U6 6. Übung

U6-1 Überblick

- Infos zur Aufgabe 5: Dateisystem, Directories
- Dateisystemschnittstelle

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

U6-2 Aufgabe 4: Verzeichnisse

- opendir(3), readdir(3), rewinddir(3), telldir(3), seekdir(3), closedir(3)
- stat(2), lstat(2)
- readlink(2)
- getpwuid(3), getgrgid(3)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 opendir / closedir

- Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);

int closedir(DIR *dirp);
```

- Argument von opendir
  - **dirname**: Verzeichnisname
- Rückgabewert: Zeiger auf Datenstruktur vom Typ **DIR** oder **NULL**
- initialisiert einen internen Zeiger des directory-Funktionsmoduls auf den ersten Directory-Eintrag
  (für den ersten readdir-Aufruf)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 readdir

- liefert einen Directory-Eintrag (interner Zeiger) und setzt den Zeiger auf den folgenden Eintrag
- Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
```

- Argumente
  - **dirp**: Zeiger auf **DIR**-Datenstruktur
- Rückgabewert: Zeiger auf Datenstruktur vom Typ **struct dirent** oder **NULL** wenn EOF erreicht wurde oder im Fehlerfall
  - ➤ bei EOF bleibt **errno** unverändert (!!! kritisch, kann vorher beliebigen Wert haben), im Fehlerfall wird **errno** entsprechend gesetzt
  - ➤ **errno** vorher auf 0 setzen, sonst kann EOF nicht sicher erkannt werden!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 … readdir

- Problem: Der Speicher für zurückgelieferte **`struct dirent`** wird von der Bibliotheksfunktion selbst angelegt und bei jedem Aufruf wieder verwendet!
  - ◆ werden Daten aus der dirent-Struktur länger benötigt, müssen sie vor dem nächsten readdir-Aufruf in Sicherheit gebracht (kopiert) werden
  - ◆ konzeptionell schlecht
    - ➤ aufrufende Funktion arbeitet mit Zeiger auf internen Speicher der readdir-Funktion
  - ◆ in nebenläufigen Programmen (mehrere Threads) nicht einsetzbar!
    - ➤ man weiss evtl. nicht, wann der nächste readdir-Aufruf stattfindet
- readdir ist ein klassisches Beispiel für schlecht konzipierte Schnittstellen in der C-Funktionsbibliothek
  - ➤ wie auch `gets`, `strdup`, `getpwent` und viele andere

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 readdir_r

- *reentrant*-Variante von readir
  - ➤ Speicher der `struct dirent` wird nicht von der Funktion bereitgestellt sondern wird vom Aufrufer übergeben und die Funktion füllt ihn aus
- Funktions-Prototyp:

```
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

- Argumente
  - ◆ **`dirp`**: Zeiger auf **`DIR`**-Datenstruktur
  - ◆ Zeiger auf dirent-Struktur
  - ◆ über das dritte Argument wird im Erfolgsfall der im zweiten Argument übergebene Zeiger zurückgeliefert, sonst NULL
- Ergebnis: im Erfolgsfall 0, sonst eine Fehlernummer

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 struct dirent

- Definition unter Linux (/usr/include/bits/dirent.h)

```
struct dirent {
    __ino_t d_ino;
    __off_t d_off;
    unsigned short int d_reclen; /* tatsächl. Länge der Struktur */
    unsigned char d_type;
    char d_name[256];
};
```

- Definition unter Solaris (/usr/include/sys/dirent.h)

```
typedef struct dirent {
    ino_t           d_ino;
    off_t           d_off;
    unsigned short  d_reclen; /* tatsächl. Länge der Struktur */
    char            d_name[1];
} dirent_t;
```

- POSIX: `d_name` ist ein Feld unbestimmter Länge, max. `NAME_MAX` Zeichen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 rewinddir

- setzt den internen Zeiger des directory-Funktionsmoduls zurück
  - ➤ nächster readdir-Aufruf liefert den ersten Directory-Eintrag
- Funktions-Prototyp:

