

```

Sep 19 14:20:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6094]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 13:27:25 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 20 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17978]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17978]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 21 17:43:26 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[1634]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6066]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[1862]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

# 5. Dateien und Verzeichnisse

## C-Funktionen zu Dateien / Verz.

- Positionierung innerhalb Datei: `lseek()`,
- Datei-Informationen: `stat()`, `lstat()`
- Links: `link()`, `symlink()`
- Datei löschen: `unlink()`
- Besitzer, Gruppe, Rechte: `(f)chown()`, `(f)chmod()`
- Verzeichnisse: `getcwd()`, `(f)chdir()`, `mkdir()`, `rmdir()`
- Verzeichnisinhalte verarbeiten

## Dateien und Verzeichnisse

### Schon gesehen:

- Datei öffnen (`open`, File Descriptor)
- Datei erzeugen (`creat`)
- Lesen, schreiben (`read`, `write`)
- Datei schließen (`close`)
- Flags fürs Öffnen (`O_RDONLY` etc.)
- Modus beim Erzeugen (`S_IRUSR` etc.)

## Nachtrag zu `creat()`, `umask()`

- Zugriffsrechte (`mode`) bei
  - `creat(f, mode)` bzw.
  - `open(f, O_CREAT, mode)`
werden durch `umask` beeinflusst:
- tatsächliche Rechte: `mode & ~umask`
- `umask` setzen mit `umask (maske)`
- Beispiel: →

## Nachtrag zu creat ( ), umask ( )

```
// umask-test.c
#include <stdlib.h>
int main ( ) {
    creat ("test1.rwx", 0777); // max. Rechte: rwxrwxrwx
    creat ("test1.-wx", 0333); // Rechte: -wx-wx-wx
    creat ("test1.r-x", 0555); // Rechte: r-xr-xr-x
    umask (0);
    creat ("test2.rwx", 0777); // max. Rechte: rwxrwxrwx
    creat ("test2.-wx", 0333); // Rechte: -wx-wx-wx
    creat ("test2.r-x", 0555); // Rechte: r-xr-xr-x
    system ("stat -c '%a %A %n' test?.???");
};

root@ubu64:~# umask
0022
root@ubu64:~# ./umask-test
755 -rwxr-xr-x test1.rwx
555 -r-xr-xr-x test1.r-x
311 --wx--x--x test1.-wx
777 -rwxrwxrwx test2.rwx
555 -r-xr-xr-x test2.r-x
333 --wx-wx-wx test2.-wx
```

## lseek ( )

- Bisher: Dateien sequenziell lesen oder schreiben
- `lseek ( )` erlaubt Positionierung des Schreib-/Lese-Zeigers
- drei Varianten:
  - `lseek (fd, offset, SEEK_SET)`: absolut
  - `lseek (fd, offset, SEEK_CUR)`: relativ
  - `lseek (fd, offset, SEEK_END)`:  
Dateiende + offset (meist: offset = 0)
- Rückgabewert: neuer Offset

## Anhängen an Datei (1)

zwei Möglichkeiten:

- Variante 1
  - Datei normal zum Schreiben öffnen
  - Sprung ans Dateiende mit `lseek ( )`
  - schreiben
- Variante 2
  - Datei im Append-Modus (`O_APPEND`) öffnen
  - schreiben

## Anhängen an Datei (2)

Vorteil der Append-Variante:

- Wenn mehrere Prozesse dieselbe Datei zum Schreiben verwenden, führen alle `write ( )`-Aufrufe garantiert zum Anhängen
- bei normalem Öffnen ggf. gegenseitiges Überschreiben möglich
- Typische Anwendung: Schreiben in Log-Datei

## Datei-Informationen: stat ()

- Eigenschaften einer Datei in Datenstruktur vom Typ struct stat
- Aufruf:
  - struct stat s;  
stat (dateiname, &s);
  - lstat (dateiname, &s);
- Wenn Datei ein Symlink ist, gibt stat () Informationen über verlinkte Datei aus; bei lstat () ist es der Link selbst

## st\_mode in struct stat (1)

- st\_mode schlecht lesbar (am besten oktal ausgeben):

```
// stattest.c
#include <sys/stat.h>
#include <stdio.h>
main () {
    struct stat s;
    lstat ("/etc/fstab", &s);
    printf ("s.st_mode: %o\n", s.st_mode);
}

root@ubu64:~# ./stattest
s.st_mode: 0100644
root@ubu64:~# ls -l /etc/fstab
-rw-r--r-- 1 root root 681 2012-04-18 13:58 /etc/fstab
```

- klar: 644 = Zugriffsrechte; Rest: → Manpage

## Aufbau von struct stat

