HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES
Fakultät Informatik
Systemsoftware

systemd als Init-Dämon

Vom Server bis zum eingebetteten System

Stefano Di Martino & Jakub Werner

Inhaltsverzeichnis

1 Einleitung....3

2 Funktionsweise....4

2.1 Socket Activation....4
2.2 Bus Activation....6
2.3 Dateisysteme-Jobs parallelisieren....6
2.4 PID 1 klein halten....6
2.5 Überwachung der Dienste....6

3 Weitere systemd-Dienste....8

3.1 journald....8
3.2 WatchDog....9
3.3 logind....9

4 Init-Dämons im Vergleich....11

4.1 SysV-Init....11
4.2 Auf Hibernate und Resume basiertes Booten....11
4.3 Upstart....11
4.4 Einsatz....12

5 Zusammenfassung und Ausblick....13

6 Verständnisfragen....14

6.1 Erklären Sie das Konzept von Socket Activation....14
6.2 Warum sollten beim Bootvorgang keine Shell-Skripte verwendet werden?....14
6.3 Wie macht sich systemd cgroups zu Nutze?....14

Quellenverzeichnis....15

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 wurde anfänglich als Init-Dämon[1] für Linux von den Red Hat Angestellten Lennart Poettering (Initiator) und Kay Sievers entwickelt und maßgeblich vorangetrieben. Er 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 SysV-Init-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. Dadurch hat sich systemd sehr schnell über die Fähigkeiten eines klassischen Init-Dämons hinaus entwickelt. Dadurch, dass die von systemd (optionalen) bereitgestellten Dienste auf systemd optimiert sind, wird eine Zuverlässigkeit und Qualität erreicht, wie man sie zuvor auf Unix noch nicht gesehen hat. systemd verleiht Linux Mächtigkeit und Funktionalitäten, die bis vor kurzem nur Mikrokerneln vorbehalten waren, wie das Überwachen und Neustarten von Diensten, falls diese sich aufgehängt haben und nicht mehr reagieren (Watchdog).

systemd ist noch ein junges und sehr schnell wachsendes Projekt, das ähnlich wie Linux, zu deutlich mehr wurde als es ursprünglich intendiert war. Die hier vorgestellten Funktionalitäten decken deshalb nur einen kleinen Teil der Mächtigkeit systemds ab. Dennoch hoffen wir, dass wir Interessierten einen guten Einstieg in systemd bieten können.

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 und wie Dienste von systemd überwacht und gesteuert werden.

2.1 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.[1] 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 [1] visualisiert die Bootreihenfolge der verschiedene Init-Dämonen:

