Betriebssystem SINIX
CES
C-Entwicklungssystem
Beschreibung – Teil 2

Ausgabe Mai 1990 (CES V5.22)

Bestell-Nr. U3900-J-Z95-3
Printed in the Federal Republic of Germany
10510 AG 5902. (13140)

X/Open XPG 3-Konform
Warenzeichen beantragt.

Herausgegeben vom Bereich
Daten- und Informationstechnik
Postfach 830951, D-8000 München 83

Siemens Aktiengesellschaft

Vorwort

Ein Blick auf die SINIX-Handbücher

Dieses Handbuch richtet sich an alle Anwender, die im Betriebssystem SINIX C-Programme schreiben möchten. Es beschreibt die Systemaufrufe, C-Funktionen und Makros, die in einem C-Programm aufgerufen werden, und es beschreibt externe Variablen und Include-Dateien die von C-Programmen verwendet werden.

Der Programmiersprache C fehlen einige Möglichkeiten, die in anderen Programmiersprachen eingebaut sind. In der Sprache C gibt es z.B. kein eingebautes System zur Ein-/Ausgabe, keine dynamische Speicherverwaltung, keine Operatoren für zusammengesetzte Datentypen.

Im Betriebssystem SINIX stellt das C-Entwicklungssystem (CES) Bibliotheken zur Verfügung, in denen Sie Funktionen für die angesprochenen Aufgaben finden. Dies sind die in diesem Handbuch beschriebenen Funktionen.

Eine Einführung in das Betriebssystem SINIX und in einige Editoren, die Sie verwenden können, um Ihre C-Programme zu erstellen, finden Sie in dem Handbuch SINIX Einführung V5.22 [3].

Die Beschreibung der Kommandos des Betriebssystems SINIX und der Dienstprogramme sowie der Werkzeuge zur C-Programmierung finden Sie in dem Handbuch SINIX Kommandos V5.22 [1].
Die Werkzeuge zur C-Programmierung werden wie SINIX Kommandos auf Shell-Ebene aufgerufen und dienen dazu, fertige C-Programme auf dem Rechner zu installieren, auszuführen, zu testen und zu verwalten.

Das Handbuch im Überblick

Die Gliederung dieses Handbuchs und der Aufbau der einzelnen Beschreibungen lehnt sich an den Standard an, der durch den X/OPEN Portability Guide [6] gesetzt wird.

Wegen des großen Umfangs wurde der Inhalt dieses Handbuchs in zwei Teile aufgeteilt. Dabei wurde die Aufteilung so gewählt, daß die Erfordernisse der Praxis berücksichtigt sind.

Aufteilung der Beschreibung CES C-Entwicklungssystem

Teil 1	1 Einführung 2 Allgemeine Hinweise Funktionsübersicht 3 Funktionen (1. Teil) Anhang Abkürzungen Fachwörter Literatur Stichwörter
Teil 2	Funktionsübersicht 3 Funktionen (2. Teil) 4 Include-Dateien Anhang Abkürzungen Fachwörter Literatur Stichwörter

1 Einführung

Die Einführung

- erläutert das Beschreibungsformat,
- erklärt wichtige Begriffe,
- gibt einige Ratschläge, wie Sie Fehler in Ihrem Programm verhindern können.

2 Allgemeine Hinweise

In diesem Kapitel finden Sie allgemeine Anmerkungen zu einzelnen Funktions-Bibliotheken oder -Paketen. So finden Sie dort Informationen über

- Dateikennzahlen und Datenströme,
- die Fehlernummern und ihre Bedeutung,
- die Signale,
- die *allgemeine Terminalschnittstelle*,
- die *NLS*-Funktionen (internationalisierte Programme),
- die Funktionen zur Interprozeß-Kommunikation,
- die *curses*-Bibliothek,
- die *termcap*-Bibliothek.

3 Funktionen

In diesem Kapitel finden Sie in alphabetischer Reihenfolge die Beschreibungen aller Systemaufrufe, C-Funktionen und Makros, die Sie in Ihren C-Programmen aufrufen können.
Die Anordnung in alphabetischer Reihenfolge entspricht den gewohnten SINIX CES-Handbüchern und dem X/Open Portability Guide; sie entspricht nicht der Aufteilung in der Standard UNIX-Literatur, wo die Systemaufrufe getrennt von den C-Funktionen und Makros beschrieben werden. Aus folgenden Gründen halten wir die Zusammenfassung in einem Kapitel für vorteilhaft:

Die Aufteilung in Systemaufrufe und C-Funktionen ist nicht zwingend; sie kann daher in verschiedenen Implementierungen unterschiedlich ausfallen.

Die Benutzerschnittstelle ist bei einem Systemaufruf und bei einer C-Funktion dieselbe, d.h. die Deklarations- und Aufrufsyntax von Systemaufruf und C-Funktion unterscheiden sich nicht.

Die alphabetische Anordnung in einem Kapitel erleichtert das Suchen nach Funktionen erheblich.

Aus Platzgründen sind einige Funktionen zusammen mit anderen beschrieben. In diesen Fällen finden Sie unter dem Namen der gesuchten Funktion eine stichwortartige Beschreibung ihres Verwendungszwecks, die Definition der Funktion und einen Verweis auf die Stelle, an der die Funktion ausführlich beschrieben ist. Dies gilt nicht für die Funktionen zur Bildschirmbehandlung, die zur *curses*-Bibliothek gehören; diese finden Sie geschlossen unter dem Stichwort *curses*, wo die Funktionen vollständig beschrieben werden.

Den beiden Teilen der Funktionsbeschreibungen wird jeweils eine Funktionsübersicht vorangestellt, in der die vorhandenen Funktionen, Systemaufrufe, Variablen und Makros gruppenweise angeführt werden, so daß Sie dort den Namen einer gesuchten Funktion schnell finden können.

4 Include-Dateien

In diesem Kapitel wird der Inhalt von Include-Dateien beschrieben, die für verschiedene Systemaufrufe und Bibliotheksfunktionen verwendet werden.
Die Include-Dateien werden in der UNIX-Literatur auch als Vorspanndateien oder Header-Files bezeichnet.

Voraussetzungen dieses Handbuchs

Wenn Sie mit dem CES und mit diesem Handbuch arbeiten wollen, sollten Sie folgende Kenntnisse bereits besitzen:

- Grundkenntnisse des Betriebssystems SINIX
  (Shell, Dateisystem, ...; siehe Handbuch SINIX Einführung)
- Kenntnis der Programmiersprache C.
  Die CES-Dokumentation ist kein Lehrbuch der Sprache C. Es wird vorausgesetzt, daß Sie mit den Sprachelementen vertraut sind und bereits C-Programme erstellen können. Die Titel einiger C-Lehrbücher finden Sie im Literaturverzeichnis im Anhang. Die CES-Dokumentation ist auch keine Anleitung für den Anfänger. Sie sollten also bereits wissen, wie Sie mit Hilfe eines Editors ein C-Programm auf dem Rechner erfassen (siehe das Handbuch SINIX Einführung) und welche prinzipiellen Schritte erforderlich sind, um ein ablauffähiges Programm zu erzeugen.

Eine Bitte an Sie

Keine erklärende Dokumentation kann perfekt sein. Eine Dokumentation lebt. Sie lebt auch von Ihren Anregungen, Ideen und Verbesserungsvorschlägen. Helfen Sie uns, indem Sie uns Ihre Stolpersteine mitteilen, damit wir sie aus dem Weg räumen können.

Manualredaktion DI ST QM2
Otto-Hahn-Ring 6, 8 München 83 der Funktionen nach thematischen Gesichtspunkten.

Inhalt

1	**Einführung**	**1-1**
1.1	Beschreibungsformat	1-1
1.1.1	Beschreibungsrahmen	1-1
1.1.2	Bedeutung der Schriftart	1-3
1.2	Glossar	1-4
1.3	Ratschläge	1-22
1.3.1	Ein-/Ausgabe: C-Funktionen und Systemaufrufe nicht mischen	1-22
1.3.2	Fehlerabfrage	1-22
1.3.3	Zeiger als Ergebnistyp	1-23
1.3.4	Zeiger als Ergebnisparameter	1-23
1.3.5	Konstante oder symbolische Konstante?	1-23
1.3.6	Include-Dateien berücksichtigen	1-24
2	**Allgemeine Hinweise**	**2-1**
2.1	Dateikennzahlen und Datenströme	2-2
2.2	Fehlernummern	2-5
2.3	Signale	2-11
2.4	Allgemeine Terminalschnittstelle	2-13
2.4.1	Schnittstellen-Eigenschaften	2-13
2.4.2	Einstellbare Parameter	2-24
2.5	Internationalisierung in SINIX	2-35
2.6	Interprozeßkommunikation,	2-49
2.6.1	Auswirkungen auf die Systemschnittstellen	2-49
2.6.2	Allgemeine Beschreibung	2-49
2.7	Bildschirmsteuerung mit curses	2-54
2.7.1	Überblick	2-54
2.7.2	Datentypen und die Include-Datei curses.h	2-56
2.7.3	Funktionen	2-60
2.7.4	Synchrone und vernetzte asynchrone Datensichtstationen	2-62
2.8	Bildschirmsteuerung mit termcap/terminfo	2-64
2.8.1	Die terminfo-Datenbank	2-64
2.8.2	Die termcap-Schnittstelle	2-90
3	**System-Schnittstellen**	**3-1**
3.1	Funktionsübersicht	3-2
	Dateibearbeitung	3-3
	Prozesse	3-7
	Speicherverwaltung und -operationen	3-10
	Systemorganisation	3-11
	Zeichen und Zeichenketten	3-12
	Reguläre Ausdrücke	3-13

	Fehlermeldungen	3-14
	Zeitfunktionen	3-14
	Mathematische Funktionen	3-15
	Zufallszahlen	3-16
	Such- und Sortierverfahren	3-17
	Manipulation einer seriellen Schnittstelle	3-17
	Bildschirmsteuerung	3-18
	Funktionen zur C-Schnittstelle des Druckspools	3-21
3.2	Funktionen a - i	3-23
3.3	Funktionen j - z	3-419
4	**Include-Dateien**	**4-1**
A	**Anhang**	**A-1**
	Abkürzungen	
	Fachwörter englisch - deutsch	**F-1**
	Fachwörter deutsch - englisch	**F-9**
	Literatur	**L-1**
	Stichwörter	**S-1**

3 System-Schnittstellen

Dieses Kapitel beschreibt die *SINIX*-Systemschnittstellen und deren Umgebung, um die Portabilität von Anwendungen auf der C-Quellebene zu unterstützen.

3.1 Funktionsübersicht

In diesem Abschnitt finden Sie eine Zusammenstellung der Funktionen nach thematischen Gesichtspunkten.

— Ein Fehlen der Klammern nach dem Funktionsnamen bedeutet, daß es sich nicht um eine Funktion handelt, sondern z.B. um eine externe Variable wie *end* in der Tabelle Speicherverwaltung und -operationen.

Die eingerückten Namen wie bei *exec*, *exit()* und *curses* bedeuten, daß Sie die Beschreibung dieser Funktionen unter dem Oberbegriff finden, unter dem sie eingerückt sind.

Durch die thematische Zusammenstellng kann es auch vorkommen, däs einzelne Funktionen mehrfach auftreten, wenn sich diese zu mehreren Themenkomplexen zuordnen lassen.

Dateibearbeitung

Dateizugriff

Name	Kurzbeschreibung
access()	Zugreifbarkeit überprüfen
close()	Datei schließen
dup()	zusätzliche Dateikennzahl einrichten
dup2()	zusätzliche Dateikennzahl einrichten
fclose()	Datei schließen und Puffer bereinigen
fdopen()	einer Dateikennzahl einen Dateizeiger zuweisen
fflush()	Dateipuffer bereinigen
fopen()	Datei öffnen
freopen()	Dateizeiger neu zuweisen
fseek()	Schreib/Lesezeiger positionieren
ftell()	Position des Schreib/Lesezeigers abfragen
fsync()	Dateizustand synchronisieren
lockf()	Dateisegmente sperren
lseek()	Schreib/Lesezeiger positionieren
open()	Datei für Schreib- oder Lesezugriff öffnen
rewind()	Schreib/Lesezeiger auf Datei-Anfang positionieren

Dateiverwaltung

Name	Kurzbeschreibung
access()	Zugreifbarkeit überprüfen
chdir()	aktuelles Dateiverzeichnis wechseln
chmod()	Dateizugriffsrechte ändern
chown()	Eigentümer und Gruppe einer Datei ändern
chroot()	Root-Dateiverzeichnis wechseln
clearerr()	Schreib/Lese-Fehler-Anzeige löschen
creat()	neue Datei anlegen oder vorhandene überschreiben
closedir()	Dateiverzeichnis schließen
fcntl()	eine geöffnete Datei steuern
fstat()	Dateiinformation ermitteln
fstatfs()	Dateisysteminformation ermitteln
fsync()	Dateizustand synchronisieren
ftw()	Dateibaum durchlaufen
getcwd()	Pfadnamen des aktuellen Dateiverzeichnisses abfragen
link()	Verweis auf eine Datei einrichten
mkdir()	Dateiverzeichnis erzeugen
mkfifo()	eine FIFO-Gerätedatei erzeugen
mknod()	Dateiverzeichnis, Gerätedatei oder normale Datei einrichten
mktemp()	Eindeutigen Dateinamen erzeugen
mount()	Dateisystem einhängen
opendir()	Dateiverzeichnis öffnen
readdir()	Eintrag in Dateiverzeichnis suchen
remove()	Datei löschen
rename()	Datei umbenennen
rewinddir()	an den Anfang des Dateiverzeichnisses positionieren
rmdir()	Dateiverzeichnis löschen
seekdir()	auf Dateiverzeichnis positionieren
stat()	Dateistatus ermitteln
statfs()	Dateisysteminformation ermitteln
telldir()	Adresse eines Dateiverzeichnisses
tempnam()	Dateinamen für temporäre Datei erzeugen
tmpfile()	temporäre Datei einrichten
tmpnam()	Dateinamen für temporäre Datei erzeugen

