

# **Pro Seminar Linux**

**Sommersemester 2002**

**Thema: Dateisysteme unter Linux**

**Christoph Berlin**

# Inhaltverzeichnis

Vorwort .....	3
VFS – Virtual File System .....	4
interessantere Dateisysteme .....	5
Das EXT2 Dateisystem .....	6
Das Blocksystem – die grundlegende Struktur eines UNIX Dateisystems .....	6
Der Bootblock .....	7
I-Nodes .....	7
Optimierungen im Linux-Dateisystem EXT2 .....	9
Warum Gruppen? .....	10
EXT2 – Umgang mit dem Dateisystem .....	12
Dateinamen.....	12
Zugriffsdatum.....	12
Eigentümer .....	12
Gruppenzugehörigkeit.....	12
Anzahl der Dateinamen.....	13
Zugriffsmodus und Dateiart .....	13
Das Mounten von Dateisystemen.....	13
Die Überprüfung beim Systemstart.....	14
Dateiarten unter Unix .....	14
Reguläre Dateien .....	14
Symbolische Links .....	15
Verzeichnisse .....	15
Blockorientierte Geräte .....	15
Zeichenorientierte Geräte .....	15
Schlusswort .....	16
Quellen: .....	17

## Vorwort

Ein Dateisystem eignet sich dafür, die Aufnahme von Dateien auf Speichern (zum Beispiel Festplatten, Disketten oder CD-ROMs) zu ermöglichen. Das System stellt ein gemeinsames Interface für alle Dateisystemtypen bereit, mit denen es arbeiten kann. Unter Linux gehören das Second Extended Filesystem (oder *ext2fs*) dazu, unter dem man Linux-Dateien ablegen kann. Die Anforderungen der Einfachheit, der Sicherheit und der Erweiterbarkeit stehen im Vordergrund und dadurch eignet sich nicht jedes Dateisystem zum Standard. Das Extended Filesystem 2 hat sich in den letzten Jahren dank der Unterstützung dieser Eigenschaften zum Standard durchgesetzt. Der Aufbau, die Funktionsweise und die Schnittstelle zum Betriebssystem (VFS) werden in dieser Ausarbeitung dargestellt.

## VFS – Virtual File System

Ein großer Vorteil des Linux-Dateisystems, gegenüber anderen Betriebssystemen, ist es, dass viele verschiedene Dateisysteme unterstützt werden. Der Linux Kernel ist so modular aufgebaut, dass es, ab der Version 1.1.8, möglich ist, neue Dateisysteme zur Laufzeit einzubinden.

Die Dateisysteme, wie das EXT2, werden von dem Virtual Filesystem Switch (VFS) verwaltet.

Das VFS dient als Schnittstelle zwischen den speziellen Dateisystemen und den zugreifenden Prozessen. Innerhalb des virtuellen Dateisystems werden die Systemaufrufe der Applikationen an das eigentliche Dateisystem weitergeleitet.

Die Idee, die dem *virtual file system* (VFS) zugrunde liegt, besteht darin, dateisystemunabhängige Datenstrukturen zur Repräsentation von Dateisystemtypen, Superblöcken, Inodes und Dateien zu schaffen.

Jede der Datenstrukturen der VFS-Schicht enthält einen Zeiger auf eine Struktur, die nur aus Funktionszeigern besteht. Diese Funktionszeiger definieren den dateisystemspezifischen Satz von Operationen, die auf der betreffenden Datenstruktur anwendbar sind. Das VFS enthält den allgemeinen Code, der mit Superblöcken, Inodes und Dateien umgeht. Wann immer in diesem Kontext dateisystemspezifische Operationen notwendig sind, wird über die mit der Datenstruktur assoziierten Funktionszeiger die zugehörige spezialisierte Funktion aufgerufen.

Wenn dem Kernel ein Dateisystemtyp bekannt ist, kann ein Dateisystem dieses Typs aktiviert werden, indem sein Superblock mit Hilfe der durch den Funktionszeiger `read_super` bezeichneten Funktion eingelesen wird. Das Ergebnis dieses Funktionsaufrufes ist ein initialisierter Superblock, der neben typunabhängigen Informationen auch die für diesen Dateisystemtyp spezifischen Informationen enthält. Linux verwaltet die Superblöcke aller angemeldeten Dateisysteme unabhängig von ihrem Typ in einem Feld `super_block`. Der Dateisystemtyp eines Superblocks ist über den in ihm enthaltenen Rückwärtsverweis `s_type` leicht feststellbar. Ein Dateisystem kann deaktiviert werden, indem man über den Zeiger `put_super` die spezifische Deaktivierungsfunktion aufruft.

