Linux/Unix Systemprogrammierung

Pablo Yánez Trujillo

Poolmanager Team
Universität Freiburg

August 15, 2006

- Wir haben gestern die Standard I/O-Funktionen von ANSI C kennengelernt
- Wir wollen jetzt mehr über die elementaren I/O-Funktionen erfahren, die von den Standard I/O-Funktionen aufgerufen werden
- Anders als bei den Standard I/O-Funktionen, arbeiten die elementaren I/O-Funktionen ohne FILE Struktur sondern mit den Filedeskriptoren

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pfad, int flags);
int open(const char *pfad, int flags, mode_t modus);
```

- Je nach `flags`, muss `modus` gesetzt werden
- `flags` kann eine der folgenden in `<fcntl.h>` definierten symbolischen Konstanten angegeben werden:

Notwendige Flags (nur eins davon)

Flag	Wirkung
`O_RDONLY`	Nur zum Lesen öffnen
`O_WRONLY`	Nur zum Schreiben öffen
`O_RDWR`	Zum Lesen und Schreiben öffnen

Folgende Flags können auch angegeben werden

Flag	Wirkung
`O_APPEND`	Zum Anhängen öffnen
`O_CREAT`	Neu anlegen, wenn nicht vorhanden. `modus` angeben
`O_TRUNC`	Datei wird vollständig geleert

Für `modus` sind eine oder mehrere mit | (bitweises OR) verknüpfte Konstanten

Konstante	Bedeutung
S_ISUID	set-user-ID Bit
S_ISGID	set-group-ID Bit
S_IRUSR	read (user)
S_IWUSR	write (user)
S_IXUSR	execute (user)
S_IRWXU	read, write, execute (user)
S_IRGRP	read (group)
S_IWGRP	write (group)
S_IXGRP	execute (group)
S_IRWXG	read, write, execute (group)
S_IROTH	read (other)
S_IWOTH	write (other)
S_IXOTH	execute (other)
S_IRWXO	read, write, execute (other)

```
open("add", O_WRONLY | O_CREAT, S_IRWXU | S_IRGRP |
S_IXGRP | S_IXOTH);
```

Neue Datei `add` mit den Zugriffsrechten `-rwxr-x-x` anlegen und diese zum Schreiben öffnen

```
open("kunden.txt", O_APPEND);
```

Datei `kunden.txt` zum Schreiben am Dateiende öffnen

```
open("tempdat", O_WRONLY | O_TRUNC);
```

Datei `tempdat` zum Schreiben öffnen. Falls sie vorhanden ist, wird ihren Inhalt gelöscht

creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *pfad, mode_t modus);
```

Mit creat kann man neue Dateien anlegen. Es sind äquivalent

```
creat(pfad, modus);

open(pfad, O_RDWR | O_CREAT | O_TRUNC, modus);
```

- Genauso wie bei den Standard I/O Funktionen, müssen geöffnete Datein geschlossen werden

close

```
#include <unistd.h>

int close(int fd);
```

- Liefert 0 bei Erfolg, -1 sonst

read

```
#include <unistd.h>

size_t read(int fd, void *puffer, size_t byteanzahl);
```

- read liefert die Anzahl der tatsächlichen gelesenen Bytes zurück.
- Ist die Rückgabe 0, so ist man am Ende der Datei (EOF)
- Ist die Rückgabe -1, dann ist ein Lesefehler aufgetreten

write

```
#include <unistd.h>

size_t write(int fd, void *puffer, size_t byteanzahl);
```

- `write` liefert die Anzahl der tatsächlichen geschriebenen Bytes zurück.
- Ist die Rückgabe 0, so ist man am Ende der Datei (EOF)
- Ist die Rückgabe -1, dann ist ein Lesefehler aufgetreten

lseek

```
#include <sys/types.h>
#include <unistd.h>

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

- Verhält sich genauso wie `fseek`
- Bei Erfolg wird die neue Position des Schreib-/Lesezeigers zurückgeliefert
- Sonst -1 zurückgeliefert

