Technische Universität Berlin

Dateien in C

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 25/26

Open Distributed Sytems
ODS

Rückblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort
VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren
VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
VL 6 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
VL 7 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
VL 8 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen
VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort
VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
VL 13 „Q & A“: Offene Vorlesung/Wiederholung

Dateisysteme

Dateisysteme

- Eine Datei ist eine Sammlung logischer Dateneinheiten
- Dateisysteme speichern z.B. Daten, Code, Programme, etc. dauerhaft in Dateien

- Dateisysteme ermöglichen
  - Abstraktion von Hintergrundspeichern,
    z.B. Platten, CD-ROM, USB, Bandlaufwerke, …
  - Einheitliche Schnittstelle

Dateiattribute

- Name: Symbolischer Name, vom Benutzer lesbar und interpretierbar
- Größe: Länge der Datei (Bytes, Blocks, ...)
- Zeitstempel: Zeitpunkt der Erstellung, letzte Modifikation, ...
- Rechte: Zugriffsrechte
- Eigentümer: Identifikation
- …

Beispiel:

```
manfred% ls -l .
…
drwxr-xr-x 10 manfred user   2048 2021-08-21 12:41 progintro
-rw-r--r--  1 manfred user   1047 2021-09-16 15:56 hello.c
…
```

Operationen auf Dateien

- Erzeugen (create)
- Schreiben (write)
- Lesen (read)
- Löschen (delete)
- Öffnen/Schließen einer Datei (open/close)
- …

Verzeichnisse

- Ein Verzeichnis ist eine Sammlung von Dateien und Verzeichnissen
- Verzeichnisattribute: Ähnlich wie Dateiattribute:
  - Name, Größe, Datum des letzten Updates, Eigentümer, Rechte
  - …

Beispiel:

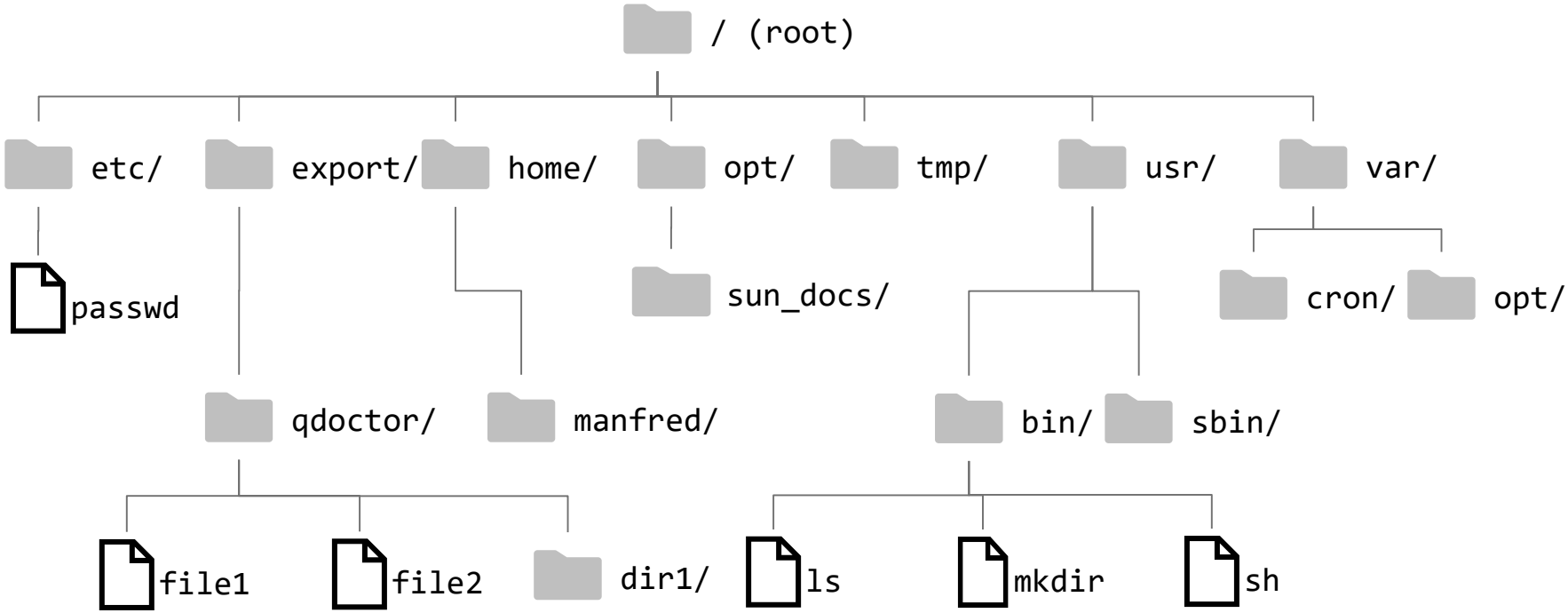
```
manfred% ls -l .
…
drwxr-xr-x 10 manfred user   2048 2021-08-21 12:41 progintro
-rw-r--r--  1 manfred user   1047 2021-09-16 15:56 hello.c
```