Durch diese Schicht zwischen den Anwendungen und Dateisystem ist es also möglich, mit einem Befehlssatz alle vorhandenen Systeme universell anzusprechen und mit ihnen individuell zu arbeiten

Beispiel: `rm`, `cp`, `mkdir` und andere Befehle sind immer einsetzbar und werden in die typabhängigen Funktionen übersetzt.

Um ein neues Dateisystem dem VFS bekannt zu machen, muss dieses mit der Funktion `register_filesystem()` hinzugefügt werden. Diese Funktion muss mit den entsprechenden Parametern, von der Initialisierungsroutine des neuen Filesystems aufgerufen werden. Die Strukturen des neuen FS werden in einer einfach verketteten Liste abgelegt.

## Einige der interessanteren Dateisysteme fasst die folgende Tabelle zusammen:

- **bfs** UnixWare Boot Filesystem
- **ext** Der Vorgänger des ext2, kaum mehr in Gebrauch
- **ext2** Das Standarddateisystem von Linux
- **ext3** Weiterentwicklung von ext2 mit zusätzlichen Funktionen eines Journaling File Systems, allerdings ist die Entwicklung noch nicht abgeschlossen
- **iso9660** Alle CD-ROMs speichern ihre Daten unter diesem Format
- **hpfs** OS-2 (nur lesend) loopback Mouneten einer einzelnen Datei als Dateisystem
- **minix** Minix ist der eigentliche Vorgänger von Linux, sein Dateisystem wird gern noch für (Installations-) Disketten eingesetzt
- **msdos** DOS, wichtig für den Zugriff auf DOS-formatierte Disketten
- **ncpfs** Novell Netware
- **nfs** Network File System
- **ntfs** WindowsNT, schreiben ist auf dieses Dateisystem zwar möglich, jedoch kann dies leicht zu irreparablen Schäden führen
- **ramdisk** Mouneten eines Bereichs des Hauptspeichers als Dateisystem
- **SMB** (Server Message Block) NT, Windows für Workgroups; zum Zugriff auf von einem Windows-Rechner freigegebene Verzeichnisse
- **swap** Swap-Partitionen oder -Dateien
- **SystemV** Verschiedene Unixe
- **proc** Prozessverwaltung
- **reiserfs** Eines der ersten Journaling File Systeme für Linux
- **umsdos** DOS-Dateisystem unter Linux verwenden, hier wird vor allem ein Mapping der unterschiedlichen Dateinamensformate und -rechte vorgenommen
- **vfat** Windows95, sowohl in der 16 bit als auch 32 bit Variante

## Das EXT2 Dateisystem

Das Extended Filesystem gilt als der Standard in der Linuxwelt. Durch seine Stabilität und Anpassungsfähigkeit eignet es sich für kleine Heimcomputer wie auch für große Serversysteme. Zum heutigen Zeitpunkt mit dem verfügbaren Kernel 2.4.x kann man im ext2, so die Abkürzung des Extended Filesystem 2, eine maximale Dateigröße von 4Gigabyte erreichen. Wie sich diese Größe zusammensetzt, lässt sich später in Verbindung mit der Struktur des Dateisystems einfach erklären. Weitere Möglichkeiten, das System unanfällig gegen Ausfälle zu machen, ist z.B. die Reservierung von Speicherplatz zur Vermeidung eines „Overflow“. D.h. es wird eine festgelegte Größe von Speicherplatz nicht zur Verfügung gestellt, damit im Fall, dass sehr viele Daten geschrieben werden und das Speichermedium ausgelastet ist, kein Systemausfall passieren kann.

### Das Blocksystem – die grundlegende Struktur eines UNIX Dateisystems



### Der Superblock

Ein Unix-Dateisystem besitzt einen so genannten Superblock, einen Block, der die grundlegenden Informationen zum Dateisystem selbst enthält. Einige wichtige Daten des Superblocks sind:

- Die Größe des Dateisystems in Blöcken
- Die Größe der Blöcke in Bytes
- Zeiger auf den ersten freien Datenblock
- Zeiger auf erste freie I-Node
- Verschiedene Statusbits (Flags)

Auch hier unterscheiden sich die verschiedenen Unix-Dateisysteme voneinander, was an zusätzlichen Informationen im Superblock gespeichert ist. Das wesentliche an dieser Struktur ist, dass der Superblock beim Mounten eines Dateisystems in den Speicher gelesen wird und alle Veränderungen dort vorgenommen werden. Es werden dann die Daten sporadisch gecached und beim Unmounten des Dateisystems werden diese Veränderungen physikalisch auf der Platte gespeichert. Das erklärt auch, dass es nach einem Systemabsturz zu Inkonsistenzen in einem Dateisystem kommen kann.

