Сборка ядра и initramfs (кросс-компилляция)

Си

Сборка проходила на ядре версии 6.5.9 на fedora 38

Задаем переменные окружения, чтобы система make исполняла скрипты для архитектуры `ARM` с кросс-комплиллятором `arm-linux-gnu-gcc`

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnu-
```

Убеждаемся, что система make теперь подготовлена для ARM архитектуры

```
make help
```

```
Architecture specific targets (arm):
* zImage        - Compressed kernel image (arch/arm/boot/zImage)
  Image         - Uncompressed kernel image (arch/arm/boot/Image)
* xipImage      - XIP kernel image, if configured (arch/arm/boot/xipImage)
  uImage        - U-Boot wrapped zImage
  bootpImage    - Combined zImage and initial RAM disk
                  (supply initrd image via make variable INITRD=<path>)
  install       - Install uncompressed kernel
  zinstall      - Install compressed kernel
  uinstall      - Install U-Boot wrapped compressed kernel
                  Install using (your) ~/bin/installkernel or
                  (distribution) /sbin/installkernel or
                  install to $(INSTALL_PATH) and run lilo
  vdso_install  - Install unstripped vdso.so to $(INSTALL_MOD_PATH)/vdso

  multi_v7_lpae_defconfig     - multi_v7_defconfig with CONFIG_ARM_LPAE enabled
```

Генерируем дефолтный конфиг ядра для ARM архитектуры

```
make defconfig
```

```
❯ make defconfig
*** Default configuration is based on 'multi_v7_defconfig'
#
# No change to .config
#
```

Теперь компилируем ядро под ARM:

```
make -j x zImage
```

Где `x` - количество потоков для сборки

```
❯ make -j 4 zImage
  SYNC    include/config/auto.conf
*
* Restart config...
*
*
* Kernel Features
*
Symmetric Multi-Processing (SMP) [Y/n/?] y
  Allow booting SMP kernel on uniprocessor systems (SMP_ON_UP) [Y/n/?] y
Support cpu topology definition (ARM_CPU_TOPOLOGY) [Y/n/?] y
  Multi-core scheduler support (SCHED_MC) [N/y/?] n
  SMT scheduler support (SCHED_SMT) [N/y/?] n
Architected timer support (HAVE_ARM_ARCH_TIMER) [Y/?] y
Multi-Cluster Power Management (MCPM) [Y/?] y
big.LITTLE support (Experimental) (BIG_LITTLE) [N/y/?] n
Memory split
> 1. 3G/1G user/kernel split (VMSPLIT_3G)
  2. 3G/1G user/kernel split (for full 1G low memory) (VMSPLIT_3G_OPT)
  3. 2G/2G user/kernel split (VMSPLIT_2G)
  4. 1G/3G user/kernel split (VMSPLIT_1G)
choice[1-4?]: 1
```

Проверяем скомпилированное ядро:

```
❯ file arch/arm/boot/zImage
arch/arm/boot/zImage: Linux kernel ARM boot executable zImage (little-endian)
```

Собираем dtb конфигурации для возможных плат

```
make dtbs
```

Запускаем эмуляцию устройств ARM с помощью QEMU и отдаем эмулятору свежесобранное ядро

```
QEMU_AUDIO_DRV=none qemu-system-arm -M vexpress-a9 -kernel zImage -dtb ./dts/arm/vexpress-v2p-ca9.dtb -append "console=ttyAMA0" -nographic
```

где:

- QEMU_AUDIO_DRV - отключение проброса звука
- M - указание платы, на котором исполняется ядро

- kernel - указание ядра, которое необходимо загрузить
- dtb - указание конфигурации платы (какие устройства есть, номера прерываний, порты ввода/вывода и т.д.)
- append - подключить вывод QEMU к stdout "гипервизора"
- nographic - не запускать VNC сервер

```
❯ QEMU_AUDIO_DRV=none qemu-system-arm -M vexpress-a9 -kernel zImage -dtb ./dts/arm/vexpress-v2p-ca9.dtb -append "consol
e=ttyAMA0" -nographic
lan9118: error: PHY write reg 13 = 0x0003
lan9118: error: PHY write reg 14 = 0x0014
lan9118: error: PHY write reg 13 = 0x4003
lan9118: error: PHY read reg 14
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 6.5.9-g8609ed56389a (andrey@fedora) (arm-linux-gnu-gcc (GCC) 13.2.1 20230728 (Red Hat Cros
s 13.2.1-1), GNU ld version 2.39-4.fc38) #2 SMP Fri Oct 27 11:44:21 +07 2023
[    0.000000] CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
[    0.000000] OF: fdt: Machine model: V2P-CA9
[    0.000000] Memory policy: Data cache writeback
[    0.000000] efi: UEFI not found.
[    0.000000] Reserved memory: created DMA memory pool at 0x4c000000, size 8 MiB
[    0.000000] OF: reserved mem: initialized node vram@4c000000, compatible id shared-dma-pool
[    0.000000] OF: reserved mem: 0x4c000000..0x4c7fffff (8192 KiB) nomap non-reusable vram@4c000000
[    0.000000] cma: Failed to reserve 64 MiB
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000060000000-0x0000000067ffffff]
[    0.000000]   Normal   empty
[    0.000000]   HighMem  empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
```

```
[    2.367800] Hardware name: ARM-Versatile Express
[    2.368960]  unwind_backtrace from show_stack+0x10/0x14
[    2.370346]  show_stack from dump_stack_lvl+0x40/0x4c
[    2.370612]  dump_stack_lvl from panic+0x10c/0x338
[    2.370893]  panic from mount_root_generic+0x2d0/0x380
[    2.371190]  mount_root_generic from prepare_namespace+0x33c/0x3b0
[    2.371424]  prepare_namespace from kernel_init+0x10/0x12c
[    2.371637]  kernel_init from ret_from_fork+0x14/0x28
[    2.371880] Exception stack(0xc8825fb0 to 0xc8825ff8)
[    2.372288] 5fa0:                                     00000000 00000000 00000000 00000000
[    2.372602] 5fc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[    2.372859] 5fe0: 00000000 00000000 00000000 00000000 00000013 00000000
[    2.373816] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

В результате, произошел kernel panic, так как у нас нет initramfs.

Сборка initramfs

> ⚠ **Warning**
>
> На Fedora пока есть только средства для компилляции ядра. Пользуйся Debian 12 или ubuntu

Скачиваем busybox. Переходим в директорию с busybox и генерируем дефолтную конфигурацию

```
ARCH=arm make defconfig
```

Включаем компилляцию со статической линковкой, прописываем директорию установки и прописываем префикс кросс-компиллятора `arm-linux-gnueabihf-` через вызов

```
ARCH=arm make menuconfig
```

После чего, начинаем сборку initramfs

```
ARCH=arm make busybox
```

Запускаем установку в ранее указанную директорию

```
ARCH=arm make install
```

Упаковываем все файлы initramfs в файл img

```
find . | cpio --quiet -H newc -o | gzip -9 -n > initramfs.img
```

Когда готов initramfs, закидываем готовый файл к нашему собранному ядру

```
cp initramfs.img ~/kernelcheck/linux-6..../arch/arm/boot
```

После чего, запускаем эмулятор

```
QEMU_AUDIO_DRV=none qemu-system-arm -M vexpress-a9 -kernel zImage -dtb ./dts/arm/vexpress-v2p-ca9.dtb -append "console=ttyAMA0" rdinit=/bin/ash -nographic -initrd initramfs.img
```

В качестве init - указываем оболочку /bin/ash