Operationen auf Verzeichnissen

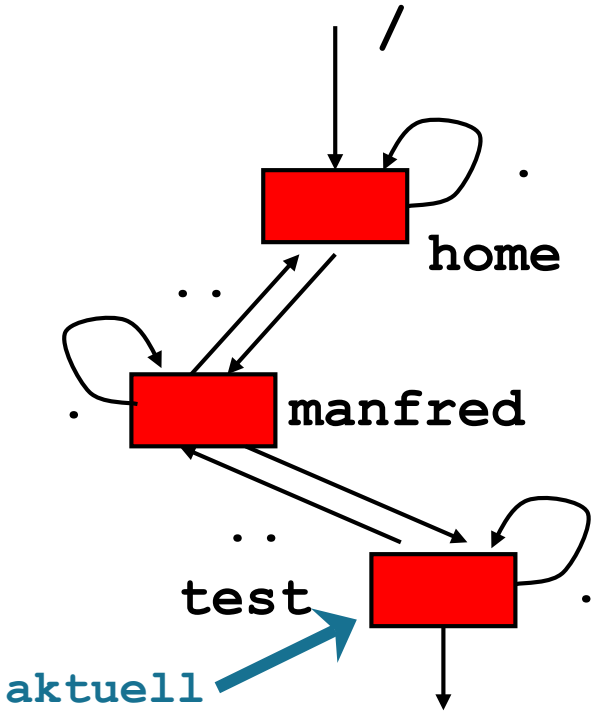
- Suchen einer Datei bzw. Verzeichnisses
- Erzeugen einer Datei bzw. Verzeichnisses
- Löschen einer Datei bzw. Verzeichnisses
- Umbenennen von Dateien bzw. Verzeichnissen
- Auslesen der Verzeichnis-Einträge
- Durchlaufen des Dateisystems

- Wichtige Unix Befehle:
  - ls, mv, cp, mkdir, rm, rmdir, find
- Details:
  - man <Befehl>

Verzeichnisse haben Baumstruktur



Unix Pfadnamen



- Benannt sind die Verbindungen zwischen Dateien und Verzeichnissen
- Rückverweise: “..“
- Selbstverweise: “.“
- Verschiedene Pfade für dieselbe Datei bzw. dasselbe Verzeichnis möglich

- Aktuelles Verzeichnis:

```
/home/manfred/test/
/home/manfred/test/.
/home/manfred/test/../test
```