## Der Bootblock

Noch vor dem Superblock steht der Bootblock auf der Festplatte. Der erste Block jeder Partition kann einen Bootloader (zum Start eines Betriebssystems) enthalten, er wird beim Einschalten des Rechners vom BIOS gelesen. Dieser Block existiert bei allen Dateisystemen. (siehe Referat „Bootkonzepte“)

## Inodes

Unix-Dateisysteme sind nach einem anderen Prinzip aufgebaut, als etwa DOS-FAT-Systeme. Es gibt zwar viele verschiedene Dateisysteme unter Unix, gemeinsam haben sie jedoch eben das Prinzip der Funktionsweise. Und dieses Prinzip beruht auf den so genannten Inodes.

Jede Partition enthält ein Dateisystem, dieses Dateisystem wiederum enthält eine Art Inhaltsverzeichnis, die *I-Node-Liste*. Die einzelnen Elemente der *I-Node-Liste* sind die Dateiköpfe, also die Orte wo Dateiattribute, Größe usw. gespeichert sind. Diese Dateiköpfe werden *Inodes* genannt.

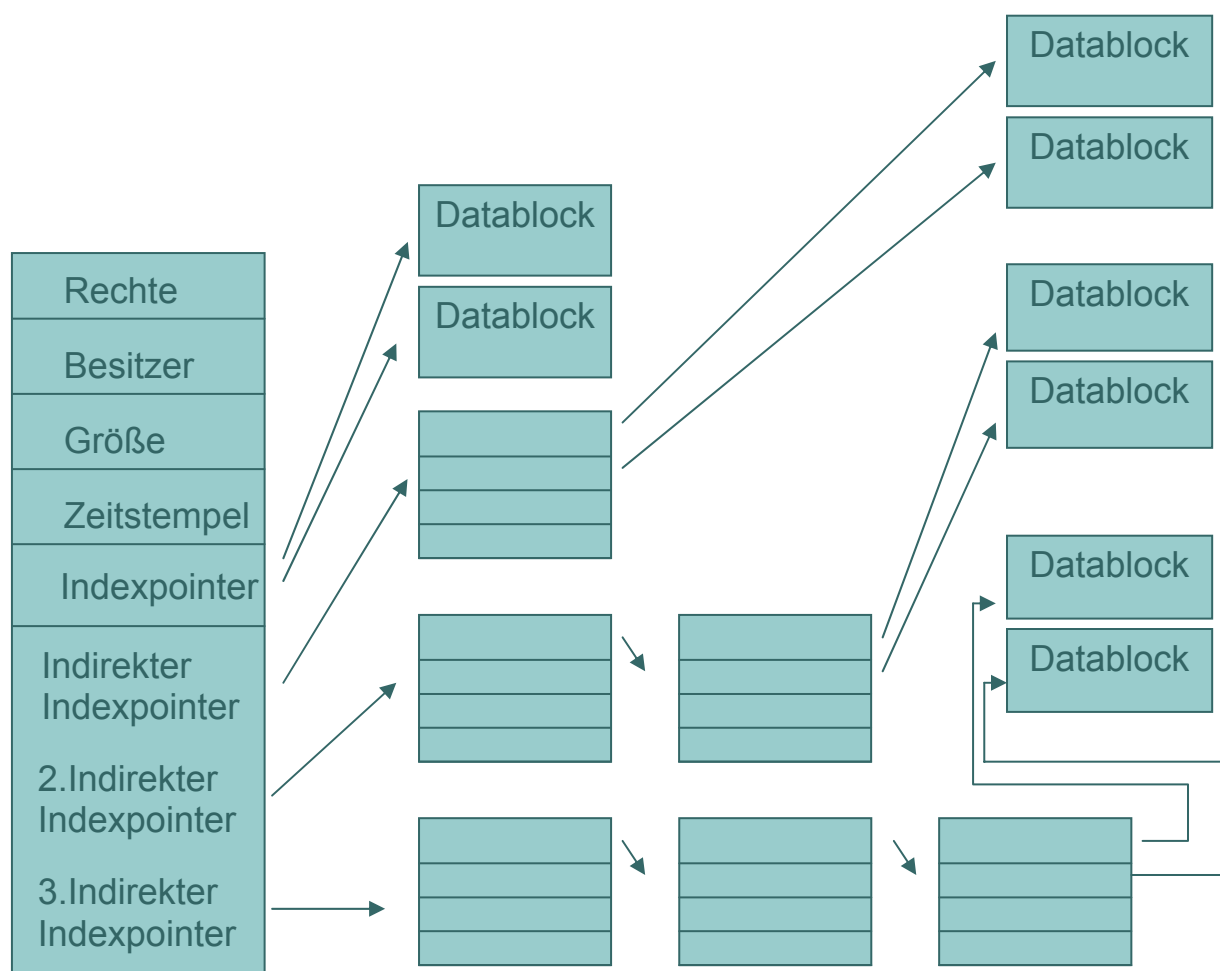
Wie unter DOS auch, so verwalten Unix-Dateisysteme nicht zwangsläufig die Sektoren einer Festplattenpartition sondern Zuordnungseinheiten, die hier aber nicht Cluster sondern *Block* heißen. Beim Anlegen eines Dateisystems kann die Blockgröße angegeben werden, die auf dieser Partition verwendet werden soll. Typische Blockgrößen sind 512, 1024 oder 2048 Byte. Voreingestellt sind meist 1024 Byte pro Block.

