Systemprogrammierung

Foliensatz C
– Prozesse
– Dateien und Verzeichnisse

Prof. Dr. Hans-Georg Eßer

Wintersemester 2023/24
v1.0 – 25.10.2023

Prozesse

Prozesse

Schon gesehen:

- Prozess-Hierarchie (`fork`, Vater/Sohn)
- Programm nachladen (`exec`)
- Warten auf Prozess (`wait`, `waitpid`)

C-Funktionen zu Prozessen

- `getpid()`, `getppid()`
- Rückgabewert in `wait (int *status)`
- `nice()`
- `setpgid()`, `setsid()`, `getpgid()`, `getsid()`

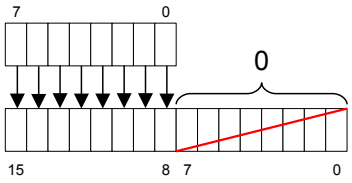
Weitere Themen:

- `/proc`-Dateisystem
- Argumente (`argc`, `argv`), `getopt()`

Zusammenhang exit / wait

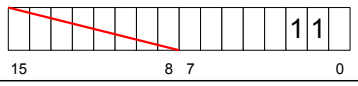
- auslesen des `exit`-Werts: über (`status >> 8`) oder mit Makro `WEXITSTATUS`
- Signalnummer != 0, falls erzwungener Abbruch durch Signal

`exit(exitcode);`



`status`

`abort();`



`status = 6`


```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

void forktest (int retval) {
  int pid, status;
  pid = fork ();
  if (pid==0) {
    // child:
    if (retval >= 0) {
      // normal termination
      printf ("Child: exiting with exit code %d\n", retval);
      exit (retval);
    } else {
      // abnormal termination
      printf ("Child: exiting abnormally\n");
      abort ();
    };
  } else {
    // parent:
    printf ("Parent: Waiting for child to terminate\n");
    wait (&status);
    if (WIFEXITED(status)) {
      printf ("Parent: Child exited with exit code %d; status=%d \n",
        WEXITSTATUS(status), status);
    } else {
      printf ("Parent: Child exited abnormally; status=%d \n\n",
        status);
    }
  };
  return;
}
```

```c
int main () {
  forktest (0);
  forktest (3);
  forktest (-1);  // terminate abnormally
  return 0;
}
```

```
esser@ubu64:forkwait$ gcc forkwait.c
esser@ubu64:forkwait$ ./a.out
Parent: Waiting for child to terminate
Child: exiting with exit code 0
Parent: Child exited with exit code 0; status=0

Parent: Waiting for child to terminate
Child: exiting with exit code 3
Parent: Child exited with exit code 3; status=768

Parent: Waiting for child to terminate
Child: exiting abnormally
Parent: Child exited abnormally; status=6
```

`nice()`: Priorität ändern (1)

- Der Aufruf `nice(x)` ändert die Priorität des Prozesses und gibt
  - die neue Priorität
  - oder -1 im Fehlerfall zurück
- gültige Werte für den Nice-Wert: –20 .. 19
  - Normalwert: 0
  - 1..19: niedrige Priorität („freundlich“)
  - –20..–1: hohe Priorität („unfreundlich“)
- Nur *root* darf höhere Prioritäten wählen

`nice()`: Priorität ändern (2)

- Wichtig: `nice()` interpretiert Argument als relative Änderung des aktuellen Nice-Werts (wie bei Shell-Kommando `nice`):
  - `nice(v)`: aktuellen Nice-Wert um v erhöhen
  - `nice -n v kommando`: Kommando mit Nice-Wert NIC+v (NIC = aktueller Nice-Wert in der Shell) ausführen
- nur *root* darf Priorität erhöhen; es ist also kein `nice (10); ... ; nice (-10)` möglich
- Sohnprozesse (`fork`) erben Nice-Wert des Vaters

Prozessgruppen und Sessions (1)

- Prozesse lassen sich zu Gruppen und diese zu Sessions zusammenfassen
- Alle Mitglieder einer Gruppe können gemeinsam signalisiert werden (z. B.: Abbruch)
- Sessions stehen oft für Terminal-Sitzungen; alle Prozesse in einer Session haben dasselbe „kontrollierende Terminal“ (TTY)

Prozessgruppen und Sessions (2)

- **Kontrollierendes Terminal:**
  - Textmodus-Login (Textkonsolen; Strg-Alt-F1 bis Strg-Alt-F6)
  - Terminal-Fenster / Tabs unter X

```
root@ubu64:/home/esser/tmp/forkwait# ps -C getty,bash,mc -o pid,sess,tty,cmd
  PID  SESS TT       CMD
  709   709 tty4     /sbin/getty -8 38400 tty4
  715   715 tty5     /sbin/getty -8 38400 tty5
  724   724 tty2     /sbin/getty -8 38400 tty2
  726   726 tty3     /sbin/getty -8 38400 tty3
  730   730 tty6     /sbin/getty -8 38400 tty6
1505  1505 pts/0    bash
3434  3434 pts/1    bash
4945  3434 pts/1    bash
5941  1010 tty1     -bash
6064  1010 tty1     mc
6066  6066 pts/2    bash -rcfile .bashrc
```

