Inhaltsverzeichnis

1 Einleitung

Mit dem Auftauchen von Multicore-Systemen ist das Bedürfnis von Parallelisierung ebenfalls beim Bootvorgang gewachsen, da Multicore-Systeme aufgrund der sequentiellen Abarbeitung von Diensten nicht bedeutend schneller booteten als Single Core Systeme.

Die Wichtigkeit vom schnellen “Sekunden-Boot” wird insbesondere in eingebetteten Geräten klar: Die Stromzufuhr kann jederzeit vom Anwender vollständig unterbrochen werden und gleichzeitig erwartet der Anwender, dass nach dem Druck auf den Power-Knopf das System binnen weniger Sekunden wieder einsatzbereit ist. Diese Anforderungen sind insbesondere bei zunehmend komplexeren eingebetteten System schwer zu gewährleisten. – systemd versucht dem Rechnung zu tragen.

systemd ist ein Init-Dämon[1] für Linux das von den Red Hat Angestellten Lennart Poettering (Initiator) und Kay Sievers entwickelt und maßgeblich vorangetrieben wird. Es wird als Init-Prozess als erster gestartet (PID 1) und dient zum Überwachen, Starten und Stoppen von weiteren Linux-Prozessen. systemd ist zu den bis dato verbreiteten SysVinit-Skripten kompatibel, bricht aber, anders als sein Vorgänger, die Kompatibilität zu anderen Unix-Betriebssystemen zu Gunsten von Features. Der Init-Dämon verspricht nicht nur Performance, sondern auch Einfachheit bei der Administration. So müssen die explizite Abhängigkeiten zwischen den Systemdiensten in vielen Fällen nicht konfiguriert werden. Auch distributionsspezifische Init-Skripts werden durch systemd obsolet. Die in systemd verwendeten Techniken erlauben es, dass alle Dienste gleichzeitig starten können ohne dass Dienste, die von anderen Diensten abhängen, in einem Fehlerzustand enden. Sowohl Administratoren und Dämon-Entwickler profitieren von der einfacheren Handhabung und letztendlich auch die Endanwender, da der *Start* der Dienste nicht mehr von der Abhängigkeit bestimmt wird.

Da man mit systemd auch Dienste über die Konsole starten, stoppen und überwachen kann, fallen ebenfalls distributionsspezifische Kommandos weg.

systemd wurde von Anfang an auf Erweiterbarkeit entwickelt und so kann es über Module über den Bootvorgang hinaus Dienste bereitstellen, worauf später eingegangen wird.

1 Ein Dämon ist ein Hintergrundprogramm der gewisse Dienste zur Verfügung stellt.

2 Funktionsweise

In diesem Kapitel erörtern wir die verschiedenen Techniken, die es erlauben, den Bootvorgang zu parallelisieren und zu beschleunigen.

Socket Activation

Die Idee der Socket Activation ist nicht neu. Dies wurde auch schon beim alten initd angewandt. Anstatt alle lokalen Internet Dienste direkt beim booten zu starten, horcht ein Superserver im Auftrag des Dienstes und startet eine Instanz des Dienstes wann immer eine eingehende Verbindung aufgebaut wird. Beim Verbindungsabbruch wurde der entsprechende Dienst wieder beendet. Dies erlaubt schwächeren Maschinen mit wenig Ressourcen zu arbeiten. Da jedoch für jede Anfrage eine eigene Instanz kreiert wurde, wurde viel Zeit beim Forken und Initialisieren verbraucht. initd geriet dadurch in Verruf langsam zu sein. Außerdem konnte es nur mit AF_INET (z. B. Internet) Sockets umgehen und nicht mit Unix-Sockets (AF_UNIX), die viele lokalen Dienste verwenden.

systemd greift das Konzept von initd auf, erweitert es jedoch mit AF_UNIX-Unterstützung, die lokale Dienste oft verwenden, so wie Bus Activation auf das später zu sprechen gekommen wird. Dadurch kann das Konzept ebenfalls auf das Booten ausgedehnt werden. Zudem wird nur *eine* Instanz des angefragten Dienstes benutzt, welcher dadurch nur einmal initialisiert werden muss.

