

Systemprogrammierung

Teil 8: POSIX

Elementare Ein-/Ausgabe, Dateizugriff

POSIX: Übersicht

POSIX (Portable Operating System Interface) ist ein von IEEE und Open Group entwickelter Standard für die Programmierschnittstelle von Betriebssystemen.

- der Standard legt **C-Systemaufrufe** und die zugehörigen **Header-Dateien** fest:
über 80 Header-Dateien mit über 1000 Funktionen und Makros
(dabei teilweise Überlappungen mit dem C-Standard)

Die meisten UNIX-Varianten und viele weitere Betriebssysteme halten sich ganz oder zumindest weitgehend an diesen Standard.

- der Standard umfasst Themen wie
Ein-/Ausgabe
Speicherverwaltung
Synchronisation paralleler Abläufe
Nachrichtenaustausch zwischen parallelen Prozessen

...

POSIX Ein-/Ausgabe: Übersicht

Wichtige Header-Dateien im Zusammenhang mit Ein-/Ausgabe:

- Ein-/Ausgabe der C-Standardbibliothek:

<stdio.h>

Umgang mit **FILE**-Streams (**fopen**, **fclose**, **fgetc**, **fputc**, ...)

- elementare Ein-/Ausgabe:

<fcntl.h> und **<unistd.h>**

Umgang mit Dateien und Datenströmen (**creat**, **open**, **read**, **write**, **close**)

kein Schreibfehler!

<sys/stat.h> und **<dirent.h>**

Umgang mit Verzeichnissen (**stat**, **mkdir**, **opendir**, **readdir**, **closedir**)

<errno.h>

Fehlerzustand und symbolische Namen für Fehlernummern (**errno**)

Ein-/Ausgabe mit der C-Standardbibliothek: <stdio.h> (1)

Bei den Ein-/Ausgabefunktionen der C Standard-Bibliothek werden die Eingabe-Quellen und Ausgabe-Ziele mit einem **FILE-Zeiger** angegeben:

- **FILE** ist ein (Alias-)Name für eine Struktur, die den Zustand einer Eingabe-Quelle bzw. eines Ausgabe-Ziels verwaltet
zum Zustand gehören Puffer, Lese-/Schreibposition, aufgetretene Fehler, ...

- vordefinierte globale Variablen für die Standard-Ein-/Ausgabe:

extern FILE *stdin;

extern FILE *stdout;

extern FILE *stderr;

Hinweis: stdin, stdout und stderr können auch Präprozessor-Makros sein

- **fopen** liefert Zeiger auf weitere FILE-Objekte:

FILE *fopen(const char *dateiname, const char *mode);

mode "r" für reinen Lesezugriff, "w" für reinen Schreibzugriff, ...

- **fclose** schließt nicht mehr benötigte Eingabe-Quellen und Ausgabe-Ziele:

int fclose(FILE *fp);

Ein-/Ausgabe mit der C-Standardbibliothek: <stdio.h> (2)

- **Ein-/Ausgabe von Einzelzeichen:**

```
int fgetc(FILE *fp);
```

liefert das nächste Zeichen (umgewandelt in int) oder EOF bei Eingabeende / Fehler

```
int fputc(int c, FILE *fp);
```

schreibt das Zeichen c und liefert c oder bei Fehler EOF

...

- **Ein-/Ausgabe von Zeichenketten:**

```
char *fgets(char *s, int n, FILE *fp);
```

liefert in s die nächsten maximal n - 1 Zeichen einer Zeile und gibt s zurück, bzw. NULL bei Eingabeende / Fehler

```
int fputs(const char *s, FILE *fp);
```

schreibt die Zeichenkette s und liefert nicht-negativen Wert bzw. bei Fehler EOF

...

Ein-/Ausgabe mit der C-Standardbibliothek: <stdio.h> (3)

- **formatierte Ein-/Ausgabe:**

```
int fscanf(FILE *fp, const char *format, ...);
```

versucht die in format genannten Werte zu lesen und liefert die Anzahl der erfolgreich gelesenen Werte oder EOF bei Eingabeende / Fehler

```
int fprintf(FILE *fp, const char *format, ...);
```

schreibt die Zeichenkette format inklusive der mit Werten gefüllten Lücken und liefert die Anzahl der insgesamt geschriebenen Bytes oder bei Fehler EOF

...

