Real Time Operating Systems

RootFS Creation: Summing Up

Luca Abeni

System Boot

- System boot $\rightarrow$ the CPU starts executing from a well-known address
- ROM address: BIOS $\rightarrow$ read the first sector on the boot device, and executes it
  - Bootloader (GRUB, LILO, U-Boot, . . .)
  - In general, load a kernel and an “intial ram disk”
  - The initial fs image isn’t always needed (example: netboot)
- Kernel: from `arm-test-*.tar.gz`
- Initial filesystem?
  - Loaded in RAM without the kernel help
  - Generally contains the boot scripts and binaries

Initial Filesystem

- Old (2.4) kernels: Init Ram Disk (initrd); New (2.6) kernels: Init Ram Filesystem (initramfs)
- Generally used for modularized disk and FS drivers
  - Example: if IDE drivers and Ext2 FS are modules (not inside the kernel), how can the kernel load them from disk?
  - Solution: boot drivers can be on initrd / initramfs
    - The bootloader loads it from disk with the kernel
    - The kernel creates a “fake” fs based on it
    - Modules are loaded from it
- Embedded systems can use initial FS for all the binaries
- Qemu does not need a bootloader to load kernel and initial FS (`-kernel` and `-initrd`)

Init Ram Filesystem

- Used in 2.6 kernels
- It is only a RAM FS: no real filesystem metadata on a storage medium
- All the files that must populate the FS are stored in a cpio package (similar to tar or zip file)
- The bootloader loads the cpio file in ram
- At boot time, the kernel “uncompresses” it, creating the RAM FS, and populating it with the files contained in the archive
- The cpio archive can be created by using the `cpio -o -H newc` command (see `man cpio`)
- Full command line: `find . | cpio -o -H newc | gzip > <file name>`

How to Populate an Init Filesystem

- Some executables in `/bin` and `/sbin`
- Configuration files in `/etc`
- Dynamically linked binaries $\rightarrow$ shared objects in `/lib`
- The kernel starts `/init` as an init process
  - If a real "init" program is used $\Rightarrow$ `/etc/inittab`
  - Your inittab might reference programs like `getty` that you need to provide
- Executables for `/bin` and `/sbin` can be provided by busybox
  - Kernel $\rightarrow$ Linux
  - User Space (init filesystem) $\rightarrow$ busybox

Init Ram Disk

- Only kind of init filesystem supported in old (2.4) kernels
- A Ram Disk device is used as a block device for the init filesystem
  - Difference respect to initramfs: there are FS metadata in the block device
  - Real filesystem: FAT, ext2, ext3, reiserfs, . . .
- An initrd can be created by:
  1. Creating an empty file (something like `dd if=/dev/zero of= ...`)
  2. Creating a filesystem on it (by using `mkfs.* ...`)
  3. Mounting the FS with a loop device (`mount -o loop ...`)
  4. Writing the files in it

Creating an InitRD

- An initrd can be created similarly to an initramfs...
- ...But you generally need to be root
  - Only the root can mount the ram disk
- Moreover, some space is wasted by the FS metadata in the initrd image
- The “Ram Disk” block device must be enabled in the kernel
- As an alternative, `e2fstools` (filesistem access implemented in user space) can be used to fill the initrd
  - No need to be root