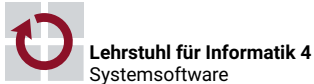


# Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2024

## Übung 9

Maxim Ritter von Onciul  
Arne Vogel

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



Friedrich-Alexander-Universität  
Technische Fakultät

## Dateien & Dateikanäle

## Vorstellung Aufgabe 5

### Dateikanäle



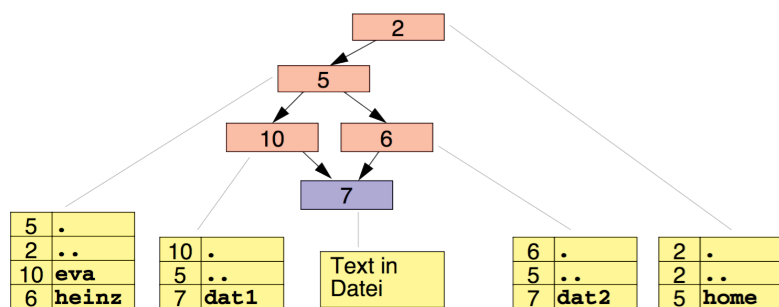
- Ein- und Ausgaben erfolgen über gepufferte Dateikanäle
- `FILE *fopen(const char *path, const char *mode);`
  - Öffnet eine Datei zum Lesen oder Schreiben (je nach mode)
  - Liefert einen Zeiger auf den erzeugten Dateikanal
  - `r` Lesen
  - `r+` Lesen & Schreiben
  - `w` Schreiben; Datei wird ggf. 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

2



**inode:** Enthält Dateiattribute & Verweise auf Datenblöcke

**Datei:** Block mit beliebigen Daten

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

- `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

3

4



```

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)

5

- 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);

```

6

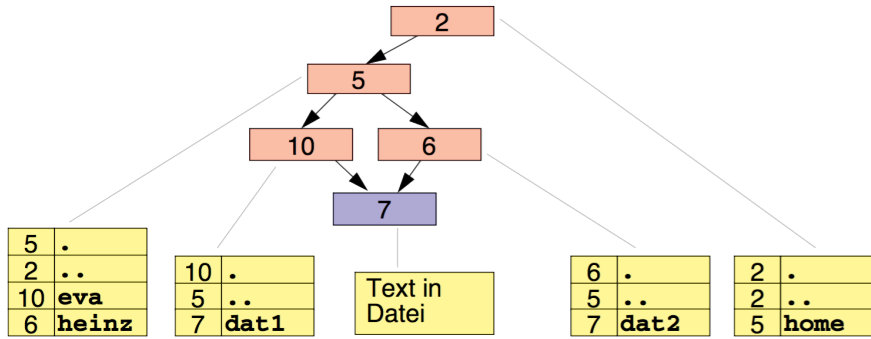


- readdir(3) liefert **nur Name und inode-Nummer** eines Verzeichniseintrags
- Weitere Attribute stehen im **inode**
- `int stat(const char *path, struct stat *buf);`
  - Abfragen der Attribute eines Eintrags (folgt symlinks)
- `int lstat(const char *path, struct stat *buf);`
  - Abfragen der Attribute eines Eintrags (folgt symlinks nicht)

7

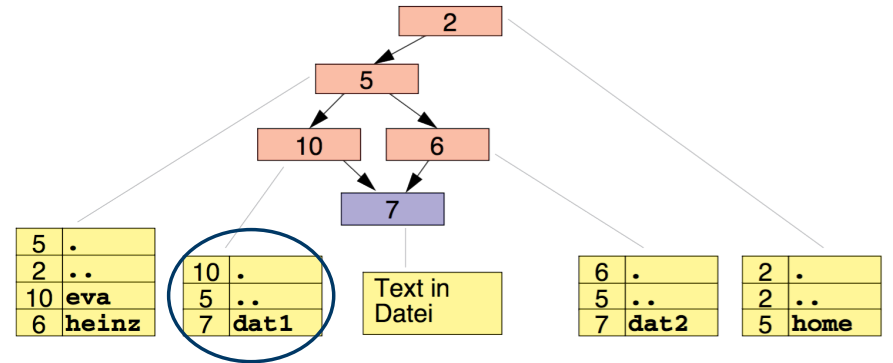
- Inhalte des inode sind u.a.:
  - Geräte- und inode-Nummer
  - Eigentümer und Gruppenzugehörigkeit
  - Dateityp und -rechte
  - Dateigröße
  - Zeitstempel (letzte(r) Veränderung, Zugriff, ...)
  - ...
- Der Dateityp ist im Feld `st_mode` codiert
  - Reguläre Datei, Ordner, symbolischer Verweis (*symbolic 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?