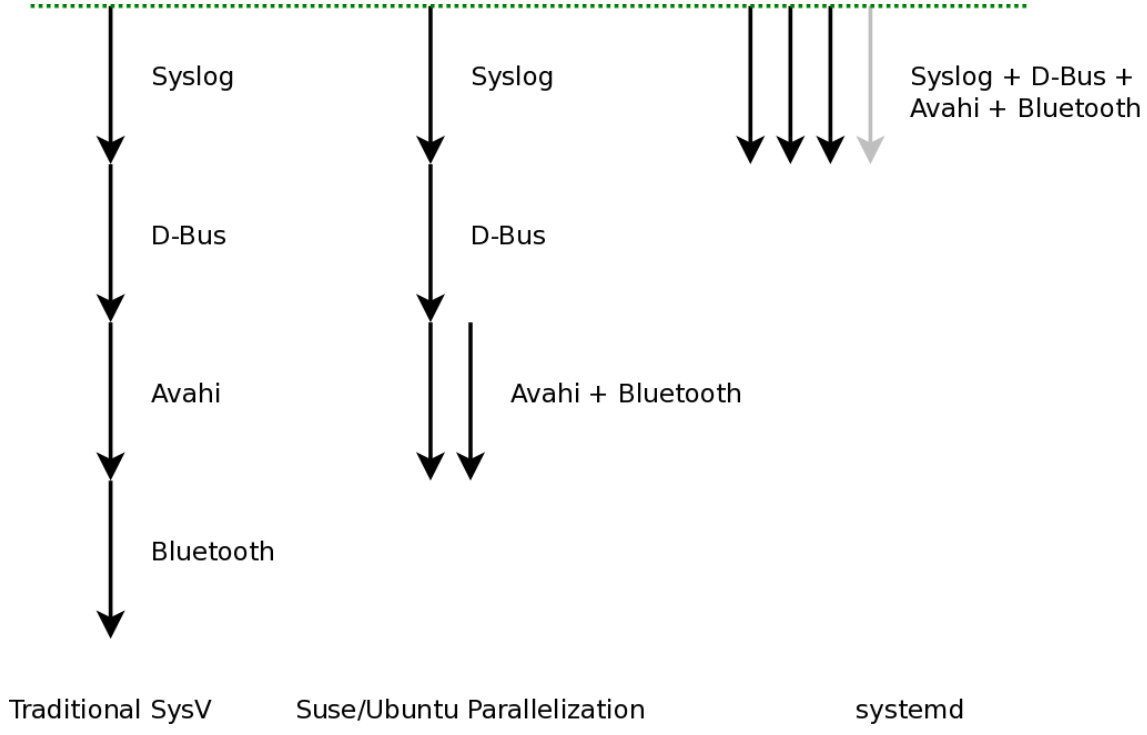

Abbildung 1: Vergleich bisheriger init-Dämonen

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 [1]) und eines zur Beschreibung des Sockets (siehe Abbildung 3 [1]).

```
[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 und gestartet werden.

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

2.2 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.

Kurz: Durch Socket und D-Bus basierte Dienste, kann man somit alle Dienste parallelisieren.

2.3 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. [1] 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, wird 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.

2.4 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 mit SysV-Init die PID 1823 aus, d. h. seit dem Systemstart wurden schon 154 bzw. 1823 Prozesse kreiert. [1] Im letzteren Fall kommt man wie oben schon erwähnt durch die Verwendung von Shell-Skripten auf eine so hohe Zahl.

2.5 Überwachung der Dienste

Ein zentraler Punkt eines Init-Dienstes ist ebenfalls die Überwachung der Dienste, was bis dato mit SysV-Init nicht zuverlässig funktioniert hat (siehe „Init-Dämons im Vergleich“).

Falls ein Dienst immer laufen sollte und er sich beendet, sollte er neugestartet werden. Falls er abstürzt sollten entsprechende Informationen ggf. aus verschiedenen Quellen gesammelt und zusammengeführt werden. Z. B. sollten die Informationen des Crash-Dump-Dienstes abrt und des Log-Dienstes syslog für den Administrator zentral zugänglich gemacht werden. – All dies unterstützt systemd.

Ebenfalls sollte der Init-Dämon Dienste komplett herunterfahren können, was nicht einfach ist, wenn ein Dienst einen fork von sich gemacht hat. Z. B. könnte ein ausgeartetes CGI-Skript sich selbst forken und es würde nicht heruntergefahren werden, wenn Apache heruntergefahren wird. [1]

3 IPC steht für Interprocess Communication

Um die Beziehungen zwischen den Diensten zu gruppieren, macht sich systemd cgroups (Control Groups) zu nutze. Ursprünglich wurde dieses Feature im Linux-Kernel aufgenommen, um Ressourcen für eine Gruppe von Prozessen zu limitieren. Allerdings hat es den nützlichen Nebeneffekt, dass alle forks eines Prozesses in derselben Gruppe landen, deren Hierarchie im virtuellem Dateisystem (VFS) als Ordner und Unterordner aufgelistet wird. Die Beziehung zwischen den Prozessen wird so für systemd sichtbar.

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 [2]) spricht, das 97% der Anwendungsfälle abdecken soll: Vom Server bis zum eingebetteten System, wobei mit eingebetteten Systemen eher die Hardware-Klasse der Smartphones gemeint ist. Bei „richtigen eingebetteten Systemen“ empfiehlt Lennart Poettering busybox. [3]

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

