

РУКОВОДСТВО ДЛЯ ВЕРСИИ 0.1.0

By Krzysztof Krystian Jankowski for the community

Translated by Bs()Dd for the Russian community

- Разумеется, основная цель - уместить дистрибутив (ОС+программы) на одну дискету размером 1440KiB
- Использовать свежее ядро Linux
- Оставить только утилиты, необходимые для поддержки моего встроенного приложения
- Документация с легкими и понятными инструкциями для самостоятельной сборки
- Разумеется, все это бесплатно и с открытым исходным кодом

- Возможность примонтировать другую дискету и сохранить туда файлы
- Добавить текстовый редактор Nano (или ему подобный)

Приступим к сборке FLOPPINUX

Рабочая директория

Создайте папку, где будут располагаться все файлы.

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

Ядро

Я использую последнюю версию. Такой шаг позволяет объединить старые и новые технологии воедино. На данный момент это версия 5.13.0-rc2.

Скачаем исходники:

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

Исходники загрузились в папку /linux/. Давайте сконфигурируем и соберем кастомное ядро. Для начала создадим минимальную конфигурацию:

```
make ARCH=x86 tinyconfig
```

Теперь нужно добавить дополнительные настройки:

```
make ARCH=x86 menuconfig
```

В меню выберите следующие опции:

- 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 #!**

Выйдите из конфигуратора (да, сохраните настройки в .config). А теперь время компиляции!

```
make ARCH=x86 bzImage
```

Это займет некоторое время, в зависимости от мощности вашего процессора. После компиляции файл ядра создастся в *arch/x86/boot/bzImage*. Переместим его в основную папку.

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

Утилиты

Без утилит мы можем лишь загрузиться в ядро, но толку от этого нет. Наиболее популярный набор утилит - BusyBox. Он заменяет (большинство) GNU утилит с достаточным функционалом для встраиваемых нужд.

Последняя версия доступна по адресу <https://busybox.net/downloads/>. На данный момент это 1.33.1. Скачайте файл, распакуйте и перейдите в распакованную папку:

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

Как и при сборке ядра, нужно создать конфигурацию:

```
make allnoconfig
```

Сейчас начинается самое веселое. Нужно выбрать необходимые вам утилиты. Каждый пункт меню показывает, сколько места займет та или иная утилита. Выбирайте с умом :)

```
make menuconfig
```

Я выбрал следующее:

- Settings > **Build static binary (no shared libs)**
- Coreutils > **cat, du, echo, ls, sleep, uname** (если хотите, можете

сменить название ОС)

- 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**

Выйдите, сохранив конфигурацию. И снова время компиляции.

```
make  
make install
```

***Прим. переводчика:** Если у вас Linux x64, то команды должны выглядеть так:*

```
CFLAGS=-m32 LDFLAGS=-m32 make  
CFLAGS=-m32 LDFLAGS=-m32 make install
```

Иначе скомпилируется BusyBox x64, который не будет работать в системе (error -8). Убедитесь, что у вас установлен gcc-multilib.

Создастся файловая система со всеми файлами в _install. Переместите ее в главную папку. Я предпочитаю переименовать ее.

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

Файловая система

Вы получили ядро и базовый набор утилит. Но системе необходима дополнительная структура директорий.

```
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
```

Теперь создадим конфигурационные файлы. Первый — файл приветствия при запуске:

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

Inittab файл обрабатывает запуск, перезапуск и выключение:

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

И скрипт инициализации:

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

Делаем файл инициализации исполняемым и делаем root владельцем всех файлов:

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

Теперь сжимаем директорию в один файл:

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

Вы можете проверить работоспособность, запустив QEMU из базовой директории:

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

Теперь нужно поместить все это на дискету!

Загрузочный образ

Создайте файл для grub-a, содержащий созданные вами ядро и файловую систему:

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

Создайте пустой образ дискеты:

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

Примонтируйте его, затем скопируйте syslinux, ядро и файловую систему в образ:

```
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
```

Готово!

Теперь у вас есть свой образ floppinux.img, который можно записать на дискету и загрузиться с нее на настоящем железе!

Если у вас нет дискет, вы просто можете протестировать образ в QEMU, как нормальный человек:

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

Итоги

Полный размер: 1440KiB / **1.44MiB**

Размер ядра: 632KiB

Утилиты: 552KiB

Свободное место (du -h): **272KiB**

Добавляем встроенное приложение

Теперь, когда у нас есть встраиваемый дистрибутив, попробуем извлечь из него пользу. Он загружается очень быстро (после загрузки дискеты) и может легко запустить какое-нибудь скомпилированное приложение. Но я хочу немного повеселиться со скриптами. Поэтому вместо скомпилированных программ я положу пару скриптов. Процесс ничем не отличается.

- Обновить файлы в директории /filesystem/
- сжать файл rootfs
- смонтировать образ дистрибутива
- заменить rootfs
- демонтировать образ
- (опционально) записать новый образ на дискету
- загрузиться в новую систему с обновленным софтом

Вы также можете изменить скрипт *etc/init.d/rc* и заменить /bin/sh на путь к скрипту/двоичному файлу.

Но на время отладки лучше запускать приложение вручную. Я зависим от скриптов, поэтому наличие редактора Vi очень удобно для тестирования фиксов непосредственно в системе.

Ресурсы

- <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