Name	Kurzbeschreibung
umask()	Schutzbitmaske abfragen und setzen
umount()	Dateisystem aushängen
unlink()	Eintrag in Dateiverzeichnis löschen
ustat()	Information über ein eingehängtes Dateisystem
utime()	Zeiteinträge für Dateizugriff und -änderung setzen

Datei-Information

Name	Kurzbeschreibung
access()	Zugreifbarkeit prüfen
feof()	auf Datei-Ende prüfen
ferror()	Datei auf Schreib/Lesefehler prüfen
fileno()	Dateikennzahl abfragen
fstat()	Datei-Informationen abfragen
fstatfs()	Dateisysteminformation ermitteln
stat()	Datei-Informationen abfragen
statfs()	Dateisysteminformation ermitteln
ustat()	Information über ein eingehängtes Dateisystem

Ein/Ausgabe

Name	Kurzbeschreibung
fgetc()	Zeichen einlesen
fgets()	Zeichenkette einlesen
fprintf()	formatierte Ausgabe in eine Datei
fputc()	ein Zeichen in eine Datei schreiben
fputs()	Zeichenkette in eine Datei schreiben
fread()	objektorientiertes Lesen
fscanf()	formatiertes Lesen aus Datei
fwrite()	objektorientiertes Schreiben
getc()	ein Zeichen einlesen
getchar()	Zeichen von Standardeingabe lesen
getopt()	Schalter aus dem Argumentvektor abfragen
gets()	Zeichenkette von Standardeingabe lesen
getw()	Maschinenwort-weise einlesen
putc()	Zeichen ausgeben
putchar()	Zeichen auf Standardausgabe schreiben
printf()	formatierte Ausgabe
puts()	Zeichenkette auf Standardausgabe schreiben
putw()	Maschinenwort-weise in eine Datei schreiben
read()	aus Datei lesen
scanf()	Formatierte Eingabe
setbuf()	Ein/Ausgabepuffer zuordnen
setvbuf()	Ein/Ausgabepuffer zuordnen (mit Typwahl)
sprintf()	formatierte Ausgabe in eine Zeichenkette
sscanf()	formatiertes Lesen aus einer Zeichenkette
stderr	Standarfehlerausgabe-Datenstrom
stdin	Standareingabe-Datenstrom
stdout	Standarausgabe-Datenstrom
ungetc()	ein Zeichen in den Puffer zurückstellen
vfprintf()	formatierte Ausgabe einer Argumenteliste
vprintf()	formatierte Ausgabe einer Argumenteliste
vsprintf()	formatierte Ausgabe einer Argumenteliste
write()	in Datei schreiben

Prozesse

Prozeßverwaltung

Name	Kurzbeschreibung
alarm()	Alarmuhr stellen
clock()	verbrauchte Rechenzeit angeben
cuserid()	Benutzerkennung als Zeichenkette
getegid()	effektive Gruppennummer abfragen
getenv()	Umgebungsvariable abfragen
geteuid()	effektive Benutzernummer abfragen
getgid()	reale Gruppennummer abfragen
getpgrp()	Prozeßgruppennummer abfragen
getpid()	Prozeßnummer abfragen
getppid()	Vaterprozeßnummer abfragen
getuid()	reale Benutzernummer abfragen
gsignal()	Software-Signal auslösen
kill()	Signal an Prozeß oder Prozeßgruppe senden
logname()	Benutzerkennung zurückgeben
pause()	Prozeß bis zum Eintreffen eines Signals anhalten
putenv()	Umgebungsvariable ändern oder ergänzen
setgid()	reale und effektive Gruppennummer setzen
setpgrp()	Prozeßgruppennummer setzen
setuid()	reale und effektive Benutzernummer setzen
sigaction()	Signalbehandlung
sigaddset()	Signal zu Signalmenge zufügen
sigdelset()	Signal aus Signalmenge entfernen
sigemptyset()	Signalmenge initialisieren
sigfillset()	Signalmenge initialisieren
sigismember()	Signalmenge auf Signal überprüfen
signal()	Signalbehandlung
sigpending()	anstehende Signale prüfen
sigprocmask()	Signalmaske prüfen und ändern
sissuspend()	auf Signal warten
sleep()	Prozeß für festgesetzte Zeitspanne anhalten
ssignal()	Software-Signale
times()	Laufzeit eines Prozesses und seiner Sohnprozesse abfragen
ulimit()	Prozeßschranken abfragen und setzen

Interprozeßkommunikation

Name	Kurzbeschreibung
ftok()	Standard-Interprozeß-Kommunikations-Paket
ipc()	Überblick über die Interprozeßkommunikation
msgctl()	Nachrichtensteuerung
msgget()	Nachrichtenwarteschlange einrichten
msgop()	Nachrichtenoperationen
msgrcv()	Nachricht empfangen
msgsnd()	Nachricht senden
semctl()	Semaphorsteuerung
semget()	Semaphorenmenge abfragen
semop()	Semaphorsteuerung
shmat()	gemeinsamen Speicherbereich anhängen
shmctl()	Steuerungsoperationen gemeinsamer Speicherbereiche
shmdt()	gemeinsamen Speicherbereich abhängen
shmget()	gemeinsamen Speicherbereich abfragen

Pipes

Name	Kurzbeschreibung
pclose()	Pipe zu einem Kommando schließen
pipe()	Pipe einrichten
popen()	Pipe zu einem Kommando öffnen

Zusammenwirken von Prozessen

Name	Kurzbeschreibung
abort()	anormaler Prozeßabbruch
exec	Programmaufrufe
execl()	
execle()	
execlp()	
execv()	
execve()	
execvp()	
exit()	Prozeß beenden
_exit()	
fork()	neuen Prozeß erzeugen
system()	Shell-Kommando aufrufen
wait()	auf Prozeßbeendigung warten

Steuerung von Programmabläufen

Name	Kurzbeschreibung
longjmp()	nicht-lokaler Sprung
nice()	Prozeßpriorität ändern
profil()	Zeitprofil der Programmausführung
monitor()	Auswertung der Programmausführung
setjmp()	nicht-lokaler Sprung
siglongjmp()	nicht-lokaler Sprung
sigsetjmp()	nicht-lokaler Sprung

Programmtest

Name	Kurzbeschreibung
assert()	Programmaussage prüfen
nlist()	Einträge aus der Symboltabelle abfragen
ptrace()	Prozeßüberwachung

Prozeßuniversum

Name	Kurzbeschreibung
universe()	Prozeßuniversum abfragen/wechseln

Speicherverwaltung und -operationen

Name	Kurzbeschreibung
brk()	den "break" neu setzen
calloc()	Speicherplatz für einen Vektor reservieren
edata	Ende-Adresse des initialisierten Speicherbereiches
end	Ende-Adresse des nicht initialisierten Speicherbereichs
etext	Ende-Adresse des Speicherbereiches für den Programmcode
free()	Speicherplatz freigeben
mallinfo()	Speicherplatzauslastung abfragen
malloc()	Speicherplatz reservieren
mallopt()	Algorithmus für Speicherplatzreservierung steuern
memccpy()	Zeichen kopieren (im Speicher)
memchr()	Zeichen suchen (im Speicher)
memcmp()	Zeichen vergleichen (im Speicher)
memcpy()	Zeichen kopieren (im Speicher)
memset()	Zeichen setzen (im Speicher)
realloc()	Speicherplatz verändern
sbrk()	den "break" verändern

Systemorganisation

Name	Kurzbeschreibung
crypt()	Kennwort verschlüsseln
encrypt()	ver-/entschlüsseln
endgrent()	Gruppendatei schließen
endpwent()	Kennwortdatei schließen
endutent()	utmp-Datei schließen
fgetgrent()	Zeiger auf die nächste Group-Struktur
fgetpwent()	Zeiger auf die nächste Kennwort-Struktur
fpathconf()	konfigurierbare Pfadnamenvariablen ermitteln
getgrent()	Eintrag aus der Gruppendatei lesen
getgrgid()	Gruppennummer in der Gruppendatei suchen
getgrnam()	Gruppennamen in der Gruppendatei suchen
getgroups()	weitere Gruppennummern ermitteln
gethz()	Taktfrequenz ermitteln
getlogin()	Login-Namen abfragen
getpass()	Kennwort lesen
getpw()	Benutzernummer in der Kennwortdatei suchen
getpwent()	nächste Zeile der Kennwortdatei lesen
getpwnam()	Benutzernamen in der Kennwortdatei suchen
getpwuid()	Benutzernummer in der Kennwortdatei suchen
getutent()	utmp-Eintrag lesen
getutline()	utmp-Eintrag lesen
pathconf()	konfigurierbare Pfadnamenvariablen ermitteln
putpwent()	Eintrag für die Kennwortdatei schreiben
setgrent()	auf den Anfang der Gruppendatei positionieren
setkey()	DES-Algorithmus einstellen
setpwent()	auf den Anfang der Kennwortdatei positionieren
setutent()	auf den Anfang der utmp-Datei positionieren
sync()	Superblock aktualisieren
sysconf()	konfigurierbare Systemvariablen ermitteln
uname()	Name des aktuellen SINIX Systems abfragen
utmpname()	Name der utmp-Datei festlegen

Zeichen und Zeichenketten

Einzelne Zeichen bearbeiten

Name	Kurzbeschreibung
isalnum()	alphanumerisches Zeichen?
isalpha()	Buchstabe?
isascii()	ASCII-Zeichen?
iscntrl()	Lösch- oder Steuerzeichen?
isdigit()	Ziffer?
isgraph()	abdruckbares Zeichen außer Leerzeichen?
islower()	Kleinbuchstabe?
isprint()	abdruckbares Zeichen?
ispunct()	Sonderzeichen?
isspace()	Zwischenraum?
isupper()	Großbuchstabe?
isxdigit()	hexadezimales Zeichen?
toascii()	Umwandlung in ASCII-Zeichen
tolower()	Umwandlung in Kleinbuchstaben
_tolower()	Umwandlung in Kleinbuchstaben
toupper()	Umwandlung in Großbuchstaben
_toupper()	Umwandlung in Großbuchstaben

Konvertierung von Größen

Name	Kurzbeschreibung
a64l()	ASCII-Zeichenkette in long integer
atof()	Zeichenkette in Gleitkommazahl
atoi()	Zeichenkette in integer
atol()	Zeichenkette in long integer
ecvt()	Gleitkommazahl in Zeichenkette
fcvt()	Gleitkommazahl in Zeichenkette
gcvt()	Gleitkommazahl in Zeichenkette
l3tol()	3-Byte-integer in long integer
l64a()	long integer in ASCII-Zeichenkette
ltol3()	long integer in 3-Byte-integer
strtod()	Zeichenkette in Gleitkommazahl Typ double
strtol()	Zeichenkette in long integer
swab()	Kopieren mit Vertauschen benachbarter Bytes

Zeichenketten bearbeiten

Name	Kurzbeschreibung
strcat()	Verkettung von zwei Zeichenketten
strchr()	erstes Vorkommen eines Zeichen in einer Zeichenkette
strcmp()	Vergleich zweier Zeichenketten
strcoll()	Zeichenketten gemäß Sortierreihenfolge vergleichen
strcspn()	Länge der Zeichenkette s1 ohne Zeichen aus s2
strcpy()	Zeichenkette kopieren
strdup()	Zeichenkette duplizieren
strlen()	Länge einer Zeichenkette abfragen
strncat()	Verkettung bis zur Länge n
strncmp()	Vergleich bis zur Länge n
strncpy()	Kopie bis zur Länge n
strpbrk()	erstes Zeichen in Zeichenkette s1 aus Zeichenkette s2
strrchr()	letztes Vorkommen eines Zeichens in einer Zeichenkette
strspn()	Länge der Zeichenkette s1 aus Zeichen aus s2
strstr()	Zeichenkette in Zeichenkette suchen
strtok()	Zeichenkette s1 auf Trennzeichen aus s2 durchsuchen
strxfrm()	Zeichenketten umwandeln
swab()	Kopieren mit Vertauschen benachbarter Bytes

Reguläre Ausdrücke

Name	Kurzbeschreibung
regexp	reguläre Ausdrücke übersetzen und Muster erkennnen
compile()	reguläre Ausdrücke übersetzen
step()	Muster erkennen
advance()	Muster erkennen

Fehlermeldungen

Name	Kurzbeschreibung
perror()	Fehlermeldung auf Standard-Fehlerausgabe schreiben
errno	externe Variable mit Fehlercode
strerror	Fehlermeldungstexte

Zeitfunktionen

Name	Kurzbeschreibung
asctime()	Datum mit Uhrzeit in Englisch
ctime()	Datums- und Zeitangaben in Zeichenketten umwandeln
daylight	Sommerzeit-Kennzeichen
gmtime()	aktuelle Zeit GMT als Struktur
localtime()	aktuelle Zeit MEZ als Struktur
mktime()	Zeit als Struktur in Zeit sein Epochenbeginn umwandeln
stime()	Systemuhr stellen
strftime()	Datum und Zeit als Zeichenkette darstellen
time()	aktuelle Zeit abfragen
timezone	Zeitzonen-Information
tzname	Vektor mit Zeitzonen-Namen
tzset()	externe Variablen setzen

