Zustandsmanagement mit init

Nils Magnus, Kester Habermann
Unix-AG, Universität Kaiserslautern
{magnus, kester}@unix-ag.uni-kl.de

16. Mai 1997

Zusammenfassung

Das Programm init hat eine zentrale Bedeutung für den Betrieb eines Unix. Hauptsächlich kommt unter Linux das an System V angelehnte Paket zu Einsatz. Obwohl viele gute Ansätze erkennbar sind, fehlt bis heute eine in allen Belangen zufriedenstellende Lösung. Dazu werden wesentliche Kriterien für ein solches System herausgearbeitet und eine zusammenfassende Anforderungsspezifikation vorgestellt.

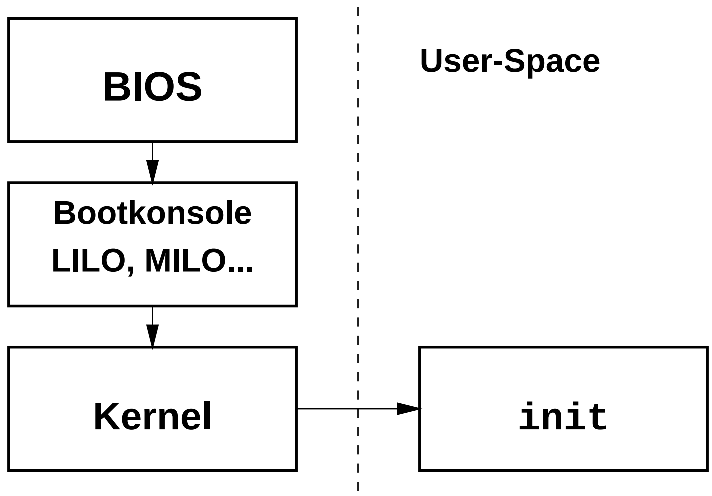
Inhaltsverzeichnis

1 Gliederung

- Vorstellung verschiedener Realisierungen des init-Konzeptes
- Gegenüberstellung von gegenwärtigen Vor- und Nachteilen
- Herausarbeiten wesentlicher Kriterien
- Möglichkeiten einer Realisierung
- Ableitung eines Anforderungsprofiles

2 Vorstellung des init-Konzeptes unter Linux

Booten von Linux



Vorbereitung des Systemes:

- Identifikation und Initialisierung der Hardware
- Laden und Starten des Kernels
- Initialisierung der Kernel-internen Subsysteme (Device-Treiber, Memory-Management, Filesysteme)

Vergleich bei verschiedenen Betriebssystemen

- **DOS:** Ein textbasierter Kommandozeileninterpreter (Shell)
  → `/bin/sh`
- **Windows:** Mehrere Fenster, eines aktiv, „statisches Multitasking“
  → Shellscript, das diverse Applikationen aufruft (`.xinitrc`).

Unix-Varianten:

- BSD-Style
  wenige Skripte für alle Aufgaben
- System V-Style
  Aufteilung in Subdirectories und einzelne Skripte

