FLOPPINUX - An Embedded 🐧 Linux on a Single 💾Floppy

2025 Edition (v0.3.0)

October 19, 2025

> FLOPPINUX was released in 2021. After four years people find it helpful. Because of that I decided to revisit FLOPPINUX in 2025 and make updated tutorial. This brings bunch of updates like latest kernel and persistent storage.

Table of Contents

Main Project Goals

Think of this as Linux From Scratch but for making single floppy distribution.

> It is meant to be a full workshop (tutorial) that you can follow easily and modify it to your needs. It is a learning exercise. Some base Linux knowledge is needed.

The final distribution is very simple and consists only of minimum of tools and hardware support. As a user you will be able to boot any PC with a floppy drive to a Linux terminal, edit files, and create simple scripts. There is 264KB of space left for your newly created files.

Core features:

- Fully working distribution booting from the single floppy
- Latest* Linux kernel
- Supporting all 32-bit x86 CPUs since Intel 486DX
- Have a working text editor (Vi) and basic file manipulation commands (move, rename, delete, etc.)

- Support for simple scripting
- Persistent storage on the floppy to actualy save files (264KB)
- Works on real hardware and emulation

** latest kernel that supports Intel 486 CPUs*

Minimum Hardware Requirements:

- Intel 486DX 33MHz
- 20MB RAM
- Internal floppy disk

Linux Kernel

The Linux kernel drops i486 support in 6.15 (released May 2025), so **6.14** (released March 2025) is the latest version with full compatibility.

64-bit Base OS

This time I will do everything on **Omarchy** Linux. It is 64-bit operating system based on Arch Linux. Instructions should work on all POSIX systems. Only difference is getting needed packages.

Working Directory

Create directory where you will keep all the files.

```
mkdir ~/my-linux-distro/
BASE=~/my-linux-distro/
cd $BASE
```

Host OS Requirements

> You need supporting software to build things. This exact list may vary depending on the system you have.

Install needed software/libs. On Arch/Omarchy 3.1:

```
sudo pacman -S ncurses bc flex bison syslinux cpio
```

Cross-compiler:

```
wget https://musl.cc/i486-linux-musl-cross.tgz
tar xvf i486-linux-musl-cross.tgz
```

Emulation

> 86Box is also good but slower. Bochs is the best but for debugging, not needed here.

For emulation I will be using qemu.

```
sudo pacman -S qemu-full
```

Kernel

Get the sources for the latest compatible kernel 6.14.11:

```
git clone --depth=1 --branch v6.14.y
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 base configuration:

```
make ARCH=x86 tinyconfig
```

> This is a bootstrap with absolute minimum features. Just enough to boot the system. We want a little bit more.

Add additonal config settings on top of it:

```
make ARCH=x86 menuconfig
```

From menus choose those options:

- Processor type and features
    - x86 CPU resources control support
    - Processor family
        - 486DX
- Enable the block layer
- Device Drivers
    - Block devices
        - Normal floppydisk support
        - RAM block device support (1 partition)
    - Character devices
        - Enable TTY
- General Setup
    - Configure standard kernel features (expert users)
        - Enable support for printk
    - Initial RAM filesystem and RAM disk (initramfs/initrd)

        - Support initial ramdisk/ramfs compressed using XZ and uncheck everything else
- Executable file formats
    - Kernel support for ELF binaries
    - Kernel support for scripts starting with #!
- File systems
    - DOS/FAT/EXFAT/NT Filesystems
        - MSDOS fs support
    - Pseudo filesystems
        - sysfs file system support
    - Native language support
        - Codepage 437
- Library routines
    - XZ decompression and uncheck everything else

Exit configuration (yes, save settings to .config).

Time for compiling!

Compile Kernel

```
make ARCH=x86 bzImage -j$(nproc)
```

This will take a while depending on the speed of your CPU. In the end the kernel will be created in **arch/x86/boot/** as **bzImage** file.

Move kernel to our main directory:

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

Toolset

Without tools kernel will just boot and you will not be able to do anything. One of the most popular lightweight tools is BusyBox. It replaces the standard GNU utilities with way smaller but still functional alternatives, perfect for embedded needs.

Get the **1.36.1** version from busybox.net or Github mirror. Download the file, extract it, and change directory:

> Remember to be in the working directory.

