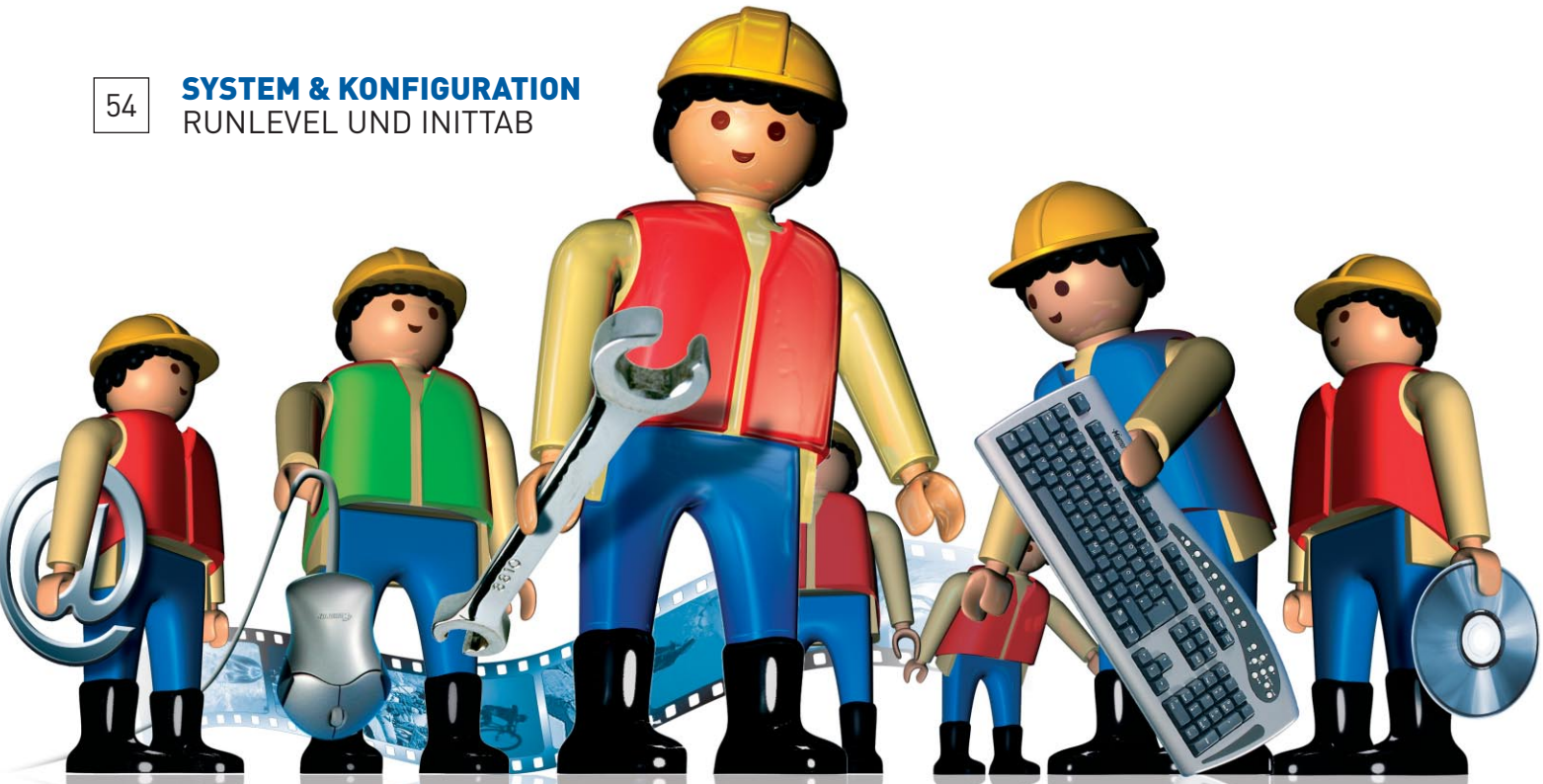


DER BOOTVORGANG UNTER LINUX

START ME UP!

Ist gleich zu Beginn des Linux-Boots der Kernel geladen, ist „Prozess Nr. 1“ an der Reihe: *System V Init*, welches mit einer Menge Shell-Skripte Dienste startet – zumindest bei den meisten Distributionen. Wir zeigen die Funktionsweise und stellen moderne Alternativen vor.

