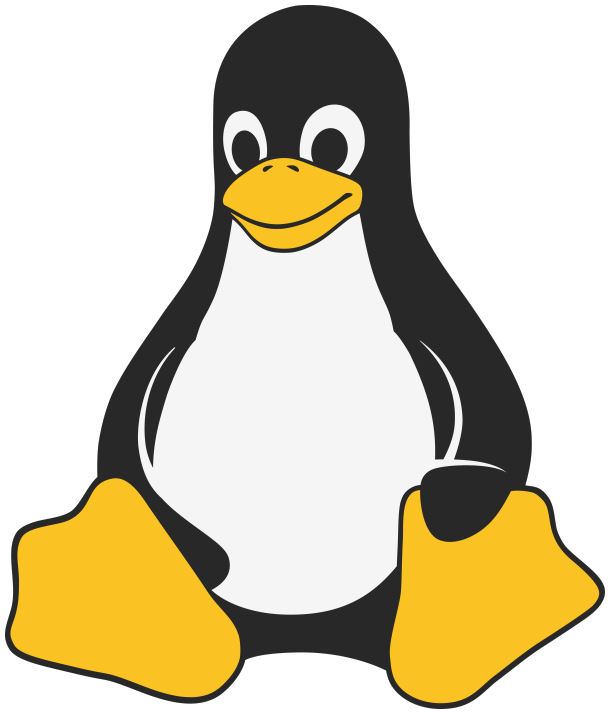
Dateitypen



Inhaltsverzeichnis

Unix-Dateikonzept

In Unix- und Linux-Systemen wird (fast) alles als Datei betrachtet - auch Hardware-Geräte, Kommunikationskanäle und selbst Prozesse.

Was bedeutet das? - Alle diese Objekte werden als Dateien behandelt und können mit denselben System-Aufrufen (system calls) für Dateioperationen bearbeitet werden. Dabei betrachtet der Kernel eine Datei einfach als einen strukturlosen Byte-Stream, also eine Sequenz von Bytes. (Bei Verzeichnissen verhält es sich etwas anders, dazu später mehr.)

Zum Inhaltsverzeichnis ...

Die Dateiinhalte sind aus Sicht des Kernels irrelevant. Der Kernel kümmert sich nicht darum, ob die Datei Text, HTML, PDF-Daten oder eine Bilddatei enthält. Um die Inhalte kümmern sich die Anwendungsprogramme. Sie müssen die Dateiinhalte interpretieren und entsprechend verarbeiten. Z.B. muss ein Foto-Viewer die Bilddaten interpretieren und anzeigen können. Er muss die Strukturen und Formate der verschiedenen Bilddateien (JPG, PNG etc.) kennen.

Eine Ausnahme bilden Verzeichnisse. Diese werden vom Kernel nicht als strukturlose Byte-Streams betrachtet. Sie enthalten eine definierte Struktur, die der Kernel kennt und verwaltet: Jeder Verzeichniseintrag enthält außer dem Namen des Eintrags auch alle Dateiattribute des Eintrags (z.B. Dateityp, Größe, Zugriffsrechte, Zeitstempel etc.).

Außerdem muss jedes Verzeichnis auf sich selbst (Eintrag `.`) und auf sein übergeordnetes Verzeichnis (Eintrag `..`) verweisen.

Mit diesen Informationen kann der Kernel den ganzen tief verschachtelten Dateibaum aufbauen und verwalten.

Grundlegende Dateioperationen (system calls) in Unix/Linux

- **open()** - öffnet eine Datei zum Lesen, Schreiben oder zum Anhängen
- **read()** - liest Daten sequentiell aus einer Datei (ab der aktuellen Position)
- **write()** - schreibt Daten sequentiell in eine Datei (ab der aktuellen Position)
- **seek()** - setzt die aktuelle Position in einer Datei
- **close()** - schließt eine Datei

Dateitypen - Übersicht

- `d` (directory) - Verzeichnis
- `-` (regular file) - reguläre Datei
- `l` (symbolic link) - symbolischer Link
- `c` (character device) - Zeichenorientiertes Gerät
- `b` (block device) - Blockorientiertes Gerät
- `p` (named pipe) - benannte Pipe
- `s` (socket) - Socket

Dateityp: Verzeichnis (d)

Verzeichnisse genießen eine Sonderstellung im Unix-Dateisystem. Sie sind (wie oben beschrieben) keine strukturlosen Byte-Streams, sondern das Struktur gebende Element des Dateisystems, mit der gesamte Dateibaum aufgebaut ist.