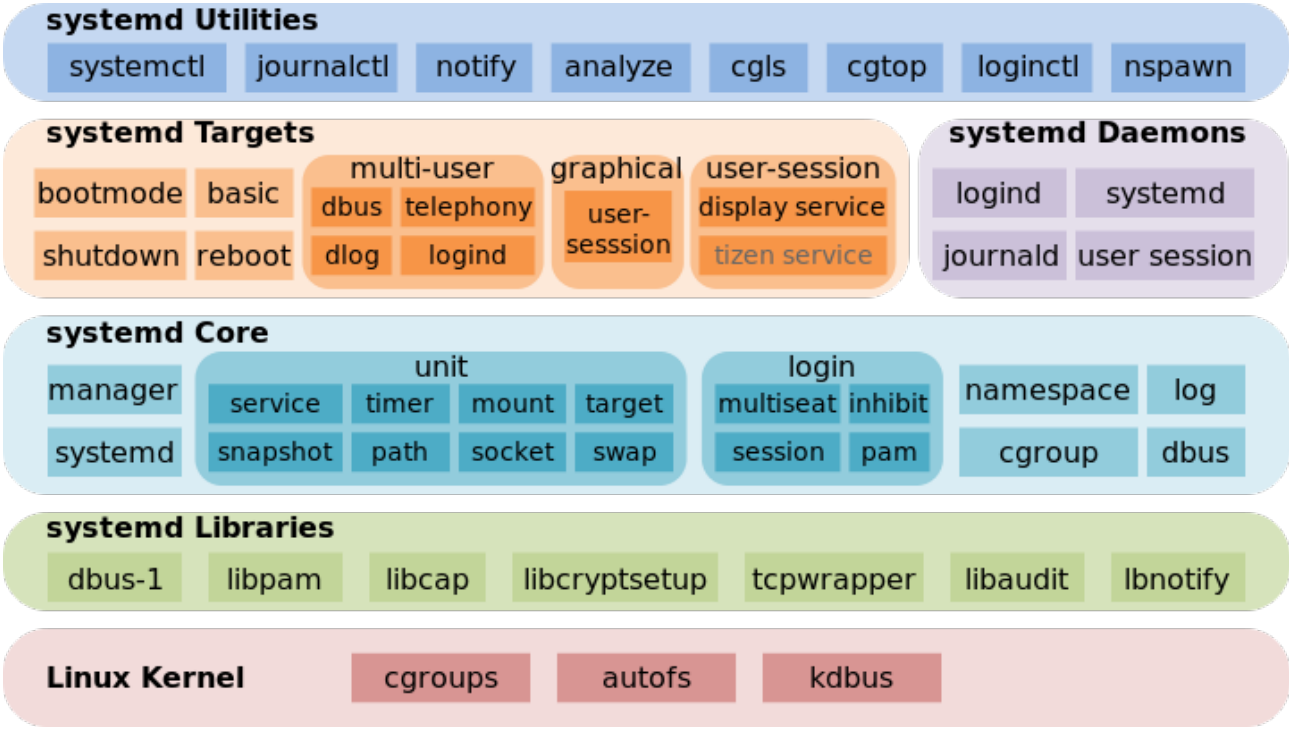

Abbildung 4: systemd-Komponenten

3.1 journald

journald ist ein Dienst von systemd und bietet viele Vorteile, die die alten log-Dienste wie rsyslog nicht oder nur eingeschränkt bieten. Um nur einige aufzuzählen[4]:

- 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 konnte. systemd vermeidet dies durch Verwendung von Metadaten wie _EXE, die die ausführbare Datei angibt und von Anwendungen nicht selbst gesetzt werden können.
- Alle Logs werden, ähnlich wie bei git, gehasht. Nachträgliche Manipulationen des Logs sind

somit nicht möglich wie das bei alten Log-Dateien, mit entsprechenden Rechten, die man sich evtl. gehackt hat, der Fall war.
- 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 oder dhclient betreffend: *journalctl /usr/sbin/vpnc /usr/sbin/dhclient*
- Alle Logs mit der PID 1: *journalctl _PID=1*

3.2 WatchDog

systemd versucht drei Bereichen gerecht zu werden: Der Embedded-, Desktop- und Server-Welt. Trotz der gewaltigen Unterschiede in der zur Verfügung stehenden Ressourcen, haben die Embedded- und die Server-Welt eine große gemeinsame Anforderung: Watchdog. Watchdog ist in eingebetteten Geräten als Hardware integriert. Wenn eine Software innerhalb einer bestimmten Zeit nicht reagiert, wird sie einfach neu gestartet, um die Verfügbarkeit und Zuverlässigkeit zu erhöhen. Bei Servern, die Dienste zur Verfügung stellen, spielte Verfügbarkeit und Zuverlässigkeit ebenso eine große Rolle (bei Desktop-Systemen weniger).

systemd unterstützt seit Version 183 sowohl Hardware- als auch Supervisor-Watchdog (Software) für einzelne Dienste. [5] Falls diese Funktionalität aktiviert ist, wird systemd regelmäßig die Watchdog-Hardware anpingen. Falls sich systemd oder der Linux-Kernel aufgehängt haben, wird der Ping ausgesetzt und das System von Watchdog neugestartet. Darüber hinaus stellt systemd ein Interface für Software-Watchdog für einzelne Dienste zur Verfügung, so dass diese ebenfalls von systemd neugestartet werden können (oder andere denkbare, konfigurierbare Aktionen), falls diese sich aufhängen.

Nimmt man alles zusammen (Hardware-Watchdog, der systemd und Linux überwacht und Software-Watchdog, der einzelne Dienste überwacht), hat man ein hoch zuverlässiges System.

3.3 logind

logind ist unter anderem für das Session- und Seat-Management verantwortlich. Multiseats und Multisessions werden von logind von Haus aus unterstützt. D. h., dass an einem Computer mehrere Nutzer mit eigenem Bildschirm, Tastatur und Maus arbeiten können, was man bspw. von Rechenzentren an Hochschulen kennt. Was jedoch nicht unüblich ist, ist dass ein einzelner Nutzer das System für alle anderen unbrauchbar machen kann, in dem es gewollt oder ungewollt, das System auf das Maximum auslastet. Da systemd cgroups verwendet, kann man die Ressourcen pro User ohne weiteres limitieren, so dass ein einzelner User nicht mehr das komplette System für andere Nutzer ausbremsen kann. [3]