An einem Fallbeispiel werden traditionelle Init-Dämons mit systemd verglichen: Die Dienste Syslog, D-Bus, Avahi und Bluetooth sollen gestartet werden. Dabei ist bei den traditionellen SysV basierten Diensten zu beachten, dass D-Bus von Syslog abhängig ist, Avahi von D-Bus und Syslog und Bluetooth ebenfalls von D-Bus und Syslog. Dies hat zur Folge, dass bei SysV basierten Diensten die Bootreihenfolge folgendermaßen aussieht: Syslog → D-Bus → Avahi → Bluetooth (wobei Bluetooth und Avahi in der entgegengesetzten Reihenfolge gebootet werden können, da sie nicht voneinander abhängig sind).

Abbildung 1 visualisiert die Bootreihenfolge der verschiedene Init-Dämonen:

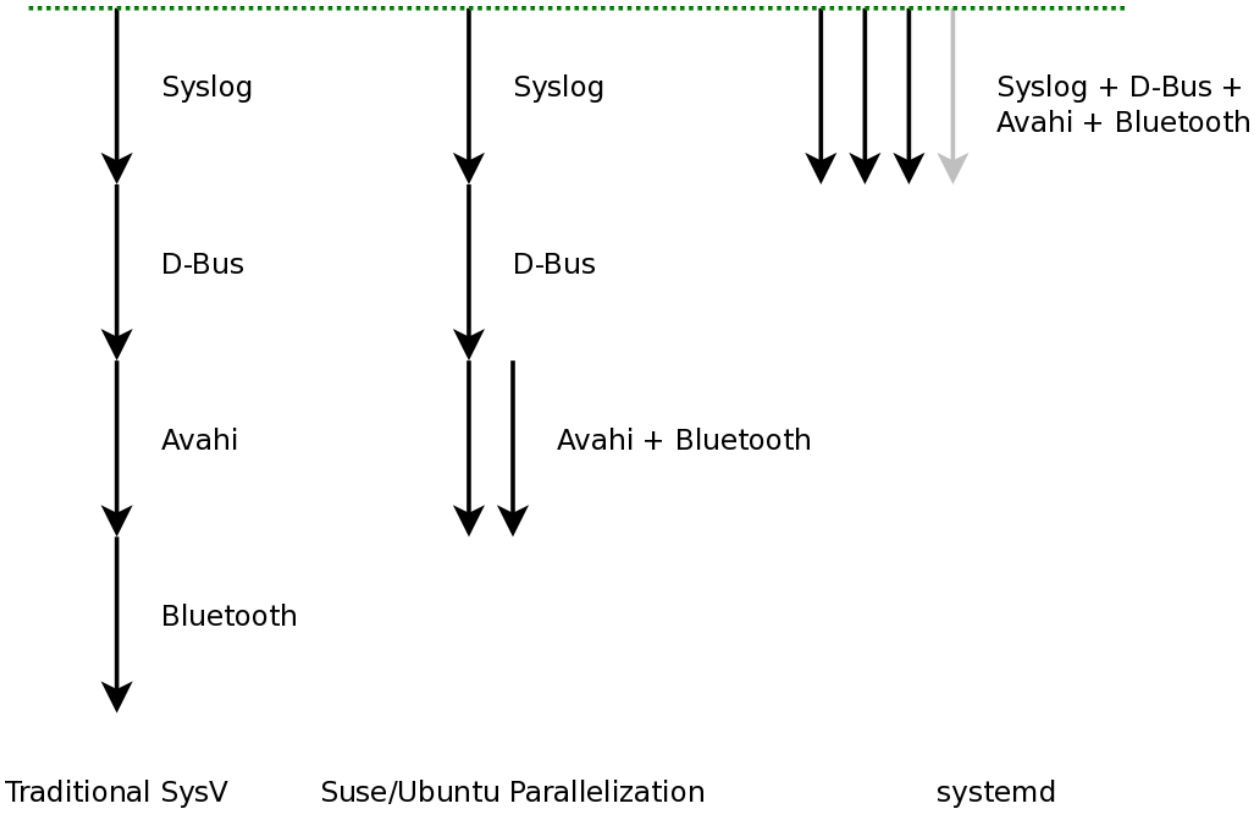

Abbildung 1: Vergleich bisheriger init-Dämonen [2]

Da Avahi und Bluetooth nicht voneinander abhängig sind, können diese parallel gestartet werden, was jedoch nur eine minimalen Bootzeitverkürzung zur Folge hat (siehe „Suse/Ubuntu Parallelization“[2] in Abbildung 1).