- Alle anderen Datei-Objekte (Dateien, Geräte etc.) sind Einträge in diesen Verzeichnissen.

Zum Inhaltsverzeichnis ...

Dateityp: Reguläre Datei (-)

Reguläre Dateien sind (aus Sicht des Kernels) strukturlose Daten-Container, die auf einem Massenspeicher (Festplatte, SSD, USB-Stick etc.) abgelegt sind. Nachdem eine Datei geöffnet wurde, steht der Dateiinhalt als sequentieller Byte-Stream zum Lesen oder Schreiben zur Verfügung.

Dateityp: Symbolischer Link (`l`)

Symbolische Links (auch: Softlinks) sind spezielle Dateien, die auf andere Dateien oder Verzeichnisse verweisen (analog zu Web-Links). Auch sie sind (kleine) Daten-Container, die einen kurzen Text-String enthalten, nämlich den Pfad des Objekts, auf das sie verweisen.

Der Kernel wertet den Inhalt des symbolischen Links aus und greift dann auf das Objekt zu, auf das der Link verweist. Für die Korrektheit des Links ist der Kernel nicht zusändig, sondern der Benutzer, der den Link erstellt. Zeigt ein Softlink "ins Leere" (broken link), dann meldet der Kernel einen Fehler, der besagt, dass das verwiesene Objekt nicht existiert.

Zum Inhaltsverzeichnis ...

Dateityp: Zeichenorientiertes Gerät (`c`)

Dieser Dateityp bezeichnet Geräte, die Daten zeichenweise (Byte für Byte) verarbeiten. Dies sind in erster Linie Terminals (TTYs und Pseudo-TTYs), aber auch alles andere, was an einem seriellen Port angeschlossen ist (z.B. Scanner, Drucker, Modems etc.).

Lesen vom Terminal bedeutet Lesen von der Tastatur, Schreiben auf das Terminal bedeutet Schreiben auf den Bildschirm. Die Gerätedatei eines physischen Terminals ist z.B. `/dev/tty1`, `/dev/tty2` etc. Die Gerätedatei eines virtuellen Terminals (Pseudo-TTY) ist z.B. `/dev/pts/0`, `/dev/pts/1` etc.

Zum Inhaltsverzeichnis ...

Ein Pseduo-TTY (PTY) ist ein virtuelles Terminal, das von einem Terminal-Emulator (z.B. `xterm`, `gnome-terminal`, `lxterminal` etc.) mit der Shell-Sitzung des Benutzers verknüpft wird. Ein Pseudo-TTY erhält man auch, nachdem eine SSH-Verbindung zu einem entfernten Rechner hergestellt wurde. Der Benutzer kann dann auf dem entfernten Rechner arbeiten, als säße er direkt davor (so als säße er an einem Terminal).

Dateityp: Blockorientiertes Gerät (b)

Blockorientierte Geräte sind Geräte, deren Daten blockweise verarbeitet werden. Dazu gehören alle Massenspeicher (Festplatten, SSDs, USB-Sticks, CD/DVD-Laufwerke, Diskettenlaufwerke etc.) Der Kernel liest und schreibt Daten in Blöcken von diesen Geräten und auf sie. Die Blockgröße ist gerätespezifisch und kann zwischen 512 Byte und 16 KByte liegen.

Dateityp: Named Pipe (p)

Named Pipes (auch FIFOs genannt) sind spezielle Dateien, die der unidirektionalen Interprozesskommunikation (IPC) dienen. Sie ermöglichen den Datenaustausch zwischen Prozessen, die nicht miteinander verwandt sind. Der Zugriff wird nur über die Datei-Rechte der Named Pipe geregelt.

Das Funktionsprinzip ist dasselbe wie bei den anonymen Pipes, die uns als Shell-Benutzer geläufig sind. Beide arbeiten nach dem FIFO-Prinzip (First In, First Out). Der Unterschied besteht darin, dass Named Pipes einen Dateinamen haben und damit über das Dateisystem zugänglich sind.