⇒ Linux-eigene Änderungen

3 Beschreibung der Funktionalität von init und Runleveln

Aufgaben von init

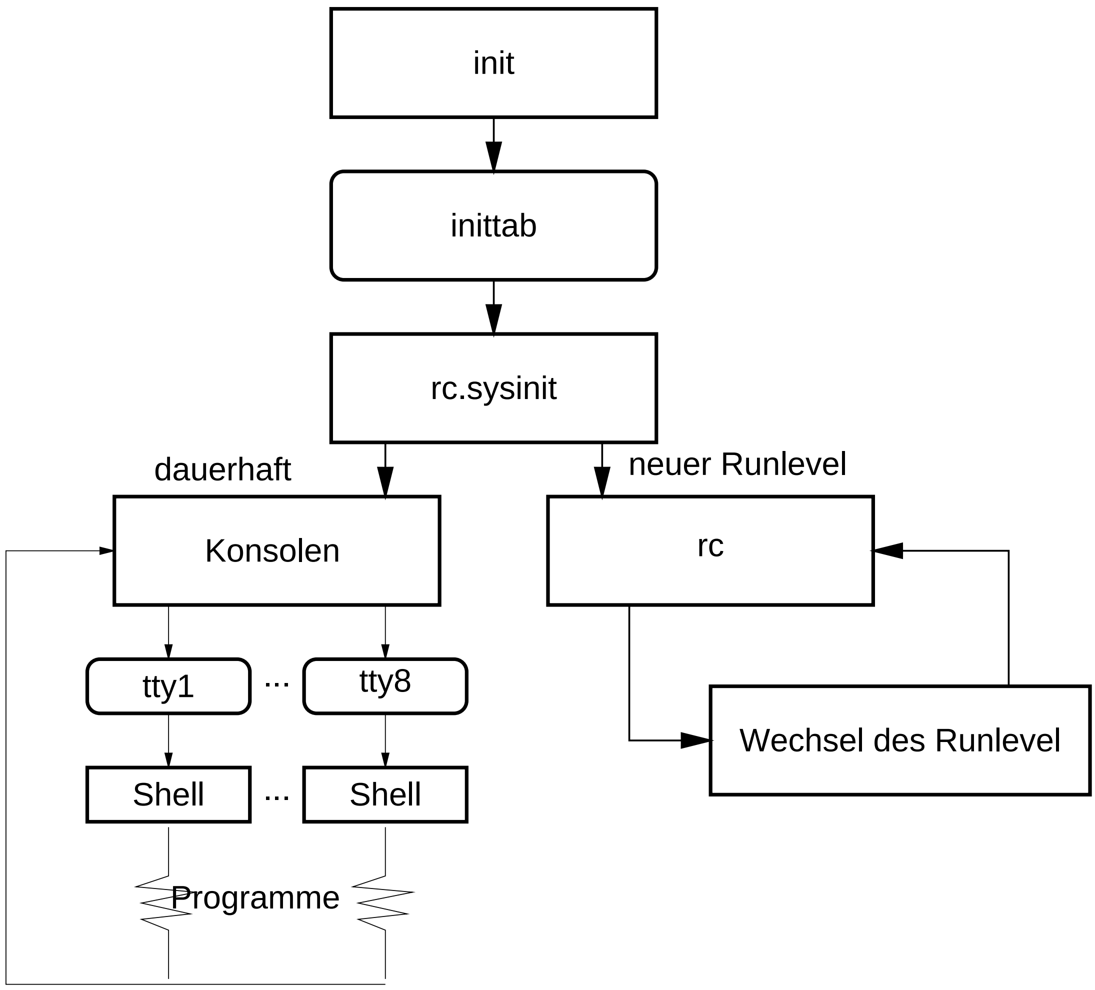
- Initialisierungen (`rc.sysinit`),
- dauerhaft zu kontrollierende Prozesse (Logins, USV-Kontrolle),
- Subsysteme in bestimmten Systemzuständen (Netz-Server, Host-Server, Remote-Filesystem, ...)

Initialiserungen und dauerhafte Einstellungen ändern sich selten.

Subsystemne müssen häufiger umkonfiguriert werden

⇒ Runlevel.

Ablauf von init:



Was sind Runlevel?

