systemd

ein init-replacement

c¼h von sECuRE

NoName e.V., 2011-08-18

powered by LaTeX

Teil 1: Was ist init?

init

- PID 1, vom Kernel gestartet
- startet daemons, inklusive X11 bzw. xdm
- adoptiert elternlose Prozesse (fork),
  startet sie ggf. neu (z.B. getty)

SysV init

- seit UNIX System V
- verschiedene „Runlevels“ geben an was gestartet wird:
  1 = single user mode, 2 = grafisches system, …
- Reihenfolge (sequenziell) durch Nummern in `/etc/rc*.d`
- Shellscripts in `/etc/init.d/*` starten die daemons

Probleme mit SysV init

- (leicht) verschiedene Scripts, je nach Distribution
- komplexe Scripts (125 Zeilen, 78 SLOC im Schnitt)
- jeder Daemon muss dasselbe implementieren:
  background, `setsid`, privilege dropping, PID files, log, …
- daemons werden nicht überwacht (nur gestartet)
- langsamer (meist sequenzieller) Systemstart

SysV init Beispiel: /etc/init.d/thinkfan (1/9)

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          thinkfan
# Required-Start:    $remote_fs
# Required-Stop:     $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: thinkfan initscript
# Description:     starts thinkfan if enabled in /etc/default/th
### END INIT INFO

# Author: Evgeni Golov <evgeni@debian.org>

# Do NOT "set -e"
```

SysV init Beispiel: /etc/init.d/thinkfan (2/9)

```
# PATH should only include /usr/* if it runs after mountnfs.sh
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="fan control tool"
NAME=thinkfan
DAEMON=/usr/sbin/$NAME
DAEMON_ARGS="-q"
# This one is compiled-in, you can't change it here!
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
START=no
```

SysV init Beispiel: /etc/init.d/thinkfan (3/9)

```
# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Check if daemon is to be started
[ "$START" = "yes" ] || exit 0

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure this file is present.
. /lib/lsb/init-functions
```

SysV init Beispiel: /etc/init.d/thinkfan (4/9)

```
# Function that starts the daemon/service
do_start()
{
 # Return
 #   0 if daemon has been started
 #   1 if daemon was already running
 #   2 if daemon could not be started
 start-stop-daemon --start --quiet --pidfile $PIDFILE \
     --exec $DAEMON --test > /dev/null || return 1
 start-stop-daemon --start --quiet --pidfile $PIDFILE \
     --exec $DAEMON -- $DAEMON_ARGS || return 2
}
```

SysV init Beispiel: /etc/init.d/thinkfan (5/9)

```
# Function that stops the daemon/service
do_stop()
{
 # [...]
 start-stop-daemon --stop --quiet --retry=TERM/30/KILL/5 \
     --pidfile $PIDFILE --name $NAME
 RETVAL="$?"
 [ "$RETVAL" = 2 ] && return 2
 # [...]
 start-stop-daemon --stop --quiet --oknodo \
     --retry=0/30/KILL/5 --exec $DAEMON
 [ "$?" = 2 ] && return 2
 # Many daemons don't delete their pidfiles when they exit.
 rm -f $PIDFILE
 return "$RETVAL"
}
```

SysV init Beispiel: /etc/init.d/thinkfan (6/9)

```
#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
 #
 # If the daemon can reload its configuration without
 # restarting (for example, when it is sent a SIGHUP),
 # then implement that here.
 #
 start-stop-daemon --stop --signal 1 --quiet --pidfile $PIDFILE
 return 0
}
```

SysV init Beispiel: /etc/init.d/thinkfan (7/9)

```
case "$1" in
  start)
   [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME
    do_start
    case "$?" in
            0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
             2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
    esac
    ;;
  stop)
   [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME
    do_stop
    case "$?" in
            0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
             2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
    esac
    ;;
```

SysV init Beispiel: /etc/init.d/thinkfan (8/9)

```
status)
  status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
  ;;
reload|force-reload)
  log_daemon_msg "Reloading $DESC" "$NAME"
  do_reload
  log_end_msg $?
  ;;
restart)
  log_daemon_msg "Restarting $DESC" "$NAME"
  do_stop
  case "$?" in
    0|1)
      do_start
```

SysV init Beispiel: /etc/init.d/thinkfan (9/9)

```
        case "$?" in
          0) log_end_msg 0 ;;
        1) log_end_msg 1 ;; # Old process is still running
          *) log_end_msg 1 ;; # Failed to start
        esac
        ;;
      *)
        # Failed to stop
        log_end_msg 1
        ;;
    esac
    ;;
  *)
   echo "Usage: $SCRIPTNAME {start|stop|restart|status|restart|
    exit 3
    ;;
esac
```

Teil 2: systemd

systemd

- kompatibel mit SysV init
- möglichst wenig (shell, unnötigen Code) starten
- Parallelisierung ($\Rightarrow$ Geschwindigkeit!)
- **einfache** unit files ($\approxeq$ initscript)
- implizite Abhängigkeiten, socket/dbus activation
- cgroups

systemd Beispiel: thinkfan.service