Mathematische Funktionen

Name	Kurzbeschreibung
abs()	Absolutbetrag
acos()	Arcus Cosinus x
asin()	Arcus Sinus x
atan()	Arcus Tangens x
atan2()	Arcus Tangens y/x
ceil()	ganzzahlig aufrunden
cos()	Cosinus x
cosh()	Cosinus Hyperbolicus x
erf()	Fehlerfunktion
erfc()	Komplement der Fehlerfunktion
exp()	Exponentialfunktion
fabs()	Absolutbetrag einer Gleitkommazahl
floor()	ganzzahlig abrunden
fmod()	Gleitkommarest von x/y
frexp()	Gleitkommazahl zerlegen in Mantisse und Exponent zur Basis 2
gamma()	logarithmische Gammafunktion
hypot()	Euklidische Distanzfunktion
isnan()	Test auf NaN (Not a Number)
j0()	Besselfunktion
j1()	
jn()	
ldexp()	Umkehrfunktion zu frexp
lgamma()	logarithmische Gammafunktion
log()	natürlicher Logarithmus x
log10()	Logarithmus x zur Basis 10
matherr()	Funktion zur Fehlerbehandlung
modf()	Aufspalten in Ganzteil und Bruchteil
pow()	allgemeine Exponentialfunktion
signgam	externe Variable zu *gamma*(3X)

Name	Kurzbeschreibung
sin()	Sinus x
sinh()	Sinus Hyperbolicus x
sqrt()	Quadratwurzel
tan()	Tangens x
tanh()	Tangens Hyperbolicus x
y0()	Besselfunktion
y1()	
yn()	

Zufallszahlen

Name	Beschreibung
drand48()	nicht negative Zufallszahl Typ double
erand48()	nicht negative Zufallszahl Typ double
jrand48()	Zufallszahl Typ long
lcong48()	Initialisierungsfunktion
lrand48()	nicht negative Zufallszahl Typ long
mrand48()	Zufallszahl Typ long
nrand48()	nicht negative Zufallszahl Typ long
rand()	einfacher Zufallszahlengenerator
seed48()	Initialisierungsfunktion
srand()	Initialisieren des Zufallszahlen-Generators
srand48()	Initialisierungsfunktion

Such- und Sortierverfahren

Name	Beschreibung
bsearch()	binäres Durchsuchen einer sortierten Tabelle
hcreate()	Hashtabelle anlegen
hdestroy()	Hashtabelle zerstören
hsearch()	Suchroutine für Hashtabellen
lfind()	lineare Suchroutine ohne Anhängen
lsearch()	lineare Suchroutine mit Anhängen
qsort()	Quicksort
tdelete()	Knotenpunkt aus Baumstruktur entfernen
tfind()	Baumstruktur durchsuchen
tsearch()	Baumstruktur aufbauen und durchsuchen
twalk()	Baumstruktur durchlaufen

Manipulation einer seriellen Schnittstelle

Name	Kurzbeschreibung
cfgetispeed()	Eingabe-Baudrate ermitteln
cfgetospeed()	Ausgabe-Baudrate ermitteln
cfsetispeed()	Eingabe-Baudrate setzen
cfsetospeed()	Ausgabe-Baudrate setzen
ctermid()	Dateinamen für Datensichtstation generieren
dial()	Kommunikationsleitung einrichten
ioctl()	Gerätesteuerung
isatty()	Datensichtstation?
tcdrain()	Auf Übertragung der Ausgabe warten
tcflow()	Flußkontrolle für Datenübertragung
tcflush()	Ein-/Ausgabepuffer leeren
tcgetattr()	Attribute einer Datensichtstation ermitteln
tcgetpgroup()	Vordergrund-Prozeßgruppennummer ermitteln
tcsendbreak()	Pause für bestimmte Zeit senden
tcsetattr()	Attribute einer Datensichtstation setzen
tcsetpgroup()	Vordergrund-Prozeßgruppennummer setzen
ttyname()	Name einer Datensichtstation abfragen
ttyslot()	Eintrag für aktuellen Benutzer in /etc/utmp suchen
undial()	Kommunikationsleitung freigeben

Bildschirmsteuerung

Name	Kurzbeschreibung
curses	
addch()	Zeichen in Fenster ausgeben
mvaddch()	
mvwaddch()	
waddch()	
addstr()	Zeichenkette in Fenster ausgeben
mvaddstr()	
mvwaddstr()	
waddstr()	
attroff()	Fenster-Attribute behandlen
attron()	
attrset()	
standend()	
standout()	
wattroff()	
wattron()	
wattrset()	
wstandend()	
wstandout()	
baudrate()	Baudrate der Datensichtstation
beep()	audiovisuelle Signale
flash()	
box()	Rahmen zeichnen
cbreak()	CBREAK-Modus einschalten
nocbreak()	CBREAK-Modus ausschalten
clear()	Fenster löschen
wclear()	
clearok()	Bildschirmlöschen aktivieren
clrtobot()	Löschen bis Ende Bildschirm
wclrtobot()	
clrtoeol()	Löschen bis Zeilenende
wclrtoeol()	
def_prog_mode(),def_shell_mode()	Modus der Datensichtstation speichern
delay_output()	verzögerte Ausgabe
delch()	Zeichen löschen
mvdelch()	
mvwdelch()	
wdelch()	

noch Bildschirmsteuerung

Name	Kurzbeschreibung
curses	
deleteln()	Zeile löschen
wdeleteln()	
delwin()	Fenster löschen
echo()	Ausgabe einschalten
noecho()	Ausgabe unterdrücken
endwin()	*curses* beenden
erase()	Fenster löschen
werase()	
erasechar()	ERASE-Zeichen ermitteln
flushinp()	Zeichenpuffer löschen
getch()	Zeichen einlesen
mvgetch()	
mvwgetch()	
wgetch()	
getstr()	Zeichenkette einlesen
mvgetstr()	
mvwgetstr()	
wgetstr()	
getyx()	Cursorposition ermitteln
has_ic()	Zeichen einfügen/löschen prüfen
has_il()	Zeilen einfügen/löschen prüfen
idlok()	hardwaremäßiges einfügen/löschen aktivieren
inch()	Zeichen aus Fenster liefern
mvinch()	
mvwinch()	
winch()	
initscr()	Umgebung der Datensichtstation initialisieren
insch()	Zeichen einfügen
mvinsch()	
mvwinsch()	
winsch()	
insertln()	Zeile einfügen
winsertln()	
intrflush()	Warteschlange bei Signal löschen
keypad()	Funktionstastenblock aktivieren
killchar()	KILL-Zeichen ermitteln
leaveok()	Cursorposition unverändert lassen
longname()	Namen der Datensichtstation ermitteln

noch Bildschirmsteuerung

Name	Kurzbeschreibung
curses	
move()	Cursor bewegen
wmove()	
newpad()	neuen *Pad* einrichten
newterm()	neue Datensichtstation eröffnen
newwin()	neues Fenster eröffnen
nl()	Umsetzung des NL-Zeichens einschalten
nonl()	Umsetzung des NL-Zeichens ausschalten
nodelay()	Blockierung beim Einlesen ausschalten
overlay()	Fenster überlagern
overwrite()	
prefresh()	*Pad* aktualisieren
pnoutrefresh()	
printw()	formatierte Ausgabe im Fenster
mvprintw()	
mvwprintw()	
wprintw()	
raw()	RAW-Modus einschalten
noraw()	RAW-Modus ausschalten
refresh()	Fenster aktualisieren
wrefresh()	
reset_prog_mode(), reset_shell_mode()	Datensichtstation: Betriebsart zurücksetzen
resetty()	Betriebsart der Datensichtstation zurücksetzen
savetty()	Betriebsart der Datensichtstation speichern
scanw()	formatiertes Lesen im Fenster
mvscanw()	
mvwscanw()	
wscanw()	
scroll()	Bildlauf durchführen
scrollok()	"Blättern" einschalten
set_term()	Umschalten zwischen Datensichtstationen
setscrreg()	Bildlaufbereich einstellen
wsetscrreg()	
subwin()	Unterfenster eröffnen
touchwin()	Fensteränderungs-Information löschen
typeahead()	Puffer prüfen

noch Bildschirmsteuerung

Name	Kurzbeschreibung
curses	
unctrl()	Zeichen in darstellbares Format bringen
wnoutrefresh()	effiziente Aktualisierung
doupdate()	
tgetent()	*termcap*-Eintrag lesen
tgetflag()	boolesches *termcap*-Feld lesen
tgetnum()	numerisches *termcap*-Feld lesen
tgetstr()	*termcap*-Steuerzeichenkette lesen
tgoto()	*termcap*-Cursorpositionierung vorbereiten
tputs()	*termcap*-Steuerzeichenketten ausgeben

Funktionen zur C-Schnittstelle des Druckspools

Druckerbeschreibung

Name	Kurzbeschreibung
getcfadnam()-	
getcftyent()	Druckerbeschreibung
getcfadnam()	Zeiger auf eine Struktur liefern
getcfgrnam()	
getcfprnam()	
getcftynam()	
getcfadent()	nächste Zeile einer Struktur lesen
getcfgrent()	
getcfprent()	
getcftyent()	
setcfadent()	erneutes Durchsuchen des Strukturfeldes
setcfgrent()	
setcfprent()	
setcftyent()	
endcfent()	Speicher freigeben

Durchsuchen der Auftragsdateien

Name	Kurzbeschreibung
getpdjbent()- getpdprent()	Durchsuchen der Auftragsdateien
getpooldat()	POOLDAT in den Speicher lesen
getpdjbent()	Zeiger auf Strukturen liefern, Strukturen lesen und durchsuchen
getpdjbnam()	
getpdjbnum()	
getpdprent()	
getpdprnam()	
setpdjbent()	erneutes Durchsuchen des Strukturfeldes
setpdprent()	
endpdent()	Speicher freigeben

Die Seiten 3-23 bis 3-418 (Abschnitt 3.2 Funktionen a - i) sind im SINIX CES V5.22 Teil 1 enthalten).

3.3 Funktionen j - z

NAME

j0, j1, jn - Bessel functions of the first kind
Bessel-Funktionen der ersten Art

DEFINITION

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
int n;
double x;
```

BESCHREIBUNG

Die Funktionen *j0()*, *j1()* und *jn()* liefern Bessel-Funktionen der ersten Art für *x* der Ordnungen 0, 1 und *n*.

ERGEBNIS

Bei erfolgreicher Beendigung liefern die Funktionen *j0()*, *j1()* und *jn()* den relevanten Bessel-Wert der ersten Art für *x*.

Ist das Argument *x* zu groß, so wird der Wert 0 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Anderfalls wird entweder *errno* gesetzt, um den Fehler anzuzeigen oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktionen *j0()*, *j1()* und *jn()* schlagen fehl, wenn gilt:

[ERANGE] Der Wert von *x* ist zu groß.

Die Funktionen *j0()*, *j1()* und *jn()* können in anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von *x* ist ein NaN.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *j0()*, *j1()*, oder *jn()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

```
cc programm.c -lm
```

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

isnan(), *matherr()*, *y0()*, *<math.h>*.

NAME

jrand48 - generate uniformly distributed pseudo-random long signed integers
Gleichmäßig verteilte, vorzeichenbehaftete Pseudo-Zufallszahlen vom Typ long erzeugen.

DEFINITION

```
long jrand48 (xsubi)
unsigned short xsubi[3];
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *jrand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

kill - send a signal to a process oder a group of processes
Signal an Prozeß oder Prozeßgruppe senden

DEFINITION

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

int kill (pid, sig)
pid_t pid;
int sig;
```

BESCHREIBUNG

Die Funktion *kill()* sendet ein Signal an einen Prozeß oder eine Prozeßgruppe, angegeben durch *pid*. Das zu sendende Signal wird durch *sig* angegeben und ist entweder eines der in der Datei *<signal.h>* angegebenen Signale oder gleich 0. Wenn *sig* gleich 0 ist (*Nullsignal*), dann wird die Fehlerüberprüfung durchgeführt, ohne daß ein Signal wirklich gesendet wird. Das Nullsignal kann verwendet werden, um die Gültigkeit von *pid* zu überprüfen.

Damit ein Prozeß die Erlaubnis hat, ein Signal an den durch *pid* angegebenen Prozeß zu senden, muß die reale oder effektive Benutzernummer des sendenden Prozesses mit der entsprechenden Benutzernummer des empfangenden Prozesses übereinstimmen, sofern der sendende Prozeß keine besonderen Rechte besitzt. Die gesicherte Benutzernummer des empfangenden Prozesses wird anstelle von dessen effektiver Benutzernummer überprüft. Wenn die effektive Benutzernummer des empfangenden Prozesses durch die Verwendung des S_ISUID-Bits geändert wurde (siehe auch *<sys/stat.h>*), so kann er dennoch Signale empfangen, die durch den Vaterprozeß oder einen Prozeß mit derselben Benutzernummer wie der Vaterprozeß gesendet wurden.

Wenn *pid* größer als 0 ist, dann wird *sig* an den Prozeß gesendet, dessen Prozeßnummer gleich *pid* ist.

Wenn *pid* gleich 0 ist, dann wird *sig* an alle Prozesse außer einer Anzahl von Systemprozessen gesendet, deren Prozeßgruppennummer gleich der Prozeßgruppennummer des Senders ist, und für die der Prozeß die Erlaubnis hat, ein Signal zu senden.

Wenn *pid* gleich -1 ist, dann wird *sig* an alle Prozesse gesendet, für die der Prozeß die Erlaubnis hat, ein Signal zu senden, außer den Systemprozessen.

Wenn *pid* negativ, aber ungleich -1 ist, dann wird *sig* an alle Prozesse gesendet, deren Prozeßgruppenummer gleich dem Absolutbetrag von *pid* ist, und für die der Prozeß die Erlaubnis hat, ein Signal zu senden.

Wenn der Wert von *pid* veranlaßt, daß *sig* für den sendenden Prozeß generiert wird, und wenn *sig* nicht blockiert ist, dann wird entweder *sig* oder zumindest ein anstehendes, nicht blockiertes Signal an den sendenden Prozeß zugestellt, bevor die Funktion *kill()* zurückkehrt.

Die Funktion *kill()* ist erfolgreich, wenn der Prozeß die Erlaubnis hat, *sig* an einen der durch *pid* angegebenen Prozesse zu senden. Wenn die Funktion *kill()* fehlschlägt, dann wird kein Signal gesendet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird der Wert -1 geliefert und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *kill()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert des Arguments *sig* ist eine ungültige oder nicht unterstützte Signalnummer.

[EPERM] Der Prozeß besitzt nicht die Erlaubnis, das Signal an einen empfangenden Prozeß zu senden.

[ESRCH] Es kann kein Prozeß oder keine Prozeßgruppe gefunden werden, die der durch *pid* angegebenen entspricht.

HINWEIS

In dieser Version des CES wird ein neuer Algorithmus für die Ermittlung der Berechtigung zum Senden eines Signals verwendet. Dies bedeutet, daß jetzt auch solchen Prozessen ein Signal zugestellt werden kann, deren effektive Benutzernummer zwar nicht mit der effektiven Benutzernummer des sendenden Prozesses übereinstimmt, aber deren reale, effektive oder gesicherte Benutzernummer mit der effektiven oder realen Benutzernummer des sendenden Prozesses übereinstimmt.

PORTABILITÄT

Die Funktion *kill()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Wenn eine andere X/Open-kompatible Implementierung das Signal SIGCONT unterstützt, dann finden die oben beschriebenen Prüfungen der Benutzernummer dann nicht statt, wenn das Signal SIGCONT an einen Prozeß gesendet wird, der Mitglied in derselben Sitzung wie der sendende Prozeß ist.

Eine andere X/Open-kompatible Implementierung, die erweiterte Sicherheitskontrollen bietet, kann zusätzliche, implementierungsabhängige Einschränkungen des Sendens von Signalen festlegen, einschließlich für das Nullsignal. Insbesondere kann das System die Existenz einiger oder aller Prozesse verneinen, die durch *pid* angegeben werden. Dies sollte bei der Entwicklung portabler Anwendungen berücksichtigt werden.

BEISPIEL

Ein Programm, das sich selbst abbricht:

```
#include <stdio.h>
#include <signal.h>

main()

{  int i=1;

   while (1)
   {  printf("Endlosschleife: %d. Durchlauf\n", i++);
      sleep(2);
      kill(getpid(),SIGKILL);
   }
}
```

SIEHE AUCH

getpid(), *setsid()*, *sigaction()*, *<signal.h>*, *<sys/types.h>*.

NAME

l3tol - Umwandlung von 3-Byte-Ganzzahl in long int

DEFINITION

```
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;
```

BESCHREIBUNG

l3tol() wandelt eine Liste von *n* 3 Byte langen Ganzzahlen, die in einem durch *cp* referenzierten Zeichenvektor abgelegt sind, in eine Liste von long-Werten um, die in einem durch *lp* referenzierten Vektor abgelegt werden.

Diese Funktion und ihre Umkehrfunktion *ltol3()* sind für die Verwaltung von Dateisystemen nützlich, wo Plattenadressen drei Byte lang sind.

HINWEIS

Aufgrund möglicher Unterschiede in der Byteanordnung sind die numerischen Werte jeweils maschinenabhängig.

PORTABILITÄT

Die Funktion *l3tol()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

ltol3().

NAME

l64a - Ganze Zahl (Typ long) in ASCII-Zeichenkette (Basis 64) umwandeln

DEFINITION

```
char *l64a (l)
long l;
```

BESCHREIBUNG

l64a() ist die inverse Funktion zu *a64l()*.

Eine ausführliche Beschreibung von *l64a()* finden Sie unter *a64l()*.

PORTABILITÄT

Die Funktionen *a64l()* und *l64a()* sind im X/OPEN-Standard nicht enthalten.

NAME

lcong48 - seed uniformly distributed pseudo-random signed long integer generator
Gleichmäßig verteilte, vorzeichenbehaftete ganzzahlige Pseudo-Zufallszahl vom Typ long setzen

DEFINITION

```
void lcong48 (param)
unsigned short param[7];
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *lcong48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

ldexp - load exponent of a floating point number
Exponent einer Gleitpunktzahl laden

DEFINITION

```
#include <math.h>

double ldexp (value, exp)
double value;
int exp;
```

BESCHREIBUNG

Die Funktion *ldexp()* berechnet *value* * 2*exp*. Sie ist die Umkehrfunktion zu *frexp()*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ldexp()* einen Wert vom Typ *double*, der aus dem Wert *value* und dem Exponenten *exp* gebildet wurde.

Wenn der Wert von *value* ein NaN ist, dann wird ein NaN zurückgeliefert.

Wenn *ldexp()* einen Überlauf verursachen würde, dann wird +HUGE_VAL oder -HUGE_VAL zurückgeliefert, je nach Vorzeichen von *value*, und *errno* ist gleich [ERANGE] gesetzt.

Wenn *ldexp()* einen Unterlauf verursachen würde, dann wird der Wert 0 zurückgeliefert und *errno* ist gleich [ERANGE] gesetzt.

Andernfalls ist entweder *errno* gesetzt, um den Fehler anzuzeigen, oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktion *ldexp()* schlägt fehl, wenn gilt:

[ERANGE] Der zurückgelieferte Wert würde einen Überlauf oder einen Unterlauf verursachen.

Die Funktion *ldexp()* kann unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Das Argument *value* ist ein NaN.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *ldexp()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *ldexp()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispiel zeigt die Wirkung von *frexp()* und *ldexp()*. *frexp()* zerlegt sein Gleitkommaargument in Mantisse und Exponent zur Basis 2, und *ldexp()* berechnet aus diesen Teilen wieder den ursprünglichen Wert, hier vorgeführt für die Zahl 5.3421:

```
#include <math.h>

double frexp();
int ex;

main()
{
   double x;

   x = frexp(5.3421,&ex);

   printf("Mantisse: %lf\nExponent : %d\n", x, ex);
   printf("Ausgangswert : %lf\n", ldexp(x, ex));
}
```

SIEHE AUCH

frexp(), *isnan()*, *matherr()*, *<math.h>*.

NAME

lfind - find entry in linear search table
Eintrag in linearer Suchtabelle finden

DEFINITION

```
#include <search.h>

char *lfind (key, base, nelp, width, compar)
void *base, *key;
size_t width, *nelp;
int (*compar)();
```

BESCHREIBUNG

Siehe unter *lsearch()*.

PORTABILITÄT

Die Funktion *lfind()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

lgamma - log gamma function
logarithmische Gamma-Funktion

DEFINITION

```
#include <math.h>

double lgamma (x)
double x;

extern int signgam;
```

BESCHREIBUNG

Die Funktion *lgamma()* arbeitet genauso, wie die Funktion *gamma()*, einschließlich der Verwendung von *signgam*.

Siehe unter *gamma()*.

PORTABILITÄT

Die Funktion *lgamma()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

link - link to a file
Verweis auf Datei einrichten

DEFINITION

```
int link (path1, path2)
char *path1, *path2;
```

BESCHREIBUNG

Die Funktion *link()* erzeugt einen neuen Verweis (Dateiverzeichniseintrag) für die existierende Datei *path1*.

Das Argument *path1* zeigt auf einen Pfadnamen, der eine existierende Datei benennt. Das Argument *path2* zeigt auf einen Pfadnamen, der den neuen, zu erzeugenden Dateiverzeichniseintrag benennt. Die Funktion *link()* erzeugt automatisch einen neuen Verweis für die existierende Datei und der Verweiszähler dieser Datei wird um 1 erhöht.

Wenn *path1* ein Dateiverzeichnis benennt, dann schlägt *link()* fehl.

Bei erfolgreicher Beendigung markiert die Funktion *link()* das Feld *st_ctime* der Datei zum Ändern. Auch die Felder *st_ctime* und *st_mtime* des Dateiverzeichnisses, das den neuen Eintrag enthält, werden zum Ändern markiert.

Wenn die Funktion *link()* fehlschlägt, dann wird kein Verweis erzeugt und der Verweiszähler der Datei bleibt unverändert.

Die Implementierung kann fordern, daß der aufrufende Prozeß die Erlaubnis hat, auf die existierende Datei zuzugreifen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 geliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *link()* schlägt fehl, wenn gilt:

[EACCES] Eine Komponente eines Pfadnamenanfangs verweigert die Durchsucherlaubnis oder der geforderte Verweis erfordert das Schreiben in ein Dateiverzeichnis mit Zugriffsrechten, die die Schreiberlaubnis verweigern. Oder der aufrufende Prozeß besitzt nicht die Erlaubnis, auf die existierende Datei zuzugreifen und dies wird von der aktuellen Implementierung gefordert.

[EEXIST] Der durch *path2* benannte Verweis existiert.

[EMLINK] Die Anzahl der Verweise auf die durch *path1* benannte Datei würde {LINK_MAX} überschreiten.

[ENAMETOOLONG]
Die Länge von *path1* oder *path2* überschreitet {PATH_MAX} oder eine Pfadnamenkomponente ist länger als {NAME_MAX} und {_POSIX_NO_TRUNC} ist aktiv.

[ENOENT] Eine Komponente einer der Pfadnamenanfänge oder die durch *path1* benannte Datei existiert nicht oder *path1* oder *path2* zeigt auf eine leere Zeichenkette.

[ENOSPC] Das den Verweis enthaltende Dateiverzeichnis kann nicht erweitert werden.

[ENOTDIR] Eine Komponente einer der Pfadnamenanfänge ist kein Dateiverzeichnis.

[EPERM] Die durch *path1* benannte Datei ist ein Dateiverzeichnis und der Prozeß besitzt entweder keine besonderen Rechte oder die Implementierung verbietet es, Verweise auf Dateiverzeichnisse einzurichten.

[EROFS] Der gewünschte Verweis erfordert das Schreiben in einem Dateiverzeichnis auf einem nur zum Lesen eingehängten Dateisystem.

[EXDEV] Der durch *path2* benannte Verweis und die durch *path1* benannte Datei befinden sich auf verschiedenen Dateisystemen.

Unter SINIX V5.22 kann zusätzlich zu den im XPG3 [6] definierten Fehlern auch noch der folgende Fehler auftreten:

[EFAULT] Es wurde eine ungültige Adresse als Argument übergeben.

HINWEIS

Andere X/Open-kompatible Implementierungen erlauben zusätzlich auch Verweise zwischen Dateisystemen bzw. auf Dateiverzeichnisse.

Sehr oft wird die Funktion *link()* dazu verwendet, einen einfachen Mechanismus für das Sperren von Dateien zu gewährleisten. Soll auf eine Datei exklusiv zugegriffen werden, so wird diese Datei dadurch "gesperrt", daß ein Verweis mit einem Standardnamen für diese Datei eingerichtet wird. Existiert

dieser Verweis bereits, dann schlägt *link()* fehl und das Programm darf auf die Datei nicht zugreifen (Siehe BEISPIEL).

PORTABILITÄT

Die Funktion *link()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Ein Programm soll exklusiv die Datei */usr/xyz/daten* ändern. Da das Programm gleichzeitig mehrfach ausgeführt werden darf, muß es zunächst prüfen, ob die Datei nicht bereits bearbeitet wird:

```
#include <stdio.h>

main()
{  FILE *dat;
   unsigned old;

   old = alarm(10);          /* Hoechstens 10 Sekunden auf Sperre wa
                                dann Programmabbruch   */
   /* Versuch, die Datei zu sperren (den Verweis einzurichten): */
   while (link("/usr/xyz/daten","/usr/xyz/daten.lock") != 0)
      sleep(2);
   alarm(old);               /* Alarmuhr zuruechstellen */

   if ((dat=fopen("/usr/xyz/daten", "r+")) == NULL)
      ....   /* Fehlerbehandlung
   else
   {
      ...     /* Bearbeiten der Datei */
      fclose(dat);
   }

   /* Datei wieder freigeben: */
   unlink("/usr/xyz/daten.lock");
}
```

Diese Methode, Dateien zu "sperren", hat einen entscheidenden Nachteil gegenüber dem Sperren mit *fcntl()*. Wird das oben beschriebene Programm durch ein Signal unterbrochen, während es die Datei bearbeitet, dann bleibt der Verweis *daten.lock* auch nach dem Abbruch des Programms bestehen. Dieser Verweis muß dann von Hand gelöscht werden. Dieses Problem kann nicht völlig umgangen werden, da es auch Signale gibt, die nicht abgefangen werden können. Die Methode eignet sich daher nur für einfache Sperren bei der Entwicklung eines neuen Programms, nicht jedoch für endgültige Anwendungen. Anwendungen sollten die Funktion *fcntl()* zum Sperren von Dateien verwenden.

SIEHE AUCH

unlink().

NAME

loc1, loc2 - pointers to characters matched by regular-expression
Zeiger auf Zeichen, die zu einem regulären Ausdruck passen

DEFINITION

```
extern char *loc1;
extern char *loc2;
```

BESCHREIBUNG

Siehe unter *regexp()*.

PORTABILITÄT

Die Variablen *loc1* und *loc2* sind im X/Open-Standard (Ausgabe 3) definiert.

NAME

localtime - convert time value to broken-down local time
Zeitwert in aufgeschlüsselte lokale Zeit umwandeln

DEFINITION

```
#include <time.h>

struct tm *localtime (timer)
time_t *timer;
```

BESCHREIBUNG

Die Funktion *localtime()* wandelt die Zeit in Sekunden seit dem Epochenwert, auf die *timer* zeigt, in eine aufgeschlüsselte Zeit um, die als lokale Zeit ausgedrückt wird. Die Funktion korrigiert das Ergebnis ewntsprechend der Zeitzone und eventuellen saisonabhängigen Anpassungen. Die Zeitzonen-Information wird benutzt, als ob *localtime()* die Funktion *tzset()* aufrufen würde.

