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

Socket Activation [1]

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.

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:

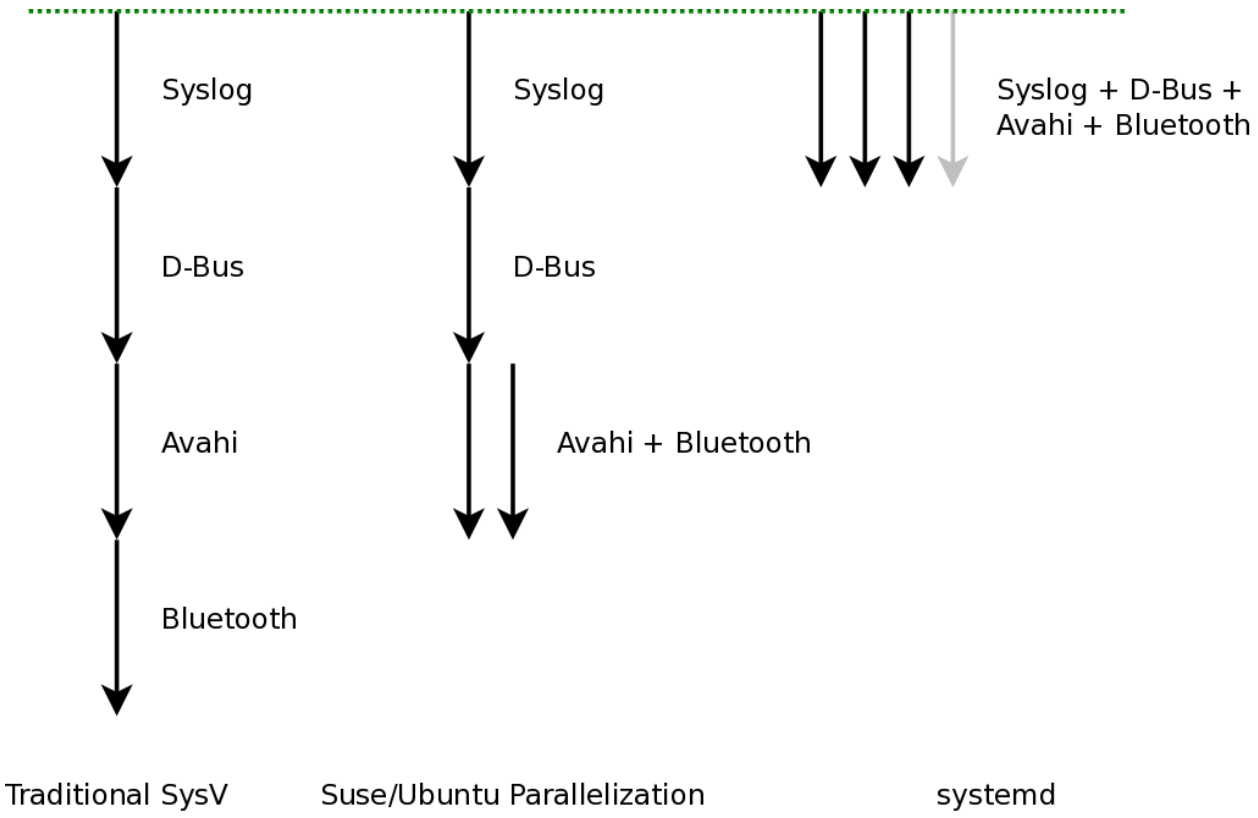

Abbildung 1: Vergleich bisheriger init-Dämonen [2]

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.
-

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

3 Quellen

[1] http://0pointer.de/blog/projects/socket-activation.html
[2] http://0pointer.de/public/parallelization.png