```
[Unit]
Description=simple and lightweight fan control program

[Service]
ExecStart=/usr/sbin/thinkfan -q -n

[Install]
WantedBy=multi-user.target
```

systemctl

```
michael ~ $ systemctl status thinkfan.service
thinkfan.service - simple and lightweight fan control pr
      Loaded: loaded (/lib/systemd/system/thinkfan.servi
      Active: active (running) since Sat, 13 Aug 2011 23
    Main PID: 1794 (thinkfan)
      CGroup: name=systemd:/system/thinkfan.service
              1794 /usr/sbin/thinkfan -q -n


# start/stop/reload/restart, wie bei init.d
```

Abhängigkeiten: socket activation

- socket activation für Socket-basierte Dienste wie syslog, D-Bus, CUPS, …
- Beispiel: syslog nutzt `/dev/log`
- Daemons connecten darauf und schreiben Logmeldungen
- Idee: Socket **vor** Programmstart von syslog erzeugen
- Daemons können bereits connecten **und** schreiben, der Kernel buffert
- Sobald syslog läuft, kann er sofort die Daten wegschreiben

Abhängigkeiten: socket activation (2)

- $\Rightarrow$ Abhängigkeit ist implizit vorhanden. Wir können syslog und Daemons gleichzeitig starten
- Erst bei tatsächlicher Verbindung wird das zugehörige Programm gestartet
- $\Rightarrow$ SSHd läuft erst, wenn man ihn tatsächlich benutzen will

Abhängigkeiten

- Klassische Abhängigkeiten lassen sich via `After=` und `Before=` angeben
- Seltenst nötig (Beispiel: `ifup.service` setzt `After=local-fs.target`)
- Standardmäßig starten services nach `basic.target`, welches low-level Zeug beinhaltet

initscript-Inhalt

- Einige Scripts erledigen Aufräumarbeiten oder legen Verzeichnisse in `/var/run` an
- Stattdessen zentrale Config in `/etc/tmpfiles.d/*.conf`, systemd legt sie dann an/räumt auf
- Kernelmodule werden analog dazu in `/etc/modules-load.d/*` konfiguriert, sollten aber besser automatisch im Kernel geladen werden
- Kleinkram wie hostname setzen, Dateisysteme mounten/`fsck`en sind direkt in C implementiert

cgroups

- Saubere Trennung der Prozesse, eine cgroup pro service
- Man kann einen Service problemlos aufräumen, selbst wenn Teile davon crashen
- Resource-Limits (CPU, Speicher, …) pro cgroup zuweisbar
- `systemd-cgls` listet die vorhandenen cgroups und deren Prozesse

systemd-analyze

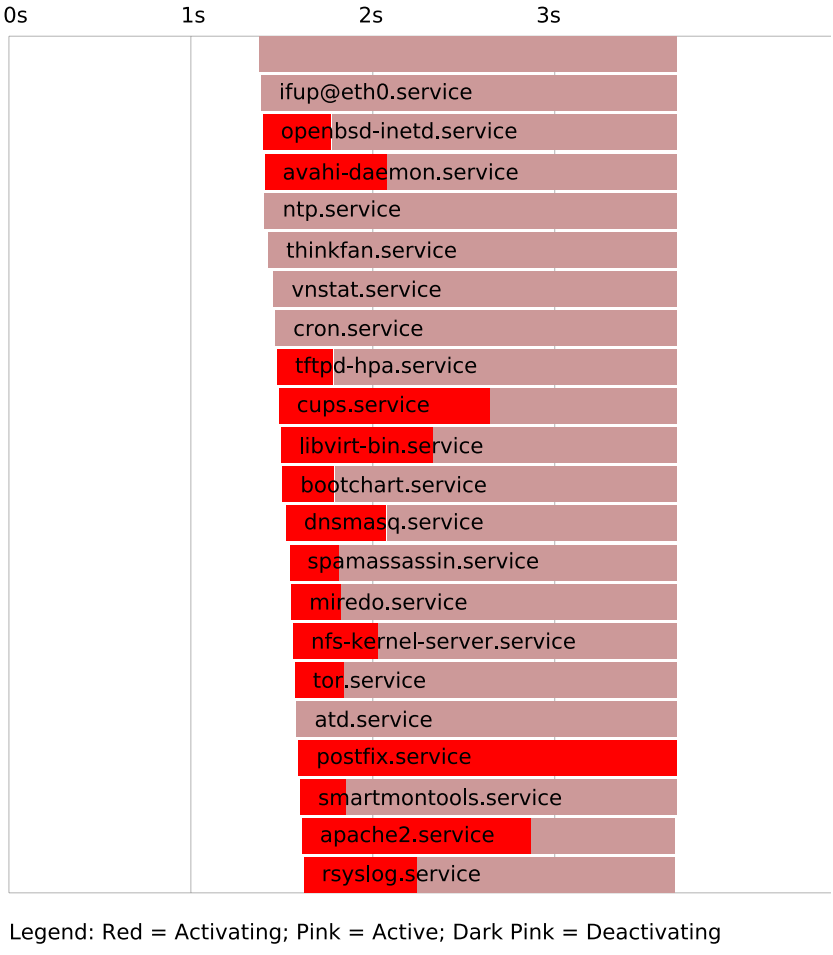

Figure: `systemd-analyze plot > plot.svg`

Weitere Resourcen

- http://0pointer.de/blog/projects/systemd.html
- http://www.freedesktop.org/wiki/Software/systemd
- http://en.wikipedia.org/wiki/Systemd
- http://wiki.debian.org/systemd
- https://github.com/falconindy/systemd-arch-units