ERGEBNIS

Die Funktion *localtime()* liefert einen Zeiger auf die aufgeschlüsselte Zeitstruktur.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Ergebnis zeigt auf einen statischen Bereich, der von einem nachfolgenden Aufruf von *ctime()*, *gmtime()* oder *localtime()* überschrieben wird.

PORTABILITÄT

Die Funktion *localtime()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispiel verdeutlicht die Zusammenarbeit von *asctime()* mit *localtime()* und *gmtime()*.

```
#include <time.h>

long time();
char *asctime();
struct tm *gmtime();
struct tm *localtime();
struct tm *zeit, *breit;
char *daten, *braten;
long clock;

main()
{
   int i;
   clock = time(0L);
   zeit = localtime(&clock);
   daten = asctime(zeit);

   printf("\nlocaltime:%d, %d, %d, %d, %d, %d, %d, %d, %d\n",
      zeit->tm_sec, zeit->tm_min, zeit->tm_hour,
      zeit->tm_mday, zeit->tm_mon, zeit->tm_year,
      zeit->tm_wday, zeit->tm_yday, zeit->tm_isdst);
   printf("\nasctime(localtime):%s\n", daten);

   breit= gmtime(&clock);
   braten= asctime(breit);
   printf("\ngmtime:%d, %d, %d, %d, %d, %d, %d, %d, %d\n,
      (*breit).tm_sec, (*breit).tm_min, (*breit).tm_hour,
      (*breit).tm_mday, (*breit).tm_mon, (*breit).tm_year,
      (*breit).tm_wday, (*breit).tm_yday, (*breit).tm_isdst);
   printf("\nasctime(gmtime):%s\n", braten);
}
```

SIEHE AUCH

asctime(), *ctime()*, *gmtime()*, *tzset()*, *<time.h>*.

NAME

lockf - record locking on files
Dateibereiche sperren

DEFINITION

```
#include <unistd.h>

int lockf (datkz, funktion, gr)
long gr;
int datkz, funktion;
```

BESCHREIBUNG

Mit *lockf()* kann man einzelne Abschnitte einer Datei sperren. Versucht ein anderer Prozeß, mittels eines *lockf()*-Aufrufs einen bereits gesperrten Dateibereich zu sperren, so wird ihm entweder ein Fehler angezeigt oder er wird so lange angehalten, bis die erste Sperre aufgehoben wird. Bei Beendigung eines Prozesses werden alle von ihm gesetzten Sperren aufgehoben. (Bei *fcntl()* sind nähere Einzelheiten bzgl. Sperren von Dateibereichen beschrieben.)

Das Argument *datkz* ist die Dateikennzahl einer geöffneten Datei. Das dazugehörige Datei-Statusbyte muß entweder auf O_WRONLY oder auf O_RDWR gesetzt sein, wenn mit diesem Aufruf eine Dateisperre gesetzt werden soll.

Mit dem Argument *funktion* gibt man die durchzuführende Aktion an. Die für *funktion* zulässigen Werte sind in *<unistd.h>* wie folgt definiert:

```
F_ULOCK 0 /* vorher gesperrten Dateibereich freigeben */
F_LOCK 1  /* Dateibereich für fremde Schreib/Lesezugriffe sperren
F_TLOCK 2 /* Datcibereich ueberpruefen und für fremde */
              /* Schreib/Lesezugriffe sperren */
F_TEST 3  /* Dateibereich auf fremde Sperren ueberpruefen */
```

Andere Angaben für *funktion* sind für spätere Erweiterungen reserviert; solange diese noch nicht implementiert sind, haben sie eine Fehlermeldung zur Folge.

Mit F_TEST wird festgestellt, ob der angegebene Dateibereich bereits von einem anderen Prozeß gesperrt wurde. Ist dies nicht der Fall, so wird mit F_LOCK oder F_TLOCK eine Sperre gesetzt. Mit F_ULOCK wird eine Sperre wieder aufgehoben.

gr gibt die Größe des zu sperrenden oder freizugebenden Abschnitts als Anzahl aufeinanderfolgender Bytes an.

Die Sperre beginnt dann an der aktuellen Position in der Datei und gilt für die entsprechende Anzahl nachfolgender Bytes (bei einem positiven Wert *gr*) oder vorausgehender Bytes (bei einem negativen Wert *gr*), wobei das aktuelle Byte nicht miteingeschlossen ist. Ist für *gr* 0 angegeben, so wird der Dateibereich ab der aktuellen Position bis einschließlich {FCHR_MAX} gesperrt (d.h. von der aktuellen Position bis zum aktuellen oder auch späteren Dateiende). Auch ein Bereich, der der Datei nicht zugewiesen ist, kann gesperrt werden, da sich eine Sperre auch über das Dateiende hinaus erstrecken kann.

Ein mit F_LOCK oder F_TLOCK gesperrter Dateiabschnitt kann ganz oder teilweise in einem bereits von demselben Prozeß gesperrten Bereich enthalten sein oder einen solchen enthalten. In diesem Fall, ebenso wie in Fällen, wo benachbarte Dateibereiche gesperrt werden sollen, wird aus den einzelnen Abschnitten ein einzelner gesperrter Bereich gebildet. Soll mit einem solchen Aufruf ein neuer Eintrag in der Tabelle der gesetzten Sperren erfolgen, und dies würde zu einem Überlauf der Tabelle führen, so wird eine Fehlermeldung abgesetzt und die angeforderte neue Sperre nicht gesetzt.

F_LOCK und F_TLOCK unterscheiden sich nur in der Reaktion, wenn der angegebene Dateiabschnitt bereits gesperrt ist. Bei F_LOCK wird der aufrufende Prozeß angehalten, bis die existierende Sperre aufgehoben wird. Bei F_TLOCK wird −1 zurückgegeben und *errno* auf [EACCES] oder [EAGAIN] gesetzt, wenn der angegebene Abschnitt bereits durch einen anderen Prozeß gesperrt ist.

Mit F_ULOCK kann ein Prozeß ein oder mehrere von ihm gesperrte Dateibereiche ganz oder auch teilweise freigeben. Dabei wird die Sperre ab der aktuellen Position für die mit *gr* angegebene Anzahl Bytes oder bis Dateiende (wenn für *gr* 0 angegeben ist) aufgehoben. Wird ein gesperrter Bereich nicht vollständig freigegeben, so gilt die Sperre weiterhin für die übrigen Teile. Wenn man z.B. den mittleren Teil eines gesperrten Dateiabschnitts freigibt, so bleiben die Teile davor und danach weiterhin gesperrt. In diesem Fall muß ein zusätzlicher Eintrag in der Tabelle der gesetzten Sperren erfolgen. Würde dies zu einem Überlauf führen, so wird −1 zurückgegeben, *errno* wird auf [EDEADLK] gesetzt, und der angegebene Teilabschnitt wird nicht freigegeben.

Zu "Verklemmungen" kann es kommen, wenn ein Prozeß, der eine Sperre gesetzt hat, angehalten wird, weil er auf die Freigabe eines von einem anderen Prozeß gesperrten Bereichs wartet. Aus diesem Grunde prüfen *lockf()* und *fcntl()*, bevor sie einen Prozeß anhalten, ob dies zu einer "Verklemmung" führen könnte. Besteht diese Gefahr, so wird eine Fehlermeldung abgesetzt.

Ein Prozeß, der auf die Freigabe eines Bereichs wartet, kann mit jedem beliebigen Signal "aufgeweckt" werden. Die Funktion *alarm()* bietet eine Zeitüberwachung für Anwendungen, die dies benötigen.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird −1 zurückgeliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *lockf()* schlägt fehl, wenn gilt:

[EACCES]
[EAGAIN]
Je nach Implementierung wird einer dieser beiden Fehler gemeldet, wenn für *funktion* F_TLOCK oder F_TEST angegeben ist und der Bereich bereits von einem anderen Prozeß gesperrt wurde.

[EBADF]
datkz ist keine gültige Dateikennzahl.

[EDEADLK]
Für *funktion* ist F_LOCK angegeben, und die Ausführung würde zu einer "Verklemmung" führen. Oder für *funktion* ist F_LOCK, F_TLOCK oder F_ULOCK angegeben, und die Ausführung würde zu einem Überlauf der Systemtabelle für Sperren führen (die Anzahl der Einträge würde {LOCK_MAX} überschreiten).

HINWEIS

Datei- oder Dateiabschnittssperren sollten nicht in Verbindung mit den Funktionen *fopen()*, *fread()*, *fwrite()* usw. benutzt werden. Stattdessen sollten dann einfachere, ungepufferte Funktionen, z.B. *open()*, verwendet werden. In Prozessen, die den Benutzeradreßraum als Puffer verwenden, kann es zu unerwarteten Ergebnissen kommen: es kann vorkommen, daß der Prozeß Daten liest/schreibt, die gesperrt sind/waren. In den meisten Fällen sind diese Funktionen die Ursache von unerwarteten Pufferinhalten.

PORTABILITÄT

Die Funktion *flock()* ist im X/Open-Standard nicht mehr enthalten. Anwendungen sollten für das Sperren von Dateibereichen die Funktion *fcntl()* verwenden.

SIEHE AUCH

chmod(), *close()*, *creat()*, *fcntl()*, *open()*, *read()*, *write()*, *<unistd.h>*.

NAME

locs - stop regular expression matching in a string
Passenden regulären Ausdruck in Zeichenkette anhalten

DEFINITION

```
extern char *locs;
```

BESCHREIBUNG

Siehe unter *regexp()*.

PORTABILITÄT

Die Variable *locs* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

log - natural logarithm function
Natürlicher Logarithmus

DEFINITION

```
#include <math.h>

double log (x)
double x;
```

BESCHREIBUNG

Die Funktion *log()* berechnet den natürlichen Logarithmus von x, $\log_e(x)$. Der Wert von x muß positiv sein.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *log()* den natürlichen Logarithmus von x.

Wenn x ein NaN ist, dann wird ein NaN zurückgeliefert.

Andernfalls wird entweder -HUGE_VAL zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen, oder ein NaN wird zurückgeliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktion *log()* schlägt fehl, wenn gilt:

[ERANGE] Der Wert von x ist gleich 0 oder der Logarithmus von x kann nicht dargestellt werden oder das Ergebnis würde einen Überlauf verursachen.

Die Funktion *log()* kann unter anderen X/Openkompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von x ist negativ oder gleich 0.

HINWEIS

Wenn *x* sehr nahe bei 0 ist, dann können Rundungsfehler im Algorithmus bedeuten, daß der Logarithmus nicht berechnet werden kann.

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *log()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis gleich -HUGE_VAL oder ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

```
cc programm.c -lm
```

PORTABILITÄT

Die Funktion *log()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

exp(), *isnan()*, *log10()*, *matherr()*, *<math.h>*.

NAME

log10 - base 10 logarithm function
Logarithmus zur Basis 10

DEFINITION

```
#include <math.h>

double log10 (x)
double x;
```

BESCHREIBUNG

Die Funktion *log10()* berechnet den Logarithmus zur Basis 10 von x, $\log_{10}(x)$. Der Wert von x muß positiv sein.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *log10()* den Logarithmus von x.

Wenn x ein NaN ist, dann wird ein NaN zurückgeliefert.

Andernfalls wird entweder -HUGE_VAL geliefert und *errno* ist gesetzt, um den Fehler anzuzeigen oder ein NaN wird zurückgeliefert und *errno* kann gesetzt sein um den Fehler anzuzeigen.

FEHLER

Die Funktion *log10()* kann fehlschlagen, wenn gilt:

[ERANGE] Der Logarithmus von x kann nicht dargestellt werden oder das Ergebnis würde einen Überlauf verursachen.

Die Funktion *log10()* kann unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von x ist nicht größer als 0.

[ERANGE] Der Wert von x ist gleich 0.

HINWEIS

Wenn *x* sehr nahe bei 0 liegt, dann können Rundungsfehler im Algorithmus bedeuten, daß der Logarithmus nicht berechnet werden kann.

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *log10()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *log10()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

isnan(), *log()*, *matherr()*, *pow()*, *<math.h>*.

NAME

logname - return login name of user
Benutzerkennung ermitteln

DEFINITION

```
char *logname( )
```

BESCHREIBUNG

Die Funktion *logname()* gibt einen Zeiger auf die mit dem Nullbyte abgeschlossene Benutzerkennung zurück. Sie ermittelt die Benutzerkennung aus der Umgebungsvariablen LOGNAME des Benutzers.

HINWEIS

Der zurückgegebene Zeiger verweist auf einen statischen Datenbereich, der bei jedem Aufruf überschrieben wird.

PORTABILITÄT

Die Funktion *logname()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

<environ.h>.

NAME

longjmp - non-local goto
Nichtlokaler Sprung

DEFINITION

```
#include <setjmp.h>

void longjmp (env, val)
jmp_buf env;
int val;
```

BESCHREIBUNG

Die Funktion *longjmp()* stellt die Umgebung wieder her, die vom letzten Aufruf des Makros *setjmp()* im selben Prozeß mit dem entsprechenden Argument *jmp_buf* gesichert worden ist. Wenn es keinen solchen Aufruf gab, oder wenn die Funktion, die den Aufruf des Makros *setjmp()* enthält, inzwischen ihre Ausführung beendet hat, dann ist das Verhalten undefiniert.

Alle zugreifbaren Objekte besitzen diejenigen Werte, die sie zu dem Zeitpunkt des Aufrufs von *longjmp()* besaßen, mit Ausnahme der Werte von automatischen Objekten, die lokal zu der Funktion sind, die den entsprechenden Aufruf des Makros *setjmp()*enthält, und die zwischen dem Aufruf von *setjmp()* und dem von *longjmp()* geändert wurden. Diese sind unbestimmt.