VON **MATTIAS SCHLENKER**

Eine der ältesten Komponenten von Linux und anderen Unix-Systemen ist das *Init* – eine Routine des Betriebssystemstarts, die all das bestimmt, was passiert, wenn der Kernel geladen ist. Ihre Wurzeln liegen im *System V* der späten 1980er. Standard wurde dieses Startverfahren bei Linux Mitte der 1990er. Bevor System V Init seinen Siegeszug antrat, herrschte das BSD-Init vor, das immer noch bei den Derivaten der *Berkeley Software Distribution* zum Einsatz kommt. Das BSD Init ist sehr übersichtlich: eine Datei */etc/ttys* definiert die (virtuellen) Konsolen, auf denen ein Login möglich sein soll, daneben wird ein Script */etc/rc* ausgeführt, welches Systemdienste startet. Den Entwicklern des System V Init war dies zu unflexibel: Für Wartungsarbeiten sollte die Möglichkeit bestehen, den Rechner mit nur einem Teil der möglichen Dienste zu starten.

```
root@mattias-desktop: ~
root@mattias-desktop:~# ls -la /etc/rc3.d/
insgesamt 12
drwxr-xr-x   2 root root 4096 2006-11-11 18:49 .
drwxr-xr-x 103 root root 4096 2006-11-14 14:43 ..
-rw-r--r--   1 root root  556 2006-10-06 13:34 README
lrwxrwxrwx   1 root root   16 2006-11-10 14:46 S01apport -> ../init.d/apport
lrwxrwxrwx   1 root root   17 2006-11-10 14:46 S05vbesave -> ../init.d/vbesave
lrwxrwxrwx   1 root root   15 2006-11-10 14:46 S10acpid -> ../init.d/acpid
lrwxrwxrwx   1 root root   18 2006-11-10 14:46 S10sysklogd -> ../init.d/sysklogd
lrwxrwxrwx   1 root root   15 2006-11-10 14:46 S11klogd -> ../init.d/klogd
lrwxrwxrwx   1 root root   13 2006-11-10 14:46 S13gdm -> ../init.d/gdm
lrwxrwxrwx   1 root root   16 2006-11-10 14:46 S19cupsys -> ../init.d/cupsys
lrwxrwxrwx   1 root root   15 2006-11-10 14:46 S19hplip -> ../init.d/hplip
lrwxrwxrwx   1 root root   14 2006-11-10 14:46 S20apmd -> ../init.d/apmd
lrwxrwxrwx   1 root root   14 2006-11-10 14:46 S20dbus -> ../init.d/dbus
lrwxrwxrwx   1 root root   18 2006-11-10 14:46 S20festival -> ../init.d/festival
lrwxrwxrwx   1 root root   22 2006-11-10 14:46 S20hotkey-setup -> ../init.d/hotkey-setup
lrwxrwxrwx   1 root root   17 2006-11-10 14:46 S20wakedev -> ../init.d/wakedev
lrwxrwxrwx   1 root root   23 2006-11-10 14:46 S20nvidia-kernel -> ../init.d/nvidia-kernel
lrwxrwxrwx   1 root root   19 2006-11-10 14:46 S20powernowd -> ../init.d/powernowd
lrwxrwxrwx   1 root root   15 2006-11-10 14:46 S20rsync -> ../init.d/rsync
lrwxrwxrwx   1 root root   13 2006-11-11 18:49 S20ssh -> ../init.d/ssh
lrwxrwxrwx   1 root root   19 2006-11-10 14:46 S25bluetooth -> ../init.d/bluetooth
lrwxrwxrwx   1 root root   17 2006-11-10 14:46 S89anacron -> ../init.d/anacron
lrwxrwxrwx   1 root root   13 2006-11-10 14:46 S89atd -> ../init.d/atd
lrwxrwxrwx   1 root root   14 2006-11-10 14:46 S89cron -> ../init.d/cron
lrwxrwxrwx   1 root root   24 2006-11-10 14:46 S90binfmt-support -> ../init.d/binfmt-support
lrwxrwxrwx   1 root root   17 2006-11-10 14:46 S90usplash -> ../init.d/usplash
lrwxrwxrwx   1 root root   22 2006-11-10 14:46 S99acpi-support -> ../init.d/acpi-support
lrwxrwxrwx   1 root root   18 2006-11-10 14:46 S99rc.local -> ../init.d/rc.local
lrwxrwxrwx   1 root root   19 2006-11-10 14:46 S99rmnologin -> ../init.d/rmnologin
root@mattias-desktop:~#
```

Beim klassischen Sys-V-Init werden die StartSkripte in ihrer alphanumerischen Reihenfolge abgearbeitet.

