Linux – Prinzipien und Programmierung

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2014

systemd

Dateisysteme

Logical Volume Manager

Motivation für systemd

`systemd` ist ein Ersatz für das SystemV-Init-System.
Dieses ist zwar relativ simpel aufgebaut, hat aber einige Nachteile:

- Keine Parallelisierung, Dienste werden hintereinander gestartet,
- dadurch lange Zeit zum Booten,
- die Dienste (Dämonen) bzw. deren Init-Skripte müssen grundlegende Funktionalitäten wie das Bringen des Daemon-Prozesses in den Hintergrund, das Anlegen der PID-Datei und der Lock-Datei unter `/var/log/subsys` selber realisieren.

Historie

Der Hauptentwickler von `systemd` ist Lennart Poettering (Red Hat), unter Mitarbeit von insbesondere Kay Sievers (früher Novell, jetzt Red Hat). Daher ist es nicht verwunderlich, dass sich `systemd` in neueren Versionen von Fedora (seit Version 15) und OpenSuse findet. Weitere Distributionen die `systemd` als Standard oder wenigstens optional unterstützen sind Mandriva, Gentoo, Arch Linux und Debian Testing.

`systemd` orientierte sich an Neuentwicklungen wie `upstart` unter Ubuntu oder `launchd` unter Mac OS X.

Grundlegende Funktionsweise

Wie in der letzten Vorlesung dargestellt, werden im SysV-Init-System die Dienste eines Runlevels hintereinander (beendet oder) gestartet, in der Reihenfolge, wie es die Nummerierung der Links auf die Initskripte im Runlevel-Verzeichnis angibt.

Bei `systemd` hingegen können einzelne *Units* prinzipiell parallel gestartet werden, wobei durchaus innerhalb der Konfiguration einer Unit Abhängigkeiten zu anderen definiert werden können (z. B. dass eine andere Unit gestartet sein muss, oder ein Konflikt zu einer anderen Unit besteht).

Ein Hauptfeature von `systemd` ist, dass es einen Puffer für Kommunikationspfade zur Verfügung stellt, wie vom System benutzte Sockets oder den Syslog. Daher können Dienste gestartet werden, die Meldungen an den Syslog-Dämon schreiben, bevor dieser überhaupt vollständig gestartet ist!

Units und Targets

Die Konfiguration des Verhaltens von `systemd` wird durch die Units bestimmt, wobei jede Unit eine Konfigurationsdatei hat, die Ini-Dateien unter Windows ähnelt.

Hierbei gibt es verschiedene Arten von Units:

.service	entspricht einem Systemdienst (Dämon) bei SysV-Init
.mount	definiert Mountpoints
.socket	legt einen Socket an (dass dieser dann aber tatsächlich genutzt wird erfolgt wieder über einen Service)
.target	ist eine Gruppe von Units.

Insbesondere gibt es für die traditionellen Runlevels eine Entsprechung in Form eines Targets. Auch Targets können Abhängigkeiten untereinander haben, so hängt z. B. das `graphical.target` von `multi-user.target` ab.

Grundlegende Konfiguration

`systemd` benutzt zwei Verzeichnisbäume, nämlich `/lib/systemd` mit der Konfiguration von `systemd` durch die Distribution und `/etc/systemd`. Hierbei hat letzteres Vorrang, wodurch lokale Änderungen möglich sind, die nicht z. B. mit Updates durch die Distribution in Konflikt geraten.

Beim Booten wird das Target benutzt, auf das der Link `/etc/systemd/system/default.target` verweist (dies entspricht dem Eintrag in der Datei `inittab` bei SysV-Init).

Welche Services z. B. für das Target `multi-user.target` gestartet werden, wird durch symbolische Links im Verzeichnis `/etc/systemd/systemd/multi-user.target.wants/` bestimmt.

Beispiel

Für die Demonstration wird der Beispieldienst aus der letzten Vorlesung genutzt:

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

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

Beispiel (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);
    }
}
```

Klassischer Dienst wie in SysV-Init

Die Konfiguration für einen Dienst kompatibel zu SysV-Init (Hintergrundprozess und Schreiben PID-Datei) sieht dann so aus:

```
[Unit]
Description=Ping To Syslog Test Service
After=syslog.service

[Service]
Type=forking
PIDFile=/var/run/log.pid
ExecStart=/usr/sbin/log -b -p /var/run/log.pid