- **Ein-/Ausgabe von Binärdaten:**

```
size_t fread(void *p, size_t size, size_t n, FILE *fp);
```

liefert in p maximal n Portionen von size Byte und gibt die Anzahl der tatsächlich gelesenen Portionen zurück

```
size_t fwrite(const void *p, size_t size, size_t n, FILE *fp);
```

schreibt maximal n Portionen von size Byte aus p und gibt die Anzahl der tatsächlich geschriebenen Portionen zurück

Ein-/Ausgabe mit der C-Standardbibliothek: <stdio.h> (4)

- Fehlerbehandlung:

`int feof(FILE *fp) ;`

liefert einen von 0 verschiedenen Wert, wenn das Eingabeende erreicht wurde

`int ferror(FILE *fp) ;`

liefert einen von 0 verschiedenen Wert, wenn ein Fehler aufgetreten ist

`void perror(const char *prefix) ;`

gibt prefix gefolgt von der Fehlermeldung des aktuellen Fehlers auf stderr aus

`void clearerr(FILE *fp) ;`

Setzt den Eingabeende- und Fehlerzustand zurück

Beispiel-Programm <stdio.h>

```
#include <stdio.h> // fopen, fgetc, fclose
int main(int argc, char *argv[])
{
    for (int i = 1; i < argc; ++i)
    {
        FILE *fp = fopen(argv[i], "r");
        if (fp == NULL) ... // Fehlerbehandlung

        int n = 0; // statt int besser intmax_t (Warum?)
        while (fgetc(fp) != EOF) {
            ++n;
        }
        if (ferror(fp)) ... // Fehlerbehandlung

        printf("%s: %d Bytes\n", argv[i], n);
        fclose(fp);
    }
    return 0;
}
```

Zählt die Bytes in Dateien

POSIX: Elementare Ein-/Ausgabe (1)

Bei den elementaren Ein-/Ausgabefunktionen nach POSIX-Standard werden Eingabe-Quellen und Ausgabe-Ziele über einen **Dateideskriptor** angesprochen:

- ein **Dateideskriptor** ist eine nicht-negative ganze Zahl
bei der C-Standardbibliothek Ein-/Ausgabe in der FILE-Struktur gespeichert
- vordefinierte Dateideskriptoren für die Standard-Ein-/Ausgabe:
 - 0 Standardeingabe
 - 1 Standardausgabe
 - 2 Standardfehlerausgabe
- **open** liefert einen Dateideskriptor für eine Datei:

```
int open(const char *dateiname, int flags); // <fcntl.h>
```

liefert den kleinsten nicht belegten Dateideskriptor oder bei Fehler -1
- **close** schließt nicht mehr benötigte Eingabe-Quellen und Ausgabe-Ziele:

```
int close(int fd); // <unistd.h>
```

liefert 0 oder bei Fehler -1

POSIX: Elementare Ein-/Ausgabe (2)

- **Ein-/Ausgabe von Bytes:**

```
ssize_t read(int fd, void *p, size_t n); // <unistd.h>
```

liefert in p maximal n Byte und gibt die Anzahl der tatsächlich gelesenen Bytes zurück, 0 bei Eingabeende, -1 bei Fehler

```
ssize_t write(int fd, const void *p, size_t n); // <unistd.h>
```

schreibt maximal n Byte aus p und gibt die Anzahl der tatsächlich geschriebenen Bytes oder bei Fehler -1 zurück

```
ssize_t // <sys/types.h>
```

Aliasname für einen ganzzahligen Typ mit Vorzeichen (int oder long)
- **Fehlerbehandlung:**

```
extern int errno; // <errno.h>, errno kann auch ein Makro sein
```

POSIX-Funktionen weisen errno im Fehlerfall eine Fehlernummer ungleich 0 zu für die Fehlernummern sind symbolische Konstanten definiert (z.B: EACCES für fehlendes Zugriffsrecht auf eine Datei)

Beispiel-Programm Dateien (1)

```
#define _POSIX_C_SOURCE 1

#include <stdio.h>    // fprintf
#include <string.h>    // strerror
#include <fcntl.h>    // open, O_RDONLY, O_WRONLY, O_CREAT, O_EXCL
#include <sys/stat.h> // mode_t, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH
#include <unistd.h>    // read, write
#include <errno.h>     // errno

int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        fprintf(stderr, "Aufruf: %s Quelle Ziel\n", argv[0]);
        return 1;
    }

    int in = open(argv[1], O_RDONLY); // Dateideskriptor Quelle
    if (in == -1) ... // Fehlerbehandlung

    ...
}
```