Anders als unter DOS werden diese Blöcke aber nicht in einer Tabelle (FAT) zusammengefasst, sondern die Inodes enthalten selbst die Verweise auf diese Blöcke. Das Format eines typischen Inodes sieht etwa so aus:

Typ und Zugriffsrechte
Anzahl der Hardlinks
Benutzernummer (UID)
Gruppennummer (GID)
Größe der Datei in Bytes
Datum der letzten Veränderung (mtime)
Datum der letzten Statusänderung (ctime)
Datum des letzten Zugriffs (atime)
Adresse von Datenblock 0
...
Adresse von Datenblock 12
Adresse des ersten Indirektionsblocks
Adresse des Zweifach-Indirektionsblocks
Adresse des Dreifach Indirektionsblocks

Nach den Datumsfeldern stehen zwölf Felder, die direkt die Adressen der Datenblöcke beinhalten können. Benutzt die Datei weniger Platz sind die Felder einfach leer.

Ist die Datei größer als 12 Blöcke (also größer als  $13 \cdot 1024$  oder  $13 \cdot 2048$  oder  $13 \cdot 512$ ), so enthält das nächste Feld der I-Node eine Adresse eines Blockes, der wiederum bis zu 128 Adressen anderer Blöcke enthalten kann. Sollte das auch noch nicht ausreichen, so enthält der nächste I-Node-Eintrag eine Adresse eines Zweifach-Indirektionsblocks, eines Blocks, der bis zu 128 Adressen auf Blöcke mit wiederum 128 Adressen enthält. Und falls auch das noch zu wenig sein sollte, so enthält der nächste Eintrag die Adresse eines Blockes, der wiederum 128 Adressen von Zweifach-Indirektionsblöcken enthalten kann. Mit dieser Möglichkeit der 3fach-Indizierung sind also, wie oben besprochen, Dateigrößen von bis zu 4Gigabyte (je nach Blockgröße von 512, 1024, 2048 oder 4096 Byte) möglich.



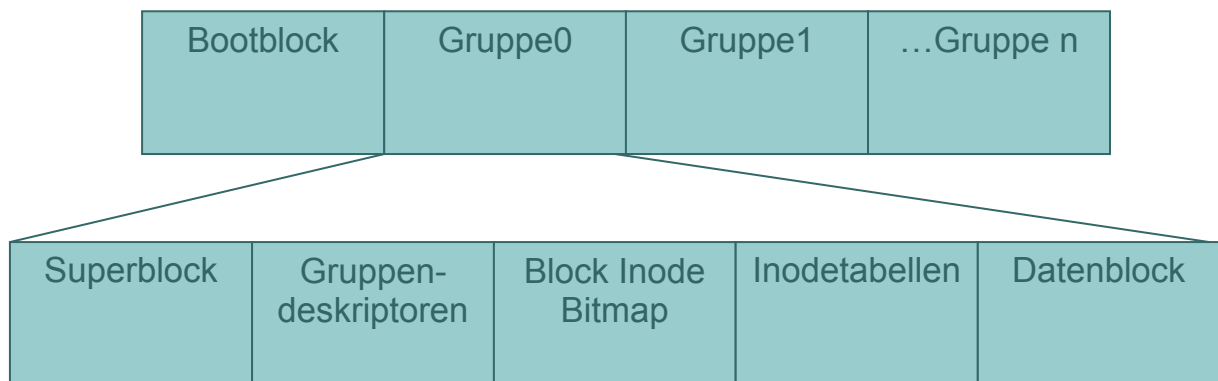
Zu beachten ist, dass der Dateiname nicht in der I-Node auftaucht. Die I-Node ist sozusagen nur die Referenz auf die Datenblöcke, die eine Datei benutzt und der Ort, an dem die Attribute wie Eigentümer, Gruppe, Größe und Zugriffsdaten gespeichert sind.