```
void rewinddir(DIR *dirp);
```

6 telldir / seekdir

- telldir fragt aktuelle Position des internen Zeigers ab (Ergebnis)
- seekdir setzt ihn auf einen zuvor abgefragten Wert (Parameter `loc`)
- Funktions-Prototypen:

```
long int telldir(DIR *dirp);
void seekdir(DIR *dirp, long int loc);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 stat / lstat

- liefert Datei-Attribute aus dem Inode
- Funktions-Prototyp:

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

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


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 stat: Ergebnisrückgabe im Vergleich zur readdir

- problematische Rückgabe auf funktions-internen Speicher wie bei **readdir** gibt es bei **stat** nicht
- Grund: **stat** ist ein Systemaufruf - Vorgehensweise wie bei **readdir** wäre gar nicht möglich
  - ➤ Vergleiche Vorlesung Seite 5-33
  - ➤ **readdir** ist komplett auf Ebene 3 implementiert (Teil der Standard-C-Bibliothek - Laufzeitbibliothek, siehe Vorl. Seite 5-26 / 5-30)
  - ➤ **stat** ist nur ein Systemaufruf(-stumpf), die Funktion selbst ist Teil des Betriebssystems (Ebene 2)
- der logische Adressraum auf Ebene 3 (Anwendungsprogramm) ist nur eine Teilmenge (oder sogar komplett disjunkt) von dem logischen Adressraum auf Ebene 2 (Betriebssystemkern)
  - ➤ Betriebssystemspeicher ist für Anwendung nicht sichtbar/zugreifbar
  - ➤ Funktionen der Ebene 2 (wie stat) können keine Zeiger auf ihre internen Datenstrukturen an Ebene 3 zurückgeben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 stat / lstat: stat-Struktur

- **`dev_t st_dev;`** Gerätenummer (des Dateisystems)
- **`ino_t st_ino;`** Inodenummer
- **`mode_t st_mode;`** Dateimode, u.a. Zugriffs-Bits
- **`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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 stat- Zugriffsrechte

- in dem Strukturelement **`st_mode`** sind die Zugriffsrechte (12 Bit) und der Dateityp (4 Bit) kodiert.
- UNIX sieht folgende Zugriffsrecht vor (davor die Darstellung des jeweiligen Rechts bei der Ausgabe des ls-Kommandos)
  - **`r`** lesen (getrennt für *User*, *Group* und *Others* einstellbar)
  - **`w`** schreiben (analog)
  - **`x`** ausführen (bei regulären Dateien) bzw. Durchgriffsrecht (bei Dir.)
  - **`s`** setuid/setgid-Bit: bei einer ausführbaren Datei mit dem Laden der Datei in einen Prozess (exec) erhält der Prozess die User (bzw. Group)-Rechte des Dateieigentümers
  - **`t`** bei Directories: es dürfen trotz Schreibrecht im Directory nur eigene Dateien gelöscht werden
  - ➤ **`s`** wird anstelle von x ausgegeben und bedeutet "**`s`** und **`x`**", **`t`** analog
  - ➤ **`S`** bedeutet, **`x`** darunter ist nicht gesetzt — hat in manchen UNIX-Systemen besondere Semantik im Zusammenhang mit file-locking

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 readlink

- Funktions-Prototyp:

```
#include <unistd.h>

int readlink(const char *path, char *buf, size_t bufsiz);
```

- Argumente
  - **path**: Dateiname
  - **buf**: Puffer für Link-Inhalt
    - ➤ Vorsicht: es wird einfach der Link-Inhalt in **buf** kopiert - die Daten werden von readlink nicht explizit mit '\0' terminiert
      - ➥ entweder buf mit Nullen initialisieren oder '\0' explizit am Ende des Link-Inhalts eintragen (Rückgabewert von readlink = Länge)
  - **bufsiz**: Größe des Puffers
- Rückgabewert: Anzahl der in buf geschriebenen Bytes oder -1