Kopiert eine Datei

Beispiel-Programm Dateien (2)

```
...

const mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH; // Zugriffsrechte
int out = open(argv[2], O_WRONLY | O_CREAT | O_EXCL, mode);
if (out == -1) ... // Fehlerbehandlung

int n;
unsigned char byte;
while ((n = read(in, &byte, 1)) > 0)
{
    int m = write(out, &byte, 1);
    if (m != 1) ... // Fehlerbehandlung
}

if (n < 0) ... // Fehlerbehandlung

close(out);
close(in);
return 0;
}
```

POSIX Ein-/Ausgabe: Verzeichnisse

Nach POSIX-Standard werden Verzeichnisse über **DIR-Zeiger** angesprochen:

- **opendir** liefert einen DIR-Zeiger für ein Verzeichnis:
`DIR *opendir(const char *verzeichnisname) ; // <dirent.h>`
liefert NULL bei Fehler
- **closedir** beendet den Verzeichniszugriff:
`int closedir(DIR *dirp) ; // <dirent.h>`
liefert 0 oder bei Fehler -1
- **readdir** liefert einen Zeiger auf den nächsten ungelesenen Verzeichniseintrag:
`struct dirent *readdir(DIR *dirp) ; // <dirent.h>`
der Verzeichniseintrag enthält unter d_name einen Dateinamen
liefert NULL bei Verzeichniseinde oder Fehler
- **stat** liefert Statusinformation zu einer Datei (Dateityp, Zugriffsrechte, ...):
`int stat(const char *dateiname, struct stat *buf) ; // <sys/stat.h>`
liefert 0 oder bei Fehler -1
Ausgabeparameter

Beispiel-Programm Verzeichnisse (1)

```
#define _POSIX_C_SOURCE 1

#include <stdio.h>    // fprintf, printf
#include <string.h>    // strerror

#include <sys/stat.h> // struct stat, S_ISDIR
#include <dirent.h>    // DIR, struct dirent, opendir, readdir
#include <errno.h>     // errno

int main(int argc, char *argv[])
{
    for (int i = 1; i < argc; ++i) {
        // Datei vorhanden?
        struct stat s; // Dateistatus
        if (stat(argv[i], &s) == -1) ... // Fehlerbehandlung

        // Dateityp Verzeichnis?
        if (!S_ISDIR(s.st_mode)) ... // Fehlerbehandlung

        ...
    }
}
```

Listet Verzeichnisse auf

Beispiel-Programm Verzeichnisse (2)

```
...  
    DIR *d = opendir(argv[i]); // geoeffnetes Verzeichnis  
    if (d == NULL) ... // Fehlerbehandlung  
  
    errno = 0;  
  
    struct dirent *e; // gelesener Verzeichniseintrag  
    while ((e = readdir(d)) != NULL) {  
        printf("%s/%s\n", argv[i], e->d_name);  
    }  
  
    if (errno) ... // Fehlerbehandlung  
    closedir(d);  
}  
  
return 0;  
}
```

Ein-/Ausgabe: Index

<dirent.h>	8-2,8-12,8-13	fputc	8-4
<errno.h>	8-2,8-9	fputs	8-4
<fcntl.h>	8-2,8-8,8-10	fread	8-5
<stdio.h>	8-3 bis 8-6	fscanf	8-5
<sys/stat.h>	8-2,8-10,8-12	fwrite	8-5
<unistd.h>	8-2,8-7 bis 8-10	open	8-2,8-8
clearerr	8-5	opendir	8-2,8-12
close	8-2,8-8	perror	8-6
closedir	8-2,8-12	POSIX	8-1
Dateideskriptor	8-8	read	8-2,8-9
DIR	8-12	readdir	8-2,8-12
errno	8-2,8-9	ssize_t	8-9
fclose	8-3	stat	8-2,8-12
feof	8-6	stderr	8-3
ferror	8-6	stdin	8-3
fgetc	8-4	stdout	8-3
fgets	8-4	struct dirent	8-12,8-13
FILE	8-3	struct stat	8-12,8-13
fopen	8-3	write	8-2,8-9
fprintf	8-5		