Boot your Linux

Summary

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

Classic boot scheme

- root=...
- FS drivers compiled into kernel
- root FS is mounted by kernel and init is executed
- Disadvantages:
  - FS drivers in kernel
  - Complicated LVM and *-RAID support
  - Complicated network boot

initrd

- Temporary FS is image is supplied
- Its mounted and /linuxrc is executed:
  - Load necessary modules and exit
- Kernel will continue as in classic boot.

Initrd - advanced

- Temporary FS is image is supplied
- Its mounted and /linuxrc is executed:
  - 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
- Difficulties with booting from network
- User space is given control too late during boot
- Anyway, this solution is sufficient for most needs
- More in “Documentation/initrd.txt”

Initramfs

- Enter user-space 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/initramfs.cpio.gz
- Kernel unpacks image to RAM disk and executes "/init"
- Now we can do what ever we want. We do not even have to mount real root.
- For example:
  - Acrnonis TrueImage software is run
  - HP RAID setup CD even runs X from here

Initrd/ramfs - implementations

- Usually distro specific script, that:
  - Detects your boot device
  - Creates initramfs “/init” (initrd's “/linuxrc”):
    - Loads required kernel modules
    - Runs RAID/LVM activation commands
    - Does the rest of the mounting and chrooting
  - Adjusts boot loader's “root=” option
- Red Hat created special “nash” interpreter
- PCLinuxOS initramfs example

Current problems

- Initialization timeouts:
  - Loading device module does guarantee that its ready for mounting
- Inconsistent device naming
  - Next slide

“root=” problem

- What is “/dev/sda1”?
  - Depends on bus scan order
  - May change when new devices are added
- It may be enough to forget your USB DOK plugged in and the system will fail to boot.
- Most distros do not deal with the problem

root=LABEL=

- Initrd:
  - Kernel has to be aware of FS headers
- Initiramfs:
  - mount tools (or Nash) have 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
  - Scary kernel command lines :)

Persistent device naming

- Udev in the nutshell:
  - Kernel exports 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 do device initialization
- Reuse standard tools (udev rule, no klibc/busbox)
- As simple as:
  - Setup environment
  - Trigger udev events
  - Wait for “/dev/disk/by-uuid/$MY_UUID” to apear
  - Mount/chroot/exec

Initramfs-NG – Lets code!

Lets write “/init” right now!