

Boot your Linux

Well

Summary

- Classic boot scheme
- Initrd
- Initramfs
- Current implementations
- Persistent device names problem
- Initramfs – the ultimate way
- The lecture goal

Classic boot scheme

- root=...
- FS drivers compiled into kernel
- Root FS is mounted by kernel. Init executed
- Disadvantages:
 - FS drivers in kernel
 - Complicated LVM and RAID support
 - Complicated network boot
- Good for embedded setups

initrd

- Temporary FS image is supplied
- Its mounted and /linuxrc executed
 - Loads necessary modules and exits
- Kernel will continue as in classic boot

initrd - advanced

- Temporary FS image is supplied
- Its mounted and `/linuxrc` executed:
 - load modules and mount `/new-root`
 - `cd /new-root; mkdir initrd`
 - `pivot_root . initrd`
 - `echo 0x0100 > /proc/sys/kernel/real-root-dev`
 - We are done
- Kernel sees that `/` is already mounted and executes `“init”`.

initrd disadvantages

- Unclean implementation
- Userspace given control too late during boot
- Anyway, the solution is sufficient for most needs
- More in “Documentation/initrd.txt”

Initramfs

- Enter userspace as early as possible
- And let it do all of the job:
 - Find and mount new root
 - chroot to new root and exec real init
- If something goes wrong:
 - “Kernel panic: attempted to kill init”

Initramfs – details

- *cpio* image is supplied to kernel
 - GRUB: `initrd=/boot/iniramfs.cpio.gz`
- Kernel unpacks image to RAM disk and executes “/init”
- Now we can do whatever we want. We do not even have to mount real root.
- Files are paged – perfect for embedded.

initrd/ramfs implementations

- Usually distro-specific script, that:
 - Detects boot device
 - Creates “/init” script that will:
 - Load required kernel modules
 - Run LVM/RAID initialization
 - Do the rest of mounting and chrooting
 - Adjust boot loader “root=” option
- Red Hat created special “nash” interpreter

Current problems

- Initialization timeouts:
 - Loading device module does not guarantee that its ready for mounting
- Inconsistent device naming (next slide)

“root=” problem

- What is “/dev/sda”?
 - Depends on bus scan order
 - May change when new devices are added
- It may be enough to forget your Disk-On-Key plugged in and the system will fail to boot
- Most distros do not deal with the problem

root=LABEL=

- Kernel
 - Kernel has to be aware of FS headers
- Initrd/Initramfs
 - Mount tools (or Nash) has to be aware of FS headers
- LABEL collisions are possible
 - Famous Red Hat's “root=LABEL=/
”

root=UUID

- Yes! File system is uniquely identified by UUID!
- But in current implementations
 - The same drawback as with LABEL (FS headers)
 - Scary kernel command lines :)

Persistent device naming

- Udev in a nutshell
 - Kernel export info via sysfs
 - Kernel events emitted
 - Udev(sysfs, events, udev rules) = /dev/layout
- Udev:
 - “/dev/disk/by-uuid” lists UUIDS of all file systems discovered so far

Initramfs-NG

- <http://mkinitramfs.sf.net>
- Let the udev to device initialization
- Reuse standard tools (udev rules, no klibc/busybox)
- As simple as:
 - Setup environment
 - Trigger udev events
 - Wait for “/dev/disk/by-uuid/\$MY_UUID” to appear
 - mount/chroot/exec

Thank you!