Socket Activation macht es möglich mehrere Dienste gleichzeitig zu starten, ohne das Beachten irgendeiner Reihenfolge. Dies wird dadurch möglich gemacht, dass die Sockets nicht mehr von dem Dienst selbst erzeugt werden, sondern außerhalb des Dienstes. Dabei ist es kein Problem, wenn der zuständige Dienst eines Sockets noch nicht gestartet ist, da Sockets immer einen Puffer haben. Ist bspw. Syslog noch nicht gestartet und D-Bus möchte loggen, dann kann er dies machen, in dem er einfach die Logs in /dev/log schreibt. Sobald Syslog gestartet ist, liest er die Logs anschließend aus. Falls der Puffer voll ist, blockiert D-Bus. Ist jedoch wieder freier Speicher zur Verfügung, weil Syslog den Puffer ausgelesen hat, wird der D-Bus wieder aufgeweckt.

Die Synchronisation wird dadurch komplett vom Linux-Kernel gemanagt. Dies erlaubt nicht nur einen hohen Parallesierungsgrad, sondern auch eine vereinfachte Konfiguration der Dienste.

Neben dem hohen Parallesierungsgrad und der vereinfachten Konfiguration, hat Socket Activation noch weitere Vorteile:

- Falls ein Dienst abstürzt, geht keine einzige Nachricht verloren. Stattdessen kann der Dienst dort weiter machen, wo er aufgehört hat.
- Falls ein Dienst aktualisiert werden muss, kann er sich neustarten und die anderen Dienste können weiterhin problemlos in sein Socket pushen. Nach dem Neustart des Dienstes kann er anschließend wieder die Nachrichten entgegen nehmen, um diese zu verarbeiten.
- Austausch der Dienste im fliegenden Wechsel: Dienste, die dieselben Nachrichten verarbeiten können, können jederzeit ausgetauscht werden, wenn beide dieselben Sockets benutzen.

systemd unterstützt folgende Socket ähnliche Technologien:

- AF_UNIX, die klassischen Unix-Sockets.
- AF_INET Sockets, z.B. TCP/IP und UDP/IP
- Unix named pipes/FIFOs.
- AF_NETLINK Sockets.
- Bestimmte virtuelle Dateien wie /proc/kmsg oder Geräteknoten wie /dev/input/*.
- POSIX Message Queues

Ein Dienst, der von Socket Activation Gebrauch machen möchte, muss einen vorinitialisierten Socket von systemd entgegen nehmen, was sich jedoch auf wenige Zeilen Code beschränkt [1].

Um ein Dienst zu aktivieren, benötigt es zwei sogenannte Unit-Dateien. – Eines zur Beschreibung des Dienstes (siehe Abbildung 2) und eines zur Beschreibung des Sockets (siehe Abbildung 3).

```
[Service]
ExecStart=/usr/bin/foobard
```

Abbildung 2: foobar.service

```
[Socket]
ListenStream=/run/foobar.sk

[Install]
WantedBy=sockets.target
```

Abbildung 3: foobar.socket

Mit *systemctl enable foobar.socket* und *systemctl start foobar.socket* kann der Dienst aktiviert

2 Es sei anzumerken, dass Suse mittlerweile selbst systemd benutzt und an der Entwicklung beteiligt ist.

und gestartet werden.

Bus Activation

D-Bus ist ein freies IPC-Framework[3] mit denen Programme miteinander kommunizieren können. Moderne Dienste unterstützten heute zunehmend (zusätzlich) D-Bus welches alle die Synchronisationsmechanismen unterstützt wie sie bei Socket Activation ebenfalls erwähnt wurden. Ein Dienst kann ebenfalls nur auf Nachfrage gestartet werden, falls er selten genutzt wird.

Dateisysteme-Jobs parallelisieren

Nicht nur der Start von Diensten bietet Parallelisierungspotential, sondern auch Dateisystem verwandte Jobs wie mounten, fscking, quota etc.

Vor systemd wurde viel Zeit im IDLE-Zustand verbracht, um darauf zu warten, dass alle Geräte in /etc/fstab aufgelistet wurden. Anschließend wurden sie gemounted, mit fschking geprüft und ggf. das Quota geprüft. Erst danach konnten die Dienste gestartet werden.

systemd startet jedoch ungeachtet davon alle Dienste und erst dann, wenn ein Dienst ein open() auf eine Datei oder ein Dateisystem macht und dieser noch nicht verfügbar ist, wird der entsprechende Dienst blockiert und wieder freigegeben, sobald die Datei bzw. das Dateisystem verfügbar ist. Möglich wird dies mit autofs, das die entsprechenden Mountpunkte vorab erstellt und sobald ein Dienst auf eine Datei zugreifen möchte, wird er vom Kernel in eine Schlange eingereiht. Sobald die angeforderte Datei gemounted und geprüft wurde, werde der Mountpunkt mit der richtigen Datei ersetzt und die entsprechenden Dienste, die darauf zugreifen wollen, geweckt.