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 getpwuid

- Funktions-Prototyp:

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
```

- struct passwd:
  - char *pw_name; /* user's login name */
  - uid_t pw_uid; /* user's uid */
  - gid_t pw_gid; /* user's gid */
  - char *pw_gecos; /* typically user's full name */
  - char *pw_dir; /* user's home dir */
  - char *pw_shell; /* user's login shell */

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 getgrgid

- Prototyp:

```
#include <grp.h>
struct group *getgrgid(gid_t gid);
```

- struct group:
  - char *gr_name; /* the name of the group */
  - char *gr_passwd; /* the encrypted group password */
  - gid_t gr_gid; /* the numerical group ID */
  - char **gr_mem; /* vector of pointers to member names */

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

U6-3 Dateisystem Systemcalls

- open(2) / close(2)
- read(2) / write(2)
- lseek(2)
- chmod(2)
- fstat(2)
- umask(2)
- utime(2)
- truncate(2)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

1 open

- Funktions-Prototyp:

```
#include <fcntl.h>
int open(const char *path, int oflag, ... /* [mode_t mode] */ );
```

- Argumente:
  - Maximallänge von path: **PATH_MAX**
  - **oflag**: Lese/Schreib-Flags, Allgemeine Flags, Synchronisierungs I/O Flags
    - Lese/Schreib-Flags: **O_RDONLY, O_WRONLY, O_RDWR**
    - Allgemeine Flags: **O_APPEND, O_CREAT, O_EXCL, O_LARGEFILE, O_NDELAY, O_NOCTTY, O_NONBLOCK, O_TRUNC**
    - Synchronisierung: **O_DSYNC, O_RSYNC, O_SYNC**
  - **mode**: Zugriffsrechte der erzeugten Datei (nur bei **O_CREAT**) - siehe **chmod**
- Rückgabewert
  - Filedeskriptor oder -1 im Fehlerfall (**errno** wird gesetzt)

SoS I - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

1 open - Flags

- **O_EXCL**: zusammen mit **O_CREAT** - nur *neue* Datei anlegen
- **O_TRUNC**: Datei wird beim Öffnen auf 0 Bytes gekürzt
- **O_APPEND**: vor jedem Schreiben wird der Dateizeiger auf das Dateiende gesetzt
- **O_NDELAY, O_NONBLOCK**: Operationen arbeiten nicht-blockierend (bei Pipes, FIFOs und Devices)
  - open kehrt sofort zurück
  - read liefert -1 zurück, wenn keine Daten verfügbar sind
  - wenn genügend Platz ist, schreibt write alle Bytes, sonst schreibt write nichts und kehrt mit -1 zurück
- **O_NOCTTY**: beim Öffnen von Terminal-Devices wird das Device nicht zum Kontroll-Terminal des Prozesses

SoS I - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

1 open - Flags (2)

- Synchronisierung
  - **O_DSYNC**: Schreibaufruf kehrt erst zurück, wenn Daten in Datei geschrieben wurden (Blockbuffer Cache!!)
  - **O_SYNC**: ähnlich **O_DSYNC**, zusätzlich wird gewartet, bis Datei-Attribute wie Zugriffszeit, Modifizierungszeit, auf Disk geschrieben sind
  - **O_RSYNC|O_DSYNC**: Daten die gelesen wurden, stimmen mit Daten auf Disk überein, d.h. vor dem Lesen wird der Buffercache geflushet
    O_RSYNC|O_SYNC: wie **O_RSYNC|O_DSYNC**, zusätzlich Datei-Attribute

SoS I - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

2 close

- Funktions-Prototyp:

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

- Argumente:
  - **fildes**: Filedeskriptor der zu schließenden Datei
- Rückgabewert:
  - 0 bei Erfolg, -1 im Fehlerfall

SoS I - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

3 read

- Funktions-Prototyp:

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

- Argumente
  - **fildes**: Filedeskriptor, z.B. Rückgabe vom open-Aufruf
  - **buf**: Zeiger auf Puffer
  - **nbyte**: Größe des Puffers
- Rückgabewert
  - Anzahl der gelesenen Bytes oder -1 im Fehlerfall