Technische Universität Berlin

Arbeiten mit Dateien in C

Open Distributed Sytems
ODS

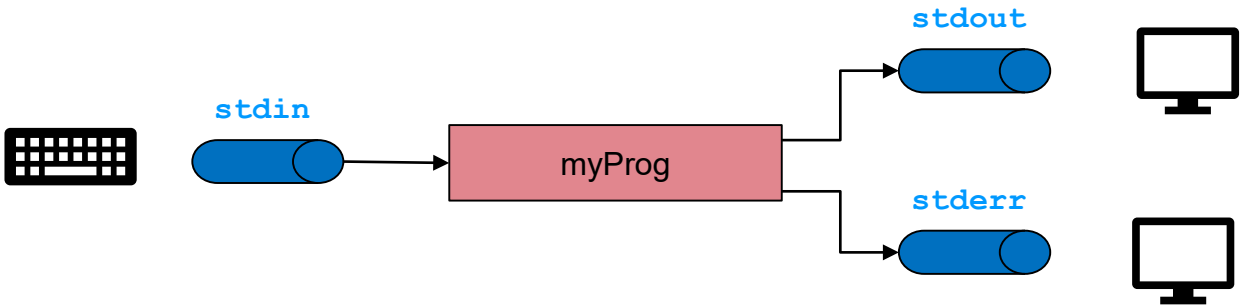
Dateien – Formatierte Ausgabe: fprintf

Aufruf: `fprintf(FILE *stream, fmt, args)`

- `fprintf()` wie printf jedoch mit Dateien / Streams, d.h. konvertiert und gibt die Parameter `args` unter Kontrolle des Formatstrings `fmt` auf `stream` aus

C-Streams

- Jedes laufende C-Programm (= Prozess) hat voreingestellt drei Kanäle für Ein-/Ausgabe:
  - `stdin` Standardeingabe
    - meist Tastatur
  - `stdout` Standardausgabe
    - meist Bildschirm
  - `stderr` Standardfehlerausgabe
    - meist Bildschirm



Dateien – Formatierte Ausgabe: fprintf

Technische Universität Berlin

Aufruf: `fprintf(FILE *stream, fmt, args)`

- `fprintf()` wie printf jedoch mit Dateien / Streams, d.h. konvertiert und gibt die Parameter `args` unter Kontrolle des Formatstrings `fmt` auf `stream` aus
- Beispiel C Streams: `stdout, stdin`

Beispiele: // fprintf(stdout, ...) entspricht printf

```
fprintf(stdout, "Hello world\n");
fprintf(stdout, "Wert von i: %d\n", i);
fprintf(stdout, "a(%d)+b(%d) ist: %d\n",
                        a, b, a+b);
```

Dateien – Formatierte Eingabe: fscanf

Technische Universität Berlin

Aufruf: `fscanf(FILE *stream, fmt, args)`

- `fscanf()` liest von `stream` und versucht, die Eingabe unter Kontrolle des Formatstrings `fmt` auf die Parameter `args` abzubilden

Beispiele: // fscanf(stdin, ...) entspricht scanf

- `int a, b; fscanf(stdin, "%d %d", &a, &b);`
- `float x; fscanf(stdin, "%f", &x);`
- `char a; fscanf(stdin, "%c", &a);`

Öffnen von Dateien: fopen

Technische Universität Berlin

Aufruf: `FILE *stream fopen(path, mode)`

- `fopen()` öffnet die Datei `path` im Modus `mode`

Beispiele:

```
FILE *file_pointer_in, *file_pointer_out;
// öffnet Datei „datei“ zum Lesen
file_pointer_in = fopen(“./datei“, “r“);

// öffnet „datei“ zum Schreiben
file_pointer_out = fopen(“./datei“, “w“);

// öffnet „datei“ zum Schreiben mittels Anhängen am Dateiende
file_pointer_out = fopen(“./datei“, “a“);
```

Schließen von Dateien: fclose

Aufruf: `fclose(FILE *stream)`

- `fclose()` schließt den Stream `stream`
- Fehlerfreies Beenden der Operationen
- Gibt Ressourcen frei, u.a. im Betriebssystem

Beispiel:

```
FILE *file_pointer_in
//schließt den Stream file_pointer_in
fclose(file_pointer_in);
```

Lesen von Datei
Ausgabe auf stdout

```
// Konvention: „fp“ kurz für „file_pointer“
FILE *fp = fopen(“datei.txt“, “r“);
int a, b;
while (fscanf(fp, “%d %d\n“, &a, &b) != EOF) {
   printf(“%d %d\n“, a, b);
}
fclose(fp);
```

- `fscanf()` gibt die Anzahl der zugewiesenen Variablen zurück oder EOF (End of File), wenn aus der Datei nicht gelesen werden kann
- Hinweis: Fehlerbehandlung fehlt, ist aber notwendig!!!