Anmeldeprozesse auf den Textkonsolen 2-6 (tty2-tty6)

Bash-Prozesse in Terminal-fenstern 0 und 1 (X)

Benutzer arbeitet auf tty1

Bash-Prozess in Fenster 2 (X)

Prozessgruppen und Sessions (3)

- Neue Gruppe erzeugen: `setpgid (0,0)`
- Neue Session erzeugen: `setsid ()`

erzeugen eine neue Gruppe bzw. Session, wobei als Gruppen-ID / Session-ID die Prozess-ID des aufrufenden Prozess verwendet wird

- Erzeugen einer neuen Session bewirkt immer auch das Erzeugen einer neuen Gruppe
- `setsid` funktioniert nicht, wenn der aufrufende Prozess „Prozessgruppenführer“ ist (PID = PGID)
- neue Session hat kein kontrollierendes Terminal (TTY)

Prozessgruppen und Sessions (4)

„`setsid` gibt -1 (für Fehler) zurück, wenn der aufru-fende Prozess bereits ein Prozessgruppenführer ist. Um dies zu verhindern, kreiert man üblicherweise mittels `fork` einen Kindprozess, der weiterläuft, während sich der Elternprozess beendet. Der Kindprozess kann nämlich kein Prozessgruppenführer sein, da er zwar die Prozessgruppen-ID vom Elternprozess erbt, aber in jedem Fall eine neue Prozess-ID erhält, die niemals eine Prozessgruppen-ID sein kann, da sie neu ist.“

(Helmut Herold, Linux-Unix-Systemprogrammierung)

Prozessgruppen und Sessions (5)

```
int main () {
  fork ();
  setsid ();       // new session (works only in child)
  fork ();
  setpgid (0,0);   // new group
  fork ();
  sleep(5);
  return 0;
}
```

```
esser@ubu64:~$ ps -C setpgid -o pid,ppid,pgid,sid,tty,cmd; pstree -p | cut -c34- | grep setpgid
  PID  PPID  PGID   SID TT     CMD
 5757  4945  5757  3434 pts/1  ./setpgid
 5758  5757  5758  5758 ?      ./setpgid
 5759  5757  5759  3434 pts/1  ./setpgid
 5760  5757  5757  3434 pts/1  ./setpgid
 5761  5759  5759  3434 pts/1  ./setpgid
 5762  5758  5762  5758 ?      ./setpgid
 5763  5758  5758  5758 ?      ./setpgid
 5764  5762  5762  5758 ?      ./setpgid
```

Session 3434: 5757, 5759, 5760, 5761
Session 5758: 5758, 5762, 5763, 5764
Vier Prozessgruppen: 5757, 5758, 5759, 5762

```
bash(3434)--su(4937)--bash(4945)--setpgid(5757)-+-setpgid(5758)-+-setpgid(5762)--setpgid(5764)
                                                |               `-setpgid(5763)
                                                |-setpgid(5759)---setpgid(5761)
                                                `-setpgid(5760)
```


Abfragen von P.-Gruppe, Session

- Über die Funktionen
  - `getpgid (pid)`
  - `getsid (pid)`

  lassen sich jederzeit die Gruppen- und Session-Zugehörigkeiten feststellen.

  (für den aufrufenden Prozess mit pid=0)
- Prozessgruppenführer: PGID==PID
- Sessionführer: SID==PID

Informationen über Prozesse (1)

- `/proc` enthält für jeden Prozess einen Ordner PID (also `/proc/1`, `/proc/2`, ...)
- In jedem dieser Ordner gibt es zahlreiche Dateien und Verzeichnisse mit Informationen über den Prozess
  - `status`: (menschen-)lesbare Statusinformationen
  - `stat`: maschinenlesbare Statusinformationen
  - `cmdline`: Kommando
  - `environ`: Umgebungsvariablen
  - `fd`, `fdinfo`: Informationen zu offenen Dateien