```
char buf[1024];
int fd;
fd = open("/etc/passwd", O_RDONLY);
if (fd == -1) ...
read(fd, buf, 1024);
```

SoSI - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Hintergrund-Info

4 write

- Funktions-Prototyp

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

- Argumente
  - äquivalent zu **read**
- Rückgabewert
  - Anzahl der geschriebenen Bytes oder -1 im Fehlerfall

SoSI - Ü

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 lseek

- Funktions-Prototyp

```
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

- Argumente
  - **fildes**: Filedeskriptor
  - **offset**: neuer Wert des Dateizeigers
  - **whence**: Bedeutung von offset
    - **SEEK_SET**: absolut vom Dateianfang
    - **SEEK_CUR**: Inkrement vom aktuellen Stand des Dateizeigers
    - **SEEK_END**: Inkrement vom Ende der Datei
- Rückgabewert
  - Offset in Bytes vom Beginn der Datei oder -1 im Fehlerfall

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 chmod

- Funktions-Prototyp:

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

- Argumente:
  - **path**: Dateiname
  - **mode**: gewünschter Dateimodus, z.B.
    - **S_IRUSR**: lesbar durch Besitzer
    - **S_IWUSR**: schreibbar durch Benutzer
    - **S_IRGRP**: lesbar durch Gruppe
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel:

```
chmod("/etc/passwd", S_IRUSR | S_IRGRP);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 fstat

- Funktions-Prototyp:

```
int fstat(int filedes, struct stat *buf);
```

- wie **stat**, aber Deskriptor einer geöffneten Datei statt Dateiname

8 umask

- Funktions-Prototyp:

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
```

- Argumente
  - **cmask**: gibt Permission-Bits an, die beim Erzeugen einer Datei ausgeschaltet werden sollen
- Rückgabewert: voriger Wert der Maske

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 utime

- Funktions-Prototyp:

```
#include <utime.h>
int utime(const char *path, const struct utimbuf *times);
```

- Argumente
  - **path**: Dateiname
  - **times**: Zugriffs- und Modifizierungszeit (in Sekunden)
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel: setze atime und mtime um eine Stunde zurück

```
struct utimebuf times;
struct stat buf;
stat("/etc/passwd", &buf); /* Fehlerabfrage */
times.actime = buf.st_atime - 60 * 60;
times.modtime = buf.st_mtime - 60 * 60;
utime("/etc/passwd", &times); /* Fehlerabfrage */
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 truncate

- Funktions-Prototyp:

```
#include <unistd.h>
int truncate(const char *path, off_t length);
```

- Argumente:
  - **path**: Dateiname
  - **length**: gewünschte Länge der Datei
- Rückgabewert: 0 wenn OK, -1 wenn Fehler

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

U6-4 POSIX I/O vs. Standard-C-I/O

- POSIX Funktionen open/close/read/write/... arbeiten mit Filedeskriptoren
- Standard-C Funktionen fopen/fclose/fgets/... arbeiten mit Filepointern
- Konvertierung von Filepointer nach Filedeskriptor

```
#include <stdio.h>
int fileno(FILE *stream);
```

- Konvertierung von Filedeskriptor nach Filepointer

```
#include <stdio.h>
FILE *fdopen(int fd, const char* type);
```

  - type kann sein "r", "w", "a", "r+", "w+", "a+"
    (**fd** muß entsprechend geöffnet sein!)
- Filedeskriptoren in **<unistd.h>**:
  STDIN_FILENO, **STDOUT_FILENO**, **STDERR_FILENO**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.