8



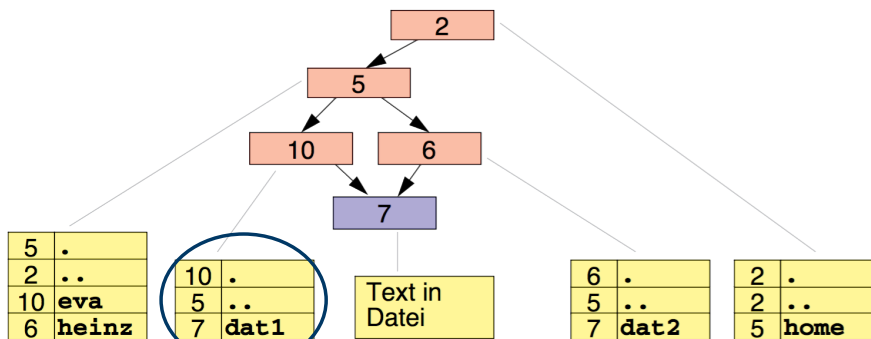
```

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



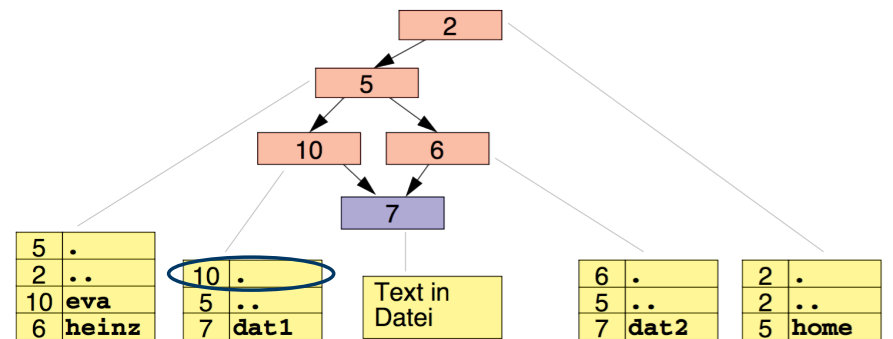
```

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



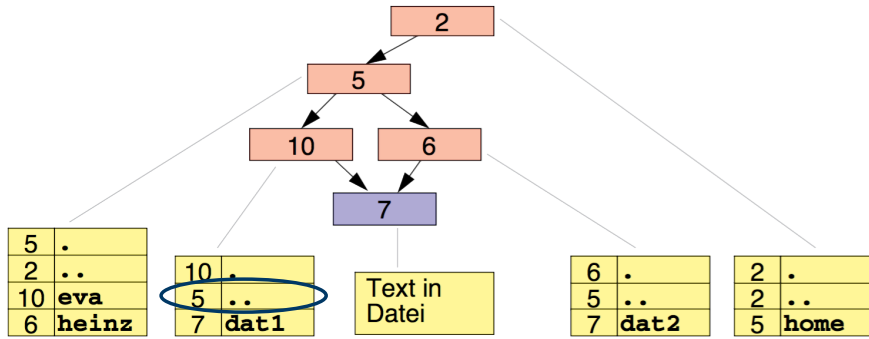
```

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



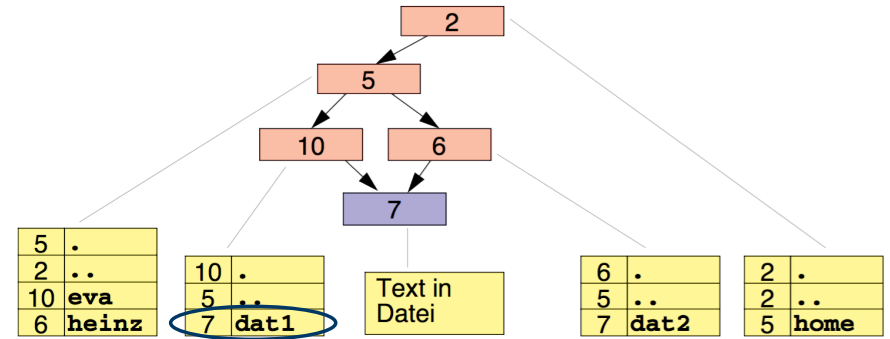
```