Die Lösung fanden die Entwickler in so genannten *Runlevels*. Theoretisch gibt es zehn davon (0 bis 9), die unterschiedlich belegt werden können, in der Praxis der letzten zehn Jahre haben sich aber drei für das laufende System eingebürgert:

- In Runlevel 1 werden keine Netzwerkdienste und kein X-Server gestartet. Dieser Modus dient Notfallarbeiten an havarierten Systemen.
- Runlevel 3 enthält bereits die Netzwerk-Interfaces und Server-Dienste und ist deshalb bei Server-Systemen Standard.
- In Runlevel 5 wird der Displaymanager gestartet, der ein grafisches Login bereitstellt, weshalb dieses meist bei Desktop-Systemen zum Einsatz kommt.

Aus der Reihe tanzen Ubuntu, das bis 6.06 nur die Runlevels 1 und 2 kennt, und ältere SUSE- sowie Debian-Distributionen (vor 2004).

Eine besondere Bedeutung haben 0 und 6: Runlevel 0 steht für den Shutdown eines Systems, in Runlevel 6 wird rebootet. Das Default-Runlevel auf einen dieser Werte einzustellen ist demnach keine gute Idee: Ihr System würde sofort herunterfahren oder endlos neu starten. Wie das BSD Init verwendet das System V Init zwei wesentliche Orte

für die Speicherung der Konfiguration: Eine Datei */etc/inittab* enthält Einstellungen für Konsolen und definiert das Standard-Runlevel. Und anstelle eines Startscripts für alle Dienste existiert für jedes Runlevel ein Verzeichnis mit Skripten für die einzelnen Dienste.

Komfortable Tools

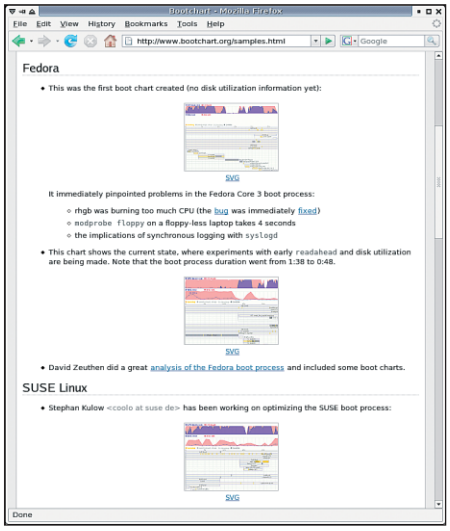
System V Init wartet mit komfortablen Hilfen für Wartungsarbeiten auf. Um in ein bestimmtes Runlevel zu wechseln, genügt es, den Befehl *init* gefolgt vom gewünschten Runlevel zu verwenden. Soll beispielsweise für Änderungen am X-Server das grafische Login abgeschaltet werden, reicht

```
init 3
```

um von Runlevel 5 nach 3 zu wechseln. Wollen Sie nur einen bestimmten Dienst stoppen, rufen Sie das Start-Stop-Script in */etc/init.d* auf:

```
/etc/init.d/xdm stop
```

Für den Neustart des gewünschten Dienstes



Mit dem Tool *Boot-Chart* können Bremsen beim Systemstart ermittelt und beseitigt werden.

verwenden Sie den Parameter *start*. Daneben sollten die meisten Skripte *restart* und *reload* kennen. Letzteres weist den Dienst nur an, eine geänderte Konfiguration neu einzulesen.

Die /etc/inittab

Die zentrale Konfigurationsdatei des System V Inits ist die */etc/inittab*. Heute wird sie praktisch nur noch dazu verwendet, das Default Runlevel und die virtuellen Konsolen zu definieren.

