Übungen Betriebssysteme (BS)

U5 – Dateioperationen

https://moodle.tu-dortmund.de/course/view.php?id=34604

Peter Ulbrich

peter.ulbrich@tu-dortmund.de

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

technische universität
dortmund

Agenda

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

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

  - Abstrakte Stream-Schnittstelle der C-Bibliothek („high-level“), z.B. **fopen(3)**

```
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(int argc, char *argv[]) {
    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(int argc, char *argv[]) {
    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
    ```

fstat(2) - Beschreibung

- 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

fstat(2) - struct stat

- st_mode → Rechte/Dateityp. Abfrage mit Makros:
  - S_ISREG(st_mode) → reguläre Datei
  - S_ISDIR(st_mode) → Verzeichnis

Diese existieren bereits

- 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 */ }
    }
}
```

fstat(2) – Verwendung bei Streams

- Zugriff auf Dateideskriptor bei C-Streams
- Übersetzung von `FILE*` zu `fd` via Bibliotheksaufruf
  - fileno(3)
  - Nimmt einen Filestream als Argument und gibt den dazugehörigen File-Deskriptor zurück

```
int fileno(FILE *stream);
```

„High-Level“-Verzeichnisoperationen

- Öffnen von Verzeichnissen mit **opendir(3)**

```
DIR *opendir(const char *path);
```

- Auslesen der Verzeichnisinhalte mit **readdir(3)**

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

  - Rückgabe = NULL → alle Verzeichniseinträge gelesen, keine weiteren vorhanden oder Fehler
  - sonst: in dirent.d_name steht der Name des Verzeichniseintrages

- Schließen von geöffneten Verzeichnissen mit **closedir(3)**

```
int closedir(DIR *dir);
```

- Benötigte Header: <sys/types.h>, <dirent.h>

Beispiel: Verzeichnis Auslesen

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

int main(int argc, char *argv[]) {
  int retval = 0, count = 0;
  DIR *dir = NULL;
  struct dirent *entry = NULL;

  /* Verzeichnis öffnen, DIR-Zeiger erstellen */
  dir = opendir("/tmp");
  if (!dir) { /* ... Fehler ... */ }

  /* jeder Aufruf liefert einen Eintrag, NULL -> Listenende */
  while ((entry = readdir(dir))) {
    printf("Eintrag %d: %s\n", ++count, entry->d_name);
  }
  printf("Anzahl: %d\n", count);

  retval = closedir(dir); /* Verzeichnis wieder schließen */
  if (retval == -1) { /* ... Fehler ... */ };

  return 0;
}
```

Dateipfade konstruieren

- Der benötigte Speicherplatz hängt von Verzeichnis- und Dateinamenlänge ab
  - „foo/bar“ vs. „irgendein/furchtbar/langer/Pfad/mit einer Datei.mp3“
- Größe mit **strlen(3)** bestimmen
- Speicher mit **malloc(3)** belegen
- Wenn nicht mehr benötigt: **free(3)** nicht vergessen!

```
int bufsize = strlen("foo") + strlen("bar") + 2;

char *buf = malloc(bufsize);
if (buf == NULL) { /* Fehler */ }

snprintf(buf, bufsize, "%s/%s", "foo", "bar");
// buf enthält jetzt „foo/bar“
// (inkl. terminierendem Nullbyte)
```

Warum 2?

Hilfreiche Stringfunktionen

- Notwendiger Header

```
#include <string.h>
```

- Kopieren eines Strings

```
char *strncpy( char *to, const char *from, size_t count );
```

- Konkatenieren zweier Strings

```
char *strncat( char *str1, const char *str2, size_t count );
```

- Länge eines Strings (ohne terminierende Null)

```
size_t strlen( char *str );
```

ID3v1-Tags

- MP3-Dateien können Metadaten enthalten
- Nach ID3v1[1)] befinden sich diese Daten in den letzten 128 Bytes der Datei

Offset	Länge	Bedeutung
0	3	Kennung „TAG“ zur Kennzeichnung eines ID3v1-Blocks
3	30	Songtitel
33	30	Künstler/Interpret
63	30	Album
93	4	Erscheinungsjahr
97	30	Beliebiger Kommentar
127	1	Genre

1) http://www.id3.org/

ID3v1-Tags im Speicher

```
00002db0  00 00 00 00 00 54 41 47  48 61 6c 6c 6f 21 00 00  |.....TAGHallo!..|
00002dc0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002dd0  00 00 00 00 00 00 45 53  53 00 00 00 00 00 00 00  |......ESS.......|
00002de0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002df0  00 00 00 00 42 53 52 76  53 20 31 20 53 61 6d 70  |....BSRvS 1 Samp|
00002e00  6c 65 72 00 00 00 00 00  00 00 00 00 00 00 00 00  |ler.............|
00002e10  00 00 32 30 30 39 4b 6f  6d 6d 65 6e 74 61 72 20  |..2009Kommentar |
00002e20  6d 69 74 20 4d 61 78 69  6d 61 6c 6c 61 65 6e 67  |mit Maximallaeng|
00002e30  65 21 00 2a 1c                                    |e!.*.|
00002e35
```

- Setzen der Lese/Schreibposition

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

- Rückgabe: -1 im Fehlerfall, sonst 0
- SEEK_SET, SEEK_END oder SEEK_CUR
- Offset: positiv oder negativ
- Als „low-level“-Operation:

```
off_t lseek(int fd, off_t offset, int whence);
```

fseek(3)

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

int main(int argc, char *argv[]) {
  FILE *file = NULL;


  /* Datei öffnen */

  if (fseek(file, -128L, SEEK_END) == -1) {
        perror("Fehler!");
        return 1;
  }

  /*Datei bearbeiten*/

  /*Datei schließen*/

  return 0;
}
```

Verarbeitung der Informationen

- Auslesen des entsprechenden Bereiches in ein `char`-Array
- Array der ID3v1-Spezifikation entsprechend durchlaufen
- Inhalte in das vorgegebene `struct` speichern
  - `char interpret[31];`
  - `char titel[31];`
  - `char album[31];`
  - `char kommentar[31];`
  - `char jahr[5];`
  - `char genre;`
- Struct zurückgeben