```
wget https://github.com/mirror/busybox/archive/refs/tags/1_36_1.tar.gz
tar xzvf 1_36_1.tar.gz
cd busybox-1_36_1/
```

As with kernel you need to create starting configuration:

```
make ARCH=x86 allnoconfig
```

> You may skip this following fix if you are building on Debian/Fedora

Fix for ArchLinux based distributions:

```
sed -i 's/main() {}/int main() {}/' scripts/kconfig/lxdialog/check-lxdialog.sh
```

> 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 :) For the first time use my selection.

Run the configurator:

```
make ARCH=x86 menuconfig
```

Choose those:

- Settings
  - Build static binary (no shared libs)
  - Support files
    - 2GB
- Coreutils
  - cat
  - cp
  - df
  - echo
  - ls
  - mv
  - rm
  - sync
  - test
- Console Utilities
  - clear
- Editors
  - vi
- Init Utilities
  - init
  - uncheck everything else (inside init)
- Linux System Utilities
  - mdev
  - mount (just -o flag, rest off)

    - umount
- Shells
    - Choose alias as (ash)
    - ash
    - Optimize for size instead of speed
    - Alias support
    - Help support
- Miscellaneous
    - uncheck readahead

Now exit with save config.

Cross Compiler Setup

> Our target system needs to be 32-bit. To compile it on 64-bit system we need a cross compiler. You can setup this by hand in the menuconfig or just copy and paste those four lines.

Setup paths:

```
sed -i "s|.*CONFIG_CROSS_COMPILER_PREFIX.*|CONFIG_CROSS_COMPILER_PREFIX=\"${BASE}i486-linux-musl-cross/bin/i486-linux-musl-\"|" .config

sed -i "s|.*CONFIG_SYSROOT.*|CONFIG_SYSROOT=\"${BASE}i486-linux-musl-cross\"|" .config

sed -i "s|.*CONFIG_EXTRA_CFLAGS.*|CONFIG_EXTRA_CFLAGS=-I$BASE/i486-linux-musl-cross/include|" .config

sed -i "s|.*CONFIG_EXTRA_LDFLAGS.*|CONFIG_EXTRA_LDFLAGS=-L$BASE/i486-linux-musl-cross/lib|" .config
```

Compile BusyBox

Build tools and create base filesystem ("install"):

```
make ARCH=x86 -j$(nproc) && make ARCH=x86 install
```

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

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

Filesystem

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

> This created minimum viable directory structure for satisfying the basic requirements of a Linux system.

> Remember to be in the filesystem/ directory.

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

Next step is to add minimum configuration files. First one is a welcome message that will be shown after booting.

> Here is the first real opportunity to go wild and make this your own signature.

```
cat >> welcome << EOF
Your welome message or ASCII art.
EOF
```

Or download my welcome file.

```
wget https://krzysztofjankowski.com/floppinux/downloads/0.3.0/welcome
```

It looks like that:

```
$ cat welcome


                 _________________
               /_/ FLOPPINUX  /_/;
              / ' boot disk  ' //
             / '------------' //
            /   .---------.   //
           /   /          /  //
          .___/__________/__//   1440KiB
          '===\__________\=='   3.5"

________FLOPPINUX_V_0.3.0 ____________________________________
________AN_EMBEDDED_SINGLE_FLOPPY_LINUX_DISTRIBUTION _________
________BY_KRZYSZTOF_KRYSTIAN_JANKOWSKI _____________________
________2025.10 ______________________________________________
```

> Back to serious stuff. Inittab tells the system what to do in critical states like starting, exiting and restarting. It points to the initialization script rc that is the first thing that our OS will run before dropping into the shell.

Create an inittab file:

```
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 init rc script:

```
cat >> etc/init.d/rc << EOF
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
mdev -s
ln -s /proc/mounts /etc/mtab
mkdir -p /mnt /home
mount -t msdos -o rw /dev/fd0 /mnt
mkdir -p /mnt/data
mount --bind /mnt/data /home
clear
cat welcome
cd /home
/bin/sh
EOF
```

Make the script executable and owner of all files to root:

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

Compress this directory into one file:

```
find . | cpio -H newc -o | xz --check=crc32 --lzma2=dict=512KiB -e > ../rootfs.cpio.xz
cd ..
```

Create booting configuration.

