

DATEIVERWALTUNG

Hinweis: Sämtliche Ausführungen gelten für Linux-Betriebssysteme

1. GRUNDLAGEN

Alle Anwendungen müssen Informationen speichern und abrufen. Der Arbeitsspeicher reicht hierfür nicht aus:

- Die Größe des Arbeitsspeichers ist limitiert.
- Die Informationen im Adressraum des Prozesses gehen verloren, wenn der Prozess beendet wird.
- Mehrere Prozesse können nicht gleichzeitig auf die Informationen im Arbeitsspeicher zugreifen.

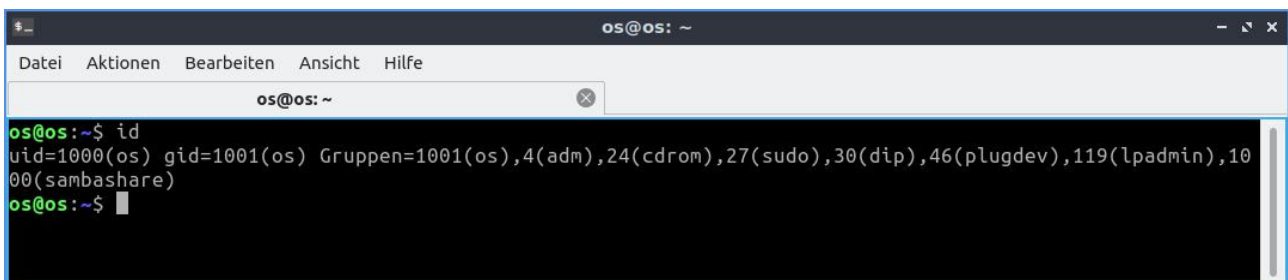
Um diesen Limitierungen zu begegnen, können Betriebssysteme Informationen auf Festplatten oder anderen persistenten Medien speichern.

Wenn mehrere Prozesse, Nutzer und Nutzergruppen diese Informationen nutzen sollen, wird eine Rechteverwaltung unerlässlich. Nutzer müssen feingranular steuern können, wer Zugriff auf die generierten Informationen hat.

Linux verwendet eine relativ einfache Rechteverwaltung. Grob gesagt existiert nur der root-User, der alle Rechte hat, sowie die gewöhnlichen Nutzer, deren Berechtigungen durch die Rechteverwaltung geregelt werden:

Jede Datei hat genau einen Besitzer, der durch die UID identifiziert wird. Die UID ist die eindeutige Nutzernummer, mit der das Betriebssystem intern anstatt des Nutzernamens arbeitet. Die UID kann mit dem Befehl `id` in einem Terminal abgefragt werden.

Jede Datei wird genau einer Gruppe zugewiesen, identifiziert durch die GID. Die GID ist die eindeutige Gruppennummer, mit der das Betriebssystem intern anstatt des Gruppennamens arbeitet. Der Befehl `id` liefert auch die Gruppen, denen ein Nutzer angehört, samt GID.

A screenshot of a terminal window titled "os@os: ~". The terminal shows the command `id` being executed. The output is: `uid=1000(os) gid=1001(os) Gruppen=1001(os),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),119(lpadmin),1000(sambashare)`. The terminal has a menu bar with "Datei", "Aktionen", "Bearbeiten", "Ansicht", and "Hilfe". There is a tab labeled "os@os: ~" with a close button.

Jeder Datei kann grundsätzlich ein Leserecht `r`, ein Schreibrecht `w` und ein Ausführungsrecht `x` vergeben werden. Diese drei Rechte werden jeweils festgelegt für den Besitzer `u` (user), für die Gruppe `g` (group) sowie für alle anderen Nutzer `o` (other).

