

udev

and the mysteries of /sys filesystem

Phil Sutter

Linux/Unix Usergroup Sankt Augustin
Nela, Bonn

14. Dezember 2006

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

Gliederung

- 1 **Benötigtes Vorwissen**
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

Gerätedateien

- liegen alle in `/dev`
- zwei abstrakte Typen: `char` und `block`
- Major–Nummer: Treiberidentifikation
- Minor–Nummer: Geräteidentifikation

Wozu braucht man das?

- für fast alle Geräte

Wozu braucht man es nicht?

- physikalische Netzwerkdevices
- neuere HBCI-Cardreader
- ...

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund**
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

Bis Linux-2.3.45: statisches /dev

- „If you say Y here and create a character special file . . . with major number . . . and minor number . . . you get read and write access to . . .”
- `mknod [OPTION]... NAME TYPE [MAJOR MINOR]`
- alternativ: `/dev/MAKEDEV`

Ab Linux-2.3.46: DevFS

- Kerneldienst (repräsentiert als Dateisystem)
- Verwendung von Major/Minor-Nummern (theoretisch) überflüssig
- Zugriffsrechte werden innerhalb des Treibers gesetzt

Seit Mitte 2006: udev

- Userspace Device Filesystem
- alles Weitere später

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)**
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

SysFS–Struktur

- Objekte werden durch Verzeichnisse repräsentiert
- Dateien enthalten Attribute eines Objekts
- Beziehungen unter Objekten durch Symlinks dargestellt
- wird vom Kernel verwaltet

Was sind SysFS–Objekte?

- Systembusse
- Geräteklassen
- Geräte
- Schnittstelle zur Firmware
- Kerneleinstellungen
- ...

SysFS–Funktionalitäten

- Kommunikation mit Geräten aus dem Userspace
- Auslesen von Geräteinformationen
- Lesen/Schreiben von Geräte/Kernel–Einstellungen

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`**
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

Trivialitäten

- Implementierung im Userspace
- `udev` läuft im Hintergrund
- konfigurierbar mittels `udev`-Regeln
- Dienst erkennt Regeländerungen (inotify?)

Initialisierungsprozess

- `/dev` als `tmpfs` mounten
- initiales Set von Gerätedateien anlegen
- `udev` starten
- mit `udevtrigger` Colplugging anstoßen

Geräteregistrierung

- 1 `udev` empfängt Device-Events direkt vom Kernel
- 2 passende `udev`-Regeln werden gesucht und ausgewertet
- 3 entsprechende Device-Nodes werden angelegt

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel

Zusammensetzung einer `udev`-Regel

- einzeilige, zweiteilige Anweisungen aus kommaseparierten Statements
- erster Teil: „Matches“
- zweiter Teil: „Actions“
- Nutzung von Wildcards und internen Umgebungsvariablen

Beispiele für Matches

ACTION „add“ oder „remove“

KERNEL Kernelname des Geräts

SUBSYSTEM Gerätesubsystem (z.B. „net“)

BUS Gerätebus (z.B. „scsi“)

Operatoren: „==“, „!=“

Beispiele für Actions

NAME setzt den Gerätenamen (z.B. „%k“)

SYMLINK legt symbolischen Link zur Datei an (z.B. „tts/%n“)

OWNER, GROUP, MODE Permissions setzen

RUN Befehl absetzen

Operatoren: „=“, „+ =“, „:=“

Beispiele für Regeln

Netzwerkdevice umbenennen

```
SUBSYSTEM=="net", DRIVERS=="?*", \  
ATTRS{address}=="00:0a:e4:2e:c5:26", NAME="e1000"
```

Dateirechte setzen

```
KERNEL=="nvram", NAME="%k", GROUP="nvram", \  
MODE="0660"
```

Programm ausführen

```
ACTION=="add", SUBSYSTEM=="usb", \  
ENV{PRODUCT}=="c4b/300/100", \  
RUN="/etc/init.d/chipcardd2 --quiet start"
```

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging**
- 7 Wir erstellen eine `udev`-Regel

udevinfo

- zeigt Informationen zu existenten Geräten an
- nach Pfad:

```
udevinfo --path=/sys/class/input/input2/js0 \  
--query=all
```

- nach Namen:

```
udevinfo --name=hda --query=all
```

udevtest

- simuliert einen `udev`-Lauf und printet Aktionen nach `stdout`
- Beispiel:

```
udevtest /sys/devices/platform/tpm_atmel
```

udevmonitor

- lauscht kontinuierlich auf `udev`-Events
- printet Kernelinformationen, SysFS-Attribute und Umgebungsvariablen

Gliederung

- 1 Benötigtes Vorwissen
- 2 Geschichtlicher Hintergrund
- 3 Das neue Gerätemodell (SysFS)
- 4 Die Funktionsweise von `udev`
- 5 `udev`-Regeln
- 6 Debugging
- 7 Wir erstellen eine `udev`-Regel**

Wir erstellen eine `udev`-Regel

- 1 `udevmonitor` anschmeißen
- 2 Gerät einstecken / Modul laden
- 3 eindeutiges Merkmal im Log-Output finden
- 4 neue `udev`-Regel anlegen