logind beinhaltet mehrere Komponenten in sich und diese weiter auszuführen würde den Rah-

men dieser Ausarbeitung sprengen. Dennoch wollten wir kurz auf das Session-Management mit logind und cgroups zu sprechen kommen.

4 Init-Dämons im Vergleich

4.1 SysV-Init

Das Konzept von SysV-Init stammt ursprünglich vom Unix System V ab, daher der Name. Der Dienst startet als erster Prozess beim Booten (PID 1) und startet nacheinander andere Dienste, wobei hier die Abhängigkeiten berücksichtigt werden müssen. Ist beim Booten eine Festplatte angeschlossen, wird diese erst initialisiert, wobei andere Dienste erst mal warten müssen. Tritt dabei ein Fehler auf, wird der komplette Bootvorgang unterbrochen. [6]

SysV-Init selbst und auch die Dienste, die er startet, sind i. d. R. Shell-Skripte. Wie das System initialisiert wird, wird in /etc/inittab festgelegt. Hier unterscheidet man zwischen verschiedenen Run-Levels, die bspw. Dinge wie Einbenutzerbetrieb, Mehrbenutzerbetrieb, Netzwerkunterstützung etc. festlegen. Je nach dem, welcher Run-Level konfiguriert ist, wird der entsprechende Run-Level-Ordner ausgeführt. [7] In der inittab kann ebenfalls festgelegt werden, ob ein Dienst bei einem Absturz oder einer Beendigung neugestartet werden soll. Der entsprechende Dienst hinterlegt dabei meist in /run/var seine PID und SysV-Init versucht die Dienste entsprechend zuzuordnen. Allerdings können sich Dienste der Kontrolle schnell entziehen, in dem sie sich forken und dadurch eine neue PID zugewiesen bekommen. Mit SysV-Init können auch nicht Gruppen von zusammengehörigen Diensten wie z. B. Apache, MySQL die Ressourcen limitiert werden, da SysV-Init keinen Bezug zu Kindprozessen herstellen kann.

4.2 Auf Hibernate und Resume basiertes Booten

Chinesische Software-Ingenieure der „Chinese Academy of Sciences Beijing“ haben einen auf Hibernate und Resume basierte Bootmethode entwickelt, der mit dem Schlafenlegen und Aufwecken von PCs verwandt ist. Die Bootzeit hat sich dadurch von 14,47 Sekunden auf 12 Sekunden verkürzt, was ein Performancegewinn von 17% ist. [8] Die Testumgebung war ein MSTAR 6198 TV Board. Dies soll hier nur am Rande erwähnt werden, da richtige Init-Dienste sehr viel mehr Mächtigkeit bieten als ein simples Einfrieren und Laden des Systemzustandes (zur Erinnerung: Selbst Sys-V-Init konnte rudimentär Dienste überwachen). Darüber hinaus scheint es sich eher um unfertige Forschungsarbeit zu handeln, deren Produkt für die Allgemeinheit nicht in Form von Quellcode zur Verfügung steht wie das bei anderen init-Diensten der Fall ist.

4.3 Upstart

