

06-4 initramfs

6.4 Initial RAM Disk

- Need to mount an early root file system for certain startup-related initialization
- Two approaches
- **initrd**
 - Used in angstrom
- **initramfs**
 - Newer method

Traditional booting sequence

Bootloader

- Executed by the hardware at a fixed location in ROM / Flash
- Initializes support for the device where the kernel image is found (local storage, network, removable media)
- Loads the kernel image in RAM
- Executes the kernel image (with a specified command line)

Kernel

- Uncompresses itself
- Initializes the kernel core and statically compiled drivers (needed to access the root filesystem)
- Mounts the root filesystem (specified by the `root` kernel parameter)
- Executes the first userspace program (specified by the `init` kernel parameter)

First userspace program

- Configures userspace and starts up system services

From Free Electrons: <http://free-electrons.com/blog/beagle-labs/>

Drawbacks

- ▶ Assumption that all device drivers needed to mount the root filesystem (storage and filesystem drivers) are statically compiled inside the kernel
- ▶ Assumption can be *correct* for most embedded systems, where the hardware is known and the kernel can be fine-tuned for the system
- ▶ Assumption is mostly *wrong* for desktop and servers, since a single kernel image should support a wide range of devices and filesystems
 - ▶ More flexibility was needed
 - ▶ Modules have this flexibility, but they are not available before mounting the root filesystem
 - ▶ Need to handle complex setups (RAID, NFS, etc.)

Solution

- ▶ A solution is to include a small temporary root filesystem with modules, in the kernel itself. This small filesystem is called the *initramfs*
- ▶ This *initramfs* is a gzipped cpio archive of this basic root filesystem
 - ▶ A gzipped cpio archive is a kind of zip file, with a much simpler format
- ▶ The *initramfs* scripts will detect the hardware, load the corresponding kernel modules, and mount the real root filesystem
- ▶ Finally the *initramfs* scripts will run the *init* application in the real root filesystem and the system can boot as usual
- ▶ The *initramfs* technique completely replaces *init ramdisks* (*initrds*). *Initrds* were used in Linux 2.4, but are no longer needed

Bootling sequence with initramfs

Bootloader

- Executed by the hardware at a fixed location in ROM / Flash
- Initializes support for the device where the images are found (local storage, network, removable media)
- Loads the kernel image in RAM
- Executes the kernel image (with a specified command line)

Kernel

- Uncompresses itself
- Initializes the kernel core and statically compiled drivers
- Uncompresses an *initramfs* cpio archive (if existing, in the kernel image or copied to memory by the bootloader) and extracts it to the kernel file cache (no mounting, no filesystem).
- If found in the *initramfs*, executes the first userspace program: `/init`

Userspace: `/init` script (what follows is just a typical scenario)

- Runs userspace commands to configure the device (such as network setup, mounting `/proc` and `/sys`...)
- Mounts a new root filesystem. Switch to it (`switch_root`)
- Runs `/sbin/init`

Userspace: `/sbin/init`

- Runs commands to configure the device (if not done yet in the *initramfs*)
- Starts up system services (daemons, servers) and user programs

unchanged

Initramfs features and advantages

- ▶ Root filesystem directly embedded in the kernel image, or copied to RAM by the bootloader, simple solution
- ▶ Just a plain compressed cpio archive extracted in the file cache. Neither needs a block nor a filesystem driver
- ▶ Simpler to mount complex filesystems from flexible userspace scripts rather than from rigid kernel code. More complexity moved out to user-space!
- ▶ Possible to add non GPL files (firmware, proprietary drivers) in the filesystem. This is not linking, just file aggregation (not considered as a derived work by the GPL)

How to populate an initramfs

Using `CONFIG_INITRAMFS_SOURCE` in kernel configuration (General Setup section)

- ▶ Either give an existing `cpio` archive (file name ending with `.cpio`)
- ▶ Or give a directory to be archived
- ▶ Any other regular file will be taken as a text specification file (see next page)

see .../Documentation/filesystems/ramfs-rootfs-initramfs.txt and .../Documentation/early-userspace/README in kernel sources.

See also <http://www.linuxdevices.com/articles/AT4017834659.html> for a nice overview of initramfs (by Rob Landley).

Initramfs specification file example

```
dir /dev 0755 0 0
nod /dev/console 0600 0 0 c 5 1
dir /root 0700 0 0
```

major minor
permissions

No need for `root` user access!

Initramfs specification file example

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
nod /dev/loop0 644 0 0 b 7 0
dir /bin 755 1000 1000
file /bin/busybox
/stuff/initramfs/busybox 755 0 0
slink /bin/sh busybox 777 0 0
dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
file /init /stuff/initramfs/init.sh 755 0 0
```

major minor
permissions
user id group id

No need for `root` user access!

How to handle cpio archives

Useful when you want to build the kernel with a ready-made `cpio` archive, instead of letting the kernel do it for you

- ▶ Extracting:

```
host$ cpio -id < dir.cpio
```

- ▶ Creating:

```
host$ cd dir
host$ find . | cpio -H newc -o > ../dir.cpio
```

Note that the `-H newc` option is required to generate a `cpio` archive that can be used by the Linux kernel.

Summary

- ▶ For embedded systems, two interesting solutions
 - ▶ No initramfs: all needed drivers are included inside the kernel, and the final root filesystem is mounted directly
 - ▶ Everything inside the initramfs