Fehlerbehandlung mit perror

```
#include <errno.h>
FILE *fp = fopen("datei.txt", „r");
int a, b;
if (fp == NULL ) {
    perror("Fehler beim öffnen der Datei");
    return 1; }
while (fscanf(fp, "%d %d\n", &a, &b) != EOF) {
    printf("%d %d\n", a, b); }
fclose(fp);
```

- `void perror(msg)`
  - Gibt die letzte Systemfehlermeldung auf `stderr` aus

Lesen von Datei
Ausgabe in Datei

```
FILE *fpin = fopen("datei_in", „r");
FILE *fpout = fopen("datei_out", „a");
int a, b;
while (fscanf(fpin, "%d %d\n", &a, &b) != EOF) {
fprintf(fpout, "%d + %d = %d\n", a,b,a+b);
}
fclose(fpin); fclose(fpout);
```

- Hinweis: Fehlerbehandlung fehlt, ist aber notwendig!!!

Problem: Fehlerhafte Eingabedaten

- Eingabedaten entsprechen nicht unbedingt den Erwartungen
- Z.B. anstelle einer Zahl ein String

- Deshalb immer überprüfen, ob das Lesen erfolgreich war!
- Kann bei scanf, fscanf problematisch sein.

Lösung:

- fgets zum Lesen einer Zeile
- gefolgt von sscanf zum Konvertieren des Strings in Token (int, float, etc.)

Dateien – Formatierte Eingabe: fgets & sscanf

Technische Universität Berlin

Aufruf:

```
char *fgets(char *buf, int n, FILE *stream)
```

`fgets()` liest bis zum ersten Newline „\n“ von `stream`

- dabei aber maximal n-1 Zeichen, und fügt „\0“ am Ende hinzu

  oder

- bis EOF
- je nachdem welche Bedingung als erste zutrifft

Dateien – Formatierte Eingabe: fgets & sscanf

```
char buf[200];
int a, b;
char *h = fgets(buf, 200, stdin);
if (h==NULL) {
        printf(„error in fgets“);
        exit(1);
}
int ret = sscanf(buf, „%d %d“, &a, &b);
if (ret == 2) {
   printf(„read 2 integers: %d %d\n“, a, b);
} else {
        fprintf(stderr, „error reading 2 integers\n“);
}
```

Dateien – Formatierte Eingabe: fgets & sscanf

Technische Universität Berlin

Aufruf:

```
char *fgets(char *buf, int n, FILE *stream)
```

- `fgets()` liest bis zum ersten Newline „\n“ von `stream` dabei aber maximal n-1 Zeichen, und fügt „\0“ hinzu oder bis EOF (je nachdem welche Bedingung als erste zutrifft)