Im Terminal können Sie sich die Berechtigungen der Dateien im aktuellen Verzeichnis mit dem Befehl `ls -al` ausgeben lassen:

```

os@os:~$ ls -al
insgesamt 360
drwxr-xr-x 20 os os 4096 Mär 19 15:13 .
drwxr-xr-x 6 root root 4096 Mär 19 15:21 ..
-rw----- 1 os os 722 Mär 21 11:07 .bash_history
-rw-r--r-- 1 os os 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 os os 3771 Feb 25 2020 .bashrc
drwxrwxr-x 6 os os 4096 Mär 19 15:13 betriebssysteme
drwxr-xr-x 2 os os 4096 Mär 21 10:54 Bilder
drwxr-xr-x 12 os os 4096 Mär 18 20:42 .cache
drwx----- 15 os os 4096 Mär 21 10:54 .config
drwxrwxr-x 3 os os 4096 Mär 19 10:59 Desktop
drwxr-xr-x 3 os os 4096 Mär 10 12:27 Dokumente
drwxr-xr-x 3 os os 4096 Mär 18 16:33 Downloads
drwxrwxr-x 6 os os 4096 Mär 18 17:01 .eclipse
drwxrwxr-x 3 os os 4096 Mär 18 17:01 eclipse
drwx----- 3 os os 4096 Mär 10 12:21 .gnupg
drwxrwxr-x 3 os os 4096 Mär 10 12:21 .local
drwx----- 5 os os 4096 Mär 18 16:29 .mozilla
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Musik
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Öffentlich
drwxrwxr-x 6 os os 4096 Mär 18 21:18 .p2
-rw-r--r-- 1 os os 807 Feb 25 2020 .profile
-rw-r--r-- 1 os os 0 Mär 10 12:22 .sudo_as_admin_successful
drwxrwxr-x 2 os os 4096 Mär 18 17:12 .swt
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Videos
-rw----- 1 os os 1867 Mär 19 15:13 .viminfo
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Vorlagen
-rw----- 1 os os 47 Mär 19 10:59 .xauthority
-rw-rw-r-- 1 os os 9114 Mär 18 22:15 .xscreensaver
-rw----- 1 os os 242798 Mär 21 11:07 .xsession-errors
os@os:~$

```

Der erste Abschnitt der Ausgabe zu den einzelnen Dateien thematisiert die Berechtigungen:

`-rw-r--r--`

Das erste (gelb hinterlegte) Zeichen hat mit den Berechtigungen nichts zu tun. Ein `-` steht für eine Datei, `d` steht für einen Ordner, `l` kennzeichnet einen *Softlink*, bei `c` und `b` handelt es sich um Gerätedateien und `p` kennzeichnet eine *Named Pipe*.

Die Berechtigungen des Eigentümers (`u`) sind grün hinterlegt. Im konkreten Beispiel darf der Eigentümer der Datei lesen (`r`) und schreiben (`w`). Die Datei darf aber nicht ausgeführt werden (`-` statt `x`). Zum Speichern der Eigentümerrechte werden 3 Bit verwendet:

Eigentümerrechte (u)			
Bit	3	2	1
Bedeutung	r	w	x
Wertigkeit	4	2	1

Das vorliegende Beispiel `rw-` kann also auch mit der Oktalzahl 6 ausgedrückt werden ($4 + 2 + 0$).

Die Gruppenberechtigungen (g) sind rot hinterlegt. Im Beispiel darf die Gruppe `os` die Datei lesen (r) aber nicht schreiben (- statt w) oder ausführen (- statt x). Auch zum Speichern der Gruppenrechte werden wieder 3 Bit verwendet:

Gruppenrechte (g)			
Bit	3	2	1
Bedeutung	r	w	x
Wertigkeit	4	2	1

Das vorliegende Beispiel `r--` kann also auch mit der Oktalzahl 4 ausgedrückt werden ($4 + 0 + 0$).

Die Berechtigungen der anderen Nutzer (o) sind grau hinterlegt. Auch hier funktionieren die Berechtigungen analog der Eigentümer- und Gruppenrechte. Die anderen Nutzer dürfen folglich die Datei lesen (r) aber nicht schreiben (- statt w) oder ausführen (- statt x). Auch zum Speichern der Other-Rechte werden wieder 3 Bit verwendet:

Other-Rechte (o)			
Bit	3	2	1
Bedeutung	r	w	x
Wertigkeit	4	2	1

Das vorliegende Beispiel `r--` kann also auch mit der Oktalzahl 4 ausgedrückt werden ($4 + 0 + 0$).