Upstart ist ein eventbasierter Init-Dienst der von Canonical, die Firma hinter Ubuntu, noch vor systemd erfunden. Er wurde, ähnlich wie systemd, entwickelt, um den Bootvorgang durch Parallelisierung zu beschleunigen.

Es gibt (mindestens) zwei Wege, einen Dienst bereit zu stellen, sobald er gebraucht wird [9]:

1. Beobachte alle anderen Dienste, die ihn möglicherweise in Anspruch nehmen und achte beim Starten auf die richtige Reihenfolge.
2. Warte einfach, bis jemand den Dienst in Anspruch nehmen möchte und starte ihn auf Nachfrage.

SysV-Init und auch Upstart benutzen die erste Methode, systemd die zweite. D. h., dass bei Upstart bei der Administration der Dienste ebenfalls ihre Abhängigkeiten konfiguriert werden müssen und darüber hinaus, dass Dienste, die von anderen Diensten abhängig sind, gestartet werden, sobald der Dienst, von dem sie abhängen, gestartet wurde, ganz gleich, ob sie gebraucht werden oder nicht. Konkret heißt das, dass wenn D-Bus gestartet wird, auch NetworkManager – der von D-Bus abhängt – gestartet wird. [1] systemd geht genau umgekehrt vor. Wenn NetworkManager startet und

D-Bus beanspruchen will, wird dieser gestartet. Upstart startet hingegen vorsichtshalber alle Dienste im Voraus, die *möglicherweise* gebraucht werden. Ein weiteres Beispiel, diesmal aber auf Events bezogen: Der Druckdienst Cups startet nach dem Event „Dateisystem eingebunden“. Ob er nun gebraucht wird oder nicht, spielt keine Rolle.[6]

Upstart ermöglicht nur dann Parallelität, wenn Events und Dienste voneinander unabhängig sind. Bspw. können alle Dienste, die vom Event „Netzwerk aktiviert“ unabhängig sind, parallel starten. Allerdings gilt das nicht unbedingt für die Netzwerkdienste selbst. Hier müssen Abhängigkeiten definiert werden, was dann oft wieder zu einer sequentiellen Abarbeitung führt.

4.4 Einsatz

Bei der Diskussion in der Debian Mailingliste, ob sie in Zukunft upstart oder systemd als Default-Init-System wählen sollen, hat sich der Musik-Streaming-Dienst Spotify in einem Bugreport eingeschaltet [10]. Sie würden Debian GNU/Linux mit systemd auf 5000 physischen Servern verwenden und sind überzeugt, dass das Abhängigkeitsmodell von systemd einfacher zu verstehen, zu erklären und anwendbar sei als bei upstart. Außerdem helfen die zahlreichen Features die von cgroups Gebrauch machen, in einem hoch skalierbarem System wie der ihren. Darüber hinaus glauben sie, dass man mit der systemd-Community für zukünftige Technologien besser zusammenarbeiten kann.

Tatsächlich gibt es keine Distribution außer Ubuntu und seine Derivate die (noch) upstart einsetzen. Alle sind entweder auf systemd gewechselt (und das sind die meisten) oder verwenden noch den klassischen SysV-Init-Dämon (dazu gehört Debian noch dazu).

