

Übungen Betriebssysteme (BS)

U5 – Dateioperationen

 CORONA-EDITION

<https://sys.cs.tu-dortmund.de/DE/Teaching/SS2021/BS/>

Peter Ulbrich

peter.ulbrich@tu-dortmund.de

<https://sys.cs.tu-dortmund.de/EN/People/ulbrich/>



technische universität
dortmund

Agenda

- Besprechung Aufgabe 4 – Speicherverwaltung
- Dateieigenschaften und fstat(2)
- Arbeiten mit Dateien
- Aufgabe 5 – Dateioperationen

Besprechung Aufgabe 4

→ Foliensatz Besprechung

Arbeiten mit Dateien

- Typische Operationen: Öffnen, Lesen/Schreiben, Schließen
- Unter Linux mehrere Möglichkeiten
 - Syscall-Wrapper der C-Bibliothek („low-level“), z.B. **open(2)**
- Abstrakte Stream-Schnittstelle der C-Bibliothek („high-level“), z.B. **fopen(3)**

```
int open(const char *path, int flags)
    syscall(__NR_open, path, flags)
```

```
FILE* fopen(const char *path, const char *mode)
```

Abstraktion von Filedeskriptor → Stream (FILE*)

```
open(path, flags)
```

„low-level“-Dateioperationen

- **open(2):** Datei öffnen
 - `int open(const char *path, int flags)`
 - flags: Nur lesen (`O_RDONLY`), ggf. erstellen (`O_CREAT`), ...
 - Gibt Dateideskriptor zurück, der für andere Funktionen verwendet wird
- **read(2):** Aus Datei lesen
 - `size_t read(int fd, void *buf, size_t count)`
- **lseek(2):** Schreib-/Leseposition verändern
 - `off_t lseek(int fd, off_t offset, int whence)`
- **close(2):** Offene Datei schließen
 - `int close(int fd)`
- Standardisiert u.a. in POSIX.1-2001

„low-level“-Beispiel

```
#define READ_SIZE 64

int main(void) {
    int fd = 0;
    char buf[READ_SIZE+1] = {0};

    if ((fd = open("somefile", O_RDONLY)) == -1) { /* ... Fehler ... */ }
    if (read(fd, buf, READ_SIZE) < READ_SIZE) {
        /* Die Datei ist kleiner als 64 Byte */
    }

    buf[READ_SIZE] = 0;
    printf("Die ersten 64 Zeichen von somefile sind %s\n", buf);

    if (lseek(fd, 1000, SEEK_SET) != 1000) {
        /* Fehler */
    }

    if (read(fd, buf, 1) < 1) { /* ... Fehler ... */ }
    printf("An Position 1000 steht das Zeichen %c\n", buf[0]);

    if (close(fd) == -1) { /* ... Fehler ... */ }
}
```

C-Streams

- Abstrakter, plattformübergreifender Kommunikationskanal zu
 - einem Gerät / einer Datei
 - einem Prozess
 - ...
- Interne Pufferung
 - Blockweises Lesen → Höhere Verarbeitungsgeschwindigkeit
- Abstraktion vom eigentlichen Datenstrom
 - Unabhängiges Lesen und Schreiben an verschiedenen Positionen
- Stream-Repräsentation durch struct FILE
 - Zeiger auf Lese-/Schreibpuffer, aktuelle Positionen, ...
- Definierte Standardstreams: stdin, stdout, stderr

„high-level“-Dateioperationen

■ **fopen(3)**: Datei öffnen

- `FILE *fopen(const char *path, const char *mode)`
- mode: Z.B. „r“ (lesen), „r+“ (schreiben+lesen), „a“ (anhängen), ...
- Gibt Zeiger auf neu erstellte FILE-Struktur zurück

■ **fread(3)**: Aus Datei lesen

- `size_t fread(void *buf, size_t itemsize, size_t count, FILE *stream)`

■ **fseek(3)**: Schreib-/Leseposition verändern

- `int fseek(FILE *stream, long offset, int whence)`

■ **fclose(3)**: Offene Datei schließen

- `int fclose(FILE *stream)`

■ Ebenfalls u.a. in POSIX.1-2001 standardisiert

■ Teil des C-Standards → plattformunabhängig

„high-level“-Beispiel

```
#define READ_SIZE 64

int main(void) {
    FILE *somefile;
    char buf[READ_SIZE+1] = {0};

    if ((somefile = fopen("somefile", "r")) == NULL) { /* Fehler */ }
    if (fread(buf, sizeof(*buf), READ_SIZE, somefile) < READ_SIZE) {
        /* Die Datei ist kleiner als 64 Zeichen */
    }

    buf[READ_SIZE] = 0;
    printf("Die ersten 64 Zeichen von somefile sind %s\n", buf);

    if (fseek(somefile, 1000, SEEK_SET) == -1) {
        /* Fehler */
    }

    if (fread(buf, sizeof(*buf), 1, somefile) < 1) { /* Fehler */ }
    printf("An Position 1000 steht das Zeichen %c\n", buf[0]);

    if (fclose(somefile) == -1) { /* Fehler */ }
}
```