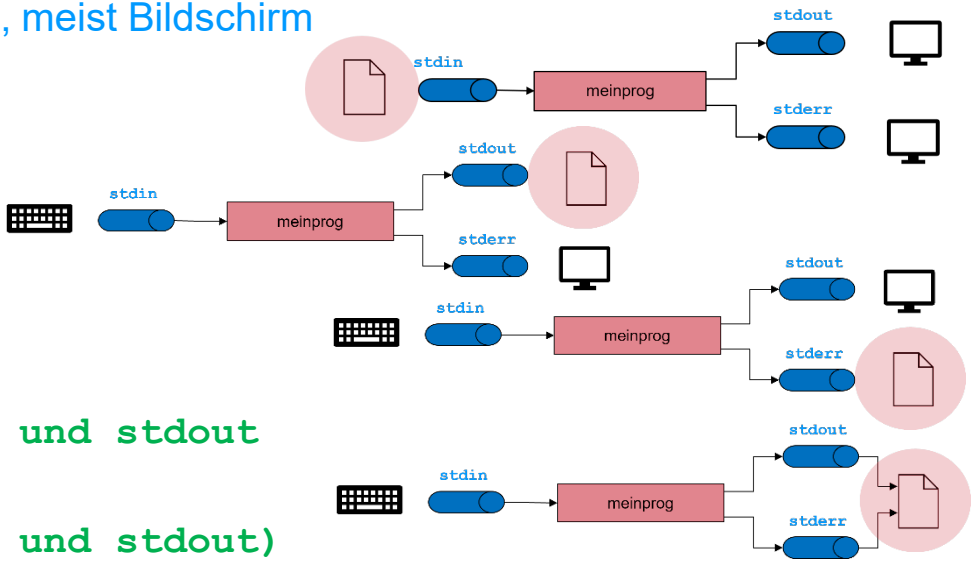
- Finger weg von `gets(char *buf)` !!!
  `gets` liest unabhängig von der Größe des Buffers bis zum ersten Newline „\n“ von `stream`. Gefahr eines “Buffer Overflow“ !!!

C-Streams

- Jedes laufende C-Programm (= Prozess) hat voreingestellt drei Kanäle für Ein-/Ausgabe:
  - `stdin` Standardeingabe, meist Tastatur
  - `stdout` Standardausgabe, meist Bildschirm
  - `stderr` Standardfehlerausgabe, meist Bildschirm

- Die Standardkanäle sind umlenkbar:

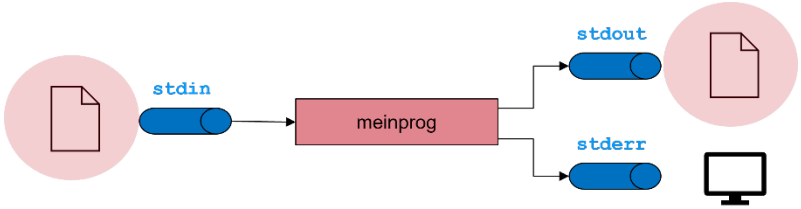
```
$ ./meinprog < InFile

$ ./meinprog > OutFile

$ ./meinprog 2> OutFile
  // bash Umleiten von stderr

$ ./meinprog &> OutFile
  // bash Umleiten von stderr und stdout

($ ./meinprog >& OutFile
  // tcsh Umleiten von stderr und stdout)
```



C-Streams

- Die Standardkanäle sind kombinierbar:

```
$ ./meinprog < InFile > OutFile
```

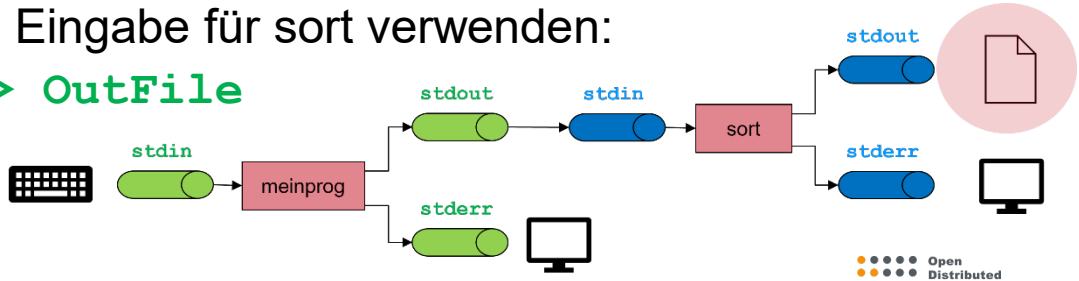


- Unix Tools:

```
cat, sort, more, less, grep, wc, tail, cut,
sed, split, uniq
```

- Ausgabe von ./meinprog1 als Eingabe für sort verwenden:

```
$ ./meinprog1 | sort > OutFile
```



Ausblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort
VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren
VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
VL 6 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
VL 7 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
VL 8 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen
VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort
VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
VL 13 „Q & A“: Offene Vorlesung/Wiederholung