Alternative Init-Systeme für Linux

Startbereit

Michael Plura

Die Schlacht um PID 1 hat *systemd* gewonnen, SysVInit liegt in Trümmern – doch es gibt erstaunlich leistungsfähige Init-Alternativen, die nicht mit bewährten Prinzipien brechen.

PCs auf x86-Basis starten üblicherweise, indem das BIOS (über den Umweg eines Bootloader) oder UEFI den Betriebssystem-Kernel lädt. Der Kernel initialisiert die Hardware und stellt Systemfunktionen bereit, um schließlich bei Unix-Systemen */sbin/init* aufzurufen. Diesem Init-Prozess kommt die zentrale Rolle zu, den Rest des Systems zu konfigurieren und alle benötigten Dienste zu starten. *init* erhält daher die Prozess-ID (PID) 1.

Im einfachsten Fall könnte *init* das einzige Programm sein, bei vielen Unix-Systemen kommen mehr oder weniger übersichtliche Shell-Skripte (SysV, BSD) oder komplexe binäre Frameworks hinzu, die durch deklarative Konfigurationsdateien mit spezieller Syntax gesteuert werden (neben *systemd* etwa Upstart, *launchd* bei Apples macOS, Service Management Facility ab Solaris 10).

Systemstart schlicht und übersichtlich halten

Beim früher unter Linux verwendeten SysVInit definiert die Datei */etc/inittab* die zu startenden Prozesse und Dienste. Eine Zeile enthält dort durch Doppelpunkte getrennt eine eindeutige ID, die Runlevels, in denen der Prozess laufen soll, dessen Lebensdauer und den eigentlichen Startbefehl samt Parametern. Zusätzliche Dienste lassen sich über ein Skript in */etc/init.d/* ins System einfügen, das Parameter wie *start*, *stop*, *status* oder *reload* auswertet.

Die Linux Standard Base (LSB) definiert Vorgaben für Init-Skripte, damit diese möglichst unabhängig von einer Distribution funktionieren. Über LSB-Header in den Skripten und Werkzeugen wie *insserv* (Debian, SUSE) oder *chkconfig* (Red Hat) lässt sich die Administration vereinfachen. Wer Dienste parallel starten will (bei neueren Init-Systemen wie *systemd* üblich), kann in der */etc/init.d/rc* an die Zeile „startup boot“ ein *&* anhängen.

Canonical versuchte, mit Ubuntu 9.10 das ereignisgesteuerte Init-System Upstart durchzusetzen, das automatisch und parallel *.conf*-Dateien aus */etc/init/* verarbeitet – ist damit aber gescheitert. Mittlerweile starten alle großen Distributionen mit *systemd*. Die Verflechtung von Red Hats *systemd* mit anderen Systemkomponenten wie GNOME 3 erlaubt es jedoch kaum noch, Mainstream-Distributionen mit einem frei gewählten Init-System zu bestücken. Davon gibt es etliche – wenig beachtet werden etwa OpenRC und runit. Beides sind moderne, einfache, etablierte und vor allem gegenüber der *systemd*-Architektur geradezu schlichte Alternativen für den Systemstart.

OpenRC ist ein abhängigkeitsbasiertes Init-System, das auf dem bekannten *init* aufsetzt, dabei POSIX-konform und somit zu SysVInit kompatibel ist und sogar auf diverse BSD-Varianten portiert wurde. OpenRC wird klassisch über Skripte konfiguriert, wertet aber auch komplexe Konfigurationsdateien in */etc/conf.d/* samt deren Abhängigkeiten aus. Es kann beispielsweise den SSH-Daemon im Runlevel „default“ nur auf *eth0* starten, im frei definierten Runlevel „office“ jedoch auf *wlan0*. Über Hardware-Events (Hotplugging) lassen sich Aktionen auslösen, etwa automatisches Netzwerk-Tethering beim Verbinden eines Smartphones. Einen Sicherheitsgewinn bietet die Möglichkeit, Prozesse in *chroot*-Umgebungen zu starten und über Control Groups (cgroups) den Zugriff auf Systemressourcen zu reglementieren. Optional kann OpenRC zum Beschleunigen des Startprozesses Dienste parallel starten und deren Funktion als „Supervise-Daemon“ überwachen.

Während sich OpenRC eher als Erweiterung zu *init* sieht, ist runit von Gerrit Pape mehr ein stark erweiterter *init*-Ersatz. Auch ist Letzteres unter dem Gesichtspunkt der Portabilität zu anderen freien Betriebssystemen konzipiert worden und läuft daher unter Linux, den BSDs, macOS und Solaris. runit beerbt die daemontools von Daniel J. Bernstein, die bereits vor 15 Jahren als richtungweisendes Prozess-Supervisor-Toolkit galten.

Im Kern besteht runit aus drei Stufen (Stages): */etc/runit/1* fährt das System hoch, */etc/runit/2* läuft in einer Schleife und */etc/runit/3* schließlich fährt das System herunter. Die zweite Stufe ruft über *runsvdir* alle Dienste auf, die jeweils einzeln als Skript in Verzeichnissen unter */etc/sv/* definiert werden. Ein symbolischer Link in */service/*, das runit beim Booten und alle paar Sekunden scannt, macht einen Dienst verfügbar. Normale Benutzer können über ein *~/service/*-Verzeichnis eigene Dienste administrieren.

OpenRC, runit und S6 als Alternativen

Reichen OpenRC und/oder runit nicht aus, bietet S6 einen prall gefüllten Werkzeugkasten für Low-Level-Prozess-Management und Serviceadministration. Dazu gehören ein verbessertes *syslogd*, umfangreiches Dienste-Monitoring als Grundlage für stabiles Abhängigkeitsmanagement und eine eigene Skriptsprache (execline) bis hin zu der nicht unumstrittenen Socket-Aktivierung, die bei *systemd* gerne herausgestellt wird. Auch S6 bleibt dabei POSIX-kompatibel und wird, da konsequent modular aus kleinen, auf genau eine Aufgabe spezialisierten Werkzeugen konzipiert, auch gerne beispielsweise mit *busybox* und in Docker-Containern eingesetzt.

Wer diese Init-Alternativen ausprobieren will, sollte dafür Distributionen ohne Abhängigkeiten wie zwischen *systemd* und GNOME 3 wählen. Dazu zählen etwa das technisch orientierte Gentoo Linux (systemd, OpenRC und runit), das kompakte und auf BusyBox aufbauende Alpine Linux (OpenRC) oder das schnelle Void Linux (runit). (tiw)

Michael Plura

lebt in Schweden und ist freier Autor mit den Schwerpunkten IT-Sicherheit, Virtualisierung und freie Betriebssysteme.

Alle Links: www.ix.de/ix1706108