

PR: Programmiersprache C (Linux) Einheit 8: Files, Directories, Processes

Roland A. Eggetberger

Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

Files

Filename
Löschen
Rechte
Status
Links
Temporäre Files

Filename

- File Umbenennen

```
#include <stdio.h>
int rename(const char *oldpath,
           const char *newpath);
```

Wechsel zwischen Directories erlaubt
Falls newpath existiert wird es ersetzt
Rückgabewert 0 bzw. -1 im Fehlerfall

Löschen

- File Löschen

```
#include <stdio.h>
int remove(const char *pathname);
```

Es wird nur der angegebene Link gelöscht
Andere Referenzen sind nicht betroffen
Rückgabewert 0 bzw. -1 im Fehlerfall

Rechte (I)

- Prüfen (I)

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

Überprüft Rechte auf einen Pfad zuzugreifen
Rückgabewert 0 bzw. -1 im Fehlerfall

Rechte (II)

- Prüfen (II)

Modi
 R_OK ... Lesen
 W_OK ... Schreiben
 X_OK ... Ausführen
 F_OK ... Existenz eines Files

Rechte (III)

- Ändern (I)

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *path, mode_t mode);
int fchmod(int filedes, mode_t mode);
```

Ändern der Zugriffsrechte

Modi wie in Einheit 7, Seite 29/30

Rechte (IV)

- Ändern (II)

```
#include <sys/types.h>
#include <unistd.h>
int chown(const char *path,
          uid_t owner, gid_t group);
int fchown(int fd,
           uid_t owner, gid_t group);
```

Ändern der Besitzrechte

Rechte (V)

- User Identity (I)

```
#include <unistd.h>
int setuid(uid_t uid)
```

UID (effektive) setzen

Rechte (VI)

- User Identity (II)

```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void);
uid_t geteuid(void);
```

Real User ID (aufrufender Prozess) bzw.
Effective User ID zurückgeben

Rechte (VII)

- Group Identity (I)

```
#include <unistd.h>
int setgid(gid_t gid)
```

GID (effektive) setzen

Rechte (VIII)

- Group Identity (II)

```
#include <unistd.h>
#include <sys/types.h>
gid_t getgid(void);
gid_t getegid(void);
```

Real Group ID (aufrufender Prozess) bzw.
Effective Group ID zurückgeben

Status (I)

- Status bestimmen

```
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *file_name,
         struct stat *buf);
int fstat(int filedes, struct stat *buf);
```

Filestatus des angegebenen Files

Status (II)

- Statusstruktur (I)

```
struct stat {
    dev_t    st_dev;      /* device */
    ino_t    st_ino;      /* inode */
    mode_t   st_mode;    /* protection */
    nlink_t  st_nlink;   /* hard links */
    uid_t    st_uid;     /* owner user ID */
    gid_t    st_gid;     /* owner grp ID */
    dev_t    st_rdev;    /* device type */
```

Status (III)

- Statusstruktur (II)

```
off_t    st_size;      /* size (bytes) */
unsigned long st_blksize;
unsigned long st_blocks;
time_t   st_atime;    /* last access */
time_t   st_mtime;    /* last modified */
time_t   st_ctime;    /* last change */
};
```

Links (I)

- Festlegen

```
#include <unistd.h>
int link(const char *oldpath,
         const char *newpath);
```

Link auf `newpath` festlegen

Links (II)

- Löschen

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

Link auf `pathname` löschen

Links (III)

- Symbolische Links (I)

```
#include <unistd.h>
int symlink(const char *oldpath,
            const char *newpath);
```

Symbolischen Link auf `newpath` festlegen

Links (IV)

- Symbolische Links (II)

```
#include <unistd.h>
int readlink(const char *path, char *buf,
             size_t bufsiz);
```

Symbolischen Link bestimmen
 Rückgabewert ist die Anzahl der übertragenen Zeichen bzw. -1 im Fehlerfall

Temporäre Files (I)

- Anlegen (I)

```
#include <stdio.h>
FILE *tmpfile (void);
```

Temporäres File anlegen
 Wird nach dem Schliessen automatisch gelöscht

Temporäre Files (II)

- Anlegen (II)

```
#include <stdlib.h>
int mkstemp(char *template);
```

Temporäres File anlegen
 Eingreifen in die Namengebung möglich

Directories

Anlegen
 Wechseln
 Löschen
 Verarbeitung

Anlegen

- Directory Anlegen

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
int mkdir(const char *pathname,
          mode_t mode);
```

