Linux – Prinzipien und Programmierung

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2014

Boot-Prozess

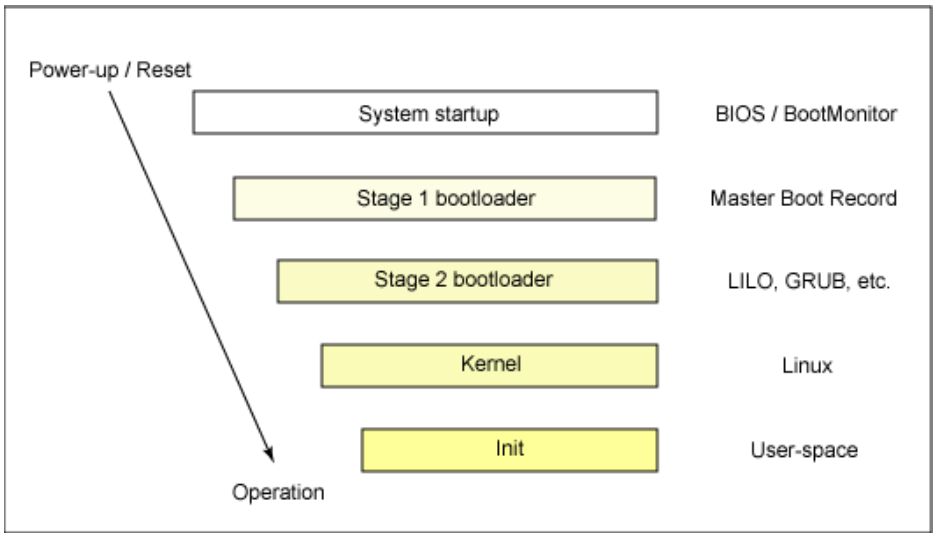
Dienste

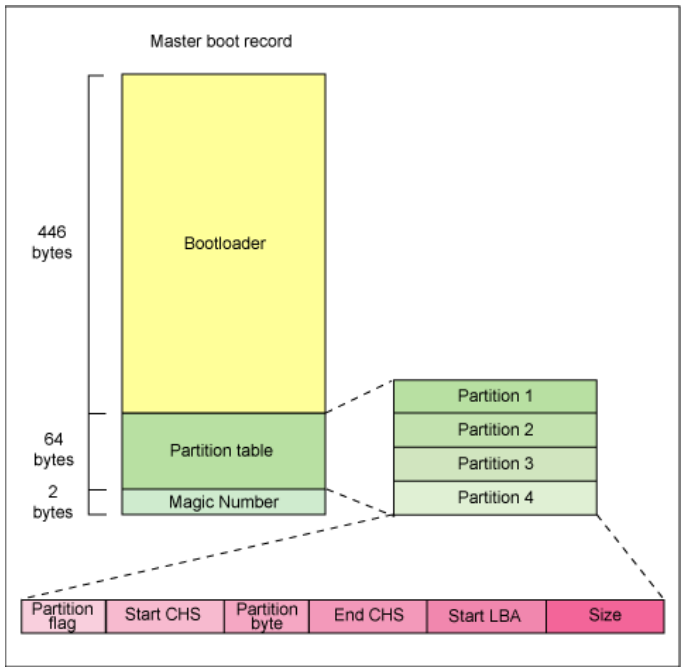
Der Linux-Boot-Prozess

Hier sollen die einzelnen Phasen des Boot-Prozesses von Linux betrachtet werden:

1. Systemstart im BIOS
2. Stage 1 bootloader im MBR
3. Stage 2 bootloader
4. Starten des Linux-Kernels
5. Init

Übersicht





Bilder: Tim Jones,
http://www.ibm.com/developerworks/library/l-linuxboot/

Die ersten Stufen

Beim Starten des PC werden die ersten Schritte durch das BIOS des Rechners ausgeführt, dazu gehört insbesondere das Suchen nach einem bootbaren Medium (Festplatte, CD, USB-Stick, früher Diskette) in der im BIOS-Setup festgelegten Reihenfolge.

Die Identifizierung eines Mediums als bootbar erfolgt hierbei durch das Vorhandensein eines gültigen Master Boot Records (MBR) auf dem Medium.

Dieser MBR enthält den Linux-Bootloader der ersten Stufe (Stage 1). Dieser lädt im Wesentlichen den Bootloader Stage 2.