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



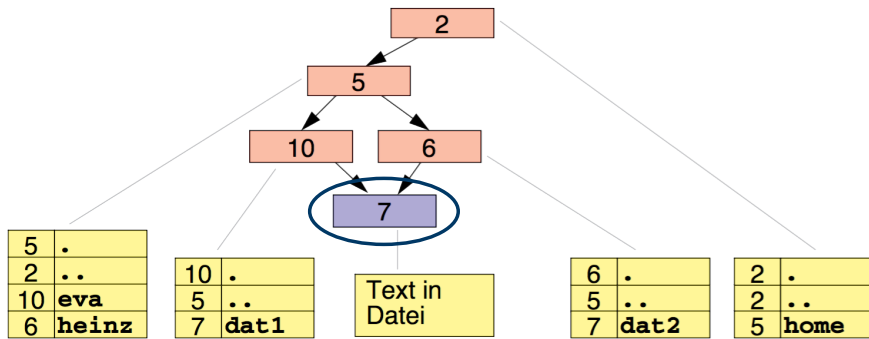
```

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



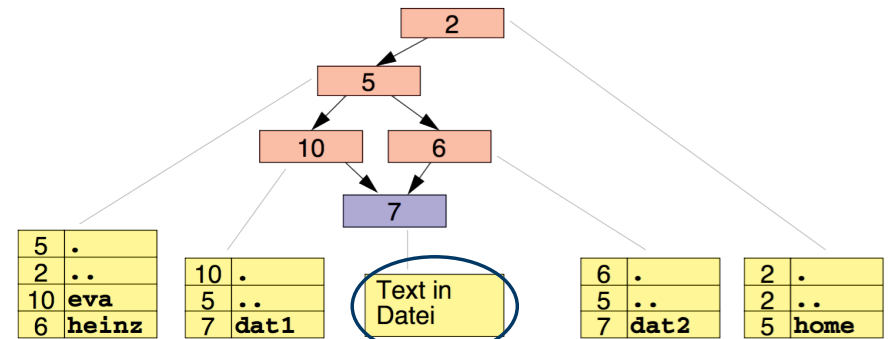
```

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



```

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 }
    
```



Minimale Implementierung von cat:

```
01 FILE *f = fopen(path, "r");
02 if(f == NULL) die("fopen");
03
04 char buf[1024];
05 while(fgets(buf, 1024, f) != NULL) {
06     printf("%s", buf);
07 }
08
09 if(ferror(f) != 0) die("fgets");
10 if(fclose(f) != 0) die("fclose");
```

```
01 $> tail -n 1 num.dat
02 499999
03 $> ./cat num.dat && echo "Success" || echo "Failed"
04 1
05 2
06 [...]
07 499999
08 Success
```

10



Minimale Implementierung von cat:

```
01 FILE *f = fopen(path, "r");
02 if(f == NULL) die("fopen");
03
04 char buf[1024];
05 while(fgets(buf, 1024, f) != NULL) {
06     printf("%s", buf);
07 }
08
09 if(ferror(f) != 0) die("fgets");
10 if(fclose(f) != 0) die("fclose");
```

```
01 $> ./cat num.dat > dir/file && echo "Success" || echo "Failed"
02 Success
03 $> tail -n 1 dir/file
04 35984
```

- Warum wird nicht die ganze Datei geschrieben?
- Warum wird kein Fehler ausgegeben?

10



```
01 $> ls -lh num.dat
02 -rw-rw-r-- user group 3,3M Jan 01 00:00 num.dat
03
04 $> ls -lh dir/file
05 -rw-rw-r-- user group 200K Jan 01 00:00 tmp/file
06
07 $> df dir/
08 Filesystem      Size  Used Avail Use% Mounted on
09 tmpfs           200K  200K   0 100% /home/user/dir
```

- stdout kann in eine Datei umgeleitet werden
- Das Schreiben in eine Datei kann fehlschlagen
  - Kein Speicherplatz mehr
  - Fehlende Schreibberechtigung
  - Festplatte kaputt
- Fehlerbehandlung für *wichtige* Ausgaben
  - Was ist wichtig?
  - Fehlerbehandlung für `printf(3)` schwierig
- Für den Übungsbetrieb: Keine Fehlerbehandlung für `printf(3)`