- Bezeichnung durch eine Ziffer (oder einige Buchstaben)
- Modellieren den „Gesamtzustand“ des Systems
- Gruppieren die zur Verfügung stehenden Services und Subsysteme
- Aktivierung von dynamischen Systemkomponenten, z. B. Netzwerkdaemonen oder NFS

Beispiele für Runlevel

1 Eingeschränktes System nur für den Systemverwalter zu Administrationszwecken (sog. Single-User-Mode, auch Runlevel S)

2 Komplettes System, fertig für Benutzerbetrieb, ohne Netzwerk

3 Komplettes System, fertig für Benutzerbetrieb, mit Netzwerk

Spezielle Runlevel:

0 Halt (nur in diesem Zustand kann der Rechner ohne Datenverlust ausgeschaltet werden)

6 Reboot

Initialisieren eines Runlevels

- Auslösen des Default-Runlevels (aus `/etc/inittab`)
- Übergabe als Parameter beim Booten durch die Boot-Konsole (LILO, MILO, SILO, ...)

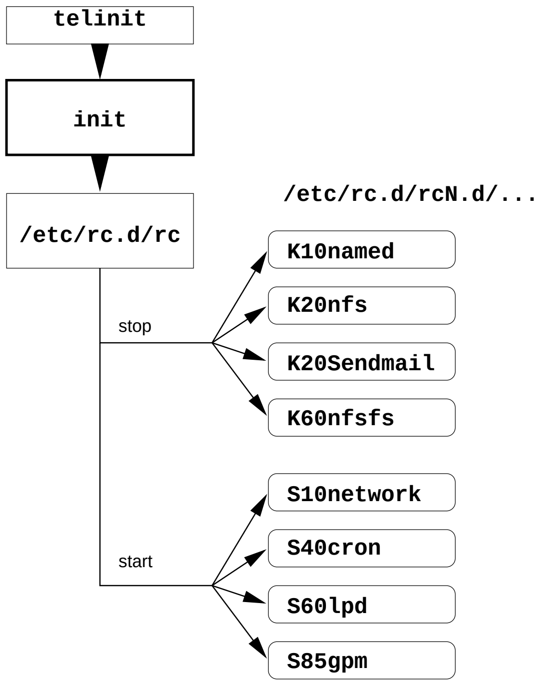
```
lilo boot: linux 3
```

- Expliziter Aufruf durch Systemverwalter bei laufendem Betrieb:

```
# telinit 4
```

- Shutdown durch Befehl oder Hotkeys (Ctrl-Alt-Del)
- Sicheres Herunterfahren durch `powerd` wegen Stromausfall etc.

Wechsel des Runlevel



- `telinit` schickt dem init-Prozess ein Signal
- init liest `inittab` neu aus, startet `rc`
- im neuen Runlvel werden **nicht** benötigte Subsysteme abgeschaltet
- im neuen Runlevel werden neue Subsysteme aktiviert

Vorgänge beim Wechsel

- `rc` wird aufgerufen (Parameter: neuer (und alter) Runlevel)
- im neuen Runlevel-Verzeichnis werden Stoppskripte aufgerufen
- im neuen Runlevel-Verzeichnis werden Startskripte aufgerufen
- Skripte liegen in `/etc/rc.d/rcN.d/...`
- die Skripte sind Symlinks nach `/etc/rc.d/init.d/...`,wo *alle* Skripte für *alle* Runlevel gesammelt werden
- Jedes Skript startet bzw. stoppt genau *ein* Subsystem, je nachdem, ob es mit Paramter `start` oder `stop` aufgerufen wird

4 Anfordungsanalyse

Bestehende Probleme

- Inkonsistenz (Zustand in einem Runlevel z. T. abhängig von „Vorgeschichte")
- uneinheitliche Semantik der Start- und Stoppskripte
- Bootvorgang unstrukturiert und unübersichtlich
- unklares Verhalten im Fehlerfall
- administrative Aufgaben schlecht automatisierbar

⇒ Anforderungsprofil erforderlich

Subsysteme

Eine logische Einheit bildet ein *Subsystem* (auch mehrere Kommandos, z. B.: `rpc.mountd`, `rpc.nfsd` bei `nfs`).