Auch in eingebetteten System wird systemd verwendet. So verwendet das Smartphone Jolla, das schon seit einigen Monaten verkauft wird, systemd. [11]

Auch das künftige Smartphone-OS Tizen verwendet systemd und soll den Bootvorgang bis zum Homescreen in weniger als 7 Sekunden schaffen. [12]

5 Zusammenfassung und Ausblick

systemd wird schon in fast allen Linux-Distributionen eingesetzt, obwohl das Projekt jünger als upstart ist und schon einige Distributionen wie SUSE und Fedora upstart im Einsatz hatten, was für die Qualität systemds spricht. Es wurde anfänglich als Init-Dämon konzipiert das die Synchronisation zwischen den Diensten über Sockets und D-Bus vollzieht – beide etablierte und getestete Technologien. systemd muss sich selbst nicht um die Synchronisation zwischen den Diensten kümmern. Im Falle von Socket Activation wird das vom Kernel selbst erledigt und im Falle von Bus-Activation von D-Bus.

Schnell hat sich systemd zu mehr als ein Init-Dämon entwickelt und durch seine Anforderungen hat es schon manche Entwicklungen im Kernel angestoßen, wie z. B. das Redesign von cgroups [13]. Ferner wird bald ein D-Bus-Äquivalent im Linux-Kernel (kdbus) zu finden sein. Die Arbeiten sind auch dank dem Einsatz der systemd-Entwickler Lennart Poettering und Kay Sievers demnächst abgeschlossen und die Unterstützung in systemd schon integriert. [14] Der Einsatz von kdbus verspricht vor allem Performancevorteile in eingebetteten Systemen, da weniger Kontextwechsel nötig sind und Technologien wie Zero-Copy unterstützt werden, was den Einsatz systemds in eingebetteten Systemen nochmals zu Gute kommt.

Es gibt viele Parallelen zur Linux-Kernel-Entwicklung: So ist systemd vielseitig einsetzbar, vom Server bis zum eingebetteten System, wobei bei letzterem eher Smartphone-Klassen gemeint sind. Es arbeiten mittlerweile viele Entwickler an systemd mit und steuern einzelne Komponenten bei, die systemd erweitern. Dadurch wird die Gewichtigkeit und die Bedeutung systemds zunehmen, so dass es im Produktiveinsatz kaum noch wegzudenken sein wird.

6 Verständnisfragen

6.1 Erklären Sie das Konzept von Socket Activation

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 von systemd. Dabei ist es kein Problem, wenn der zuständige Dienst eines Sockets noch nicht gestartet ist, da Sockets immer einen Puffer haben. Sobald der zuständige Dienst gestartet ist, übergibt systemd ihm den entsprechenden Socket der ggf. schon Daten von anderen Diensten enthält. Falls der Fall eintritt, dass der Puffer eines Sockets voll ist, wird der Dienst, der in den Socket schreibt, einfach solange blockiert, bis er wieder schreiben darf.

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

6.2 Warum sollten beim Bootvorgang keine Shell-Skripte verwendet werden?