Je nach verwendetem Dateisystem liegt das Wurzelverzeichnis auf einer festgelegten I-Node, meist 1 oder 2. Grundsätzlich ist es aber dem Dateisystem bekannt, welche I-Node das Wurzelverzeichnis enthält.

Jedes Verzeichnis (Ordner, Directory) - auch das Wurzelverzeichnis ist unter Unix nichts anderes als eine Datei, deren Inhalt einfach die Dateinamen der enthaltenen Dateien samt ihren I-Node-Nummern enthält. Damit wird auch klar, warum Unix kein Problem mit Hard-Links hat, also mit Dateien mit mehreren Namen. Es handelt sich ja nur um verschiedene Namenseinträge, die eben dieselbe I-Node-Nummer besitzen.

Das Standard Linux-Dateisystem ext2 hat zusätzlich zu den gezeigten I-Node Einträgen noch verschiedene andere, die das System in mancherlei Hinsicht noch leistungsfähiger macht. Für den Anwender ergeben sich dadurch keinerlei Veränderungen, in der Praxis sind aber einige positive Effekte feststellbar.

## Optimierungen im Linux-Dateisystem EXT2



Scheinbar besitzt das ext2 keinerlei Ähnlichkeit mit herkömmlichen Unix-Dateisystemen. Abgesehen vom Bootblock, aber über diesen verfügt bekanntlich jedes Dateisystem. Jedoch erscheinen bei einem direkten Vergleich beider Strukturen n Gruppen mit dem ursprünglichen Unix-Blocksystem. Diese Gruppen beinhalteten die Struktur aller weiteren Blöcke, wie Super,- Inode- und Datenblock. Zusätzlich existieren aber noch weitere Blöcke mit den Bezeichnungen: Gruppdeskriptor und Inode Bitmap Block.

<b>Superblock</b>	Die Superblöcke der einzelnen Gruppen sind exakte Kopien des Superblockes aus Gruppe 0. Er enthält in einer 1024 Bytes großen Struktur alle wichtigen Informationen zum Dateisystem, wie die Anzahl von Inodes und Datenblöcken (gesamt/frei), Blockgröße der Datenblöcke (ein Inode ist immer 128 Byte groß), Zeitpunkt des letzten Mountens, Zähler der Anzahl der Mountvorgänge, Status, Zeit der letzten Überprüfung usw.
-------------------	---

<b>Gruppendeskriptoren</b>	Hier wurde zusätzliche Sicherheit eingebaut. Die Deskriptoren enthalten alle notwendigen Informationen zu allen anderen Gruppen, um diese nach einem Defekt restaurieren zu können. Sind in einer Gruppe also die Verwaltungsinformationen (Superblock, Gruppendeskriptoren oder Bitmaps) zerstört worden, können diese repariert werden und die Daten sind weiterhin verfügbar.
<b>Bitmaps</b>	Sie dienen dem schnellen Auffinden von freien Inode/Datenblöcken. Ihre Größe entspricht der Datenblockgröße und beschränkt die Anzahl der Datenblöcke einer Gruppe (bei einer Blockgröße von 4096 Bytes sind es somit 32768).
<b>Inodetabelle Datenblöcke</b>	/ Wie oben.

## Warum Gruppen?

Der entscheidende Performanceverlust auf heutigen Festplatten ist Bewegung der Schreib-Lese-Köpfe. Die gewählte Aufteilung stellt nun sicher, dass die Datenblöcke relativ nahe bei den zugehörigen Inodes liegen. Also muss der Schreib-Lese-Kopf nicht ständig bewegt über große Entfernungen bewegt werden. Darüber hinaus wird ebenso versucht, die Inodes der Dateien nahe dem zugehörigen Verzeichnis-Inode zu halten.