> Another place to tweak parameters for your variant. Text after SAY is what will be displayed on the screen as first, usualy a name of the OS.

> The tsc=unstable is useful on some (real) computers to get rid of randomly shown warnings about Time Stamp Counter.

> Remember to be in the working directory.

```
cat >> syslinux.cfg << EOF
DEFAULT floppinux
LABEL floppinux
SAY [ BOOTING FLOPPINUX VERSION 0.3.0 ]
KERNEL bzImage
INITRD rootfs.cpio.xz
APPEND root=/dev/ram rdinit=/etc/init.d/rc console=tty0 tsc=unstable
EOF
```

Make it executable:

```
chmod +x syslinux.cfg
```

Create sample file

> To make the system a little bit more user friendly I like to have a sample file that user will be able to read and edit. You can put anything you want in it. A simple help would be also a good idea to include.

```
cat >> hello.txt << EOF
Hello, FLOPPINUX user!
EOF
```

Filesystem is ready. Final step is to put this all on a floppy!

Boot Image

First we need an empty file in exact size of a floppy disk. Then format and make it bootable.

Create empty floppy image:

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

Format it and create bootloader:

```
mkdosfs -n FLOPPINUX floppinux.img
syslinux --install floppinux.img
```

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

```
sudo mount -o loop floppinux.img /mnt
sudo mkdir /mnt/data
sudo cp hello.txt /mnt/data/
sudo cp bzImage /mnt
sudo cp rootfs.cpio.xz /mnt
sudo cp syslinux.cfg /mnt
sudo umount /mnt
```

Done!

Test in emulator

It's good to test before wasting time for the real floppy to burn.

Boot the new OS in qemu:

```
qemu-system-i386 -fda floppinux.img -m 20M -cpu 486
```

If it worked that means You have successfully created your own distribution! Congratulations!

The **floppinux.img** image is ready to burn onto a floppy and boot on real hardware!

Floppy Disk

<!> Important <!>

Change XXX to floppy drive name in your system. In my case it is **sdb**. Choosing wrongly will NUKE YOUR PARTITION and REMOVE all of your files! Think twice. Or use some GUI application for that.

```
sudo dd if=floppinux.img of=/dev/XXX bs=512 conv=notrunc,sync,fsync oflag=direct
status=progress
```

After 5 minutes I got freshly burned floppy.

Summary

- FLOPPINUX: 0.3.0
- Linux Kernel: 6.14.11
- Busybox: 1.36.1
- Image size: 1440KiB / 1.44MiB
- Kernel size: 868KiB (bzImage)
- Tools: 138KiB (rootfs.cpio.xz)
- Free space left (df -h): 264KiB

System Tools

File & Directory Manipulation

- cat - display file contents
- cp - copy files and directories
- mv - move/rename files and directories
- rm - remove files and directories
- ls - list directory contents

System Information & Management

- df - display filesystem disk space usage
- mount - mount filesystems
- umount - unmount filesystems
- sync - force write of buffered data to disk

Text Processing & Output

- echo - display text output
- more - page through text output

Utilities

- ln - create links between files
- clear - clear terminal screen
- test - evaluate conditional expressions

Applications

- vi - text editor

Download

- FLOPPINUX 0.3.0 Floppy Image 1.44MB

```
                    _________________
                   /_/ FLOPPINUX  /_/;
                  / ' boot disk  ' //
                 / '------------' //
                /   .--------.   //
               /   /         /  //
              .___/_________/__//   1440KiB
              '===\_________\=='   3.5"


          Now go and make something fun with it!
```

FLOPPINUX - An Embedded Single Floppy Linux Distribution
By Krzysztof Krystian Jankowski
2025.10