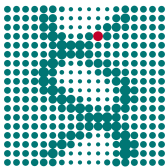
LinuxBoot and booting fast

Paul Menzel (Max Planck Institute for Molecular Genetics)

September 28th, 2018

Who am I?



- (Economic) Mathematician by studies at TU Berlin
- Free Software enthusiast
- Active in coreboot since 2005 (still LinuxBIOS back then)
- System architect at Max Planck Institute for Molecular Genetics

Presentation

Used Markdown and Pandoc, sources available online.

TinyURL: https://tinyurl.com/linuxbootfast

https://github.com/paulmenzel/linuxboot-and-booting-fast

Introduction

Why this talk?

1. Warning: Focus on x86
2. Goal of fast boot
3. Bad experiences with proprietary firmware
4. Interesting topics
5. Cool community

LinuxBoot

Motivation

We do not trust and do not like firmware.

1. Everywhere
2. High privileges, and is essentially an OS
3. SMM is bad (System Management Mode)
4. Slow
5. Buggy
6. Pain to update
7. Often proprietary
8. Different and unfamiliar code base
9. Quite limited in functionality
10. Necessary to write to flash ROM chip for update

Solution: LinuxBoot – Let Linux do it

Advantages of Linux kernel

1. Familiar code base
2. Well tested
3. Great hardware support (Braille, WiFi devices, ...)
4. Kexec as boot loader
5. Familiar user space in initrd
6. Fix issue by reboot without flashing something

Implementation

1. Make firmware as small as possible
2. Move as much as possible into the Linux kernel
3. Use a small Linux kernel as *boot kernel*
4. Use Linux as bootloader with kexec

History of LinuxBIOS

1. Started by Ron Minnich as LinuxBIOS at LANL
2. „Press F1 to continue.“
3. The Linux BIOS
4. https://www.coreboot.org/Clusters
5. 1024-node linuxbios cluster with Dual-P4 systems and Myrinet

The present

Power architecture (ppc64)

1. Since Power 8 Petitboot

Other talk on LinuxBoot

Netboot21: Bootloaders in the 21st Century
User-space bootloaders with LinuxBoot

. . . on Saturday, 4:30 p. m. by Chris Koch

What is LinuxBoot?

1. Use Linux as boot-kernel and with initrd as bootloader
2. Feasible due to increased sizes of flash ROM chips (thanks to UEFI firmware)
3. Use defined interfaces (coreboot romstage, UEFI PEI, U-Boot SPL)

Initrd/initramfs

1. Heads
2. u-root
3. Petitboot (Buildroot)
4. Everything that fits
   4.1 OpenEmbedded/Yocto

Demo time

1. ASRock E350M1
2. AMD Fusion (APU, integrated graphics device)
3. socketed 4 MB flash ROM chip by default
4. 4 GB RAM

Booting fast

Motivation

1. Systems get faster, but still take some time
2. Differentiate between firmware and OS
3. Suspend to RAM is bad, and just a workaround (and adds unneded complexity)
4. If you want to keep the state, use suspend to disk.
5. How many power plants could be shut down?
6. Customers should request fast boot times.
7. Chromebooks and -boxes have boot time requirements.
8. Even on servers, so you can just reboot with a downtime less than the TCP time-out.

Past efforts

1. LPC: Booting Linux in five seconds
2. Ten years ago: September 2008
3. Eee PC

Demo platform

1. Seven year old ASRock E350M1
2. AMD Fusion
3. Kingston SSD

Firmware

1. Use coreboot
2. 1 second with loading GRUB payload
3. Option ROM and AGESA integration slow
4. Siemens MB TCU3 with coreboot and SeaBIOS payload: Total Time: 377,319 (`siemens/mc_tcu3/4.4-108-g0d4e124/2016-05-09T06_14_45Z`)

Operating system

1. Linux kernel
2. Initrd/initramfs
3. User space

Linux kernel

1. Build it yourself
2. Use `initcall_debug`
3. kprobes
4. systemd-bootchart
5. `bootgraph.py` (with ftrace)
6. Doesn't seem much focus

Initrd/Initramfs

1. Use LZ4 with SSD

```
[    0.484102] calling  populate_rootfs+0x0/0x10f @ 1
[    0.484127] Unpacking initramfs...
[    0.538943] Freeing initrd memory: 29020K
[    0.538955] initcall populate_rootfs+0x0/0x10f returned 0 after 535
```

2. Make it smaller by only using necessary modules

   `MODULES=dep` in `/etc/initramfs-tools/initramfs.conf`

3. Get rid of initramfs (most systems static)

User space

1. systemd-analyze
2. systemd-bootchart
3. strace – trace system calls and signals
4. perf – Performance analysis tools for Linux
5. Deactivate services
   5.1 for example ModemManager not needed on desktop systems
6. Reorder services depending on system
7. systemd-journal flush takes long
8. udev rules

ACPI S3

1. `sleepgraph.py`

What to do?

Users

1. Support vendors caring about these things.
2. Use Power
   2.1 OpenPower Foundation
   2.2 Workstations available: https://www.raptorcs.com/TALOSII/
3. Resellers for older devices
4. Purism devices
5. Reseller for used Facebook Open Compute Project systems: http://www.horizon-computing.com/
6. Google Chromebooks and -boxes (Intel and ARM)
7. Dell: Systems with GNU/Linux preinstalled, has Linux developers, and LVFS support for a long time

What is needed to improve the situation?

1. Interfaces to avoid reinitializing devices
2. Pressure on device manufactures to care about boot time (NVMe, ...)
3. Different target types for desktops, servers, ...
4. Focus on fast startup times
   4.1 Integrate profiling tools in systemd