Oktal ausgedrückt entspricht die Berechtigung `rw-r--r--` damit `644`.

Grundsätzlich existiert noch eine vierte Gruppe von Berechtigungen. Diese vierte Gruppe wird relativ selten gebraucht. Sie beschreibt folgende **Sonderrechte**:

Ausführbare Dateien, bei welchen das `Set-UID-Bit` gesetzt ist, werden immer mit der UID des Eigentümers ausgeführt. Es spielt also keine Rolle, welcher Nutzer die Datei aufruft: Sie wird immer im Kontext des Eigentümers ausgeführt.

Beim `Set-GID-Bit` verhält es sich ähnlich: Die ausführbaren Dateien werden immer im Kontext der Eigentümergruppe ausgeführt.

Weiterhin existiert noch ein `Sticky-Bit`, welches aber sehr selten zum Einsatz kommt: Bei gesetztem `Sticky-Bit` in Verzeichnissen kann nur der Eigentümer Dateien löschen. Auch dann, wenn andere Nutzer Schreibrechte haben. Früher wurde das `Sticky-Bit` dafür verwendet, Anwendungen dauerhaft im Arbeitsspeicher zu behalten. Für die Sonderrechte werden ebenfalls 3 Bit verwendet:

Sonderrechte (s)			
Bit	3	2	1
Bedeutung	Set-UID	Set-GID	Sticky-Bit
Wertigkeit	4	2	1

Die Sonderrechte werden mit `ls -al` **nicht** als eigener Dreierblock angezeigt. Stattdessen wird bei gesetztem Set-UID-Bit das Eigentümer-Ausführungsrecht mit einem `s` angezeigt. Beim Set-GID-Bit wird das Gruppenausführungsrecht mit einem `s` angezeigt. Beim sticky-Bit wird das Other-Ausführungsrecht mit einem `t` angezeigt.

Werden keine Sonderrechte vergeben, so ergibt sich eine oktale 0. Dies sollte bei den meisten Dateien der Fall sein. Im Beispiel von weiter oben bedeutet dies, dass die vollständigen Berechtigungen oktal folgendermaßen dargestellt werden: 0644

Eine oktale Berechtigung 7777 wird `rwSrwsrwt` dargestellt.

Weitere Erklärungen zu den Berechtigungen finden Sie unter <https://www.howtoforge.de/anleitung/was-ist-umask-unter-linux/> und <https://www.informatik-aktuell.de/betrieb/betriebssysteme/rechte-im-dateisystem-mehr-als-nur-rwx.html>.

Die Berechtigungen für eine Datei können mit dem Befehl `chmod` angepasst werden:

```

os@os: ~
Datei Aktionen Bearbeiten Ansicht Hilfe

os@os: ~
os@os:~$ chmod 0750 Test.sh
os@os:~$ ls -al
insgesamt 360
drwxr-xr-x 20 os os 4096 Mär 21 15:10 .
drwxr-xr-x 6 root root 4096 Mär 19 15:21 ..
-rw----- 1 os os 722 Mär 21 11:07 .bash_history
-rw-r--r-- 1 os os 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 os os 3771 Feb 25 2020 .bashrc
drwxrwxr-x 6 os os 4096 Mär 19 15:13 betriebssysteme
drwxr-xr-x 2 os os 4096 Mär 21 11:09 Bilder
drwxr-xr-x 12 os os 4096 Mär 18 20:42 .cache
drwx----- 15 os os 4096 Mär 21 11:09 .config
drwxrwxr-x 3 os os 4096 Mär 19 10:59 Desktop
drwxr-xr-x 3 os os 4096 Mär 10 12:27 Dokumente
drwxr-xr-x 3 os os 4096 Mär 18 16:33 Downloads
drwxrwxr-x 6 os os 4096 Mär 18 17:01 eclipse
drwxrwxr-x 3 os os 4096 Mär 18 17:01 eclipse
drwx----- 3 os os 4096 Mär 10 12:21 .gnupg
drwxrwxr-x 3 os os 4096 Mär 10 12:21 .local
drwx----- 5 os os 4096 Mär 18 16:29 .mozilla
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Musik
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Öffentlich
drwxrwxr-x 6 os os 4096 Mär 18 21:18 .p2
-rw-r--r-- 1 os os 807 Feb 25 2020 .profile
-rw-r--r-- 1 os os 0 Mär 10 12:22 .sudo_as_admin_successful
drwxrwxr-x 2 os os 4096 Mär 18 17:12 .swt
-rwxr-x-- 1 os os 0 Mär 21 15:08 Test.sh
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Videos
-rw----- 1 os os 1867 Mär 19 15:13 .viminfo
drwxr-xr-x 2 os os 4096 Mär 10 12:21 Vorlagen
-rw----- 1 os os 47 Mär 19 10:59 .Xauthority
-rw-rw-r-- 1 os os 9114 Mär 18 22:15 .xscreensaver
-rw----- 1 os os 244412 Mär 21 15:08 .xsession-errors
os@os:~$