Der Zugriff auf / kann natürlich nicht parallelisiert werden, da dort die Dienste selbst liegen, die gestartet werden müssen.

PID 1 klein halten

SysV-Skripte die zuvor beim Booten verwendet wurden, sind ganze normale Shell-Skripte die eine ganze Reihe von Core-Tools wie sed, grep, cat, awk etc. aufriefen, teils sogar für banale Aufgaben. Bei jedem Aufruf wird ein neuer Prozess erzeugt und nach der Abarbeitung stirbt er.

Darüber hinaus ist die Ausführungsgeschwindigkeit von Shell sehr langsam, da die Befehle interpretiert werden müssen und nicht in Maschinencode vorliegen.

systemd erlaubt zwar das Ausführen von Shell-Scripts, Distributoren, die systemd verwenden, sind jedoch nach und nach dazu übergegangen die Dienste in C zu schreiben. Zum Vergleich: Mac OS X, welches Launchd eingeführt und von dessen Ideen sich systemd bedient hat, gibt bei dem Befehl *echo $$* die PID 154 und Linux die PID 1823 aus, d. h. seit dem Systemstart wurden schon 154 bzw. 1823 Prozesse kreiert. Im letzteren Fall kommt man wie oben schon erwähnt durch die Verwendung von Shell-Skripten auf eine so hohe Zahl.

3 IPC steht für Interprocess Communication

3 Weitere systemd-Dienste

systemd hat sich sehr schnell vom Init-System zu einer Ansammlung von Kern-Diensten erweitert, so dass der Hauptentwickler Lennart Poettering mittlerweile von „Core OS“ (siehe Abbildung 4) redet.

Einige dieser Dienste, die den Kern eines jeden Betriebssystems ausmachen, wollen wir hier vorstellen.