- ein `rc`-Skript pro Subsystem
- Locking-Mechanismus auf Subsystemebene
- Subsystem kann gestart und gestoppt werden
- Restart sollte möglich sein (z. B. nach Software-Update)
- Einheitliche Konsolenmeldung bei Start und Stopp
- Definition von Abhängigkeiten

Abhängigkeiten

- Definition der Abhängigkeiten (z. B. `amd` nach Netz)
- automatische Generierung der Start- und Stopp-Reihenfolge der Subsysteme

- Definition (virtueller) Services, Skriptnummer ab der verfügbar
- Abhängigkeiten von Subsystemen ('provides', 'requieres')
- Tool, das Subsysteme automatisch einfügt

Konsistenz

Zustand in einem Runlevel muß immer gleich sein (unabhängig von der „Vorgeschichte").

Beispiel für Inkonsistenz:

Runlevel 3 und 5 seien gleich, bis auf Startskript für `xdm` in `rc5.d`.

Kommando	Runlevel	Aktion
`initdefault 5`	5	`xdm` läuft
`telinit 3`	3	`xdm` läuft
`initdefault 3`	3	`xdm` läuft nicht

Definition einer Semantik

In jedem Runlevel müssen alle verwendeten Subsyteme entweder gestartet oder gestoppt werden. In jedem Runlevel entweder Starts- oder Stoppskript

Neuer Runlevel:

- Subsysteme stoppen:
  - die im neuen Runlevel nicht mehr laufen sollen
  - und die bereits laufen
- Subsysteme starten:
  - die im neuen Runlevel laufen sollen
  - und die noch nicht laufen

Man beachte die nicht notwendig zwingende Semantik!

Konformität mit Standards

- FSSTND (FHS)-compliant
- Verhalten und Aufbau sollte Standards entsprechen (POSIX, SVID, andere Unices)
- Kompatibilität zu vorhandenen Tools, existierende Pakete (z. B. RPMs) sollten weiterhin installierbar sein.
- Portierbarkeit
- Problem: Widerspruch verschiedener Ansätze

Nach Linux Filesystemstandard 1.2 sind als Platz für die `rc`-Skripte sowohl

- das System V-Modell (`/etc/rc.d/*`)
- als auch das BSD-Modell (`/etc/rc.*`)
- sowie eine Kombination aus beiden

zugelassen.

Es sollten aus Portierbarkeitsgründen keine non-Standard Features (wie z. B. `bash`, `/usr`) in `inittab`, `rc` und `rc.sysinit` verwendet werden.

Behandlung von Ausnahmefällen

- Nach `shutdown` sollten Lockfiles entfernt sein
- Überprüfen von „Stale Locks“ beim Booten
- Bail out in Maintenance Mode bei Fehlern (z. B. `fsck`)
- Paßwortschutz aus Sicherheitsgründen

Simulation

Simulation ermöglichen:

- Was ändert sich beim Wechsel zu Runlevel `N`?
- Welche Subsysteme werden gestartet, gestoppt?

Automatisierbarkeit

- Subsysteme hinzufügen, löschen (z. B. per RPM)
- Administrative Aufgaben
  dauerhaft: Subsysteme aktivieren, deaktivieren
  einmalig: Subsysteme starten, stoppen, neustarten
- Konsistenzchecks bzgl. definierter Runlevelsemantik
- weiterhin mit herkömmlichen Mitteln wartbar

Schnittstellen

Definierte Schnittstelle zur Anbindung anderer Software, z. B. grafisches Frontend

- bei formatierter Ausgabe ist Parsing der Boot-Ausgaben möglich
- Einzelskripte ermöglichen gezielte Manipulation von Subsystemen
- Status kann über Lockfiles eingesehen werden

Frontend

Zustandsmanager, User-Interface, …

5 Zusammenfassung

- Runlevel als Gliederungsstruktur
- Konzept des autonomen Subsystems
- Definierte Semantik der Subsysteme in einem Runlevel (Konsistenz)
- Einhaltung von Standards
- Simulation, Automatisierung
- Benutzerschnittstelle