Übungen zu Systemnahe Programmierung in C (SPiC)

Moritz Strübe, Rainer Müller
(Lehrstuhl Informatik 4)

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Wintersemester 2013

Inhalt

opendir, closedir, readdir

- Funktions-Prototypen (details siehe Vorlesung)

```
1 #include <sys/types.h>
2 #include <dirent.h>
3 DIR *opendir(const char *dirname);
4 int closedir(DIR *dirp);
5 struct dirent *readdir(DIR *dirp);
```

- Rückgabewert von `readdir`
  - Zeiger auf Datenstruktur vom Typ `struct dirent`
  - `NULL`, wenn `EOF` erreicht wurde **oder** im Fehlerfall

  ⇝ bei `EOF` bleibt errno unverändert (auch wenn `errno != 0`), im Fehlerfall wird `errno` entsprechend gesetzt

Fehlerbehandlung bei readdir

- Fehlerprüfung durch Setzen und Prüfen von `errno`

```
1  #include <errno.h>
2  ...
3      struct dirent *ent;
4      while(1) {
5          errno = 0;
6          ent = readdir(...);
7          if(ent == NULL) break;
8          ... /* keine weiteren break-Statements in der Schleife */
9      }
10     /* EOF oder Fehler? */
11     if(errno != 0) {
12         /* Fehler */
13         ...
14     }
```

- `errno=0` unmittelbar vor Aufruf der problematischen Funktion

  ⇒ `errno` wird nur im Fehlerfall gesetzt und bleibt sonst evtl. unverändert
- Abfrage der errno unmittelbar nach Rückgabe des pot. Fehlerwerts

  ⇒ `errno` könnte sonst durch andere Funktion verändert werden

Fehlerbehandlung bei readdir

- Fehlerprüfung durch Setzen und Prüfen von `errno`

```c
#include <errno.h>
...
    struct dirent *ent;
    while(errno=0, (ent=readdir()) != NULL) {



        ... /* keine weiteren break-Statements in der Schleife */
    }
    /* EOF oder Fehler? */
    if(errno != 0) {
        /* Fehler */
        ...
    }
```

- `errno=0` unmittelbar vor Aufruf der problematischen Funktion

⇒ `errno` wird nur im Fehlerfall gesetzt und bleibt sonst evtl. unverändert

- Abfrage der errno unmittelbar nach Rückgabe des pot. Fehlerwerts

⇒ `errno` könnte sonst durch andere Funktion verändert werden

Datei-Attribute ermitteln: stat

- `readdir(3)` liefert nur Name und Typ eines Verzeichniseintrags
- Weitere Attribute stehen im Inode
- `stat(2)` Funktions-Prototyp:

```c
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *path, struct stat *buf);
```

- Argumente:
  - `path`: Dateiname
  - `buf`: Zeiger auf Puffer, in den Inode-Informationen eingetragen werden
- Rückgabewert: `0` wenn OK, `-1` wenn Fehler
- Beispiel:

```c
struct stat buf;
stat("/etc/passwd", &buf); /* Fehlerabfrage ... */
printf("Inode-Nummer: %ld\n", buf.st_ino);
```

Das struct stat

- Ausgewählte Elemente
  - `dev_t st_dev` Gerätenummer (des Dateisystems) = Partitions-Id
  - `ino_t st_ino` Inodenummer (Tupel `st_dev`, `st_ino` eindeutig im System)
  - `mode_t st_mode` Dateimode, u.a. Zugriffs-Bits und Dateityp
  - `nlink_t st_nlink` Anzahl der (Hard-) Links auf den Inode
  - `uid_t st_uid` UID des Besitzers
  - `gid_t st_gid` GID der Dateigruppe
  - `dev_t st_rdev` DeviceID, nur für Character oder Blockdevices
  - `off_t st_size` Dateigröße in Bytes
  - `time_t st_atime` Zeit des letzten Zugriffs (in Sekunden seit 1.1.1970)
  - `time_t st_mtime` Zeit der letzten Veränderung (in Sekunden ...)
  - `time_t st_ctime` Zeit der letzten Änderung der Inode-Information (...)
  - `unsigned long st_blksize` Blockgröße des Dateisystems
  - `unsigned long st_blocks` Anzahl der von der Datei belegten Blöcke

stat / lstat: st_mode

- `st_mode` enthält Informationen über den Typ des Eintrags:
  - `S_IFMT` 0170000 bitmask for the file type bitfields
  - `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
- Zur einfacheren Auswertung werden Makros zur Verfügung gestellt:
  - `S_ISREG(m)` - is it a regular file?
  - `S_ISDIR(m)` - directory?
  - `S_ISCHR(m)` - character device?
  - `S_ISLNK(m)` - symbolic link?