„high-level“ Operationen #2

■ ftell(3)

- `long ftell(FILE *stream)`
- Gibt den Datei-Positionszeiger für den Stream stream aus

■ fscanf(3)

- `int fscanf(FILE *stream, const char *format, ...)`
- Liest den Inhalt einer Datei aus und filtert über den Formatstring Variablen heraus
- Funktioniert prinzipiell genau wie `scanf(3)`, mit denselben Platzhaltern im Formatstring

■ fprintf(3)

- `int fprintf(FILE *stream, const char *format, ...)`
- Schreibt einen formatierten String in eine Datei
- Funktioniert prinzipiell genau wie `printf(3)`, mit denselben Platzhaltern im Formatstring

Dateigröße bestimmen

```
int main(void) {
    FILE *somefile;
    long filesize = 0;
    if ((somefile = fopen("somefile", "r")) == NULL) { /* Fehler */ }

    /* ans Ende springen */
    if (fseek(somefile, 0L, SEEK_END)) { /* Fehler */ }
    /* aktuelle Position innerhalb Datei == Dateigröße */
    filesize = ftell(somefile);
    if (filesize == -1) { /* Fehler */ }

    printf("Die Datei hat %d bytes", filesize);

    if (fclose(somefile) == -1) { /* Fehler */ }
}
```

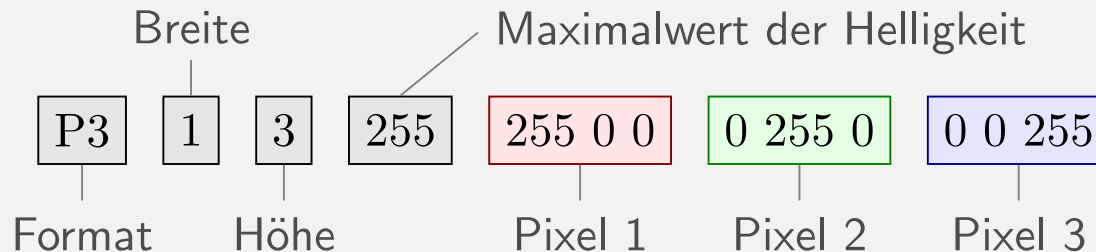
Portable Anymaps

- Sehr einfaches, von Menschen lesbares Dateiformat für Bilddateien
- 3 Unterformate:
 - Portable Bitmap (PBM) für schwarzweiße Bilder
 - Portable Graymap (PGM) für Bilder mit Graustufen
 - **Portable Pixmap (PPM) für Bilder mit Farben**
- Für das Übungsblatt werden wir uns nur mit Portable Pixmap beschäftigen.

Portable Anymaps – Beispiel

- Dateianfang mit Header:
 - Eigenschaften des Bildes (Format, Breite, Höhe, Maximalwert)
 - Leerraum (ein oder mehrere Leerzeichen, Tabulatorzeichen, *Carriage Return*, *Line Feed*) zwischen Parametern

beispiel1.ppm



Portable Anymaps

- Beispiel-PPM-Dateien in der Vorlage
- PPM anzeigen / editieren: z.B. mit GIMP
 - Debian Linux: `sudo apt install gimp`
 - Windows, Mac: <https://www.gimp.org/>

Exkurs: fstat(2)

- Linux / Unix bildet (fast) alles auf Dateien ab
 - Geräte, Sockets, ...
- Dateien haben verschiedenste Eigenschaften, u.a.
 - Dateigröße
 - Berechtigungen
 - Anlege- und Änderungszeitpunkt
 - Typ: Verzeichnis / Datei / Zeichen-/Blockbasiertes Gerät / ...
- Abfrage dieser Eigenschaften mit **fstat(2)**

```
int fstat(int fd, struct stat *fileinfo);
```

- fd: Dateideskriptor der Datei
- struct stat: Datenstruktur für Dateieigenschaften

Exkurs: struct stat

- `st_mode` → Rechte und Dateityp. Abfrage mit fertigen Makros:
 - `S_ISREG(st_mode)` → reguläre Datei
 - `S_ISDIR(st_mode)` → Verzeichnis
- `st_size`: Dateigröße in Bytes

```
int main(int argc, char **argv) {
    struct stat fileinfo;
    int fd;
    for (int i = 1; i < argc; i++) {
        if ((fd = open(argv[i], O_RDONLY)) == -1) { /* Fehlerbehandlung */ }
        if (fstat(fd, &fileinfo) == -1) { /* Fehlerbehandlung */ }

        if (S_ISREG(fileinfo.st_mode)) {
            printf("%s ist eine %d Byte große Datei\n",
                argv[i], fileinfo.st_size);
        }
        else if (S_ISDIR(fileinfo.st_mode)) { /* Verzeichnis */ }
        else { /* Sonstwas (z.B. FIFO/Socket/Link/Gerät) */ }
        if (close(fd) == -1) { /* Fehlerbehandlung */ }
    }
}
```


Exkurs: fileno(3)

- `fileno(3)`
 - `int fileno(FILE *stream)`
 - Nimmt einen Filestream als Argument und gibt den dazugehörigen File-Deskriptor zurück