

Übungen zu Systemnahe Programmierung in C

Abschnitt 11.1: Dateien und Dateisystem

06.07.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Dateien & Dateikanäle

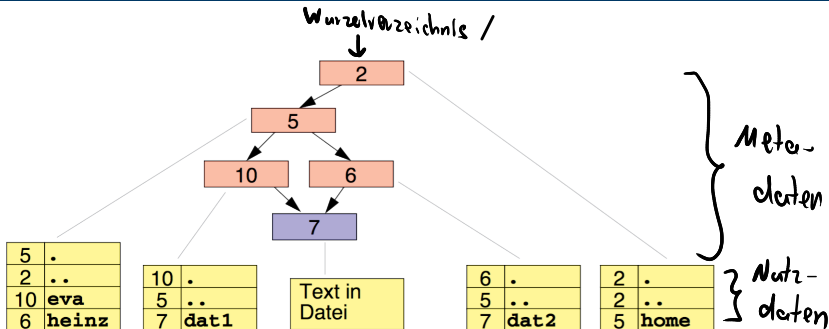


- Ein- und Ausgaben erfolgen über gepufferte Dateikanäle
- `FILE *fopen(const char *path, const char *mode);`
 - Öffnet eine Datei zum Lesen oder Schreiben (je nach mode)
 - Liefert einen Zeiger auf den erzeugten Dateikanal
 - `r` Lesen
 - `r+` Lesen & Schreiben
 - `w` Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
 - `w+` Lesen & Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
 - `a` Schreiben am Ende der Datei; Datei wird ggf. erstellt
 - `a+` Schreiben am Ende der Datei; Lesen am Anfang; Datei wird ggf. erstellt
- `int fclose(FILE *fp);`
 - Schreibt ggf. gepufferte Ausgabedaten des Dateikanals
 - Schließt anschließend die Datei



- Standardmäßig geöffnete Dateikanäle
 - `stdin` Eingaben
 - `stdout` Ausgaben
 - `stderr` Fehlermeldungen
- `int fgetc(FILE *stream);`
 - Liest ein einzelnes Zeichen aus der Datei
- `char *fgets(char *s, int size, FILE *stream);`
 - Liest max. size Zeichen in einen Buffer ein
 - Stoppt bei Zeilenumbruch und EOF
- `int fputc(int c, FILE *stream);`
 - Schreibt ein einzelnes Zeichen in die Datei
- `int fputs(const char *s, FILE *stream);`
 - Schreibt einen null-terminierten String (ohne das Null-Zeichen)

POSIX Verzeichnisschnittstelle



inode: Enthält Dateiattribute & Verweise auf Datenblöcke

Datei: Block mit beliebigen Daten

Verzeichnis: Spezielle Datei mit Paaren aus Namen & inode-Nummer

hard links: /home/eva/dat1 /home/heinz/dat2



- `DIR *opendir(const char *name);`
 - Öffnet ein Verzeichnis
 - Liefert einen Zeiger auf den Verzeichniskanal
- `struct dirent *readdir(DIR *dirp);`
 - Liest einen Eintrag aus dem Verzeichniskanal und gibt einen Zeiger auf die Datenstruktur `struct dirent` zurück
- `int closedir(DIR *dirp);`
 - Schließt den Verzeichniskanal



```
01 struct dirent {
02     ino_t          d_ino;          // inode number
03     off_t          d_off;          // not an offset; see NOTES
04     unsigned short d_reclen;       // length of this record
05     unsigned char  d_type;         // type of file; not supported
06                                     // by all filesystem types
07     char           d_name[256];    // filename
08 };
```

- Entnommen aus Manpage `readdir(3)`
- Nur `d_name` und `d_ino` Teil des POSIX-Standards
- Relevant für uns: Dateiname (`d_name`)



- Fehlerprüfung durch Setzen und Prüfen der `errno`:

```
01 #include <errno.h>
02 // [...]
03     DIR *dir = opendir("/home/eva/"); // Fehlerbehandlung!!
04
05     struct dirent *ent;
06     while(1) {
07         errno = 0;
08         ent = readdir(dir);
09         if(ent == NULL) {
10             break;
11         }
12         // keine weiteren break-Statements in der Schleife
13         // [...]
14     }
15
16     // EOF oder Fehler?
17     if(errno != 0) { // Fehler
18         // [...]
19     }
20     closedir(dir);
```



- Funktionsweise:

1. Auswertung des ersten Ausdrucks (Verwerfen dieses Ergebnisses)
2. Auswertung des zweiten Ausdrucks (Rückgabe dieses Ergebnisses)

```
01 int c = (add(3,2), sub(3,2));
```

- Geeignet für Initialisierungen vor Überprüfung der Schleifenbedingung

⇒ cli()/sei()

```
01 while(cli(), event != 0) {  
02     sleep_enable();  
03     sei();  
04     sleep_cpu();  
05     ...  
06 }  
07 sei();
```

- Elegant, aber keine Notwendigkeit!