```

Aufgabe 1.1: Wie wurden die Berechtigungen der Datei `Test.sh` (vgl. obige Abbildung) abgeändert? Wer darf nun wie auf diese Datei zugreifen?

Aufgabe 1.2: Ändern Sie die Berechtigungen derart ab, dass Sie als Eigentümer alles mit der Datei machen dürfen, Nutzer in der gleichen Gruppe und alle anderen Nutzer sollen die Datei aber nur ausführen dürfen.

Aufgabe 1.3: `chmod` funktioniert nicht nur mit den oktalen Berechtigungen. Insbesondere wenn man mit mehreren Dateien gleichzeitig arbeitet, können auch symbolische Ausdrücke verwendet werden. Recherchieren Sie, wie diese Ausdrücke funktionieren. Einen guten Einstiegspunkt bietet <https://pubs.opengroup.org/onlinepubs/9699919799.2016edition/utilities/chmod.html>.

2. DATEIEN IN C

Mit Low-Level-Funktionen kann man in C direkt auf die Schnittstellen des Betriebssystems zugreifen. Die Schnittstellen sind im POSIX-Standard (Portable Operating System Interface) festgelegt. Nicht nur Linux implementiert diese Schnittstellen, sondern der Großteil der unixoiden Betriebssysteme. `chmod` gehört ebenfalls zu diesen standardisierten Schnittstellen. Auf <https://pubs.opengroup.org/onlinepubs/9699919799.2016edition/> können Sie sich auch über alle anderen Schnittstellen informieren. Da die Schnittstellen mit C angesprochen werden können, sind sie selbstverständlich entsprechend dokumentiert: https://www.gnu.org/software/libc/manual/html_node/index.html.

Im Folgenden werden insbesondere die Low-Level-Funktionen von C behandelt, welche bei Dateioperationen Verwendung finden.

`open`

Mit `open` wird einer Datei ein File-Deskriptor zugeordnet. Dabei handelt es sich um eine positive Zahl, die vom Betriebssystem vergeben wird. Für `open` existieren zwei Versionen:

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int open(const char *pfad, int flags);
int open(const char *pfadname, int flags, mode_t zugriffsrechte);
```

Mit dem Deskriptor wird über den Flags-Parameter auch angegeben, was mit der Datei passieren soll. Die wichtigsten Modi sind:

Modus	Bedeutung
O_WRONLY	Nur zum Schreiben öffnen.
O_RDWR	Zum Lesen und Schreiben öffnen.
O_RDONLY	Nur zum Lesen öffnen.
O_CREAT	Erzeugt eine neue Datei, falls die Datei noch nicht existiert. Ansonsten wirkungslos.
O_APPEND	Datei öffnen zum Schreiben am Ende.
O_EXCL	wird in Kombination mit O_CREAT verwendet: Eine Datei kann nicht geöffnet werden, wenn sie bereits existiert.
O_TRUNC	Die zum Schreiben geöffnete Datei wird vor dem ersten Schreiben geleert.

Die Modi können bitweise ODER-verknüpft werden.

Für den Parameter **zugriffsrechte** können die symbolischen Konstanten für Zugriffsrechte verwendet werden: siehe dazu https://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html. Diese können ebenfalls wieder bitweise ODER-verknüpft werden. Alternativ können auch die Oktal-Zahlen aus dem ersten Kapitel verwendet werden.

`open` liefert `-1` zurück, falls der Deskriptor nicht angelegt werden konnte.

Ein beispielhafter `open`-Befehl sieht folgendermaßen aus:

