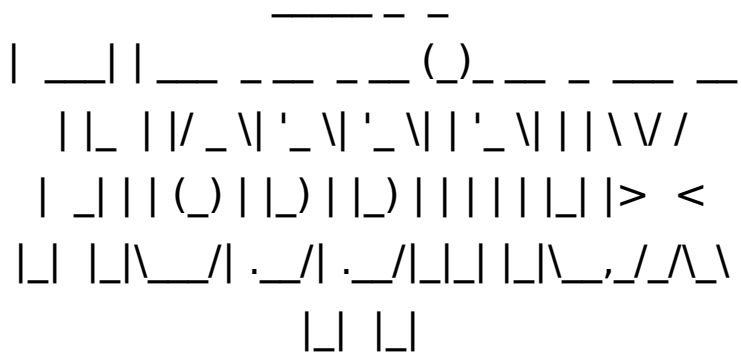
Floppinux - An Embedded Linux on a Single Floppy

MANUAL FOR VERSION 0.1.0



By Krzysztof Krystian Jankowski for the community

Goals for the project

The obvious most important goal is to fit everything (OS+software) on one floppy or 1440KiB
Latest Linux kernel
Tools reduced to those needed to support my embedded application
Documentation with easy and understandable steps to reproduce the build
As always free and open source

Additional future upgrades:

Ability to mount another floppy to save files
Nano text editor (or anything similar)

Let's Build FLOPPINUX Distribution

Working Directory

Create directory where you will keep all the files.

```
mkdir ~/my-linux-distro/
cd ~/my-linux-distro/
```

Kernel

I'm using the latest revision. It's a feat of it's own that connects old and new technologies togheter. At the moment it is Kernel 5.13.0-rc2.

Get the sources:

```
git clone --depth=1 https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
cd linux
```

Now that you have them in /linux/ directory lets configure and build our custom kernel. First create tiniest configuration:

```
make ARCH=x86 tinyconfig
```

Now you need to add additonal config settings on top of it:

```
make ARCH=x86 menuconfig
```

From menus choose those options:

Processor type and features > Processor family > **486**
Device Drivers > Character devices > **Enable TTY**
General Setup > Configure standard kernel features (expert users) > **Enable support for printk**
General Setup > **Initial RAM filesystem and RAM disk (initramfs/initrd)**

Executable file formats > **Kernel support for ELF binaries**
Executable file formats > **Kernel support for scripts starting with #!**

Exit configuration (yes, save settings to .config). Now it's time for compiling!

```
make ARCH=x86 bzImage
```

This will take a while depending on the speed of your CPU. In the end the kernel will be created in *arch/x86/boot/bzImage*. Move it to our main directory.

```
mv arch/x86/boot/bzImage ../
```

Tools

Without tools kernel will just boot and you will not be able to do anything. One of the most popular lightweight tools are BusyBox. Those replaces (bigger) GNU tools with just enough functionality for embedded needs.

Check the latest version at https://busybox.net/downloads/. At the moment it is 1.33.1. Download this file, extract it and change directory:

```
wget https://busybox.net/downloads/busybox-1.33.1.tar.bz2
tar xjvf busybox-1.19.3.tar.bz2
cd busybox-1.33.1/
```

As with kernel you need to create starting configuration:

```
make allnoconfig
```

Now the fun part. You need to choose what tools you want. Each menu entry will show how much more KB will be taken if you choose it. So choose it wisely :)

```
make menuconfig
```

I chosed those:

Settings > **Build static binary (no shared libs)**
Coreutils > **cat, du, echo, ls, sleep, uname** (change Operating system name to anything you want)
Console Utilities > **clear**
Editors > **vi**
Init Utilities > **poweroff, reboot, init, Support reading an inittab file**
Linux System Utilities > **mount, umount**
Miscellaneous Utilities > **less**
Shells > **ash**

Now exit with save config. Compile time.

```
make
make install
```

This will create a filesystem with all the files at _install. Move it to our main directory. I like to rename it also.

```
mv _install ../filesystem
```

Filesystem

You got kernel and basic tools but the system still needs some additional directory structure.

```
cd ../filesystem
mkdir -pv {dev,proc,etc/init.d,sys,tmp}
sudo mknod dev/console c 5 1
sudo mknod dev/null c 1 3
```

Now create few configuration files. First one is a welcome message that will be shown after booting:

```
cat >> welcome << EOF
Some welcome text...
EOF
```

Inittab file that handles starting, exiting and restarting:

```
cat >> etc/inittab << EOF
::sysinit:/etc/init.d/rc
::askfirst:/bin/sh
::restart:/sbin/init
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
EOF
```

And the actual init script:

```
cat >> etc/init.d/rc << EOF
#!/bin/shmount -t proc none /proc
mount -t sysfs none /sys
clear
cat welcome
/bin/sh
EOF
```

Make init executable and owner of all files to root:

```
chmod +x etc/init.d/rc
sudo chown -R root:root .
```

Lastly compress this directory into one file:

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

You can test if everything goes as planned by runing QEMU from the base directory:

```
qemu-system-i386 -kernel bzImage -initrd
rootfs.cpio.gz
```

Next step is to put this on a floppy!

Boot Image

Create this grub file that will point to your newly created kernel and filesystem:

```
cat >> syslinux.cfg << EOF
DEFAULT linux
LABEL linux
 SAY [ BOOTING FLOPPINUX VERSION 0.1.0 ]
 KERNEL bzImage
 APPEND initrd=rootfs.cpio.gz
EOF
```

Create empty floppy image:

```
dd if=/dev/zero of=floppinux.img bs=1k
count=1440
mkdosfs floppinux.img
syslinux --install floppinux.img
```

Mount it and copy syslinux, kernel and filesystem onto it:

```
sudo mount -o loop floppinux.img /mnt
sudo cp bzImage /mnt
sudo cp rootfs.cpio.gz /mnt
sudo cp syslinux.cfg /mnt
sudo umount /mnt
```

Done!

You have your own distribution image floppinux.img ready to burn onto a floppy and boot on real hardware!

If you don't have a floppy you can just test it in the QEMU like a normal person:

```
qemu-system-i386 -fda floppinux.img
```

Summary

Full size: 1440KiB / **1.44MiB**
Kernel size: 632KiB
Tools: 552KiB
Free space left (du -h): **272KiB**

Adding Embedded Application

Now as we have our embedded distribution let's make some use of it. It boots very fast (after floppy loads) and can easily run any compiled application. But I want to have some fun with scripts. So I will put .sh scripts instead of compiled software. The process then is the same.

Update files in the /filesystem/ directory
compress rootfs file
mount distro image
replace rootfs file
umonut image
(optionaly) burn new iso to the floppy
boot to a new system with your updated software

You will also want to change the *etc/init.d/rc* script and change /bin/sh to a script/binary file path.

But for time of debugging it's better to run the app by hand. I depend on scripts so having Vi editor is very handy for testing fixes live.

Resources

https://www.insentricity.com/a.cl/283
https://backreference.org/2010/07/04/modifying-initrdinitramfs-files/
https://www.centennialsoftwaresolutions.com/post/build-the-linux-kernel-and-busybox-and-run-them-on-qemu

http://blog.nasirabed.com/2012/01/minimal-linux-busybox.html
https://bootlin.com/doc/legacy/elfs/embedded_lfs.pdf