Anonyme Pipes haben keinen Dateinamen und werden von der Shell ad hoc erzeugt, wenn zwei Prozesse mit einem Pipe-Operator (|) verbunden werden. Auch sie arbeiten unidirektional nach dem FIFO-Prinzip. Sie haben eine Schreibseite und eine Leseseite.

Dateityp: Socket (s)

Unix-Domain-Sockets sind spezielle Dateien, die der Kommunikation zwischen lokalen Prozessen dienen. Sie funktionieren ähnlich wie Netzwerk-Sockets. Netzwerk-Sockets ermöglichen die Kommunikation zwischen Prozessen, die auf verschiedenen Rechnern laufen. Sie verwenden IP-Adressen und Portnummern als Adressen. Unix-Domain-Sockets verwenden die Socket-Dateien als Adressen. Die Socket-Dateien sind nur auf dem lokalen Rechner sichtbar und zugreifbar. Deshalb können sie (ähnlich wie Named Pipes) nicht zur Kommunikation im Netzwerk, sondern nur zur lokalen Interprozesskommunikation verwendet werden.

Experimente mit Named Pipes (FIFOs)

- Erzeugen einer Named Pipe (FIFO) mit dem Kommmando `mkfifo`
- Ein Prozess schreibt in die FIFO mit einer Ausgabe-Umlenkung.
- Ein anderer Prozess liest aus der FIFO mit einer Eingabe-Umlenkung.
- Macht man das Experiment in einer Terminal-Sitzung, dann muss man den Reader- oder den Writer-Prozess im Hintergrund starten. Der andere Prozess kann dann im Vordergrund gestartet werden.
- Man kann das Experiment auch in zwei Terminal-Sitzungen durchführen. Dann können beide, Reader und Writer, im Vordergrund laufen.

Writer im Hintegrund, Reader im Vordergrund

```
hermann@debian:~/my-tests$ mkfifo myFifo # create a FIFO
hermann@debian:~/my-tests$ ls -l myFifo
prw-rw-r-- 1 hermann hermann 0 Nov  7 16:14 myFifo
hermann@debian:~/my-tests$ fortune > myFifo & # write to the FIFO in bg
[1] 185539
hermann@debian:~/my-tests$ cowsay < myFifo # read from the FIFO in fg
 ___________________________________
/ You are capable of planning your \
\ future.                          /
 -----------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[1]+  Fertig                    fortune > myfifo
hermann@debian:~/my-tests$
```

Reader im Hintegrund, Writer im Vordergrund

```
hermann@debian:~/my-tests$ cowsay < myFifo & # read from the FIFO in bg
[1] 185540
hermann@debian:~/my-tests$ fortune > myFifo # write to the FIFO in fg
 ___________________________________
/ Be cheerful while you are alive. \
|                                   |
\ -- Phathotep, 24th Century B.C.   /
 -----------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[1]+  Fertig                    cowsay < myfifo
hermann@debian:~/my-tests$
```

Key Takeaways

- Das Unix-Dateikonzept behandelt (fast) alles als Datei.
- Verzeichnisse sind das Struktur gebende Element des Dateisystems.
- Alles andere (Dateien, Geräte, Kommunikationsmechanismen) sind Einträge in diesen Verzeichnissen. Sie werden als Dateien behandelt und können mit denselben System-Aufrufen (system calls) für Dateioperationen bearbeitet werden. Der Kernel betrachtet sie als strukturlose Byte-Streams.

- Unix-Dateitypen:
  - Verzeichnis (`d`) - Struktur gebendes Element des Dateisystems
  - Reguläre Datei (`-`) - strukturloser Daten-Container
  - Symbolischer Link (`l`) - Verweis auf eine andere Datei oder ein Verzeichnis
  - Zeichenorientiertes Gerät (`c`) - Geräte, die zeichenweise gelesen und geschrieben werden
  - Blockorientiertes Gerät (`b`) - Geräte, die blockweise gelesen und geschrieben werden

- Named Pipe (`p`) - Unidirektionaler Kommunikationskanal für die Interprozesskommunikation
- Socket (`s`) - Bidirektionaler Kommunikationskanal für die Interprozesskommunikation