Das eigene Linux

Prüfungstermine

FHS

Auf Unix-ähnlichen Betriebssystemen findet man immer wieder die gleichen Verzeichnisse wie

`/usr`, `/home` oder `/tmp`

Tatsächlich ist die Verzeichnisstruktur gemäß dem **Filesystem Hierarchy Standard** (FHS) normiert.

FHS

`/bin`	Binärdateien grundlegender Befehle
`/boot`	statische Dateien des Bootloaders
`/dev`	Gerätedateien
`/etc`	Konfigurationsdateien und Skripte zur Konfiguration
`/lib`	grundlegende Bibliotheken
`/media`	Mountpunkt für Datenträger
`/mnt`	temporärer Mountpunkt für Dateisysteme
`/opt`	zusätzliche Anwendungsprogramme
`/sbin`	grundlegende Binärdateien für Administratoren
`/srv`	Daten für Systemdienste
`/tmp`	temporäre Dateien
`/usr`	Software, die nicht zur Unix-Grundausstatung gehört.

Eigene bin-

und lib- Verzeichnisse sind vorhanden.

/proc und /sys

In Linux-Systemen findet man ausserdem

`/proc`: Mountpunkt für das virtuelle Dateisystem `procfs`. Es dient zur Ausgabe und Änderung von System- und Prozessinformation. Es ist eine der Schnittstellen des Kernels.

Beispiel:

`/proc/filesystems`, liefert etwa eine Liste aller Dateisysteme, die der Kernel unterstützt.

`/sys`: Ähnlich wie `/proc` Mountpunkt für das virtuelle Dateisystem `sysfs`. Es enthält Informationen über Geräte und Treiber.

Dem FHS genügen

Wir können also auf unserer SD-Karte ein Root-Filesystem anlegen, das den FHS-Konventionen entspricht.

```
sudo mkdir
bin boot dev etc lib media mnt opt proc sbin srv sys tmp var
```

Das eigene Root-Filesystem

Es fehlt natürlich noch Software, um Prozesse wie `init` oder eine Shell in Betrieb zu nehmen. Hier gibt es mehrere Möglichkeiten

- Eine Standard-Distribution
- Eine eigene Distribution
- Busybox

Eine Standard-Distribution wie Raspbian haben wir genutzt, als wir angefangen haben mit dem Target zu arbeiten.
Eine eigene Distribution kann man selbst entwickeln oder mit Werkzeugen wie `buildroot` konfigurieren und konstruieren.
Busybox ist eine **spezielle Variante** einer eigenen Distribution.

Links

Bevor wir uns Busybox zuwenden, benötigen wir etwas Hintergrundwissen.

Links

Wir erzeugen eine neue Datei

```
echo "Hello World" > base.txt
```

Anschließend legen wir einen Link auf diese Datei an:

```
ln base.txt link1.txt
```

Der Befehl

```
cat link1.txt
```

zeigt den Inhalt von `base.txt` an. Änderungen an dieser Datei sind in `base.txt` *und* `link1.txt` sichtbar.

inode

Dateien werden vom Betriebssystem nicht unter ihrem Namen, sondern über eine ID den **inode** verwaltet.
Ein Dateiname ist nur ein Alias für einen inode.
Der Befehl `ls -i` macht inodes sichtbar:

```
659972 base.txt
659972 link1.txt
```

`base.txt` und `link1.txt` referenzieren also den gleichen inode.

inode

Löschen wir `base.txt`, wird der inode weiterhin über `link1.txt` referenziert:

```
rm base.txt
cat link1.txt
```

Liefert also `Hello World`.
Die Datei wird erst gelöscht, wenn der inode nicht mehr referenziert wird.

Symbolische Links

Da die inode-Tabelle zum Dateisystem gehört, können diese Hard-Links **nicht über Dateisystemgrenzen hinweg** genutzt werden. Hier helfen symbolische Links:

```
ln -s base.txt link1.txt
```

Ein `ls -i` zeigt uns, dass symbolische Links einen eigenen inode haben.

Symbolische Links

Die Konsequenz aus dem eigenen inode für symbolische Links sehen wir jetzt:

```
rm base.txt
cat link1.txt
```

liefert

```
cat: link1.txt: No such file or directory
```

Wenn `base.txt` gelöscht ist, verweist `link1.txt` auf einen Eintrag, den es nicht mehr gibt.
In der Praxis wird vorwiegend mit symbolischen Links gearbeitet.

Ein einfaches C-Programm

Busybox nutzt intensiv symbolische Links. Den Trick den Busybox nutzt, soll das folgende Programm `main.c` demonstrieren:

```
#include <stdio.h>

int main(int argc, char *argv[]){
  for(int i=0; i<argc; i++)
    printf("%s ",argv[i]);
  return 0;
}
```

Ein einfaches C-Programm

Wir übersetzen, binden und rufen es wie folgt auf

```
gcc main.c -o main
./main hello world
```

Die Ausgabe ist

```
./main hello world
```

Bei C-Programmen wird als erstes Argument automatisch der Programmname übergeben.

Ein einfaches C-Programm

Das funktioniert auch bei Links:

```
ln -s main ls
ln -s main cd
```

Wir rufen `main` jetzt wie folgt auf:

```
./ls hello
```

Es erscheint der Name des Links (und nicht `main`):

```
./ls hello
```

Was hat das mit Busybox zu tun?

Unser C-Programm weiss also, über welchen Link es aufgerufen wurde.

- Busybox implementiert alle wichtigen Linux-Befehle auf diese Weise in einem einzigen Programm.
- Alle Kommandos sind Links, auf die ausführbare Binärdatei **`busybox`**.
- Der Vorteil besteht darin, dass wir so eine sehr kompakte Implementierung aller wichtigen Befehle erhalten.
- Im Wesentlichen gibt es nur **eine ausführbare Datei** und nicht für jeden Befehl eine eigene ausführbare Datei.