Da sie den üblichen Funktionsaufruf- und Rückkehrmechanismus umgeht, arbeitet die Funktion *longjmp()* im Zusammenhang mit Unterbrechungen, Signalen und den zugehörigen Funktionen korrekt. Trotzdem ist das Verhalten undefiniert, wenn die Funktion von einer geschachtelten Signalbehandlungsfunktion aus aufgerufen wird (d.h. von einer Funktion aus, die als Ergebnis eines Signals während der Behandlung eines anderen Signals aufgerufen wurde).

ERGEBNIS

Nach der Beendigung von *longjmp()* setzt die Programmausführung fort, als wenn der entsprechende Aufruf des Makros *setjmp()* soeben den durch *val* angegebenen Wert geliefert hätte. Die Funktion *longjmp()* kann das Makro *setjmp()* nicht veranlassen, den Wert 0 zurückzuliefern; wenn *val* gleich 0 ist, dann liefert das Makro *setjmp()* den Wert -1.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Wenn *longjmp()* aufgerufen wird, obwohl *env* niemals durch einen Aufruf von *setjmp()* initialisiert worden ist, oder wenn der letzte solche Aufruf in einer Funktion stattgefunden hat, die bereits zurückgekehrt ist, dann ist das Verhalten undefiniert.

Wenn sich der Aufruf von *longjmp()* in einer anderen Funktion befindet, als der entsprechende Aufruf von *setjmp()*, dann können alle Variablen der Speicherklasse *register* unvorherschbare Werte haben.

PORTABILITÄT

Die Funktion *longjmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Einen typischen Anwendungsfall für *setjmp()* und *longjmp* kann man sich in einem interaktiven Texteditor vorstellen, der Textausgabe durchführt.
Wird während der Ausgabe das Programm unterbrochen, indem von außen ein Signal geschickt wird (z.B. Drücken der Taste DEL), dann bedeutet dies, daß die Textausgabe beendet wird, aber ansonsten der Texteditor weitermachen soll.
Wie das mit *setjmp()* und *longjmp()* realisiert werden kann, sehen Sie in folgendem Programm (nur Signalbehandlung kein Editor!):

```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>

FILE *dz;
jmp_buf zst;

interrupt()
{
   signal(SIGINT, SIG_IGN);
   printf ("\n ***** kein Text? ***** \n");
   longjmp(zst,5);
}


main()
{
   int i;
   int c;
   char antwort;

   printf("_JBLEN: %d", _JBLEN);
   i = setjmp(zst);
   printf("i %d\n", i);

   signal(SIGINT, interrupt);
   i = setjmp(zst);  /*ohne diese Zeile ruft DEL immer interrupt a
                       mit dieser Zeile nur einmal, dann Abbruch */
   signal(SIGINT, interrupt);
   printf("im %d\n",i);
   printf("Textausgabe? (j|n): \n");
   scanf("%1s", &antwort);
   if (antwort == 'j')
   {
      dz = fopen("datei","r");
      while ((c=getc(dz)) != EOF)
       putc(c,stdout);
   }
   else
      printf("dann eben nicht\n");
   exit(0);
}
```

SIEHE AUCH

setjmp(), *sigaction()*, *siglongjmp()*, *sigsetjmp()*, *<setjmp.h>*.

NAME

lrand48 - generate uniformly distributed pseudo-random non-negative long integers
Gleichverteilte, nichtnegative Pseudo-Zufallszahlen vom Typ long erzeugen

DEFINITION

```
long lrand48 ( )
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *lrand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

lsearch, lfind - linear search und update
Lineare Suche und Änderung

DEFINITION

```
#include <search.h>

void *lsearch (key, base, nelp, width, compar)
void *base, *key;
size_t width, *nelp;
int (*compar)();

void *lfind (key, base, nelp, width, compar)
void *base, *key;
size_t width, *nelp;
int (*compar)();
```

BESCHREIBUNG

Die Funktion *lsearch()* ist eine Routine für die lineare Suche. Sie liefert einen Zeiger auf eine Tabelle, der angibt, wo ein Eintrag gefunden werden kann. Wenn der Eintrag nicht existiert, dann wird er am Ende der Tabelle angefügt. Das Argument *key* zeigt auf den Eintrag, der in der Tabelle gesucht werden soll. Das Argument *base* zeigt auf das erste Element der Tabelle. Das Argument *width* entspricht der Größe eines Element in Bytes. Das Argument *nelp* zeigt auf eine ganze Zahl, die die aktuelle Anzahl der Elemente der Tabelle enthält. Die Zahl, auf die *nelp* zeigt, wird um 1 erhöht, wenn der Eintrag zur Tabelle hinzugefügt wird. Das Argument *compar* ist der Name einer Vergleichsfunktion, die der Benutzer bereitstellen muß (z.B. die Funktion *strcmp()*). Diese wird mit zwei Argumenten aufgerufen, die auf zu vergleichende Elemente zeigen. Die Funktion muß den Wert 0 liefern wenn die Elemente gleich sind, andernfalls einen Wert ungleich 0.

Die Funktion *lfind()* arbeitet genauso wie die Funktion *lsearch()*, außer daß ein Eintrag, der nicht gefunden wird, nicht zur Tabelle hinzugefügt wird. Statt dessen wird der Nullzeiger geliefert.

ERGEBNIS

Wenn der gesuchte Eintrag gefunden wird, liefern sowohl *lsearch()* als auch *lfind()* einen Zeiger auf dieses Element. Andernfalls liefert *lfind()* den Nullzeiger und *lsearch()* liefert einen Zeiger auf ein neu hinzugefügtes Element.

Beide Funktionen liefern im Fehlerfall den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Dieses Programmstück liest höchstens TABSIZE Zeichenketten, jeweils von einer Länge von höchstens ELSIZE Zeichen ein und speichert diese in der Tabelle, wobei Duplikate vermieden werden.

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

#define TABSIZE 50
#define ELSIZE 120

   . . .
   char line[ELSIZE], tab[TABSIZE][ELSIZE];
   size_t nel = 0;
   . . .
   while (fgets(line, ELSIZE, stdin) != NULL &&
      nel < TABSIZE)
           (void) lsearch((void *) line, (void *)tab, &nel,
                     ELSIZE, strcmp);
   . . .
```

HINWEIS

Die Zeiger auf das Suchelement und auf das Element am Anfang der Tabelle sollten vom Typ *Zeiger auf Element* sein und in den Typ *void ** umgewandelt werden.

Die Vergleichsfunktion muß nicht jedes Byte vergleichen, so daß zusätzlich zu den verglichenen Werten weitere Daten in den Elementen enthalten sein können.

Obwohl der Typ des Ergebnisses als *void ** definiert ist, sollte er in den Typ *Zeiger auf Element* umgewandelt werden.

Wenn die Tabelle nicht genügend Platz bereitstellt, um ein neues Element hinzuzufügen, so können undefinierte Ergebnisse auftreten.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

bsearch(), *hsearch()*, *tsearch()*, *<search.h>*.

NAME

lseek - move read/write file offset
Dateiposition ändern

DEFINITION

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

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

BESCHREIBUNG

Die Funktion *lseek()* setzt die Dateiposition für die offene Dateibeschreibung, die der Dateikennzahl *fildes* zugeordnet ist so, wie nachfolgend beschrieben:

Wenn *whence* gleich SEEK_SET ist, dann wird die Position in der Datei gleich *offset* Bytes gesetzt.

Wenn *whence* gleich SEEK_CUR ist, dann wird die Position in der Datei gleich aktuelle Position + *offset* gesetzt.

Wenn *whence* gleich SEEK_END ist, dann wird die Position in der Datei gleich der Dateigröße + *offset* gesetzt.

Die symbolischen Konstanten SEEK_SET, SEEK_CUR und SEEK_END sind in der Include-Datei <*unistd.h*> definiert.

Das Verhalten von *lseek()* für Geräte, die nicht in der Lage sind, wahlfrei zu positionieren, ist implementierungs- und gerätebhängig. Der Wert der Dateiposition, der solch einem Gerät zugeordnet ist, ist undefiniert.

Die Funktion *lseek()* erlaubt es, die Dateiposition hinter das Ende der existierenden Daten in der Datei zu setzen. Werden später Daten an diese Position geschrieben, so liefern nachfolgende Leseoperationen in der Lücke Nullbytes, bis wirklich Daten in diese Lücke geschrieben werden.

Die Funktion *lseek()* vergrößert nicht von sich aus die Größe einer Datei.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion die neue Dateiposition, gemessen in Bytes vom Anfang der Datei. Andernfalls wird der Wert ((*off_t*) -1) zurückgeliefert, *errno* wird besetzt, um den Fehler anzuzeigen und die Dateiposition bleibt unverändert.

FEHLER

Die Funktion *lseek()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine offene Dateikennzahl.

[EINVAL] Das Argument *whence* besitzt keinen erlaubten Wert, oder die sich ergebende Dateiposition wäre nicht zulässig.

[ESPIPE] Das Argument *fildes* ist einer Pipe oder FIFO zugeordnet.

HINWEIS

Auf anderen X/Open-kompatiblen Systemen, deren Definition von *off_t* es unmöglich macht, negative Werte zu erzeugen (z.B. *unsigned long*), kann es sein, daß es nicht möglich ist, von der aktuellen Position aus rückwärts zu positionieren, außer durch die Angabe der Werts SEEK_SET für *whence*. Unter SINIX ist *off_t* als *int* definiert, so daß dieses Problem hier nicht auftreten kann.

PORTABILITÄT

Die Funktion *lseek()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Programm liest ab Positon 10 aus der Datei, die als erstes Argument beim Aufruf übergeben wird und fügt den Inhalt, falls ein zweites Argument angegeben wurde, an den Inhalt der Datei an, ansonsten schreibt es auf Standardausgabe:

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

int dk1,dk2;
char c;

main(argc, argv)
int argc;
char **argv;
{
   if (!strcmp(argv[1], argv[2]))
   {
      printf("Bitte zwei verschiedene Dateien angeben!\n");
      exit(1);
   }

   if ((dk1 = open(argv[1],0)) < 0)
      exit(1);

   if (argc < 3)
      dk2 =1;
   else
      dk2 = creat(argv[2],00777);

   lseek(dk1,(off_t) 10,SEEK_SET);
   printf("aktuelle Position in Datei1 :
      %ld\n",lseek(dk1,(off_t) 0,SEEK_CUR));
   lseek(dk2,(off_t) 0,SEEK_END);

   while(read(dk1,&c,1) > 0)
      write(dk2,&c,1);

   close(dk1); close(dk2);
}
```

SIEHE AUCH

open(), *<sys/types.h>*, *<unistd.h>*.

NAME

ltol3 - long int in 3-Byte-Ganzzahl umwandeln

DEFINITION

```
void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

BESCHREIBUNG

ltol3 wandelt eine Liste von *n* long-Werten, die in einem durch *lp* referenzierten Vektor abgelegt sind, in eine Liste von *n* 3 Byte langen Ganzzahlen um, die in einem durch *cp* referenzierten char-Vektor gespeichert werden.

Diese Funktion und ihre Umkehrfunktion *l3tol()* sind für die Verwaltung von Dateisystemen nützlich, wo Plattenadressen drei Byte lang sind.

HINWEIS

Aufgrund möglicher Unterschiede in der Byteanordnung sind die numerischen Werte jeweils maschinenabhängig.

PORTABILITÄT

Die Funktion *ltol3()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

l3tol().

NAME

mallinfo - Speicherplatzauslastung ermitteln

DEFINITION

```
#include <malloc.h>

struct mallinfo mallinfo ()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie Speicherplatz reservieren können.

Die Funktion *mallinfo()* informiert über die aktuelle Speicherplatzauslastung. Sie liefert als Ergebnis eine Struktur des Typs *mallinfo*, die aus folgenden Komponenten besteht:

```
struct mallinfo {
    int arena;      /* Speicherplatz insgesamt im Bereich */
    int ordblks;    /* Anzahl normaler Bloecke */
    int smblks;     /* Anzahl kleiner Bloecke */
    int hblks;      /* Anzahl grosser Bloecke */
    int hblkhd;     /* Platz in Koepfen der grossen Bloecke */
    int usmblks;    /* Platz in benutzten kleinen Bloecken */
    int fsmblks;    /* Platz in freien kleinen Bloecken */
    int uordblks;   /* Platz in benutzten normalen Bloecken */
    int fordblks;   /* Platz in freien normalen Bloecken */
    int keepcost;   /* Aufwand fuer Verwendung von M_KEEP */
    };
```

Auch diese Struktur ist in <malloc.h> definiert.

ERGEBNIS

Die Funktion *mallinfo()* liefert als Ergebnis eine Struktur des Typs *mallinfo*, die die gewünschten Informationen enthält.

HINWEIS

Die Standard-C-Bibliothek enthält eine Version von *malloc()* mit geringerem Funktionsumfang - sie enthält nur die Funktionen *malloc()*, *free()*, *realloc()* und *calloc()*. Die Funktion *mallinfo()* wird in der Bibliothek *libmalloc* zur Verfügung gestellt. Um diese Bibliothek zu Ihrem Programm hinzuzubinden müssen Sie beim Übersetzen den Schalter *-lmalloc* angeben:

```
cc prog.c -lmalloc
```

mallinfo() sollte nur nach erfolgreicher Speicherplatzreservierung aufgerufen werden.

PORTABILITÄT

Die Funktion *mallinfo()* ist im X/Open-Standard nicht mehr enthalten, da einige Implementierungen andere Algorithmen für die Reservierung von Speicherplatz verwenden.

SIEHE AUCH

malloc(), *<malloc.h>*

NAME

malloc - memory allocator
Speicher reservieren

DEFINITION