Informationen über Prozesse (2)

```
# cat /proc/7195/environ ; echo
SHELL=/bin/bashTERM=xtermXDG_SESSION_COOKIE=c420f37852122ff2d7e4eb894f8eb0a2-
1335107306.31018-208107520USER=rootSUDO_USER=esserSUDO_UID=1000USER
NAME=rootMAIL=/var/mail/rootPATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:/usr/gamesPWD=/home/esser/tmp/forkwaitLANG=de_DE.UTF-8SHLVL=1SUDO_COMMAND=/
bin/suHOME=/rootLOGNAME=rootLESSOPEN=| /usr/bin /lesspipe
%sSUDO_GID=1000DISPLAY=:0.0LESSCLOSE=/usr/bin/lesspipe %s %sCO LORTERM=gnome-
terminalXAUTHORITY=/var/run/gdm/auth-for-esser-Zt6Xhr/data base_=/usr/bin/nedit

# tr '\0' '\n' < /proc/7195/environ
SHELL=/bin/bash
TERM=xterm
XDG_SESSION_COOKIE=c420f37852122ff2d7e4eb894f8eb0a2-1335107306.31018-208107520
USER=root
SUDO_USER=esser
SUDO_UID=1000
USERNAME=root
MAIL=/var/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/home/esser/tmp/forkwait
LANG=de_DE.UTF-8
[...]
```


Informationen über Prozesse (3)

```
# cat /proc/7195/status
Name:   nedit
State:  S (sleeping)
Tgid:   7195
Pid:    7195
PPid:   1
TracerPid:  0
Uid:    0   0   0   0
Gid:    0   0   0   0
FDSize: 256
Groups: 0
VmPeak:    61472 kB
VmSize:    61468 kB
VmLck:         0 kB
VmHWM:      7224 kB
VmRSS:      7224 kB
VmData:     4764 kB
VmStk:        88 kB
VmExe:      1188 kB
VmLib:      6464 kB
VmPTE:       140 kB
Threads:    1
SigQ:   2/16382
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: ffffffffffffffff
CapEff: ffffffffffffffff
CapBnd: ffffffffffffffff
Cpus_allowed:   1
Cpus_allowed_list:  0
Mems_allowed:   00000000,00000001
Mems_allowed_list:  0
voluntary_ctxt_switches:    13609
nonvoluntary_ctxt_switches: 620

# cat /proc/7195/stat
7195 (nedit) S 1 7195 3434 34817 9334 4202752
2021 0 19 0 96 51 0 0 20 0 1 0 3941746
62943232 1806 18446744073709551615 4194304
5410572 140734327978864 140734327968016
140314005931272 0 0 0 0 18446744071580244841
0 0 17 0 0 0 0 0 0
```


Informationen über Prozesse (4)

```
# ps -C vi -o pid,cmd
  PID CMD
 9368 vi forkexec.c

# tr '\0' ' ' < /proc/9368/cmdline ; echo
vi forkexec.c

# ls -l /proc/9368/fd/
insgesamt 0
lrwx------ 1 esser esser 64 2012-04-24 01:07 0 -> /dev/pts/0
lrwx------ 1 esser esser 64 2012-04-24 01:07 1 -> /dev/pts/0
lrwx------ 1 esser esser 64 2012-04-24 01:07 2 -> /dev/pts/0
lrwx------ 1 esser esser 64 2012-04-24 01:07 4 ->
                      /home/esser/tmp/uebung02/.forkexec.c.swp

# cat /proc/9368/fdinfo/4
pos:   12288
flags: 0100002
```

Informationen über Prozesse (5)

Auszug aus /usr/include/asm-generic/fcntl.h:

```
#define O_RDONLY      00000000
#define O_WRONLY      00000001
#define O_RDWR        00000002
#define O_CREAT       00000100
#define O_EXCL        00000200
#define O_NOCTTY      00000400
#define O_TRUNC       00001000
#define O_APPEND      00002000
#define O_NONBLOCK    00004000
#define O_SYNC        00010000
#define FASYNC        00020000 // BSD comp.
#define O_DIRECT      00040000
#define O_LARGEFILE   00100000
#define O_DIRECTORY   00200000
#define O_NOFOLLOW    00400000
#define O_NOATIME     01000000
#define O_CLOEXEC     02000000
```

```
# cat /proc/9368/fdinfo/4
pos:    12288
flags:  0100002

 0100002 =

 0100000   (O_LARGEFILE)
+0000002   (O_RDWR)
```


argc und argv (1)

- Unix-Tools werten meist Argumente aus
- Deklariere `main()` als

  `int main (int argc, char *argv[])`
- `argc`: Anzahl der Argumente (Programmname = 1. Argument)
- `*argv[]`: Array mit Argument-Strings

argc und argv (2)

```
#include <stdio.h>   // printf
#include <stdlib.h>  // exit

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