11



- make: Build-Management-Tool
- Baut automatisiert ein Programm aus den Quelldateien
- Baut nur die Teile des Programms neu, die geändert wurden

```
01 CFLAGS = -pedantic -Wall -Werror -O3 -std=c11 -D_XOPEN_SOURCE=700
02
03 trac.o: trac.c
04     gcc $(CFLAGS) -c -o trac.o trac.c
05
06 trac: trac.o
07     gcc $(CFLAGS) -o trac trac.o
```

- Objektdatei `trac.o` wird aus Quelldatei `trac.c` gebaut (Compiler)
- Binary `trac` wird aus Objektdatei `trac.o` gebaut (Linker)

12



The screenshot shows the SPiC-IDE interface. On the left, a project tree lists folders 'aufgabe1' through 'aufgabe6'. The main window displays a 'Makefile' with two lines of code:

```
1 CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3
2
```

Below the Makefile, the 'Atom Shell Commands' panel shows the command 'make -B trac' and its output:

```
make -B trac
cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3  trac.c -o trac
[finished in 0.14 seconds]
```

A terminal window at the bottom shows a sequence of commands and their outputs:

```
jj52coty@fau10sr0:/proj/i4spic/jy52cotys cd aufgabe6
jj52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ ls
Makefile  trac.c
jj52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ make trac
cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3  trac.c -o trac
jj52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ ls
Makefile  trac  trac.c
jj52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$
```

- SPiC-IDE erkennt Makefiles (Make Button)
  - ⇒ alternativ: `make <binary>`
- make hat eingebaute Regeln (ausreichend für SPiC)
  - ⇒ nur Angabe der Compilerflags (CFLAGS) nötig

13

## Aufgabe: printdir

## Aufgabe: printdir



- Iteration über alle via Parameter übergebene Verzeichnisse
- Ausgabe aller darin enthaltenen Einträge mit Größe und Name
- Anzeige der Anzahl von regulären Dateien und deren Gesamtgröße (pro Verzeichnis)
- Relevante Funktionen:
  - `opendir(3)`
  - `readdir(3)`
  - `stat(2)`
  - Stringfunktionen
- Fehlerbehandlung:
  - Aussagekräftige Fehlermeldungen
  - Jede falsche Benutzereingabe abfangen
  - ⇒ Den (böartigen) DAU annehmen ☺

## Hands-on: sgrep

ScreenCast: <https://www.video.uni-erlangen.de/clip/id/19103>

14



```

01 # Usage: ./sgrep <text> <files...>
02 $ ./sgrep "SPiC" klausur.tex aufgabe.tex
03 Klausur im Fach SPiC
04 SPiC Aufgabe
05 SPiC ist cool

```

- Einfache Variante des Kommandozeilentools `grep(1)`
- Durchsucht mehrere Dateien nach einer Zeichenkette
- Ablauf:
  - Dateien zeilenweise einlesen
  - Zeile nach Zeichenkette durchsuchen
  - Zeile ggf. auf `stdout` ausgeben
- Sinnvolle Fehlerbehandlung beachten
  - Fehlende Dateien melden und überspringen
  - Fehlermeldungen auf `stderr` ausgeben

16



- Hilfreiche Funktionen:
  - `fopen(3)` ⇒ Öffnen einer Datei
  - `fgets(3)` ⇒ Einlesen einer Zeile
  - `fputs(3)` ⇒ Ausgeben einer Zeile
  - `fclose(3)` ⇒ Schließen einer Datei
  - `strstr(3)` ⇒ Suche eines Teilstrings

```

01 char *strstr(const char *haystack, const char *needle);

```

```

01 # Usage: ./sgrep [-i] <text> <files...>
02 $ ./sgrep -i "spic" klausur.tex aufgabe.tex
03 klausur.tex:13: Klausur im Fach SPiC
04 aufgabe.tex:32: SPiC Aufgabe
05 aufgabe.tex:56: SPiC ist cool

```

- Erweiterung
  - `strstr(3)` selbst implementieren
  - Ausgabe von Dateinamen/Zeilennummer vor jeder Zeile
  - Ignorieren der Groß-/Kleinschreibung mit Option `-i`

17