Linux-Bootloader

Die unter Linux verbreiteten Bootloader stellen sowohl den ausführbaren Code für Stage 1 als auch Stage 2 bereit. Der früher benutzte LILO (Linux Loader) wurde hierbei bei den meisten Distributionen inzwischen durch GRUB (GNU Grand Unified Bootloader) ersetzt.

Der im MBR befindliche Teil GRUB Stage 1 beschränkt sich weitgehend auf das Laden des GRUB Stage 2 (ggfs. nach Ausführen von Code, der zum Lesen von großen Festplatten notwendig ist, manchmal als Stage 1.5 bezeichnet).

In GRUB Stage 2 wird (je nach Konfiguration) ein Boot-Menü angezeigt, in dem z. B. zwischen verschiedenen Kerneln (oder zwischen Linux und anderen OS wie Windows) ausgewählt werden kann. Nach Auswahl des gewünschten Kernels wird dieser dann geladen.

Kernel-Phase

Im Allgemeinen liegt der Kernel in einer Image-Datei, die das (früher mit zlib, inzwischen mit bzip- oder lzma) komprimierte Kernel-Image enthält (zusammen mit dem Code, um das Image in das RAM zu entpacken).

Komplizierter wird es, wenn der Kernel noch einige Kernel-Module benötigt. Da das Dateisystem der Festplatte hier noch nicht zur Verfügung steht, müssen diese Module auf anderem Weg geladen werden. Hierzu wird eine Initial RAM Disk (früher initrd, heute initramfs) verwendet. Diese initrd kann z. B. Treiber enthalten, die der Kernel braucht, um das Dateisystem mit dem eigentlichen Linux-System überhaupt mounten zu können. Innerhalb der Inital RAM Disk befindet sich ein Init-Skript, ein normales Shell-Skript mit Code, der nach dem Laden des initramfs ausgeführt wird.

initramfs ist ein cpio-Archiv, das mit gzip komprimiert wird.

Init-Prozess, klassisch und modern

Unter Linux läuft der so genannte *init*-Prozess, der insbesondere Systemdienste startet und beendet.

Hier geschieht gerade ein wichtiger Umbruch:

- Ältere, stabile Linux-Distributionen verwenden ein Init-System, das kompatibel zum Großteil der Unix-Welt ist, indem dem Schema von Unix System V gefolgt wird,
- neue, moderne Distributionen verwenden mittlerweile alternative Init-Systeme, insbesondere das von RedHat entwickelte bzw. geförderte `systemd` (Fedora, OpenSuse) oder `upstart` unter Ubuntu.

Zunächst wird nun die klassische Variante dargestellt.

Der klassische Init-Prozess (SysV)

Nach dem Laden des Kernels werden die Dateisysteme des Systems gemountet (wie in /etc/fstab konfiguriert), dann führt der so genannte init-Prozess Programme aus, bis in den in /etc/inittab eingestellten Runlevel gewechselt wird.

Im vom Administrator gewählten Runlevel werden die für das System gewünschten Dienste gestartet. Neben üblichen Aufgaben wie Konfiguration der Netzwerkschnittstellen und -einstellungen gehören hierzu insbesondere das Mounten bzw. Exportieren weiterer Dateisystem (z. B. über Netzwerk) und das Starten von Services als Dämon, z. B. sshd, httpd.

Übliche Runlevel

Level	Bedeutung
1	Single User
2	Multi User (ohne Netzwerk)
3	Multi User, mit Netzwerk, ohne X11
5	Multi User mit X11

Die Angaben beziehen sich auf Red Hat, Fedora. Bei anderen Distributionen können die Zahlen abweichen.

Aktionen in den einzelnen Runlevels

Welche Dienste beim Betreten eines Runlevels gestartet (oder beim Wechseln aus einem anderen Runlevel gestoppt) werden, wird in Linux über verschiedene Skripten und Links gesteuert, die sich unterhalb des Verzeichnisses `/etc/rc.d` befinden:

/etc/rc.d/init.d Start-/Stop-Skripte (meist Init-Skript genannt) der einzelnen Dienste,

/etc/rc.d/rc?.d Init-Skripte für den betreffenden Runlevel, wobei diese jeweils symbolische Links auf das entpsrechende Skript in `/etc/rc.d/init.d` sind.