```
#include <stdlib.h>

void *malloc (size)
size_t size;
```

BESCHREIBUNG

Die Funktion *malloc()* reserviert Speicherplatz für ein Objekt, dessen Größe in Bytes durch *size* angegeben wird, und dessen Wert unbestimmt ist.

Die Reihenfolge in der durch aufeinanderfolgende Aufrufe der Funktion *malloc()* Speicher reserviert wird, ist unbestimmt. Ebenso ist es unbestimmt, ob aufeinanderfolgende Aufrufe von *malloc()* dicht zusammen liegenden Speicher reservieren. Der zurückgelieferte Zeiger bei einer erfolgreichen Reservierung ist so beschaffen, daß er an einen Zeiger auf jedes beliebige Objekt zugewiesen und dann für den Zugriff auf dieses Objekt am reservierten Platz verwendet werden kann (bis der Platz explizit freigegeben oder verändert wird). Jede solche Reservierung liefert einen Zeiger auf ein von jedem anderen Objekt verschiedenes Objekt. Der zurückgelieferte Zeiger zeigt auf den Anfang (niedrigste Byteadresse) des reservierten Platzes.

Wenn der Platz nicht reserviert werden kann, dann wird der Nullzeiger zurückgeliefert. Wenn die Größe des zu reservierenden Platzes gleich 0 ist, dann ist der zurückgelieferte Wert der Nullzeiger. Andere X/Open-kompatible Systeme können dafür auch einen anderen eindeutigen Zeiger vorsehen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *malloc()* einen Zeiger auf den reservierten Speicherplatz. Andernfalls liefert sie den Nullzeiger und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *malloc()* schlägt fehl, wenn gilt:

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

HINWEIS

Aus Gründen der Rückwärtskompatibilität mit Ausgabe 2 erlauben SINIX und andere X/Open-kompatible Systeme die Einbindung von *<malloc.h>* anstelle von *<stdlib.h>*.

Die Verwendung von *<malloc.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *malloc()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Zwei Beispiele, sollen die dynamische Speicherplatzreservierung zeigen, wobei das zweite Beispiel etwas anspruchsvoller ist. Das erste Beispiel zeigt die dynamische Speicherplatzreservierung in einem Programm für den Kauf eines Gebrauchtwagens:

```
#include <stdio.h>
#include <stdlib.h>

struct pkw
{
   char   *typ;
   int    alter;
   long   kilometer;
   char   tuev[8];
   int    zust;
   int    preis;
   struct pkw  *n;
} *list;

main()
{
   int mark;

   if((list = (struct pkw *)malloc(sizeof(struct pkw))) == NULL)
   {
      printf("Speicherplatz aufgebraucht\n");
      exit(1);
   }    /* ACHTUNG !!
           Beim vorigen malloc()-Aufruf wurde
           fuer die Komponente typ lediglich
           der Platz fuer einen Zeiger (4 Bytes)
           bereitgestellt. Der Platz fuer die
           Typbezeichnung selbst muß noch be-
           schafft werden:  */

   if((list ->typ =calloc(1, 20)) == NULL)
      exit(1);                        /* Fehler */

                                      /* Gebrauchtwagen einlesen */
   scanf("%20s %d",list ->typ, &list ->alter);
   scanf("%ld %6s %d %d",&list ->kilometer, list ->tuev,
      &list ->zust,&list ->preis);
   list ->n = NULL;

                              /* eingelesene Werte ausdrucken */
   printf("%s\n%d\n",list ->typ,list ->alter);
   printf("%ld\n%.6s\n%d\n%d",list ->kilometer,list ->tuev,
      list ->zust, list ->preis);

                           /* Speicherplatz wieder freigeben */
   free(list->typ);
   free(list);
}
```

Das zweite Beispiel ist ein baumorientiertes Sortierprogramm und zeigt die Verkettung sowie Baumgenerierung mit *calloc()*, *malloc()*, *free()*, wobei die Vorteile der Speicherplatzreservierung besser genutzt werden.

```
/*              Baumorientiertes Sortierprogramm                        */

#include <stdio.h>
#include <stdlib.h>

struct tree {
   int            knoten ;
   struct tree    *li,*re ;
} ;

main ()
{  int             eing, weiter=1 ;
   struct tree    *top=NULL, *trav_in() ;
   void            trav_out();

   do
   {  printf ("\n Bitte Zahl eingeben (Buchst. oder END fuer Abbruch):") ;
      weiter = scanf ("%d",&eing) ;
      top = trav_in(top,eing) ;
   }
   while (weiter == 1);

   trav_out (top) ;
}
```

```
struct tree * trav_in (point,e)
struct tree * point ;
int           e ;
{
   if (point == NULL)   /* Leerer Baum, d.h. Wurzel == NULL */
   {  if ((point = (struct tree *) malloc (sizeof (struct tree))) == NULL)
      {  fprintf (stderr, "Kein Speicher mehr!\n") ;
         exit (1) ;
      }
      else
      {  point->knoten = e ;
         point->li = point->re = NULL ;
         /* Adresse der neuen Baumwurzel zurueckgeben */
         return point;
      }
   }
   else if (point->knoten == e)
      {  /* Element schon im Baum */
         fprintf(stderr,"Element schon im Baum - kein Eintrag!\n");
         /* Adresse der alten Wurzel zurueckgeben */
         return point;
      }
      else if (point->knoten > e)
      {  /* Neues Element in linken Unterbaum einsortieren */
         point->li = trav_in(point->li, e);
         /* Adresse der alten Wurzel zurueckgeben */
         return point;
      }
      else
      {  /* Neues Element in rechten Unterbaum einsortieren */
         point->re = trav_in(point->re, e);
         /* Adresse der alten Wurzel zurueckgeben */
         return point;
      }
}

void trav_out (point)
struct tree    *point ;
{
   if (point == NULL)
      return ;

   trav_out (point->li);
   printf ("\n%d", point->knoten) ;
   trav_out (point->re);

   free (point) ;
}
```

SIEHE AUCH

calloc(), *free()*, *realloc()*, *<stdlib.h>*.

NAME

mallopt - Algorithmus für Speicherplatzreservierung steuern

DEFINITION

```
#include <malloc.h>

int mallopt (cmd, wert)
int cmd, wert;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie Speicherplatz reservieren können.

Die Funktion *mallopt()* steuert den Algorithmus für die Speicherplatzreservierung. *cmd* kann folgende Werte annehmen:

M_MXFAST
: *maxfast* auf *wert* setzen. Der Algorithmus faßt alle Blöcke, die kleiner als *maxfast* sind, in großen Blöcken zusammen und vergibt diese dann mit hoher Geschwindigkeit. Der Standardwert für *maxfast* ist 0.

M_NLBLKS
: *numlblks* auf *wert* setzen. Jeder der oben erwähnten "großen Blöcke" enthält *numlblks* Blöcke. *numlblks* muß einen Wert größer 1 haben; der Standardwert für *numlblks* ist 100.

M_GRAIN
: *grain* auf *wert* setzen. Bei allen Blockgrößen kleiner *maxfast* wird angenommen, daß sie auf das nächste Vielfache von *grain* aufgerundet werden. *grain* muß einen Wert größer 0 haben; der Standardwert von *grain* ist die kleinste Byteanzahl, an der jeder beliebige Datentyp ausgerichtet werden kann. *wert* wird bei gesetztem *grain* auf ein Vielfaches des Standardwerts aufgerundet.

M_KEEP
: Daten in einem freigegebenen Block bis zum nächsten *malloc()*-, *realloc()*- oder *calloc()*-Aufruf behalten. Dieses Kommando existiert einzig und allein aus Gründen der Kompatibilität mit der alten Version von *malloc()*; seine Verwendung wird nicht empfohlen.

Diese Werte sind in der Datei <*malloc.h*> definiert.

Die Funktion *mallopt()* kann wiederholt aufgerufen werden, jedoch nicht, nachdem der erste kleine Block zugewiesen ist.

ERGEBNIS

Wird *mallopt* mit ungültigem *cmd*- oder *wert*-Argument aufgerufen oder nachdem bereits eine Zuweisung erfolgt ist, so wird ein Wert ungleich 0 zurückgegeben, sonst 0.

HINWEIS

Die Standard-C-Bibliothek enthält eine Version von *malloc()* mit geringerem Funktionsumfang - sie enthält nur die Funktionen *malloc()*, *free()*, *realloc()* und *calloc()*. Die Funktion *mallopt()* wird in der Bibliothek *libmalloc* zur Verfügung gestellt. Um diese Bibliothek zu Ihrem Programm hinzuzubinden müssen Sie beim Übersetzen den Schalter *-lmalloc* angeben:

```
cc prog.c -lmalloc
```

PORTABILITÄT

Die Funktion *mallopt()* ist im X/Open-Standard nicht mehr enthalten, da einige Implementierungen andere Algorithmen für die Reservierung von Speicherplatz verwenden.

SIEHE AUCH

malloc(), *<malloc.h>*

NAME

matherr - Fehlerbehandlungsfunktion

DEFINITION

```
#include <math.h>

int matherr (x)
struct exception *x;
```

BESCHREIBUNG

Die Funktion *matherr* wird von Funktionen aus der mathematischen Bibliothek aufgerufen, wenn Fehler auftreten. Benutzer können ihre eigenen Fehlerbehandlungsroutinen definieren, wenn sie in ihren Programmen eine Funktion mit dem Namen *matherr()* verwenden. *matherr()* muß der oben angeführten Syntax genügen. Tritt nun ein Fehler auf, wird der vom Benutzer erstellten Funktion *matherr()* ein Zeiger *x* auf die Struktur *exception* übergeben. Diese Struktur ist in der Include-Datei *<math.h>* definiert und besitzt folgende Komponenten:

```
int type;
char *name;
double arg1, arg2, retval;
```

Die Komponente *type* ist ein ganzzahliger Wert und beschreibt die Art des aufgetretenen Fehlers durch eine Fehlernummer aus der folgenden Liste (die in der Include-Datei definiert ist):

DOMAIN	Bereichsfehler des Arguments
SING	Singularität des Arguments
OVERFLOW	Fehler durch Überlauf
UNDERFLOW	Fehler durch Unterlauf
TLOSS	vollständiger Signifikanzverlust
PLOSS	teilweiser Signifikanzverlust

Die Komponente *name* zeigt auf eine Zeichenkette, die den Namen der Prozedur enthält, die den Fehler verursacht hat. Die Variablen *arg1* und *arg2* sind die Argumente der Funktion bei ihrem Aufruf. *retval* wird auf den voreingestellten Wert gesetzt, den die Funktion normalerweise zurückgibt, es sei denn, daß die vom Benutzer definierte *matherr()*-Funktion einen anderen Wert zurückgibt.

Falls die benutzerdefinierte *matherr()*-Funktion einen Wert ungleich Null zurückgibt, wird keine Fehlermeldung ausgegeben und *errno* wird nicht gesetzt.

Falls *matherr()* nicht vom Benutzer gestellt wird, wird bei einem auftretenden Fehler die Standardfehlerbehandlungsprozedur aufgerufen. Diese Prozeduren sind in der nachfolgenden Tabelle aufgelistet. In allen Fällen wird *errno* auf [EDOM] oder [ERANGE] gesetzt und der Programmablauf wird fortgesetzt.

STANDARDFEHLERBEHANDLUNGSPROZEDUREN

	Fehlertyp					
Typ	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
Fehlercode in errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE	ERANGE
bessel: y0, y1, yn (arg = 0)	- M, -H	- -	- -	- -	M, 0 -	* *
exp()	-	-	H	0	-	-
log(), log10(): (arg < 0) (arg = 0)	M, -H -	- M, -H	- -	- -	- -	- -
pow(): neg ** non-int 0 ** non-pos	- M, 0	- -	± H -	0 -	- -	- -
sqrt()	M, 0	-	-	-	-	-
gamma()	-	M, H	H	-	-	-
hypot()	-	-	H	-	-	-
sinh()	-	-	± H	-	-	-
cosh()	-	-	H	-	-	-
sin(), cos(), tan()	-	-	-	M, 0	*	
asin(), acos(), atan2()	M, 0	-	-	-	-	-

ABKÜRZUNGEN:

* die größtmögliche Annäherung an den Wert wird zurückgegeben
M Ausgabe einer Fehlermeldung (EDOM-Fehler)
H Es wird HUGE_VAL zurückgegeben
-H Es wird -HUGE_VAL zurückgegeben
± H Es wird HUGE_VAL oder -HUGE_VAL zurückgegeben
0 Es wird 0 zurückgegeben

HINWEIS

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *matherr()* ist im X/OPEN-Standard nicht enthalten.

BEISPIEL

Das folgende Beispiel zeigt, wie eine *matherr()*-Funktion verwendet werden kann, um eine eigene Fehlerbehandlung für die Mathematik-Funktionen einzuführen.

```
#include <math.h>

int matherr(x)
struct exception *x;
{
        switch (x->type) {
        case DOMAIN:
                /* Aendere das Argument, so dass sqrt(-arg1)
                   zurueckgegeben wird, nicht 0 */
                if (!strcmp(x->name, "sqrt")) {
                        x->retval = sqrt(-x->arg1);
                        return (0); /* Fehlermeldung ausgeben und errno setzen */
                }
        case SING:
                /* alle anderen Bereichs- oder Singularitaetsfehler,
                   Meldung ausgeben und abbrechen */
                fprintf(stderr, "domain error in %s\n", x->name);
                abort( );
        case PLOSS:
                /* detaillierte Fehlermeldung ausgeben */
                fprintf(stderr, "loss of significance in %s(%g) = %g\n",
                        x->name, x->arg1, x->retval);
                return (1); /* keine weiteren Aktionen */
        }
        return (0); /* Standardbehandlung bei allen anderen Fehlern */
}
```

NAME

mbtowc, wctomb - multibyte character functions
Operationen auf Zeichen aus mehreren Bytes

DEFINITION

```
#include <stdlib.h>

int mbtowc(pwc,s,n)
wchar_t *pwc;
char *s;
size_t n;

int wctomb(s, wchar)
char *s;
wchar_t wchar;
```

BESCHREIBUNG

Wenn *s* nicht der Nullzeiger ist, dann bestimmt die Funktion *mbtowc()*, aus wievielen Bytes das Zeichen besteht, auf das *s* zeigt. Sie bestimmt den Code für den Wert vom Typ *wchar_t*, der diesem Zeichen entspricht. Bei Erfolg speichert die Funktion *mbtowc()* diesen Wert in dem Objekt ab, auf das *pwc* zeigt. Höchstens *n* Bytes des Vektors *s* werden untersucht.