Dennoch sollte man ihren Aufbau kennen, denn zusammen mit dem Schlüsselwörtchen *respawn* ist die *inittab* der richtige Platz für Dienste, die nicht immer wochenlang stabil laufen. Und wenn Sie dem „Affengriff“ (also *STRG-Alt-Entf*) eine andere Funktion zuordnen wollen, dann erledigen Sie das auch in der */etc/inittab*.

Der Aufbau folgt dem Schema verschiedener „tabs“ des letzten Jahrtausends, als Trenner fungiert der Doppelpunkt:

```
2:2345:respawn:/sbin/mingetty tty2
```

Das erste Feld definiert die *ID* des Eintrages. Sie kann willkürlich gewählt werden, muss aber eindeutig sein. Es folgt eine Liste der Runlevel, für die der Eintrag gilt. *respawn* gibt an, dass der Prozess, wenn er beendet wurde, neu gestartet werden soll. Und schließlich folgt der von *init* auszuführende Befehl.

Verzeichnisse für jedes Runlevel

Für jedes Runlevel existiert ein eigenes Verzeichnis, das die Startskripte als Softlink auf die bereits erwähnten Dateien in */etc/init.d* enthält. Je nach Distribution heißen die Verzeichnisse */etc/rcX.d* oder */etc/init.d/rcX.d*, wobei *X* für das jeweilige Runlevel steht. Beim Blick in die Runlevel-Ordner fallen zwei Typen von Skripten auf: Neben den mit *K* beginnenden „Kill-Skripten“ existieren die mit

Eigenes Startscript

Ein eigenes Start-Stop-Script ist schnell erstellt. Das Gerüst ist eine Case-Abfrage, die auf wenigstens drei verschiedene Übergabe-Parameter ansprechen sollte:

```
#!/bin/sh

case $1 in
    start)
        echo 'Starte Dienst...'
        start_service
    ;;
    stop)
        echo 'Stoppe Dienst...'
        stop_service
    ;;
    restart)
        echo 'Restarte Dienst'
        stop_service
        start_service
    ;;
    *)
        echo 'Usage: script [start|stop|restart]'
    ;;
esac
```

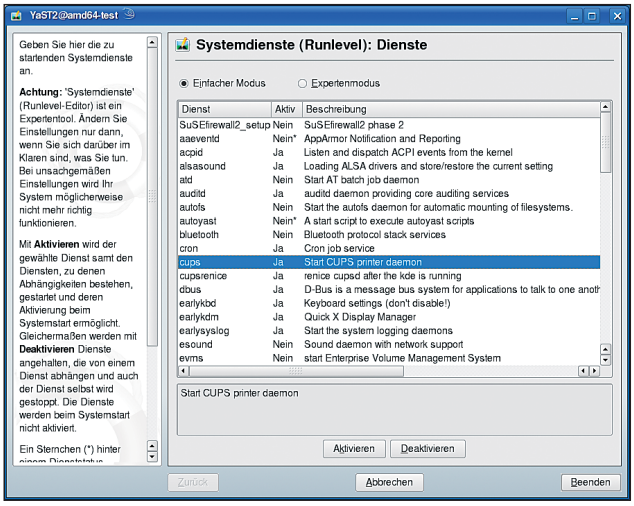
Die Funktionen „start_service“ und „stop_service“ definieren Sie separat, das erspart die Doppelungen für „restart“:

```
start_service() {
    /usr/sbin/meindaemon --pid /var/run/meindaemon
}

stop_service() {
   kill `cat /var/run/meindaemon`
}
```

Stellt der gewünschte Dienst keine PID zur Verfügung, können Sie mit *grep* in der Ausgabe von *ps -waux* nach Ihrem Prozess suchen und mit *awk* die PID ermitteln. Allerdings birgt diese Methode die Gefahr, versehentlich unschuldige Prozesse zu töten. Stellt Ihre Distribution den *start-stop-daemon* zur Verfügung, sollten Sie diesen verwenden.

Beim Verlinken des Skriptes ist darauf zu achten, dass einige Distributionen RC-Skripte, die auf *.sh* enden, nicht in einem eigenen Kindprozess, sondern im Kontext des Elternprozesses ausführt – das Script wird *ge-source-t*. In diesem Fall steigt das Risiko, durch einen Programmierfehler im Script den weiteren Startvorgang zu beeinträchtigen und bereits definierte Funktionen und Variablen zu überschreiben. Verzichten Sie auf die Endung *.sh* um das Script in einer Subshell zu starten.