- Fehlerprüfung durch Setzen und Prüfen der `errno`:

```
01 #include <errno.h>
02 // [...]
03     DIR *dir = opendir("/home/eva/");
04     if(dir == NULL) {
05         perror("opendir");
06         exit(EXIT_FAILURE);
07     }
08
09     struct dirent *ent;
10     while(errno=0, (ent=readdir(dir)) != NULL) {
11         // keine weiteren break-Statements in der Schleife
12         // [...]
13     }
14
15     // EOF oder Fehler?
16     if(errno != 0) { // Fehler
17         // [...]
18     }
19     closedir(dir);
```

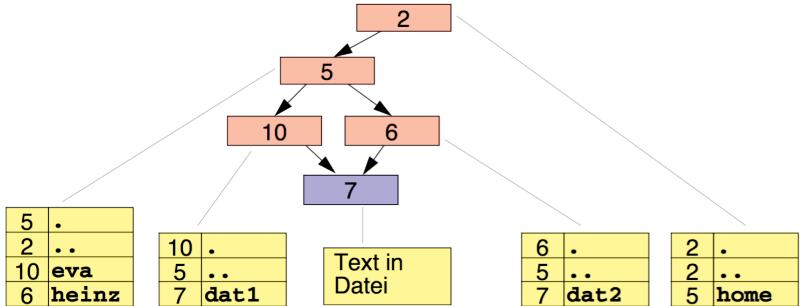


- `readdir(3)` liefert **nur Name und inode-Nummer** eines Verzeichniseintrags
- Weitere Attribute stehen im **inode**

- `int stat(const char *path, struct stat *buf);`
 - Abfragen der Attribute eines Eintrags (folgt symlinks)
- `int lstat(const char *path, struct stat *buf);`
 - Abfragen der Attribute eines Eintrags (folgt symlinks nicht)

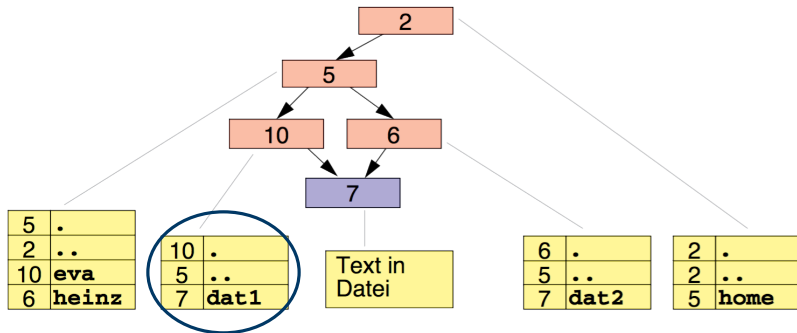


- Inhalte des inode sind u.a.:
 - Geräte- und inode-Nummer
 - Eigentümer und Gruppenzugehörigkeit
 - Dateityp und -rechte
 - Dateigröße
 - Zeitstempel (letzte(r) Veränderung, Zugriff, ...)
 - ...
- Der Dateityp ist im Feld `st_mode` codiert
 - Reguläre Datei, Ordner, symbolischer Verweis (*symbolic link*), ...
 - Zur einfacheren Auswertung
 - `S_ISREG(m)` - is it a regular file?
 - `S_ISDIR(m)` - directory?
 - `S_ISCHR(m)` - character device?
 - `S_ISLNK(m)` - symbolic link?



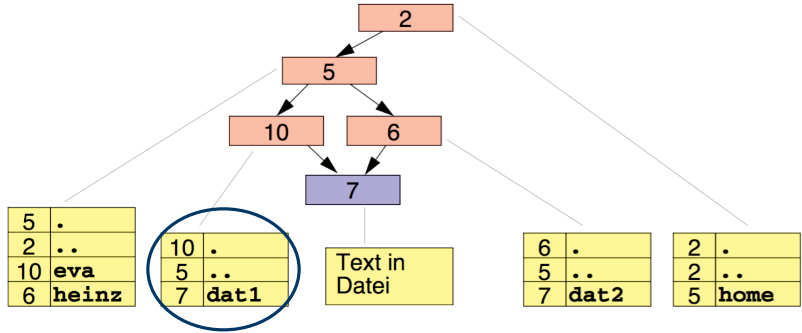
```

01 $> find /
02 /home
03 /home/eva
04 /home/eva/dat1
05 /home/heinz
06 /home/heinz/dat2
  
