu (Sur Debian # aptitude install qemu). Pour émuler l'image ISO fraîchement crée, il suffit de taper :

```
# qemu -cdrom minix86-cooking.iso
```