SUSE bringt für die Verwaltung der Startskripte ein eigenes YaST-Modul mit.

Minimales Init mit der Busybox

Wer nur ein äußerst kompaktes Init benötigt, beispielsweise um ein Minimal-Linux als Firewall zu starten, kann das minimale Init der Busybox verwenden. Es vergrößert das Universal-Tool nur um wenige Kilobyte und bietet deutlich mehr Komfort als ein *linuxrc*-Skript für den Startvorgang. Das Format der *inittab* orientiert sich am System V Beispiel:

```
::sysinit:/etc/init.d/rcS
tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
::ctrlaltdel:/sbin/reboot
```

Im ersten Feld steht das Terminal, auf dem ein Befehl gestartet werden soll – es kommt bei den drei Gettys für ein Konsolen-Login zum Einsatz. *ctraltdel* fängt den Affengriff ab, und in der ersten Zeile wird via *sysinit* ein BSD-ähnliches Startskript ausgeführt. In seiner einfachsten Form kann es benutzt werden, um Dateisysteme zu mounten, das Netzwerk zu aktivieren und einen SSH-Daemon (hier *Dropbear*) zu starten:

```
#!/bin/sh
/bin/mount -t proc none /proc
/bin/mount -t devpts none /dev/pts
/sbin/ifconfig lo inet 127.0.0.1
/sbin/ifconfig eth0 inet 192.168.1.82
/bin/sleep 5
/usr/sbin/dropbear -E
```

Das Beispiel stammt von einem nur wenige MByte großen Mini-Linux, das Sie leicht selbst nachbauen können. Eine vollständige Anleitung finden Sie auf der Webseite des Autors unter http://blog.rootserverexperiment.de/2006/09/15/das-4mb-mini-linux/

HILFREICHE LINKS

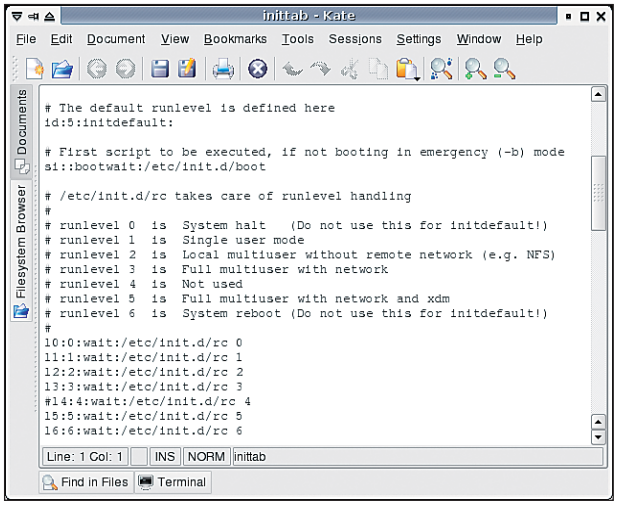
https://wiki.ubuntu.com/ReplacementInit?action=show&redirect=upstart
http://www.netsplit.com/blog/articles/2006/08/26/upstart-in-universe
http://developer.apple.com/macosx/launchd.html
http://opensolaris.org/os/community/smf
http://www.initng.org
http://blog.rootserverexperiment.de/2006/09/15/das-4mb-mini-linux

„S“ beginnenden Startskripte. Beim Wechsel in ein Runlevel werden zuerst die Kill-Skripte in der durch die Zahlen vorgegebenen Reihenfolge und anschließend die Startskripte abgearbeitet. Um einen Dienst aus einem bestimmten Runlevel zu entfernen, genügt es theoretisch, den Softlink für das S-Script zu entfernen, der K-Link sollte verbleiben, da der Dienst sonst beim Wechsel des Runlevels möglicherweise aktiv bleiben würde. In der Praxis ist das manuelle Entfernen und Hinzufügen von Softlinks recht fehleranfällig: Im Extremfall müssen für fünf Runlevels je zwei Links beachtet werden. Die meisten Distributionen bieten deshalb Tools für die Verwaltung der Runlevel. Bei Ubuntu und Debian ist dies die Kommandozeilen-Applikation *update-rc.d*, SUSE enthält ein Modul für YaST. Zudem existiert für KDE ein universelles Werkzeug für die Runlevel-Verwaltung. Allerdings lassen sich nicht bei jeder Distribution alle Dienste über Softlinks steuern: Ubuntu liest in einigen Startskripten Variablen aus */etc/alternatives* ein, die angeben, ob ein Service gestartet werden soll. Ist dem nicht so, ignoriert das Startscript die Übergabe des Parameters *start*.