```
int fd;
fd = open("dateiXY.txt", O_WRONLY | O_CREAT | O_TRUNC , 0644)
```

Hinweis: Die `umask` kann die Rechtevergabe einschränken. In der Regel wird die `umask` auf `022` gesetzt sein. Wenn bei den Gruppen- und Other-Berechtigungen Schreibrechte vergeben werden sollen, muss zuvor die `umask` neu belegt werden: `umask(0)` ;

close

Ein File-Deskriptor sollte nicht nur geöffnet, sondern immer auch geschlossen werden:

```
#include <unistd.h>
int close(int fd);
```

Wenn der File-Deskriptor geschlossen werden konnte, liefert `close` den Wert `0` zurück. Bei einem Fehler wird `-1` zurückgegeben.

write

Sofern der File-Deskriptor schreibend geöffnet wurde, kann auch mit `write` in die Datei geschrieben werden:

```
#include <unistd.h>
ssize_t write(int fd, void *buf, size_t nbyte);
```

Damit werden `nbyte` Zeichen ab der Adresse `buf` in die Datei geschrieben, die vom File-Deskriptor `fd` referenziert wird. `write` gibt die Anzahl der geschriebenen Zeichen zurück bzw. `-1` im Falle eines Fehlers.

read

Wie der Name schon sagt, werden mit dem Systemaufruf `read` byteweise Daten aus einer geöffneten Datei gelesen. Grundsätzlich funktioniert `read` in gleicher Weise wie `write`:

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t nbyte);
```

Damit werden `nbyte` Zeichen aus der Datei, die vom File-Deskriptor `fd` referenziert wird, in den Speicher ab der Adresse `buf` kopiert. `read` gibt die Anzahl der kopierten Zeichen zurück bzw. `-1` im Fehlerfall.

Aufgabe 2.1: Schreiben Sie ein Programm, welches einen in den Argumenten übergebenen String in eine Datei schreibt. Der Name der Datei soll ebenfalls in den Argumenten übergeben werden. Wenn die Datei schon existiert, soll der String an die Datei angehängt werden. Ansonsten soll die Datei neu angelegt werden. Der Eigentümer der Datei soll lesen und schreiben können. Nutzer der Benutzergruppe der Datei sollen lesen können. Alle anderen Nutzer sollen keinen Zugriff auf die Datei haben.

Aufgabe 2.2: Schreiben Sie ein Programm, welches die Datei aus Aufgabe 2.1. einliest und im Terminal ausgibt. Verwenden Sie für die Ausgabe ebenfalls einen `write`-Befehl. Der File-Deskriptor für das Terminal ist in der symbolischen Konstante `STDOUT_FILENO` enthalten. Sie brauchen also keine Datei zu öffnen oder zu schließen um auf den File-Deskriptor der Terminal-Ausgabe zuzugreifen.

Meilenstein 1: Die beiden Programme von Aufgabe 2.1 und 2.2 funktionieren so wie beschrieben.

unlink

Mit `unlink` kann eine Datei gelöscht werden. Ein File-Deskriptor ist hierfür nicht notwendig:

```
#include <unistd.h>
int unlink(const char *pathname);
```

Aufgabe 2.3: Schreiben Sie ein Programm, welches eine Datei in eine neue Datei kopiert und danach die alte Datei löscht. Die Namen der Dateien sollen als Argumente übergeben werden. Wenn die neue Datei bereits existiert, soll sie überschrieben werden.

Meilenstein 2: Das Programm von Aufgabe 2.3 funktioniert so wie beschrieben.