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

Normaler Dienst für systemd

Im Normalfall kümmert sich `systemd` selbst um das Bringen des Dienstes in den Hintergrung, ebenso ist die PID-Datei nicht notwendig. Daher sieht ein normaler Service so aus:

```
[Unit]
Description=Ping To Syslog Test Service
After=syslog.service

[Service]
ExecStart=/usr/sbin/log

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

Der Befehl `systemctl`

Unter `systemd` existiert der Befehl `systemctl`, der folgende traditionelle Befehle bei SysV-Init ersetzt:

- `service` zum manuellen Starten, Beenden, etc. von Diensten
- `chkconfig` zum Aktivieren eines Dienstes in bestimmten Runlevels
- `telinit` zum Wechseln des Runlevels

Im Allgemeinen werden die alten Befehle noch verstanden und auf die entsprechenden Aufrufe von `systemctl` gemappt.

Übergang SysV-Init nach systemd

sysvinit-Befehl	systemd-Befehl
Starten eines Service (manuell, nicht persistent):	
`service frobozz start`	`systemctl start frobozz.service`
Manuelles Beenden eines Service:	
`service frobozz stop`	`systemctl stop frobozz.service`
(Bedingter) Neustart eines Service:	
`service frobozz restart`	`systemctl restart frobozz.service`
`service frobozz condrestart`	`systemctl condrestart frobozz.service`
Neueinlesen der Konfiguration (ohne Unterbrechung):	
`service frobozz reload`	`systemctl reload frobozz.service`
Information, ob ein Service läuft:	
`service frobozz status`	`systemctl status frobozz.service`

Übergang SysV-Init nach systemd (Forts.)

sysvinit-Befehl	systemd-Befehl
Dienst beim Booten (de)aktivieren:	
`chkconfig frobozz on`	`systemctl enable frobozz.service`
`chkconfig frobozz off`	`systemctl disable frobozz.service`
Ist der Dienst beim Booten aktiv?	
`chkconfig frobozz`	`systemctl is-enabled frobozz.service`
Welcher Dienst läuft wo?	