```
esser@ubu64:~/argc$ ./argumente
argc = 1
argv[0] = ./argumente
esser@ubu64:~/argc$ ./argumente eins zwei
argc = 3
argv[0] = ./argumente
argv[1] = eins
argv[2] = zwei
```

argc und argv (3)

- vereinfachte Auswertung der Argumente mit `getopt` und `getopt_long`
- `getopt` verarbeitet Kurzoptionen (`-a`, `-b`) und deren Kombinationen (`-ab`)
- `getopt_long` verarbeitet auch Langoptionen (`--longoption`)

argc und argv (4)

Quelle: http://www.gnu.org/software/libc/manual/html_node/Example-of-Getopt.html

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char **argv) {
  int aflag = 0; int bflag = 0;
  char *cvalue = NULL; int index; int c;
  opterr = 0;

  while ((c = getopt (argc, argv, "abc:")) != -1)
  switch (c) {
      case 'a':   aflag = 1; break;
      case 'b':   bflag = 1; break;
      case 'c':   cvalue = optarg; break;
      case '?':   if (optopt == 'c')
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                  else if (isprint (optopt))
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                  else
                    fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);
                  return 1;
      default:    abort ();
  }

  printf ("aflag = %d, bflag = %d, cvalue = %s\n", aflag, bflag, cvalue);
  for (index = optind; index < argc; index++)
    printf ("Non-option argument %s\n", argv[index]);
}
```


argc und argv (5)

```
# getopt-test
aflag = 0, bflag = 0, cvalue = (null)

# getopt-test -a
aflag = 1, bflag = 0, cvalue = (null)

# getopt-test -ab
aflag = 1, bflag = 1, cvalue = (null)

# getopt-test -c test
aflag = 0, bflag = 0, cvalue = test

# getopt-test argument
aflag = 0, bflag = 0, cvalue = (null)
Non-option argument argument

# getopt-test -b argument -c test -a
aflag = 1, bflag = 1, cvalue = test
Non-option argument argument

# getopt-test -b argument -c test -a mehr
aflag = 1, bflag = 1, cvalue = test
Non-option argument argument
Non-option argument mehr

# getopt-test -f
Unknown option `-f'.
```

Erklärungen zum Beispiel-Programm:

- drittes `getopt`-Argument gibt zulässige Optionen an, im Beispiel: `"abc:"`
- Option `-c` erwartet ein Argument (Postfix `:`)
- `optopt` enthält letzte Option, wenn auf `:`-Option kein passendes Argument folgt
- `opterr = 0` unterdrückt `getopt`-eigene Fehlerausgabe bei unbekannter Option

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.)

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

`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);
    ```
  - ```
    struct stat s;
    lstat (dateiname, &s);
    ```
- Wenn Datei ein Symlink ist, gibt `stat()` Informationen über verlinkte Datei aus; bei `lstat()` ist es der Link selbst

Aufbau von `struct stat`

```
 43: struct stat
 44:   {
 45:     __dev_t st_dev;             /* Device.  */
 50:     __ino_t st_ino;             /* File serial number.  */
 58:     __nlink_t st_nlink;         /* Link count.  */
 59:     __mode_t st_mode;           /* File mode.  */
 61:     __uid_t st_uid;             /* User ID of the file's owner. */
 62:     __gid_t st_gid;             /* Group ID of the file's group.*/
 66:     __dev_t st_rdev;            /* Device number, if device.  */
 71:     __off_t st_size;            /* Size of file, in bytes.  */
 75:     __blksize_t st_blksize;     /* Optimal block size for I/O.  */
 77:     __blkcnt_t st_blocks;       /* Number 512-byte blocks allocated. */
 95:     __time_t st_atime;          /* Time of last access.  */
 97:     __time_t st_mtime;          /* Time of last modification.  */
 99:     __time_t st_ctime;          /* Time of last status change.  */
112:   };
```

- 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 (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%o\n", s.st_mode);
}

root@ubu64:~# ./stattest
s.st_mode: o100644
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

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)
```

Fehler bei `stat()`; Status geöffneter Dateien

- `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
```

- Alternative `fstat()` verwendet file descriptor (einer geöffneten Datei)

```
int fd = open (...);
struct stat s;
fstat (fd, &s);
```


Symlink erzeugen / Hardlink erzeugen

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

- 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

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

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

Zugriffsrechte: (f)chmod()

- `chmod (datei, mode)`
- `lchmod (datei, m)`
  (folgt Symlinks nicht)
- `fchmod (fd, mode)`
  (mit file descriptor `fd`)
- `mode`: siehe rechts,
  und siehe Folie C-35

```
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(), (f)chdir()

- 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)
- aktuelles Arbeitsverzeichnis ändern:

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

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`)

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 (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

Verzeichnisliste (3)

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

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

  if (argv != 2) { printf ("readdir Verzeichnis\n"); return 0; };
  if ((dirp = opendir(argc[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)
```


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 file	chown(file,u,-1)
chgrp g file	chown(file,-1,g)
chown -h	lchown()
chgrp -h	lchown()
chmod	chmod()
pwd	getcwd()
cd	chdir()
mkdir	mkdir()
rmdir	rmdir()
ls, stat	opendir(), readdir(), stat()