```
43: struct stat
44: {
45:     __dev_t st_dev;          /* Device. */
46:     __ino_t st_ino;         /* File serial number. */
47:     __nlink_t st_nlink;    /* Link count. */
48:     __mode_t st_mode;      /* File mode. */
49:     __uid_t st_uid;        /* User ID of the file's owner. */
50:     __gid_t st_gid;        /* Group ID of the file's group. */
51:     __dev_t st_rdev;       /* Device number, if device. */
52:     __off_t st_size;       /* Size of file, in bytes. */
53:     __blksize_t st_blksize; /* Optimal block size for I/O. */
54:     __blkcnt_t st_blocks;  /* Number 512-byte blocks allocated. */
55:     __time_t st_atime;     /* Time of last access. */
56:     __time_t st_mtime;     /* Time of last modification. */
57:     __time_t st_ctime;     /* Time of last status change. */
58: };
```

- Quelle: /usr/include/sys/stat.h, nur Teile dargestellt
- \_\_time\_t: long int, Sekunden seit 01.01.1970 00:00 Uhr
- ls -l zeigt st\_mtime an

## st\_mode in struct stat (2)

S_IFMT	0170000	bit mask for the file type bit fields
S_IFSOCK	0140000	socket
S_IFLNK	0120000	symbolic link
S_IFREG	0100000	regular file
S_IFBLK	0060000	block device
S_IFDIR	0040000	directory
S_IFCHR	0020000	character device
S_IFIFO	0010000	FIFO
S_ISUID	0004000	set UID bit
S_ISGID	0002000	set-group-ID bit (see below)
S_ISVTX	0001000	sticky bit (see below)
S_IRWXU	00700	mask for file owner permissions
S_IRUSR	00400	owner has read permission
S_IWUSR	00200	owner has write permission
S_IXUSR	00100	owner has execute permission
S_IRWXG	00070	mask for group permissions
S_IRGRP	00040	group has read permission
S_IWGRP	00020	group has write permission
S_IXGRP	00010	group has execute permission
S_IRWXO	00007	mask for permissions for others (not in group)
S_IROTH	00004	others have read permission
S_IWOTH	00002	others have write permission
S_IXOTH	00001	others have execute permission

## st\_mode in struct stat (3)

- Makro-Definitionen zum Testen (aus stat.h):

```
#define S_IFMT 00170000
#define S_IFSOCK 0140000
#define S_IFLNK 0120000
#define S_IFREG 0100000
#define S_IFBLK 0060000
#define S_IFDIR 0040000
#define S_IFCHR 0020000
#define S_IFIFO 0010000
#define S_ISUID 0004000
#define S_ISGID 0002000
#define S_ISVTX 0001000

#define S_ISLNK(m) ((m) & S_IFMT) == S_IFLNK)
#define S_ISREG(m) ((m) & S_IFMT) == S_IFREG)
#define S_ISDIR(m) ((m) & S_IFMT) == S_IFDIR)
#define S_ISCHR(m) ((m) & S_IFMT) == S_IFCHR)
#define S_ISBLK(m) ((m) & S_IFMT) == S_IFBLK)
#define S_ISFIFO(m) ((m) & S_IFMT) == S_IFIFO)
#define S_ISSOCK(m) ((m) & S_IFMT) == S_IFSOCK)
```

## Status geöffneter Dateien

- Alternative fstat() verwendet file descriptor (einer geöffneten Datei)
  - int fd = open (...);
  - struct stat s;
  - fstat (fd, &s);

## Fehler bei stat ()

- stat ()-Aufruf kann fehlschlagen:

```
// rekstat.c
#include <sys/stat.h>
#include <stdio.h>
main () {
    struct stat s;
    int res = stat ("rekursiv", &s);
    if (res == -1) {
        perror("rekstat"); exit(0);
    }
}