`chkconfig --list`	`systemctl list-unit-files --type=service`
In welchen Targets läuft ein bestimmter Dienst?	
`chkconfig frobozz --list`	`ls /etc/systemd/system/*.wants/`

Dateisysteme

Ein Datenträger benötigt ein Dateisystem, dies ist unter Windows entweder FAT(32) oder NTFS.

Unter Linux steht eine Vielzahl von Dateisystemen zur Verfügung:

- ext3/ext4, häufig verwendete Journaling File Systems
- ext2/ext, veraltete Dateisysteme, ohne Journaling
- ReiserFS, Reiser4: ReiserFS war erstes Journaling File System unter Linux, Vorteil beim Speichern kleiner Dateien,
- XFS, CXFS von SGI,
- JFS von IBM,
- NFS, Netzwerk-Dateisystem, ursprünglich von SUN,
- AFS, Netzwerk-Dateisystem für verteilte System,
- OCFS, Oracle Cluster File System.

Wesentliche Eigenschaften von Dateisystememem

Maximale Größe Ergibt sich i. A. aus Blockgröße und Maximalzahl der Blöcke.

Inode-Konzept Speicherung von Metadaten.

Journaling Änderungen am Dateisystem werden vorher im so genannten *Journal* verzeichnet. Daher im Fall eines Systemabsturzes *Nachvollziehbarkeit* der Änderungen, keine volle Prüfung des Datenträgers auf Integrität nötig.
Unterscheidung: *Metadaten-Journaling* vs. *Full Journaling*.

Mountkonzept

- Klassisch: Mounten einer Partition in einem Verzeichnis als *Mount Point*, z. B. `/` (root file system), `/home`.
- Keine Laufwerksbuchstaben
- `/` als Verzeichnistrenner.
- Immer öfter Nutzung von Abstraktion zwischen Platte, Verzeichnis und Partition: *LVM*, Logical Volume Manager
- Mounten von verschiedenen Dateisystemen möglich.

Struktur der Partitionenen auf einer Platte

Die Platten und Partitionen der Platten in einem System werden unter Linux folgendermaßen bezeichnet:

- Festplatten vom Typ IDE (SATA) als `/dev/hda`, `/dev/hdb` usw. (es besteht hierbei die Möglichkeit, IDE/SATA-Festplatten wie SCSI-Festplatten zu behandeln)