```
lseek(fd, 0L, SEEK_SET);
```

Schreib-/Lesezeiger auf Dateianfang setzen

```
lseek(fd, 25L, SEEK_CUR);
```

Schreib-/Lesezeiger von momentaner Position aus um 25 Bytes vorrücken

```
lseek(fd, -1L, SEEK_END);
```

Schreib-/Lesezeiger auf das letzte relevante Byte (nicht auf `EOF`) setzen

- Es gibt Anwendungsfällen, in denen man existierende Filedeskriptoren duplizieren muss

dup

```
#include <unistd.h>

int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

- Beide geben bei Erfolg den neuen Filedeskriptor zurück, -1 sonst
- dup2 macht aus `newfd` eine Kopie von `oldfd` und schließt `newfd` (wenn nötig)
- Die alten und neuen Deskriptoren referenzieren dieselbe Datei, insbesondere sind ihre Schreib-/Lesezeiger stets an der selben Stelle
- Siehe Beispiel `tag02/dup`

fileno

```
#include <stdio.h>

int fileno(FILE *fp);
```

fdopen

```
#include <stdio.h>

FILE *fdopen(int fd, const char *modus);
```

Übungen

- Unter `tag02/readwrite` befinden sich die Sources zu dem letzten Beispiel
- Ändere das Programm so, dass es (bis auf `printf`) keine Standard I/O Funktionen sondern nur elementare Funktionen im Programmcode vorkommen

- Wir haben bereits gesehen, dass es im wesentlichen 3 wichtigen Datein gibt, die die Benutzerinformation enthalten
  - `/etc/passwd`
  - `/etc/shadow`
  - `/etc/group`
- Um an die Benutzerinformation ranzukommen, könnten wir diese Dateien parsen

- Bitte, das Rad nicht neu erfinden, es existieren bereits Funktionen, die Daten aus diesen Dateien lesen

Die passwd Struktur

Die passwd Struktur

```
/* definiert in pwd.h */

struct passwd {
    char   *pw_name;       /* user name */
    char   *pw_passwd;     /* user password
                              (not POSIX) */
    uid_t   pw_uid;        /* user ID */
    gid_t   pw_gid;        /* group ID */
    char   *pw_gecos;      /* real name (not POSIX) */
    char   *pw_dir;        /* home directory */
    char   *pw_shell;      /* shell program */
};
```

getpwuid/getpwnam

```
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);

struct passwd *getpwnam(const char *login);
```

- Bei Erfolg liefern beide Funktionen einen Zeiger auf eine `struct passwd` Struktur
- Sonst `NULL`
- Die Struktur ist als **static**-Variable definiert. Jeder neue Aufruf führt dazu, dass ihr alter Wert verloren geht

Die group Struktur

```
/* Definiert in grp.h */

struct group {
    char   *gr_name;        /* group name */
    char   *gr_passwd;      /* group password
                               (not POSIX) */
    gid_t   gr_gid;         /* group ID */
    char  **gr_mem;         /* group members */
};
```

getgrgid/getgrnam

```
#include <sys/types.h>
#include <grp.h>

struct group *getgrgid(gid_t gid);

struct group *getgrnam(const char *group);
```

- Bei Erfolg liefern beide Funktionen einen Zeiger auf eine `struct group` Struktur
- Sonst `NULL`
- Die Struktur ist als **static**-Variable definiert. Jeder neue Aufruf führt dazu, dass ihr alter Wert verloren geht

Die utsname Struktur

```
/* definiert in sys/utsname.h */

struct utsname {
    char sysname[];          /* Betriebsystemname */
    char nodename[];          /* Rechnername */
    char release[];          /* Release Name */
    char version[];          /* Release Version */
    char machine[];          /* Hardware */
#ifndef _GNU_SOURCE
    char domainname[];          /* Domainname */
#endif
};
```

uname

```
#include <sys/utsname.h>

int uname(struct utsname *buf);
```

- Bei Erfolg wird eine nicht negative Zahl zurückgeliefert
- -1 sonst

Übungen

- Schreib ein Programm, welches die gleiche Ausgabe hat, wie `finger`
- Schreib ein Programm, welches eine vollständige Liste von Benutzern und Gruppen
- Schreib ein Programm, welches sich wie `uname` verhält