## Löcher in Dateien

Ist der Verweis auf einen Datenblock null, obwohl der referenzierte Block einen Teil des Dateiinhaltes repräsentieren müsste, so handelt es sich dabei um ein so genanntes Loch in der Datei. Beim Lesen dieses Bereiches muss für den gesamten Block null zurückgegeben werden. Auf diese Weise ist möglich, dass große Dateien, die in weiten Bereichen nur Nullen enthalten, physisch nur wenige oder sogar keine Datenblöcke benötigen. Die tatsächliche Zahl der von dem Inodes benutzten Datenblöcke kann also gegenüber der scheinbaren Dateigröße variieren.

## Blockallokierung

Neue Datenblöcke werden immer in der Nähe des Vorgängerblocks gesucht. Wurde für den Inodes noch kein Datenblock allokiert, so sucht man in der Nähe des InodeTable-Blocks der den Inode enthält. Sollte der Vorgängerblock ein Loch sein, so sucht man solange den Vorgänger des Vorgängers bis man einen Vorgängerblock gefunden hat, oder bis es keinen mehr gibt. Im Allgemeinen werden 32 Blöcke vor und hinter dem Zielblock durchsucht. Wurde dort kein freier Block gefunden versucht man zumindest einen freien Block in derselben Blockgruppe zu allokiert. Ist auch dies nicht möglich, so wird in allen anderen Blockgruppen gesucht.

## **Preallokation**

Wurde ein freier Block gefunden, so kann man mehrere (i.a. bis zu acht) der folgenden Blöcke reservieren, sofern diese frei sind. Auf diese Weise liegen die Datenblöcke einer Datei in relativer Nähe zueinander. Wird die Datei geschlossen, werden die unbenutzten reservierten Blöcke wieder freigegeben.

# EXT2 – Umgang mit dem Dateisystem

## Dateinamen

Die ersten wesentlichen Unterschiede sind schon im Dateinamen erkennbar. Die maximale Länge eines Unix-Dateinamens ist auf 256 Zeichen beschränkt. Ein wesentlicher Unterschied zu DOS/Windows Dateinamen ist die Tatsache, dass unter Unix streng zwischen Groß- und Kleinschreibung unterschieden wird. Das heißt die Dateinamen

Datei1.txt  
datei1.txt  
DATEI1.TXT

bezeichnen                      drei                      unterschiedliche                      Dateien.

## Zugriffsdatum

Im Gegensatz zu DOS kennt Unix nicht nur ein Datum pro Datei, sondern drei. Das erste bezeichnet den letzten schreibenden Zugriff, also das Datum aus dem ersichtlich wird, wann die Datei das letzte Mal verändert wurde. Dieses Datum wird mtime (Modification-Time) genannt.

Das zweite Datum bezeichnet den letzten Zugriff überhaupt, also auch lesende Zugriffe werden hier bemerkt. Dieses Datum wird atime (Access-Time) genannt.

Das dritte Datum registriert schließlich die letzte Statusänderung, also z.B. Wechsel der Zugriffsrechte, des Eigentümers, der Gruppe usw. Dieses Datum heißt ctime (Change-Time) Natürlich kann dieses Datum nur funktionieren, wenn der Datenträger nicht als ReadOnly gemountet ist. Auf CD-Roms etwa kann es logischerweise nicht funktionieren.

## Eigentümer

Jede Datei gehört einem User. In der Regel sind die Eigentümer die User, die eine Datei angelegt haben. Die Systemdateien gehören meist dem Systemverwalter oder bestimmten Verwaltungusern. Im Dateisystem ist nicht der Username sondern die numerische UserID gespeichert.

## Gruppenzugehörigkeit

Jede Datei gehört genau einer Gruppe an. Diese Gruppenzugehörigkeit kann beliebig variiert werden. Es ist nicht zwangsweise nötig, dass der Eigentümer einer Datei auch Mitglied der Gruppe sein muss, auch wenn es in der Praxis meist der Fall ist. Die Gruppenzugehörigkeit spielt eine große Rolle bei der Frage der Zugriffsrechte auf eine Datei.

## Anzahl der Dateinamen

Unter Unix können Dateien mehrere Dateinamen haben, so genannte Hard-Links.

## Zugriffsmodus und Dateiart

Jede Datei hat einen so genannten Zugriffsmodus, der beschreibt, um was für eine Art Datei es sich handelt, wer was damit anstellen darf. Dazu wird die 16 Bit-Zahl, die diese Zusammenhänge darstellt, in fünf Werte (einmal 4 Bit für den Dateityp, viermal 3 Bit für die Rechte) aufgeteilt.