- SCSI-Festplatten mit `/dev/sda`, `/dev/sdb` etc.
- Die einzelnen Partionen werden dabei durchnummeriert, also z, B. `/dev/hda1`
- Hierbei sind i. A. 1–4 primäre und ab 5 logische Partionen

Mounten und Formatieren

- Tools zum Anlegen, Manipulieren oder Löschen von Partionen: `cfdisk`, `sfdisk`
- Formatieren einer Partion mit einem Dateisystem z. B.
  `mkfs -t ext4 /dev/hda2`
  wobei dies auf das Programm `mkfs.ext3` gemappt wird.
- Mounten einer Partition in einen Mountpoint (verzeichnis) z. B.
  `mount -t ext3 /dev/hda5 /home`
- Definition der per Default zu mountenden Verzeichnisse in der Datei `/etc/fstab`

LVM

Klassisch wird auf einer Partition einer Festplatte ein Filesystem erzeugt und dieses in einem Mountpoint (Verzeichnis) in das System eingehängt.

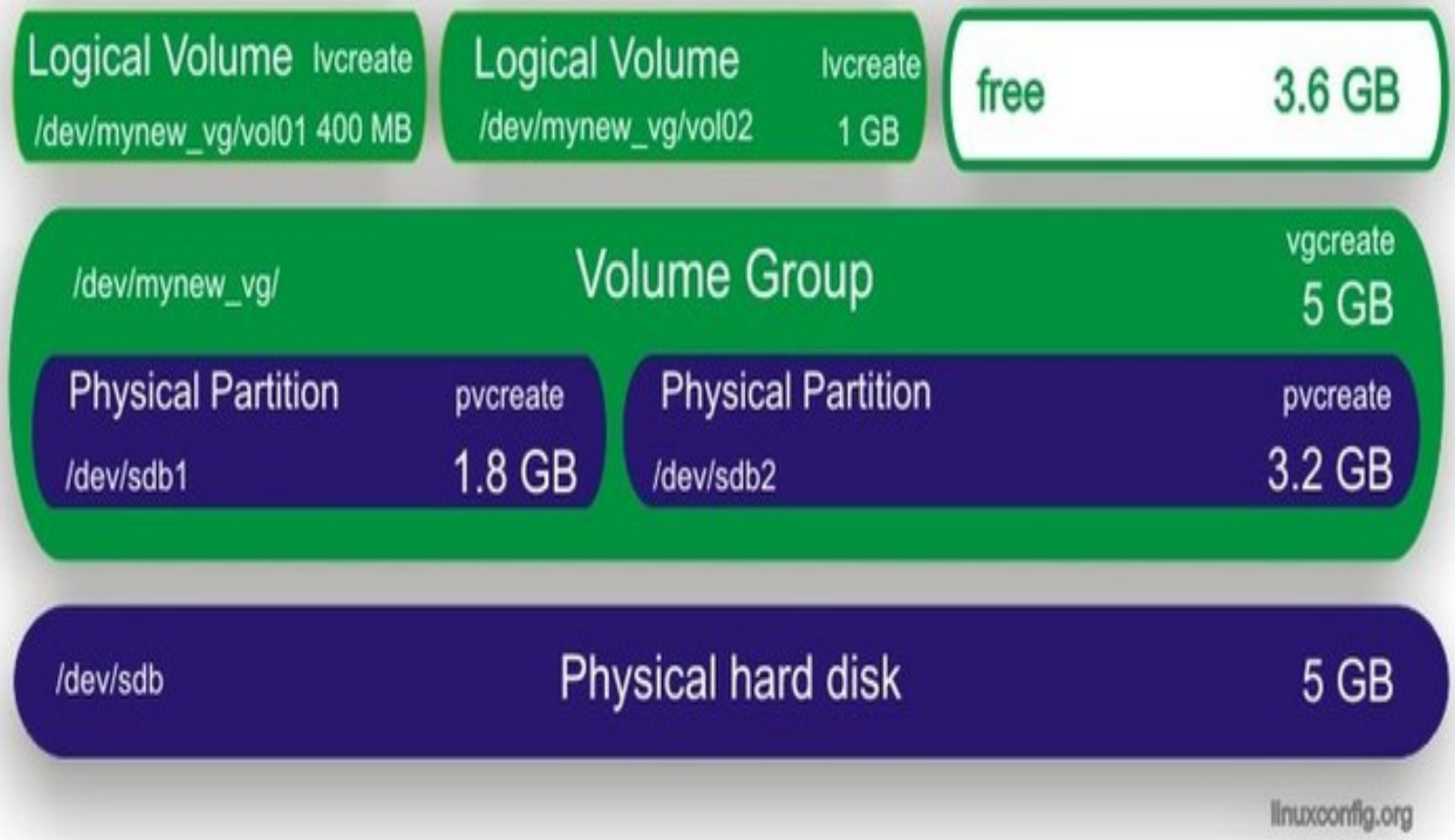
LVM, der *Logical Volume Manager* stellt eine Abstraktionsschicht dazwischen. Nun existieren:

Physical Volumes in Form einer Partition (oder einer ganzen Platte),

Volume Groups bestehend aus einem oder mehreren Physical Volumes,

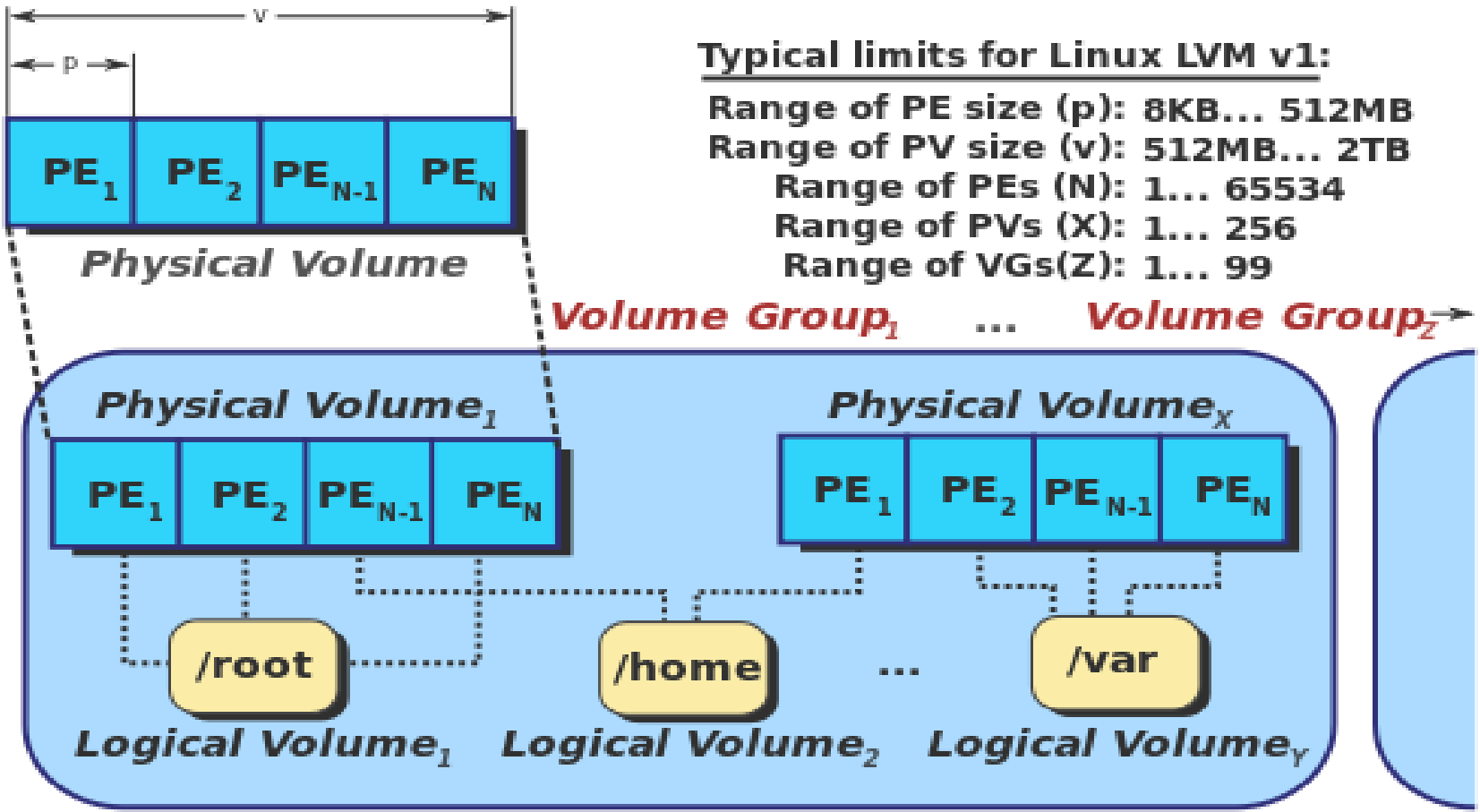
Logical Volumes die in einer Volume Group eingrichtet werden.

Schaubild


Logical Volume lvcreate
/dev/mynew_vg/vol01 400 MB
Logical Volume lvcreate
/dev/mynew_vg/vol02 1 GB
free 3.6 GB
/dev/mynew_vg/
Volume Group
vgcreate
5 GB
Physical Partition pvcreate
/dev/sdb1 1.8 GB
Physical Partition pvcreate
/dev/sdb2 3.2 GB
/dev/sdb
Physical hard disk
5 GB
linuxconfig.org

Grundbegriffe

PE: Physical Extend; PV: Physical Volume; VG: Volume Group; LV: Logical Volume



Befehle

Aktion	Phys. Vol.	Volume Group	Logical Vol.
Erzeugen	pvcreate	vgcreate	lvcreate
Entfernen	pvremove	vgremove	lvremove
Scannen	pvscan	vgscan	lvscan
Infos	pvdisplay	vgdisplay	lvdisplay
Vergrößern		vgextend	lvextend
Verkleinern		vgreduce	lvreduce
Verschieben	pvmove		