Anatomie eines Init-Skriptes

Die Start-/Stop-Skripte müssen zwingend zwei Kommandozeilenparameter zum Starten bzw. Beenden eines Dienstes verstehen:
`/etc/rc.d/init.d/myscript start` oder
`/etc/rc.d/init.d/myscript stop`

Im Allgemeinen sind noch andere Parameter zulässig:

status	zur Ausgabe des Status (running oder stopped),
restart	führt stop und start hintereinander aus,
condrestart	falls der Dienst läuft, wird er beendet und neu gestartet,
reload	weist einen laufenden Dienst an, seine Konfigurationsdateien neu einzulesen.

Anatomie eines Start-/Stop-Skriptes (Forts.)

Was soll nun das Start-/Stop-Skript machen:

- Beim Start soll natürlich ein Dienstprozess gestartet werden.
  Hierbei wird erwartet, dass der Dienstprozess seine Prozess-ID in die Datei `/var/run/dienst.pid` schreibt.
  Außerdem legt das Start-Skript eine Datei `/var/lock/subsys/dienst` an.
- Beim Stoppen wird der Prozess mit der in `/var/run/dienst.pid` stehenden Prozess-ID gestoppt, anschließend die Datei unter `/var/lock/subsys` gelöscht.
- In beiden Fällen wird auf dem Bildschirm ausgegeben, ab das Starten bzw. Stoppen des Dienstes erfolgreich war.

Für die häufig benötigten Aktionen in den Skripten gibt es vordefinierte Shell-Funktionen in der Datei `/etc/rc.d/init.d/functions`, die in den Skripten daher zu Beginn inkludiert (per Punkt-Befehl) wird.

Ablauf beim Wechseln in einen Runlevel

In dem Verzeichnis `/etc/rc.d/rc?d` befinden sich Links auf die Start-/Stop-Skripte, die jeweils die Form `SddSkriptname` bzw. `KddSkriptname` haben, d. h diese beginnen mit S oder K und einer zweistelligen Zahl.

Beim Wechseln in diesen Runlevel werden nun zunächst die mit K beginnenden Links mit dem Parameter stop aufgerufen (also K=kill), wenn es einen entsprechenden Link unter `/var/lock/subsys` gibt (!), danach die mit S beginnenden Links mit dem Parameter start aufgerufen (also S=start). Hierbei werden die Stop- bzw. Start-Befehle in aufstehender Reihenfolge der zweistelligen Zahlen ausgeführt, also lautet die Reihenfolge z. B.

```
K01abc, K85def, S25ghi,  S95jkl
```

Beispieldienst

Das folgende C-Programm schreibt als Hintergrundprozess alle 10 Sekunden eine Meldung mit dem Inhalt „ping“ über den syslog-Dämon in die Liste der Systemmeldungen:

```
#include <stdio.h>
#include <syslog.h>
#include <unistd.h>
#include <stdlib.h>

void usage(char* prog) {
    printf("Usage: %s [-b] [-p pidfile]\n");
}

int main(int argc, char** argv) {
    int do_bg = 0;
    char* pidfile = NULL;
```

Beispieldienst (Forts.)

```
int iarg = 1;
while (iarg<argc) {
    char* thisarg = argv[iarg];
    if (!strcmp(thisarg,"-b")) {
        do_bg=1;
        iarg++;
    } else if (!strcmp(thisarg,"-p")) {
        if (iarg+1>=argc) {
            usage(argv[0]);
            exit(EXIT_FAILURE);
        }
        pidfile = argv[iarg+1];
        iarg += 2;
    } else {
        usage(argv[0]);
        printf("Unknown option %s\n",thisarg);
        exit(EXIT_FAILURE);
    }
}
```

Beispieldienst (Forts.)

```
    if (do_bg) {
        int fork_pid = fork();
        if (fork_pid<0) return(EXIT_FAILURE);
        if (fork_pid>0) return(EXIT_SUCCESS);
    }

    if (pidfile) {
        FILE* pid = fopen(pidfile,"w");
        fprintf(pid,"%d\n",getpid());
        fclose(pid);
    }

    openlog("ping",LOG_ODELAY,LOG_LOCAL0);
    while (1) {
        syslog(LOG_INFO,"%s","ping");
        sleep(10);
    }
}
```