Die wichtigsten Werte davon werden durch das `ls -l` Kommando dargestellt. Dabei werden folgende Abkürzungen benutzt:

Numerisch ausgedrückt könnten wir die Rechte folgendermaßen darstellen:

<code>rw-</code>	=	$4+2+0$	=	6
<code>r--</code>	=	$4+0+0$	=	4
<code>---</code>	=	$0+0+0$	=	0

Der numerische Wert des gesamten Zugriffsmodus in Oktalziffern ist also 640.

Für Programme gibt es noch eine vierte Angabe, die sich auch mit den Oktalziffern 1,2 und 4 bzw. aus einer Kombination aus diesen Ziffern ergibt. Diese vierte Angabe wird VOR den drei üblichen Angaben gemacht, streng genommen hätte also die Datei aus dem letzten Beispiel die numerische Darstellung 0640. Diese weitere Angabe wird später, beim Usersystem genauer beschrieben.

## Das Mounten von Dateisystemen

Im Gegensatz zu anderen Betriebssystemen fasst ein Unix-System grundsätzlich alle Partitionen und angeschlossenen Laufwerke zu einem Dateibaum zusammen. Dadurch entfallen die lästigen Laufwerksbuchstaben, die in größeren Netzwerken schnell an die Grenze der Übersichtlichkeit führen.

Dieses Prinzip erfordert logischerweise eine besondere Partition, nämlich die, die die eigentliche Wurzel des ganzen großen Dateibaums enthält. Diese Partition wird als Wurzelpartition (root-partition) bezeichnet und beim booten ins Verzeichnis / gemountet. Alle weiteren Partitionen werden dann in Verzeichnisse gemountet, die ihrerseits entweder auf der Wurzelpartition oder auf schon gemounteten anderen Partitionen liegen. Die Wurzelpartition muss zwingend einige Verzeichnisse enthalten, die nicht auf anderen Partitionen liegen dürfen. Ein simples Beispiel ist das Verzeichnis, das die Bauanleitung enthält, wohin die anderen Partitionen gemountet werden sollen (/etc) oder das Verzeichnis, das den mount-Befehl selber enthält (/bin).

## Die Überprüfung beim Systemstart

Jedes Mal, wenn das System neu gebootet wird, werden alle Dateisysteme überprüft. In der Regel wird aber nur überprüft, ob sie sauber sind - nur wenn nicht wird die eigentliche Überprüfung gestartet. Das funktioniert aber nur bei der Verwendung des Ext2-Dateisystemtyps.

Dieser Typ hat im Superblock ein sogenanntes Valid-Flag (manchmal auch Clean-Flag genannt). Jedes Mal, wenn ein Dateisystem gemountet wird, wird dieses Flag auf 0 gesetzt, erst beim ordentlichen Abhängen durch umount wird es wieder auf 1 gesetzt. Beim Systemstart überprüft das fsck-Programm also zunächst, ob dieses Flag auf 1 steht, wenn ja, so gilt das Dateisystem als sauber und wird nicht weiter überprüft (es sei denn die -f Option ist dem fsck.ext2 Programm mitgegeben).

Falls das System vorher nicht sauber heruntergefahren wurde, die Dateisysteme also das Clean-Flag nicht setzen konnten, erkennt das fsck Programm dies und überprüft die Dateisysteme.

Zusätzlich enthält der Superblock einer ext2-Partition noch einen Zähler, der beschreibt, wie oft das Dateisystem gemountet wurde, seit es das letzte Mal überprüft wurde. Nach x (variabel) erfolgten Mounts wird das Dateisystem überprüft, auch wenn es das Clean-Flag gesetzt hat.

## Dateiarten unter Unix

Unter Unix werden weitaus mehr Dinge in Dateiform verwaltet, als z.B. unter DOS/Windows. Neben den regulären Dateien (regular files) gibt es noch verschiedene weitere Konstrukte, die genauso verwaltet werden. Grundsätzlich unterscheidet Unix die folgenden Dateiarten

- Reguläre Dateien (regular file)
- Verzeichnisse (directory)
- Symbolische Links (symbolic link)
- Blockorientierte Geräte (block device)
- Zeichenorientierte Geräte (char device)
- Feststehende Programmverbindungen (named pipe)
- Netzwerk-Kommunikationsendpunkte (socket)

## Reguläre Dateien

Reguläre Dateien sind alle "normalen Dateien", also das, was wir auch unter DOS als Dateien kennen. Sie haben einen Inhalt, der als Bitfolge auf einem Datenträger gespeichert ist, Die wesentlichen Eigenschaften sind schon im letzten Kapitel besprochen worden. Der einzige Punkt, der noch nicht dargestellt wurde ist die Frage, warum jede Datei die Angabe besitzt, wie viele Dateinamen sie besitzt.