Beim Anlegen werden auch Rechte festgelegt

Wechseln

- Directory Wechseln

```
#include <unistd.h>
int chdir(const char *path);
int fchdir(int fd);
```

Mit Pfadname oder File Descriptor

Löschen

- Directory Löschen

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

Verarbeitung (I)

- Auch bei Directories wird das Stream-Konzept verwendet
- Man kann daher wie mit Files arbeiten
- Es wird allerdings ein eigener Typ (`DIR`) benutzt

Verarbeitung (II)

- Öffnen

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

Durch Angabe eines Pfades wird ein Stream erzeugt

Im Fehlerfall wird `NULL` zurückgegeben

Verarbeitung (III)

- Lesen (I)

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dir);
```

Rückgabe von `struct dirent`

Im Fehlerfall wird `NULL` zurückgegeben

Verarbeitung (IV)

- Lesen (II)

```
struct dirent {
    size_t d_ino;
    size_t d_off;
    unsigned short int d_reclen;
    unsigned char d_type;
    char d_name[256];
};
```

Verarbeitung (V)

- Schliessen

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dir);
```

Rückgabewert 0 bzw. -1 im Fehlerfall

Verarbeitung (VI)

- Positionszeiger setzen (I)

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir(DIR *dir);
```

Setzen an den Anfang

Verarbeitung (VII)

- Positionszeiger setzen (II)

```
#include <dirent.h>
void seekdir(DIR *dir, off_t offset);
```

Setzen an eine beliebige Position
Rückgabewert von `telldir` verwenden

Verarbeitung (VIII)

- Position ermitteln

```
#include <dirent.h>
off_t telldir(DIR *dir);
```

Rückgabewert -1 im Fehlerfall

Verarbeitung (IX)

- Directory durchsuchen (I)

```
#include <dirent.h>
int scandir(const char *dir,
            struct dirent ***namelist,
            int (*select)(const struct
                         dirent *),
            int (*compar)(const struct
                          dirent **, const struct dirent
                          **));
```

Verarbeitung (X)

- Directory durchsuchen (II)
- Es wird `dir` durchsucht
Einträge kommen in `namelist`
`select` wählt Einträge aus (0 = alle)
`compar` sortiert sie

Rückgabewert Anzahl der Einträge bzw. -1 im Fehlerfall

Verarbeitung (XI)

- Directory durchsuchen (III)

```
int alphasort(const struct dirent **a,
              const struct dirent **b);
```

Rückgabewert kleiner, gleich oder größer Null

Verarbeitung (XII)

- Beispiel (I)


```
/* print files in current directory in */
/* reverse order */
#include <dirent.h>
main() {
    struct dirent **namelist;
    int n;
```

Verarbeitung (XIII)

- Beispiel (II)


```
n = scandir(".", &namelist, 0,
               alphasort);
if (n < 0)
    perror("scandir");
else
    while (n--)
        printf("%s\n", namelist[n]->d_name);
```

Processes

- Kommando
- Aufspalten
- Warten
- PID
- Umgebung
- Beenden

Kommando

- Kommando Ausführen


```
#include <unistd.h>
int execl(const char *path,
          const char *arg, ...);
int execv(const char *path,
          char *const argv[]);
```

Aufspalten

- Erzeugen eines Child-Prozesses


```
#include <unistd.h>
pid_t fork(void);
```

Ein Prozess wird in zwei aufgespalten

Warten

- Warten auf den Child-Prozess


```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status,
              int options);
```

PID

- PID lesen

```
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Process-ID von Child- bzw. Parent-Prozess lesen

Umgebung

- Umgebung festlegen

```
#include <unistd.h>
extern char **environ;
```

Umgebungsvariablen als Strings der Form

```
name=value
```

Beenden (I)

- Beenden selbst

```
#include <stdlib.h>
void exit(int status);
```

Rückgabe eines Status an den Parent-Prozess

Beenden (II)

- Aufräumen (I)

```
#include <stdlib.h>
int atexit(void (*function)(void));
```

Bei Beenden des Prozesses wird die übergebene Funktion aufgerufen

Die Funktion hat keine Parameter

Beenden (III)

- Aufräumen (II)

```
#include <stdlib.h>
int on_exit(void (*function)(int, void *),
           void *arg);
```

Bei Beenden des Prozesses wird die übergebene Funktion aufgerufen

Parameter sind der Rückgabewert und das zweite Argument von `on_exit`