Der Administrator muss darauf achten, dass Dienste, die voneinander abhängig sind, in der richtigen Reihenfolge aufgerufen werden. Fällt dabei ein Dienst aus, könnte dies im schlimmsten Fall den kompletten Bootvorgang lahm legen. Da die Dienste außerdem sequentiell aufgerufen werden, verlangsamt dies den Bootvorgang, was noch dadurch begünstigt wird, dass Shell-Skripte externe Tools aufrufen und jeder Aufruf einen Prozess kreiert und danach wieder terminiert wird, was sehr zeitaufwendig ist.

6.3 Wie macht sich systemd cgroups zu Nutze?

cgroups ist ein Kernel-Feature und erlaubt das Gruppieren von Prozessen. Mit systemd kann man damit bestimmten Dienstgruppen die Ressourcen limitieren (z. B. jeder angemeldete User darf die CPU maximimal zu 30% auslasten). Außerdem kann systemd damit Dienste besser überwachen, da alle Kindprozesse eines Elternprozesses in der selben Prozessgruppe landen, was systemd bspw. ermöglicht die entsprechenden Dienste rückstandsfrei zu terminieren.

Quellenverzeichnis

[1] Lennart Poettering, Rethinking PID 1, 30.04.2010, http://0pointer.de/blog/projects/systemd.html

[2] ScotXW, Wikipedia: Systemd Components, 29.09.2013, http://de.wikipedia.org/wiki/Datei:Systemd_components.svg

[3] Lennart Poettering, foss.in - systemd as the Core OS, 2012, http://www.youtube.com/watch?v=_2aa34Uzr3c

[4] Lennart Poettering, systemd journal, 2011, https://docs.google.com/document/pub?id=1IC9yOXj7j6cdLLxWEBAGRL6wl97tFxgjLUEHIX3MSTs

[5] Lennart Poettering, systemd for Administrators – Part XV, 28.06.2012, http://0pointer.de/blog/projects/watchdog.html

[6] Tim Schürmann, Fliegender Start – Upstart und Systemd im Vergleich, Januar 2011, http://www.linux-community.de/Internal/Artikel/Print-Artikel/LinuxUser/2011/01/Upstart-und-Systemd-im-Vergleich

[7] Mirko Dölle, Schneller booten mit Upstart, 29.10.2009, www.heise.de/open/artikel/Schneller-booten-mit-Upstart-844394.html

[8] Peng Chen, Quansheng Zhang, Shuzhi Wang, Xiaofeng Dong, The Research of Fastboot of Embedded Linux based on State Keepingand Resuming, 16.09.2011

[9] Jonathan Corbet, The road forward for systemd, 26.05.2010, http://lwn.net/Articles/389149/

[10] Noa Resare, tech-ctte: Decide which init system to default to in Debian, 17.01.2014, http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=727708#3546

[11] Jake Edge, Jolla: Sailfish OS, Qt, and open source, 31.07.2013, http://lwn.net/Articles/561463/

[12] Mikko Ylinen, TizenIVI Architecture, 07.05.2012, http://download.tizen.org/misc/media/conference2012/wednesday/bayview/2012-05-09-0945-1025-tizen_ivi_architecture.pdf

[13] Libby Clark, All About the Linux Kernel: Cgroup's Redesign , 15.08.2013, http://www.linux.com/news/featured-blogs/200-libby-clark/733595-all-about-the-linux-kernel-cgroups-redesign

[14] Jonathan Corbet, The unveiling of kdbus, 13.01.2014, https://lwn.net/Articles/580194/