Unter Unix existiert für Dateien so genannte Links, einer davon, der symbolische Link, wird gleich im Anschluss besprochen. Unter diesen Links ist etwas Ähnliches zu verstehen, wie unter den Verknüpfungen von Windows 95/98. Die Tatsache, dass Dateien mehrere Namen besitzen können wurde unter Unix schon sehr früh eingeführt, es handelt sich hierbei

eigentlich um mehrere Dateinamen im Inhaltsverzeichnis einer Platte, die auf dieselben Blöcke zeigen. Unter Unix spricht man hier von so genannten Hard-Links.

## **Symbolische Links**

Im Gegensatz zu Hardlinks sind symbolische Links wesentlich flexibler, aber auch fehleranfälliger. Hier handelt es sich tatsächlich um Verweise auf eine bestehende Datei, d.h., der symbolische Link ist in Wahrheit eine Datei, die nur den Dateinamen einer anderen Datei enthält. Damit das System diese Datei nicht als Textdatei mißinterpretiert wird sie als SymLink gekennzeichnet, als Dateityp steht ein l statt eines Bindestrichs.

## **Verzeichnisse**

Verzeichnisse sind unter Unix auch nur Dateien, die allerdings eben den Typ d statt Bindestrich haben um sie von regulären Dateien zu unterscheiden.

## **Blockorientierte Geräte**

Unix betrachtet auch jedes Stück Hardware als Datei. Dabei wird zwischen block- und zeichenorientierten Geräten unterschieden. Als blockorientiertes Gerät gilt jedes Gerät, das nicht einzelne Zeichen verarbeitet, sondern ganze Blocks. Dabei handelt es sich z.B. typischerweise um Festplatten, Disketten und andere Laufwerke. Diese Laufwerke werden dann als Dateien verwaltet, für den Anwender bleibt der Unterschied zwischen Gerät und Datei nahezu transparent.

Die Gerätedateien (engl. special files), sowohl die block- wie auch die zeichenorientierten befinden sich im Verzeichnis /dev. Sie belegen physikalisch keinen Platz auf der Festplatte, sind also selbst nicht die Gerätetreiber, wie oft fälschlicherweise gedacht wird. Diese Dateien haben eigentlich nur zwei Nummern, die die Adresse des Gerätetreibers im Kernel symbolisieren. Die erste Nummer, auch major number genannt zeigt, um welchen Gerätetreiber es sich handelt, die zweite (minor number) beschreibt, das wievielte Gerät es ist, das diesen Treiber benutzt.

## **Zeichenorientierte Geräte**

Grundsätzlich trifft alles, was wir für die Blockorientierten Geräte oben gesagt haben auch auf die Zeichenorientierten Geräte zu, nur dass es sich bei ihnen um Geräte handelt, die nicht Blockweise angesteuert werden sondern durch einzelne Bytes. Typischerweise handelt es sich hier um serielle oder parallele Schnittstellen, auch und gerade die Terminalleitungen (auch die der virtuellen Terminals) aber auch etwa der Soundanschluss.

Sowohl die block- als auch die zeichenorientierten Gerätedateien werden mittels dem Befehl mknod angelegt. Die einzige Information, die man dazu braucht ist die der Major- und Minornummer und die Frage, ob es sich um ein block- oder zeichenorientiertes Gerät handelt.

## **Schlusswort**

Zum heutigen Zeitpunkt arbeiten sehr viele Entwickler in der Linux-Gemeinde an Weiter- und Neuentwicklungen der Dateisysteme. Journaling-Eigenschaften bei EXT3, Verschlüsselungsmöglichkeiten zur Sicherstellung sensibler Daten und die Restauration gelöschter Daten sind nur einige Beispiele für innovative Gedanken. Manche von ihnen sind schon verwirklicht, manche auf dem Prüfstand.

Das EXT2 - Dateisystem zählt aber immer noch zu den stabilsten Systemen und wird daher immer wieder eingesetzt...

## Quellen:

Linux-Kernel-Programmierung 1. Aufl. 1994 Addison-Wesley Michael Beck

Linux - Unix - Shells 3. Aufl. 1999 Addison-Wesley Helmut Herold

[www.linuxfibel.de](http://www.linuxfibel.de)

<http://www.linux-magazin.de/ausgabe/2000/04/Ext2fs/ext2fs.html>

<http://home.fhtw-berlin.de/~s0323090/>

das große UNIX-Buch 5. Aufl. DATA Becker Dreggel-Kappel