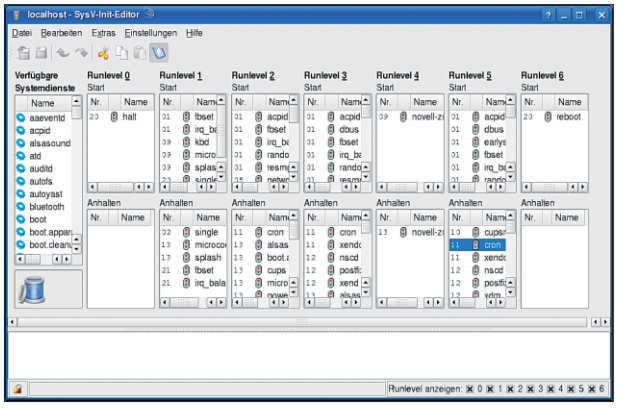
Im Gänsemarsch

Ein Kritikpunkt an Linux ist der recht langsame Start: Zwar werden Kernel und Initrd flott geladen und entpackt, die folgenden Dienste werden jedoch einer nach dem anderen gestartet. Dabei spricht aus technischer Sicht wenig gegen eine Parallelisierung. Sofern benötigt, sollte vor dem Webserver zwar der Datenbank-Server gestartet werden, ein Mailserver wird aber in der Regel noch nicht gebraucht. Der Start von Mail- und Webserver ließe sich also parallelisieren. Ideal wäre es, wenn man nur die Dienste angeben müsste, von denen ein Server abhängig ist. Eine umständliche manuelle Festlegung der Startreihenfolge entfiele.

Zudem wurde *System V Init* für Server und Workstations mit einer weitgehend festen Hardware- und Netzwerkkonfiguration entworfen, die selten neu gestartet werden. An Notebooks, die sich in wechselnden Netzen aufhalten, oder an USB-Festplatten, die, falls vorhanden, beim Start gleich eingebunden werden sollten, dachte vor fast zwanzig Jahren niemand. So haben es sich mehrere Projekte auf die Fahne geschrieben, das alte Init durch ein flexibleres System zu ersetzen: Apples *launchd* übernimmt seit etwa einem Jahr die Überwachung des Systemstarts und ersetzt auch gleich den „Herrscher über die



In der Datei */etc/inittab* sind Default-Runlevel und zu öffnende Konsolen verzeichnet.



KDE bringt mit *ksysv* ein Distributions-übergreifend nutzbares Tool für die Verwaltung der Startskripte.

Zeit“ *cron* sowie den *xinetd*, über den Netzwerkdienste gestartet werden, die nicht als Daemon laufen. Mit starkem Fokus auf den Server-Einsatz wurde Solaris *Service Management Facility* entwickelt. Daneben existiert mit *initng* ein eher auf Gentoo zugeschnittenes *Init*, das primär die Abarbeitung des klassischen Init parallelisiert.