root@ubu64:~# ln -s rekursiv rekursiv
root@ubu64:~# ./rekstat
rekstat:Too many levels of symbolic links
root@ubu64:~# file rekursiv
rekursiv: symbolic link in a loop
```

## Symlink erzeugen

- Symlink (symbolischer Link, Soft Link) ist Verweis durch Pfadangabe
- symlink (original, link)
- erzeugt Datei vom Typ link (1)
- Pfad relativ oder absolut
- dateisystem-übergreifend möglich

## Hardlink erzeugen

- Hardlink (Link) ist weiterer Verzeichniseintrag zu bestehender Datei; gleicher Inode  
(Verzeichnis = Tabelle mit Dateiname/Inode-Nr.-Paaren)
- `link (original, link)`
- `original` muss existieren, `link` muss im selben Dateisystem wie `original` liegen
- überschreibt keine vorhandenen Dateien

## Besitzer, Gruppe: (f)chown ( )

- `chown (dateiname, owner, group)`
- `lchown (dateiname, owner, group)`  
(folgt Symlinks nicht)
- `fchown (fd, owner, group)`  
(mit file descriptor `fd`, offene Datei)
- numerische IDs für `owner` und `group`
- einer der Werte darf `-1` sein (→ nicht ändern)
- keine separate `chgrp ( )`-Funktion

## Datei löschen: `unlink ( )`

- `unlink (filename)`
- löscht eine Zuordnung Dateiname/Inode-Nr. aus Verzeichnis, reduziert Link Count
- nicht identisch mit „Datei löschen“, falls Link Count vor `unlink ( )` größer als 1 war
- Datei bleibt auch bei Link Count 0 noch erhalten, solange sie noch geöffnet ist

## Zugriffsrechte: (f)chmod ( )

- `chmod (datei, mode)`
- `lchmod (datei, m)` (folgt Symlinks nicht)
- `fchmod (fd, mode)` (mit file descriptor `fd`)
- `mode:`
  - `S_ISUID (04000)` set-user-ID
  - `S_ISGID (02000)` set-group-ID
  - `S_ISVTX (01000)` sticky bit (restricted deletion flag)
  - `S_IRUSR (00400)` read by owner
  - `S_IWUSR (00200)` write by owner
  - `S_IXUSR (00100)` execute/search by owner  
("search" applies for directories, and means that entries within the directory can be accessed)
  - `S_IRGRP (00040)` read by group
  - `S_IWGRP (00020)` write by group
  - `S_IXGRP (00010)` execute/search by group
  - `S_IROTH (00004)` read by others
  - `S_IWOTH (00002)` write by others
  - `S_IXOTH (00001)` execute/search by others

## Arbeitsverzeichnis: `getcwd()`

- aktuelles Arbeitsverzeichnis abfragen
- `char pfad[100];  
getcwd (&pfad, sizeof(pfad));`
- Alternative: `getwd (&pfad)` (unsicher)
- Alternative unter Linux:  
`char *get_current_dir_name(void);`  
(reserviert mit `malloc()` freien Speicher für den Pfad, anschließend mit `free()` freigeben)

## Verzeichnis erzeugen: `mkdir()`

- `mkdir (pfad, mode)`
- `mode`: Bedeutung wie bei `chmod()`
- `mode` wird durch `umask` verändert, tatsächlicher Wert: `mode & ~umask & 0777`  
→ vgl. `umask` bei `creat()`
- es ist nicht möglich, mehrere Verzeichnisse „in einem Rutsch“ zu erzeugen (vgl. Shell-Befehl `mkdir -p a/b/c`)

## Arbeitsverzeichnis: `chdir()`

Aktuelles Arbeitsverzeichnis ändern:

- `chdir (pfad)`
- oder: `fchdir (fd)`  
mit offenem file descriptor `fd`

## Verzeichnis löschen: `rmdir()`

- `rmdir (pfad)`
- löscht leeres Verzeichnis
- es ist nicht möglich, mehrere Verzeichnisse „in einem Rutsch“ zu erzeugen (vgl. Shell-Befehl `rmdir -p a/b/c`)

## Verzeichnisliste (1)

- Die bisher vorgestellten Kommandos entsprechen direkt (meist) gleichnamigen System Calls
- Für das Auslesen eines Verzeichnisses gibt es den Syscall `getdents` (get directory entries), der nicht direkt benutzt wird  
→ Wrapper: `readdir()`
- Doku: man 3 `readdir` (*nicht* die Manpage aus Abschnitt 2!)

## Verzeichnisliste (3)

```
// readdir.c
#include <dirent.h>
#include <errno.h>
#include <stdio.h>

int main (int argv, char *argv[]) {
    DIR *dirp;
    struct dirent *entry;

    if (argv != 2) { printf ("readdir Verzeichnis\n"); return 0; };

    if ((dirp = opendir(argv[1])) == NULL) { perror(""); return -1; };

    do {
        if ((entry = readdir(dirp)) != NULL) {
            printf("%s (%d)\n", entry->d_name, (int)entry->d_ino);
        }
    } while (entry != NULL);

    closedir(dirp);
    return 0;
}
```

```
root@ubu64:~# ./readdir .
uebung04 (103804)
stat.c (100534)
. (38091)
.. (435)
readdir.c (100533)
a.out (100539)
```

## Verzeichnisliste (2)

- Aufrufe von `readdir()` geben immer Zeiger auf einen `struct dirent` zurück:

```
struct dirent {
    ino_t      d_ino;      /* inode number */
    off_t      d_off;     /* offset to the next dirent */
    unsigned short d_reclen; /* length of this record */
    unsigned char d_type;  /* type of file; not supported
                           by all file system types */
    char       d_name[256]; /* filename */
};
```

- für Namensliste: nur `d_name` auswerten

## Übersicht Shell / C

Shell-Kommandos	C-Funktionen
umask	umask()
>	creat()
>>	open (... , O_APPEND)
stat	stat()
ln	link()
ln -s	symlink()
rm	unlink()
chown u:g file	chown(file,u,g)
chown u f	chown(f,u,-1)
chgrp g f	chown(f,-1,g)
chown -h	lchown()
chgrp -h	lchown()
chmod	chmod()
pwd	getcwd()
cd	chdir()
mkdir	mkdir()
rmdir	rmdir()
ls	opendir(), readdir(), stat()