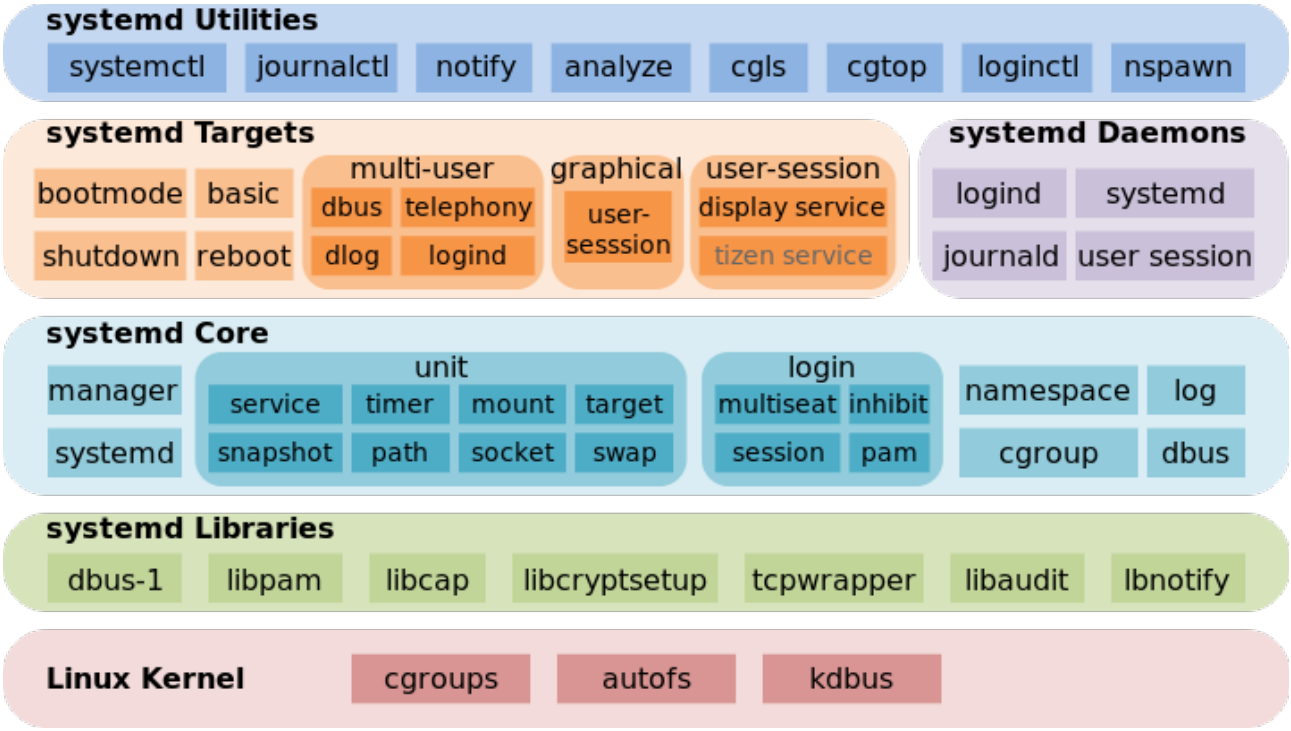

Abbildung 4: systemd-Komponenten [6]

journald

journald ist ein Dienst von systemd bietet viele Vorteile, die die alten log-Dienste wie rsyslog nicht oder nur eingeschränkt bieten. Um nur einige aufzuzählen:

- POSIX-ACL: Welcher Nutzer bzw. welche Gruppe darf mit welcher Berechtigung auf welche Log-Datei zugreifen?
- Zugriff auf die Logs mit O(log(n)), statt O(n).
- Deduplizierung: Z. B. Wird Hostname systemweit nur einmal gespeichert. Andere Einträge können darauf referenzieren.
- Automatische Komprimierung der Daten.
- Binäre BLOBs können abgespeichert werden.
- Prozesse die loggen, werden automatisch authentifiziert (bei alten Loggern war es möglich, dass sich z. B. jeder als „apache“ ausgeben kann).
- Die Festplattengröße wird beim Loggen beachtet, so dass die Festplatte unter keinen Umständen durch das Loggen voll läuft.
- Skaliert vom eingebettetem System bis zum Supercomputern und Clustern.

Grundlagen

Um die letzten 10 Log-Einträge anzuzeigen, hat man zuvor *tail -f /var/log/messages* benutzt. Das Äquivalent dazu ist *journalctl -f*. Echtzeit-Überwachung ist bei beiden Varianten möglich.

Die Möglichkeiten mit journalctl gehen jedoch weit über bisherige Möglichkeiten hinaus, die man zuvor nicht, oder nur mit größerem Aufwand umsetzen konnte:

- Alle Logs seit dem Bootvorgang: *journalctl -b*
- Alle Logs seit gestern: *journalctl –since=yesterday*
- Alle Logs von httpd zwischen 0:00 und 9:30 Uhr:
  journalctl -u httpd --since=00:00 –until=9:30
- Alle Logs /dev/sdc betreffend: *journalctl /dev/sdc*
- Alle Logs vpnc und dhclient betreffend: *journalctl /usr/sbin/vpnc /usr/sbin/dhclient*
- Alle Logs mit der PID 1: *journalctl _PID=1*

„Minimal set of components to build an OS“. „The foundation to build real-world system (not just the absolute minimum that boots)“. [5]

Zero Maintability: Checkt, ob die Festplatte genügend Platz hat, bevor es loggt. Kann gecrashte Dienste wieder starten (gab es bei SysV-Skripts nicht!).

Watchdog-Support. Wenn apache nicht antwortet → Neustarten. [5]

Soll 97% der Anwendungsfälle abdecken. Vom Server bis zum eingebetteten System, wobei mit eingebetteten Systemen eher die Hardware-Klasse in Smartphones gemeint sind. Bei „richtigen eingebetteten Systemen“ empfiehlt Lennart Poettering busybox. [5]

Session Management: Verschiedene User sollen über cgroups ungefähr gleich viel CPU bekommen. [5]

4 Quellen

[1] http://0pointer.de/blog/projects/socket-activation.html
[2] http://0pointer.de/public/parallelization.png
[3] systemd journal, https://docs.google.com/document/pub?id=1IC9yOXj7j6cdLLxWEBAGRL6wl97tFxgjLUEHIX3MSTs
[4] „systemd for Administrators, Part XVII“, http://0pointer.de/blog/projects/journalctl.html
[5] foss.in – „systemd as the Core OS“, http://www.youtube.com/watch?v=_2aa34Uzr3c
[6] http://de.wikipedia.org/wiki/Datei:Systemd_components.svg