Flexibler mit Upstart

Alle drei existierenden Projekte waren den Ubuntu-Entwicklern nicht flexibel genug, schließlich strebt Mark Shuttleworth für die Zukunft auch Marktanteile im Server-Bereich an und möchte dafür keine Desktop-Nutzer vergraulen. Die Folge war ein eigenes Projekt unter dem Namen *Upstart*, das nicht nur wie Apples *launchd* die Dienste *init*, *xinetd* und *crond* vereint, sondern weitestgehend abwärtskompatibel zu System V Init ist und mit einer Event-gesteuerten Architektur einen hohen Schutz vor Fehlfunktionen bietet. Gerade die Integration der Dienste hat handfeste Vorteile auf Maschinen, die nicht ständig laufen. Ein Beispiel ist das System-Update, das man üblicherweise dann durchführt, wenn die Nutzer nicht gestört werden. Wurde es aber verpasst, weil der Rechner nicht 24 Stunden am Stück lief, ist es ratsam, auch nach dem Booten nach Updates zu suchen. Realisiert man sein Update über ein RC-Script und einen Crontab-Eintrag, besteht die Gefahr, dass Reboot und Crontab-Eintrag zusammenfallen und so zwei Updates gleichzeitig gestartet werden, teils mit nicht erwünschten Nebeneffekten. Unter *Upstart* wird erkannt, wenn die Boot-Sequenz bereits einen Task ausführt, den der Scheduler starten möchte. Bis zu dieser Stelle bietet Upstart kaum Neues gegenüber Apples launchd. Der Clou liegt in der Verarbeitung beliebiger Events. Launchd kennt im Wesentlichen Events durch die Boot-Sequenz, den Systemstart und Netzwerkaufrufe, Upstart fügt dem Events durch das Hotplug-System oder das Powermanagement hinzu. Wer mit dem Notebook an Distributed Computing Projekten teilnimmt, kann festlegen, dass beim Einstecken des Netzkabels mitgearbeitet und beim Trennen vom Stromnetz die Arbeit beendet wird. Da auch Upstart-Ereignisse Events auslösen können, ist es einfacher, den Start von Daemons, die nicht nur von anderen Daemons, sondern auch von Netzwerk und Stromversorgung abhängig sind, zu verwalten.

Upstart in der Praxis

In der Praxis sind bei Ubuntu 6.10 kaum Unterschiede zum System V Init zu erkennen: Die RC-Skripte liegen an ihrem gewohnten Platz. Die unter */etc/event.d* zu findenden Upstart-Skripte machen nichts anderes, als die klassischen System V Init-Skripte in ihrer gewohnten Reihenfolge abzuarbeiten. Immerhin lässt sich in */etc/event.d* bereits ein Blick auf den künftigen Aufbau der Upstart-Skripte werfen. Leider sind nur Skripte für die traditionellen Startup- und Shutdown-Events enthalten, es fehlen die interessanteren Hotplug-Events und solche von Änderungen an der Stromversorgung ausgelösten. Auch Cron wurde in Ubuntu 6.10 noch nicht mit Upstart ersetzt, sodass der theoretische Vorteil, dass ein Job nicht mehrfach gestartet werden kann, noch nicht zum Tragen kommt. Zudem ist die Dokumentation lückenhaft: Manual Pages existieren nicht, die Einträge auf Ubuntus Launchpad beschreiben die Architektur und hinken hinter der realen Implementierung hinterher. Bis die Vorteile von Upstart zum Tragen kommen, werden wohl noch einige Monate vergehen.

Fazit

Nach fast zwanzig Jahren ist die Zeit reif für einen neuen „Prozess Eins“. Das alte System V Init hat lange Zeit gute Dienste geleistet, ist aber in den letzten Jahren an seine Grenzen gestoßen. Der vielseitigste Nachfolger kündigt sich mit Ubuntus *Upstart* an. Die angestrebte Kombination aus Cron, Init und Xinetd macht ihn für den Server interessant, während Desktops und Notebooks von der Event-gestützten Architektur profitieren. Allerdings wird es noch eine Weile dauern, bis sich Upstart durchgesetzt hat: Programmierer müssen ihre Skripte auf die neue Architektur anpassen und kleinere Änderungen an den Schnittstellen können Anpassungen erzwingen. Noch ist Upstart auf Ubuntu beschränkt, allerdings sind für andere Distributionen Pakete erhältlich. Wer im Jahr 2007 Änderungen an den Init-Skripten vornehmen möchte, sollte mit beiden Konzepten vertraut sein. **jkn**

Init für Eselspinguine

Ein reiner Init-Ersatz ist das von Jimmy Wennlund entwickelte und vor allem auf Gentoo-Systemen anzutreffende initng. Es parallelisiert die zu startenden Dienste falls möglich und beschleunigt so den Bootvorgang erheblich. Auch bietet initng die Möglichkeit, abgestürzte Dienste zu respawnen. Ein vollständiger Ersatz für inetd und crond möchte initng jedoch nicht sein. Auch verzichtet initng auf die Steuerung durch Events. Durch die Konzentration auf das Wesentliche ist initng deshalb robust und kompakt, die geringere Flexibilität gegenüber Upstart lässt jedoch eher einen Einsatz auf dem Server sinnvoll erscheinen.