Das Start-/Stop-Skript

Nun legen wir das Start-/Stop-Skript als `/etc/init.d/log` an:

```
#! /bin/bash
#
# log       service to send pings to syslog
#
# chkconfig: 345 99 01
# description: pings to syslog as test service
#
# pidfile: /var/run/log.pid

. /etc/init.d/functions

RETVAL=0
prog=log
progdir=/root
lockfile=/var/lock/subsys/$prog
pidfile=/var/run/$prog.pid
CMD=$progdir/$prog
OPTIONS=-b -p $pidfile
```

Das Start-/Stop-Skript (Forts.)

```
start() {
    [ "$EUID" != 0 ] && exit 4
    [ -x "$CMD" ] || exit 1

    echo -n $"Starting $prog: "
    daemon $CMD $OPTIONS
    RETVAL=$?
    [ $RETVAL -eq 0 ] && success || failure
    echo
    [ $RETVAL -eq 0 ] && touch $lockfile
    return $RETVAL
}
```

Das Start-/Stop-Skript (Forts.)

```
stop() {
    [ "$EUID" != 0 ] && exit 4

    echo -n $"Stopping $prog: "
    if [ -n "$(pidfileofproc $prog)" ]; then
        killproc $prog
    else
        failure $"Stopping $prog"
    fi
    RETVAL=$?

    [ $RETVAL -eq 0 ] && rm -f $lockfile
    echo
    return $RETVAL
}
```

Das Start-/Stop-Skript (Forts.)

```
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    status)
        status $prog
        ;;
```

Das Start-/Stop-Skript (Forts.)

```
    condrestart)
        if status $prog >/dev/null; then
            stop
            start
        fi
        ;;
    reload)
        exit 4
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|re
        exit 2
esac
```

Arbeiten mit Start-/Stop-Skripten

Im Prinzip kann unser Start-/Stop-Skripte von Hand gestartet werden:
`/etc/init.d/log start` oder
`/etc/init.d/log stop` oder
`/etc/init.d/log status`

Stattdessen kann aber (auf Redhat-Systemem) das Programm `service` genutzt werden, das sich in `/usr/sbin` befindet:
`service log start` usw.

Nebenbei unterstützt service noch den Aufruf
`service --status-all`

Verwalten der Start-/Stop-Skripte

Zu einem ordnungsgemäß konfigurierten Systemdienst gehören neben dem eigentlichen Init-Skript die entsprechenden Links in den einzelnen Runlevel-Verzeichnissen, so dass der Dienst beim Wechsel in den Runlevel entweder gestartet oder beendet wird.

Soll unser Dienst `log` in den Runlevels 3, 4 und 5 laufen, so müssen also folgende Links existieren:

`/etc/rc.d/rc0.d/K01log` (Halt)
`/etc/rc.d/rc1.d/K01log`
`/etc/rc.d/rc2.d/K01log`
`/etc/rc.d/rc3.d/S99log`
`/etc/rc.d/rc4.d/S99log`
`/etc/rc.d/rc5.d/S99log`
`/etc/rc.d/rc6.d/K01log` (Reboot)

chkconfig

Auf RedHat-Systemen existiert (übernommen von IRIX) das Programm `chkconfig`, mit dem diese Links in den Runlevel-Verzeichnissen verwaltet werden können. Dabei wird der Kommentar

```
# chkconfig: 345 99 01
```

im Init-Skript ausgewertet, die die Runlevel angibt, in denen der Dienst laufen soll, sowie die zweistellige Zahl für die Start- bzw. Stop-Links angibt.

Benutzen von `chkconfig`

- `chkconfig log` prüft, ob der Dienst `log` korrekt konfiguriert ist,
- `chkconfig --add log` erzeugt dabei die Links für den Dienst `log`,
- `chkconfig --list log` listet auf, in welchen Runlevels der Dienst läuft.
- `chkconfig --del log` entfernt alle Start-/Stop-Links zum Dienst.

Die Einstellung für einzelne Runlevel kann mit
`chkconfig --levels {level} log {on|off}`
geändert werden.

Beispiele für Dienste

- `ntpd`, Dämon zum laufenden Synchronisieren mit Zeitserver über das *Network Time Protocl*, optional auch als Server für andere Rechner.
- `httpd`, Webserver (i. A. Apache).
- `nfs`, zum Exportieren von Verzeichnissen an andere Rechner über NFS.
- `sshd`, zum Zulassen von Verbindungen über *Secure Shell*.
- `crond`, Dämon zum zeitgesteuerten Ausführen von Aufgaben.