```



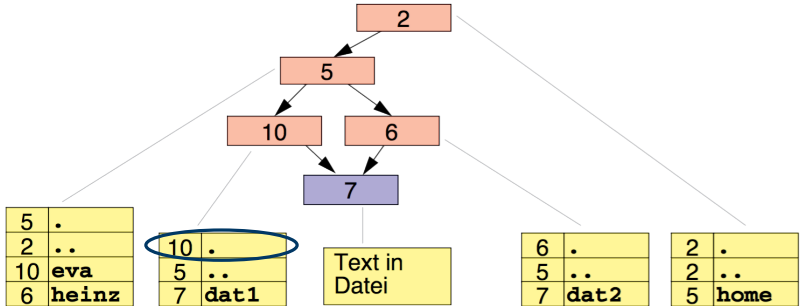
```

01 DIR *dir = opendir("/home/eva/");
02 if(dir == NULL) {
03     perror("opendir");
04     exit(EXIT_FAILURE);
05 }
    
```



```

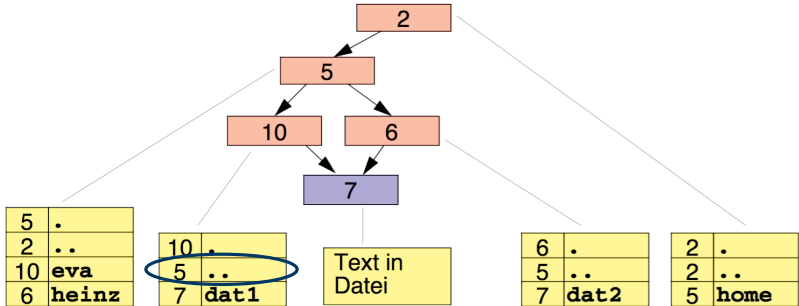
01 struct dirent *ent;
02 while(errno = 0, (ent = readdir(dir)) != NULL) {
03     //...
04 }
05
06 if(errno != 0) {
07     perror("readdir"); exit(EXIT_FAILURE);
08 }
    
```

```

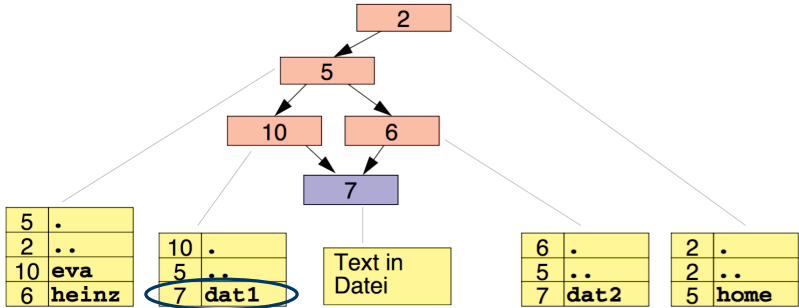
01 struct dirent *ent;
02 while(errno = 0, (ent = readdir(dir)) != NULL) {
03     //...
04 }
05
06 if(errno != 0) {
07     perror("readdir"); exit(EXIT_FAILURE);
08 }

```



```

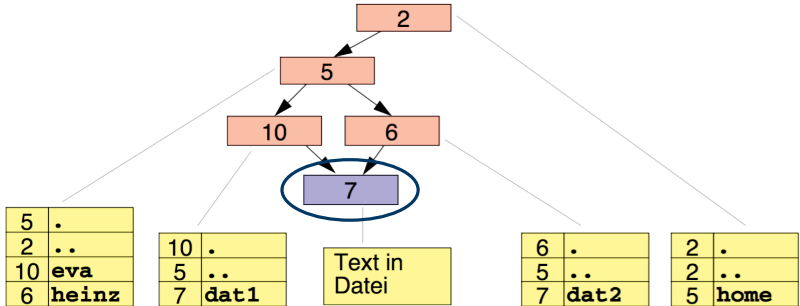
01 struct dirent *ent;
02 while(errno = 0, (ent = readdir(dir)) != NULL) {
03     //...
04 }
05
06 if(errno != 0) {
07     perror("readdir"); exit(EXIT_FAILURE);
08 }
    
```



```

01 struct dirent *ent;
02 while(errno = 0, (ent = readdir(dir)) != NULL) {
03     //...
04 }
05
06 if(errno != 0) {
07     perror("readdir"); exit(EXIT_FAILURE);
08 }

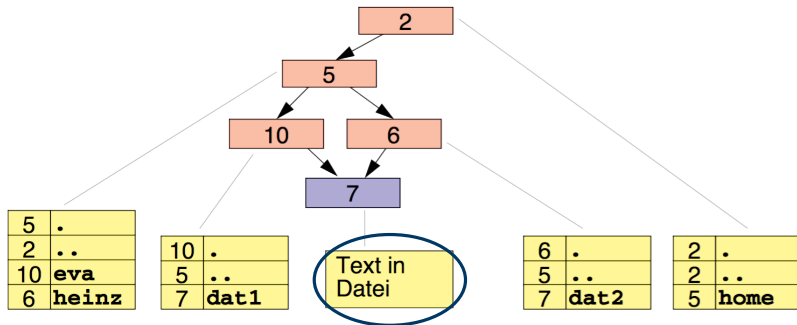
```



```

01 char path[len];
02 strcpy(path, "/home/eva/");
03 strcat(path, ent->d_name); // d_name = "dat1"
04
05 struct stat buf;
06 if(lstat(path, &buf) == -1) {
07     perror("lstat"); exit(EXIT_FAILURE);
08 }

```



```

01 FILE *file = fopen(path, "r");
02 if(file == NULL) {
03     perror("fopen");
04     exit(EXIT_FAILURE);
05 }
    
```