Von Null bis Grafik in 5 Sekunden

Paul Menzel

17. März 2019

Wer bin ich?

- Systemarchitekt beim Max-Planck-Institut für molekulare Genetik
- Diplom-Wirtschaftsmathematiker (TU Berlin)
- FLOSS-Befürworter
- Seit 2005 im Projekt coreboot aktiv (damals LinuxBIOS)
- 2 Artikel im Linux Magazin

Präsentation/Folien

Mit Markdown und Pandoc erstellt, Quellen verfügbar.

TinyURL: https://tinyurl.com/vonnullbisgrafikinfuenfsekunden

https://github.com/paulmenzel/von_null_bis_grafik_in_5_sekunden

Einführung

Warum?

1. Warnung: Fokus auf x86
2. Ziel: Schnell starten
3. Interessantes und spannendes Thema ohne Ende
4. Tolle Gemeinschaft aus unterschiedlichen Bereichen

Schneller Start

Motivation

1. Trotz immer leistungsfähigeren Geräten dauert der Start immer noch lange.
2. Unterschied durch SSD
3. Wenige Hersteller fokussieren sich darauf. Google Chromebooks positive Ausnahme mit Bedingungen in weniger als 10 Sekunden zu starten.
4. Unklar, warum von Kunden akzeptiert. (auch bei anderen Komponenten: Fernseher, Telefone)

Einschub Ruhemodus (ACPI S3, neue Idle-Zustände S0ix)

1. Bereitschaft (ACPI S3, Suspend to RAM) schlechte Lösung zum Lösung des Problems des langsamen Starts
2. Wie viel Ressourcen in Entwicklung, Fehlerbehebung (Firmware)
3. Suspend to Disk Lösung zum Speichern des Zustandes
4. Wie viel Kraftwerke könnten weltweit gespart werden?

Motivation

1. Auch auf Servern könnten mit schnellen Startzeiten Wartung vereinfachen, wenn Neustart schneller ist als Standard-TCP-Zeitüberschreitung.

Ähnliche Vorhaben in Vergangenheit

1. LPC: Booting Linux in five seconds
2. Von vor 10 Jahren: September 2008
3. Eee PC

Demo

Demo

1. sieben Jahre altes ASRock E350M1
2. AMD Fusion (APU, integriertes Grafikgerät)
3. ausgeliefert mit steckbarer 4-MB-Flash-ROM-Chip (Löschen von Block bis 400 ms)
4. 4 GB RAM
5. Kingston SSD

Demo: Ergebnis

1. gut sechs Sekunden
2. Firmware: coreboot mit GRUB-Payload: 1,5 Sekunden
3. selbstgebautes Linux von master-Zweig ohne initrd: 1,1 Sekunden
4. Debian mit systemd 241 und Weston 5.0.0 (Wayland): 3 bis 4 Sekunden

Komponenten

1. Firmware
2. Betriebssystem
   - 2.1 Linux-Kernel
   - 2.2 Initrd/Initramfs
   - 2.3 Userspace

Firmware

1. Standardmäßig BIOS/UEFI und alleine über 5 s – meist 8 bis 10 s
2. Lösung: coreboot-basierende Firmware
3. 1 Sekunde mit GRUB-Payload (minimiertes `default_payload.elf`)
   3.1 `make default_payload.elf FS_PAYLOAD_MODULES="" EXTRA_PAYLOAD_MODULES="boottime"`
   3.2 Komprimiert 177 KB in CBFS
4. Option ROM and AGESA integration slow
5. Siemens MB TCU3 with coreboot and SeaBIOS payload

   `Total Time: 377,319`

   aus *Board Status*
   `siemens/mc_tcu3/4.4-108-g0d4e124/2016-05-09T06_14_45Z`

Geschichte von coreboot/LinuxBIOS

1. Gegründet von Ron Minnich als LinuxBIOS beim 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

Firmware: LinuxBoot

1. Linux as Payload: 416 KB mit `make tinyconfig`
2. größer als GRUB-Payload und keine Shell
3. Trotzdem Vorteile

Betriebssystem

1. Linux Kernel
2. Initrd/initramfs
3. Userspace

Linux Kernel

1. Selbstbauen mit minimaler Konfiguration
2. `initcall_debug` zeigt, wie lange Modul-Init-Routinen brauchen
3. Kprobes
4. systemd-bootchart bereitet Daten auf
5. `bootgraph.py` (with ftrace)
6. Schnelles Starten kein Schwerpunkt bei Entwicklerinnen und Entwicklern

initcall_debug

Aus `init/main.c`:

```
static __init_or_module void
trace_initcall_finish_cb(void *data, initcall_t fn, int ret)
{
        ktime_t *calltime = (ktime_t *)data;
        ktime_t delta, rettime;
        unsigned long long duration;

        rettime = ktime_get();
        delta = ktime_sub(rettime, *calltime);
        duration = (unsigned long long) ktime_to_ns(delta) >> 10;
        printk(KERN_DEBUG "initcall %pF returned %d after %lld usecs\n",
                 fn, ret, duration);
}
```

Initrd/Initramfs

1. Nutze LZ4 zum Komprimieren bei 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. Verkleinern durch Einfügen von nur benötigten Modulen (statisches System)

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

3. Kein initramfs (schlecht bei Verschlüsselung)

Userspace

1. systemd-analyze
2. systemd-bootchart
3. strace – trace system calls and signals
4. perf – Performance analysis tools for Linux
5. Deaktiviere Dienste
   5.1 zum Beispiel ModemManager wird oft nicht benötigt
6. Anordnung der Dienste ändern
7. systemd-journal flush brauchte lange
8. udev-Regeln in `/usr/lib/rules.d/` durchschauen

ACPI S3

1. `sleepgraph.py`

Was kann gemacht werden?

Nutzer

1. Unterstütze Hersteller, die darauf achten
2. Reseller für alte Geräte
3. Geräte von Purism
4. Reseller für gebrauchte Geräte von Facebook (Open Compute Project): http://www.horizon-computing.com/
5. Google Chromebooks and -boxes (Intel and ARM)
6. Dell: Systems with GNU/Linux preinstalled, has Linux developers, and LVFS support for a long time

Wie kann die Situation verbessert werden?

1. Schnittstellen, um mehrfache Initialisierung zu verhindern
2. Druck auf Hersteller ausüben, um Startzeit zu verringern (NVMe, . . .)
3. Spezielle Konfiguration für Arbeitsplatzsysteme, Server, . . .
4. Fokus auf schnelle Startzeiten
   4.1 Integration von Profiling-/Tracing-Werkzeugen in systemd