Die Funktion *wctomb()* ermittelt, wieviele Bytes benötigt werden, um das Zeichen, dessen Wert gleich *wchar* ist, abzuspeichern. Es speichert dieses Zeichen unter der Adresse *s* ab.

Die Bestimmung erfolgt gemäß der internationalen Umgebung des Programms (Kategorie LC_CTYPE), wenn zuvor die Funktion *setlocale()* erfolgreich aufgerufen wurde. Ansonsten richtet sich die Bestimmung nach der Umgebungsvariablen LANG. Ist diese nicht definiert, dann finden die Regeln des US-ASCII-Zeichensatzes Anwendung.

ERGEBNIS

Bei Erfolg liefert die Funktion *mbtowc()* die Anzahl der Bytes, aus denen das Multibyte-Zeichen *s* besteht. Wenn *s* der Nullzeiger ist, dann liefert sie entweder den Wert 0 oder, wenn die Zeichen an der Adresse NULL ein gültiges Multibyte-Zeichen darstellen, die Anzahl der Bytes dieses Multibyte-Zeichens.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) nicht enthalten.

SIEHE AUCH

Abschnitt 2.5.

NAME

memccpy - copy bytes in memory
Bytes im Speicher kopieren

DEFINITION

```
#include <string.h>

void *memccpy (s1, s2, c, n)
void *s1, *s2;
int c;
size_t n;
```

BESCHREIBUNG

Die Funktion *memccpy()* kopiert Bytes aus dem Speicherbereich *s2* in *s1*, wobei sie nach dem Kopieren des ersten Auftretens von Byte *c* (umgewandelt in ein *unsigned char*), oder nach dem Kopieren von *n* Bytes aufhört, je nachdem, was zuerst auftritt. Wenn das Kopieren zwischen überlappenden Bereichen stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *memccpy()* liefert einen Zeiger auf das Byte hinter der Kopie von *c* in *s1*, oder den Nullzeiger, wenn *c* nicht in den ersten *n* Bytes von *s2* gefunden wurde.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Funktion *memccpy()* prüft nicht, ob der Ziel-Speicherbereich überläuft.

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<memory.h>* anstelle von *<string.h>*. Die Verwendung von *<memory.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *memccpy()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

memchr - find byte in memory
Byte im Speicher finden

DEFINITION

```
#include <string.h>

void *memchr (s, c, n)
void *s;
int c;
size_t n;
```

BESCHREIBUNG

Die Funktion *memchr()* ermittelt das erste Auftreten von *c* (umgewandelt in ein *unsigned char*) in den ersten *n* Bytes (jedes interpretiert als *unsigned char*) des Objekts, auf das *s* zeigt.

ERGEBNIS

Die Funktion *memchr()* liefert einen Zeiger auf das gefunden Byte oder den Nullzeiger, wenn das Byte im Objekt nicht auftritt.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<memory.h>* anstelle von *<string.h>*. Die Verwendung von *<memory.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *memchr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

memcmp - compare bytes in memory
Bytes im Speicher vergleichen

DEFINITION

```
#include <string.h>

int memcmp (s1, s2, n)
void *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *memcmp()* vergleicht die ersten *n* Bytes (jedes interpretiert als *unsigned char*) des Objekts, auf das *s1* zeigt, mit den ersten *n* Bytes (jedes interpretiert als *unsigned char*) des Objekts, auf das *s2* zeigt.

ERGEBNIS

Die Funktion *memcmp()* liefert einen Wert größer als, gleich oder kleiner als 0, je nachdem ob das Objekt, auf das *s1* zeigt, größer als das, gleich dem oder kleiner als das Objekt ist, auf das *s2* zeigt.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<memory.h>* anstelle von *<string.h>*. Die Verwendung von *<memory.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *memcmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

memcpy - copy bytes in memory
Bytes im Speicher kopieren

DEFINITION

```
#include <string.h>

void *memcpy (s1, s2, n)
void *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *memcpy()* kopiert *n* Bytes aus dem Objekt, auf das *s2* zeigt in das Objekt, auf das *s1* zeigt. Wenn das Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *memcpy()* den Wert von *s1*.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<memory.h>* anstelle von *<string.h>*. Die Verwendung von *<memory.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *memcpy()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

memset - set bytes in memory
Bytes im Speicher initialisieren

DEFINITION

```
#include <string.h>

void *memset (s, c, n)
void *s;
int c;
size_t n;
```

BESCHREIBUNG

Die Funktion *memset()* kopiert den Wert von *c* (umgewandelt in ein *unsigned char*) in jedes der ersten *n* Bytes des Objekts, auf das *s* zeigt.

ERGEBNIS

Die Funktion *memset()* liefert den Wert von *s*.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<memory.h>* anstelle von *<string.h>*. Die Verwendung von *<memory.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen werden wird.

PORTABILITÄT

Die Funktion *memset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

mkdir - make a directory
Dateiverzeichnis erzeugen

DEFINITION

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

int mkdir (path, mode)
char *path;
mode_t mode;
```

BESCHREIBUNG

Die Funktion *mkdir()* erzeugt ein neues Dateiverzeichnis mit dem Namen *path*. Die Schutzbits des neuen Dateiverzeichnisses werden aus dem Parameter *mode* initialisiert

Diese Schutzbits gemäß dem Argument *mode* werden durch die Schutzbit-Maske des Prozesses verändert (siehe *umask()*).

Die Benutzernummer des Dateiverzeichnisses wird gleich der effektiven Benutzernummer des aufrufenden Prozesses gesetzt. Die Gruppennummer des Dateiverzeichnisses wird entweder gleich der effektiven Gruppennummer des Prozesses oder gleich der Gruppennummer des übergeordneten Dateiverzeichnisses gesetzt.

Das neu erzeugte Dateiverzeichnis ist leer.

Bei erfolgreicher Beendigung markiert die Funktion *mkdir()* die Felder *st_atime*, *st_ctime* und *st_mtime* des Dateiverzeichnisses zum Ändern.

Auch die Felder *st_ctime* und *st_mtime* des Dateiverzeichnisses, das den neuen Eintrag enthält, werden zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *mkdir()* den Wert 0 als Ergebnis. Andernfalls wird der Wert -1 geliefert, kein Dateiverzeichnis erzeugt und die Variable *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *mkdir()* schlägt fehl wenn gilt:

[EACCES] Das Suchrecht für eine Komponente des Pfadnamen-Anfangs oder das Schreibrecht für das übergeordnete Dateiverzeichnis des zu erzeugenden Dateiverzeichnisses ist nicht vorhanden.

[EEXIST] Die benannte Datei exisitiert bereits.

[EMLINK] Der Verweiszähler des übergeordneten Dateiverzeichnisses würde {LINK_MAX} überschreiten.

[ENAMETOOLONG]
Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX} und {_POSIX_NO_TRUNC} ist aktiv.

[ENOENT] Eine Komponente des Pfadnamen-Anfangs existiert nicht oder das Argument *path* zeigt auf die leere Zeichenkette.

[ENOSPC] Das Dateisystem enthält nicht genügend Platz, um den Inhalt des neuen Dateiverzeichnisses aufnehmen zu können oder um das übergeordnete Dateiverzeichnis um das neue Dateiverzeichnis zu erweitern.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

[EROFS] Das übergeordnete Dateiverzeichnis befindet sich in einem nur zum Lesen eingehängten Dateisystem.

PORTABILITÄT

Die Funktion *mkdir()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Wenn in *mode* andere Bits gesetzt sind als die für die Zugriffsrechte der Datei, so ist die Bedeutung dieser zusätzlichen Bits implementierungs-abhängig.

SIEHE AUCH

umask(), *<sys/stat.h>*, *<sys/types.h>*.

NAME

mkfifo - make a FIFO special file
Eine FIFO-Gerätedatei erzeugen

DEFINITION

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

int mkfifo (path, mode)
char *path;
mode_t mode;
```

BESCHREIBUNG

Die Funktion *mkfifo()* erzeugt eine neue Gerätedatei für ein FIFO mit dem Pfadnamen, der durch *path* angegeben wird. Die Zugriffsrechte des neuen FIFO werden durch *mode* initialisiert. Die Schutzbits des Arguments *mode* werden durch die Schutzbit-Maske des Prozesses verändert (siehe auch *umask()*).

Die Benutzernummer des FIFO wird gleich der effektiven Benutzernummer des aufrufenden Prozesses gesetzt. Die Gruppennummer des FIFO wird entweder gleich der effektiven Gruppennummer des Prozesses oder gleich der Gruppennummer des übergeordneten Dateiverzeichnisses gesetzt.

Bei erfolgreicher Beendigung markiert die Funktion *mkfifo()* die Felder *st_atime*, *st_ctime* und *st_mtime* der Datei zum Ändern.

Auch die Felder *st_ctime* und *st_mtime* des Dateiverzeichnisses das den neuen Eintrag enthält, werden zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *mkfifo()* den Wert 0 als Ergebnis. Andernfalls wird der Wert -1 geliefert, keine FIFO-Datei erzeugt und die Variable *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *mkfifo()* schlägt fehl, wenn gilt:

[EACCES] Das Suchrecht für eine Komponente des Pfadnamen-Anfangs ist nicht vorhanden.

[EEXIST] Die angegebene Datei exisitiert bereits.

[ENAMETOOLONG]
Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX} und {_POSIX_NO_TRUNC} ist aktiv.

[ENOENT] Eine Komponente des Pfadnamen-Anfangs existiert nicht oder das Argument *path* zeigt auf die leere Zeichenkette.

[ENOSPC] Das Dateiverzeichnis, das die neue Datei enthalten würde, kann nicht erweitert werden oder das Dateisystem kann keine Dateien mehr reservieren.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

[EROFS] Die angegebene Datei befindet sich in einem nur zum Lesen eingehängten Dateisystem.

PORTABILITÄT

Die Funktion *mkfifo()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Wenn in *mode* andere Bits gesetzt sind als die für die Zugriffsrechte der Datei, so ist die Bedeutung dieser zusätzlichen Bits implementierungs-abhängig.

SIEHE AUCH

umask(), *<sys/stat.h>*, *<sys/types.h>*.

NAME

mknod - make node
FIFO, Dateiverzeichnis, Gerätedatei oder normale Datei einrichten

DEFINITION

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>  /* nur erforderlich,     */
                            /* wenn dev durch        */
                            /* makedev erstellt      */
                            /* werden soll (s.u.)    */

int mknod (pfad, modus, dev)
char *pfad;
int modus;
```

BESCHREIBUNG

Die Funktion *mknod()* legt eine Datei beliebigen Typs an. üblicherweise werden mit *mknod()* Dateiverzeichnisse und FIFO's angelegt. Dazu stehen jedoch die Funktionen *mkdir()* und *mkfifo()* zur Verfügung, die anstelle von *mknod()* verwendet werden sollten.

Das Argument *pfad* gibt den Pfadnamen an, mit dem die neue Datei eingerichtet wird.

Vom Bitmuster, das im Argument *modus* angegeben wird, hängen der Typ der eingerichteten Datei und die Zugriffsrechte ab. Andere als die folgenden Werte für *modus* sind undefiniert und dürfen deshalb nicht verwendet werden.

Der Dateityp wird mit einem der folgenden Werte definiert:

S_IFIFO	0010000	FIFO-Datei
S_IFCHR	0020000	zeichenorientiertes Gerät
S_IFDIR	0040000	Dateiverzeichnis
S_IFBLK	0060000	blockorientiertes Gerät
S_IFREG	0100000	normale Datei
	0000000	normale Datei

Achtung
mknod() darf für alle Dateitypen außer FIFO-Dateien nur unter der effektiven Benutzernummer des Systemverwalters aufgerufen werden.

Die s-Bit-Belegung ergibt sich aus folgenden Werten:

S_ISUID	0004000	s-Bit für Eigentümer
S_ISGID	0002000	s-Bit für Gruppe
	0001000	Sticky-Bit

Die Schutzbit-Belegung ergibt sich aus folgenden Werten:

S_IRWXU	0000700	lesen, schreiben, ausführen/durchsuchen (Eigentümer)
S_IRUSR	0000400	lesen (Eigentümer)
S_IWUSR	0000200	schreiben (Eigentümer)
S_IXUSR	0000100	ausführen (Dateiverzeichnis durchsuchen) (Eigentümer)
S_IRWXG	0000070	lesen, schreiben, ausführen/durchsuchen (Gruppe)
S_IRGRP	0000040	lesen (Gruppe)
S_IWGRP	0000020	schreiben (Gruppe)
S_IXGRP	0000010	ausführen (Dateiverzeichnis durchsuchen) (Gruppe)
S_IRWXO	0000007	lesen, schreiben, ausführen/durchsuchen (Andere)
S_IROTH	0000004	lesen (Andere)
S_IWOTH	0000002	schreiben (Andere)
S_IXOTH	0000001	ausführen (Dateiverzeichnis durchsuchen) (Andere)

Als Dateieigentümer wird die effektive Benutzerkennung des Prozesses eingetragen. Als Gruppennummer der Datei wird die effektive Gruppennummer des Prozesses eingetragen.

Das Argument *dev* ist eine Kombination aus der Major- und der Minor-Devicenummer der Gerätedatei, die neu erstellt werden soll. *dev* wird erstellt durch den Aufruf des Makros *makedev()* das über die Include-Datei <*sys/sysmacros.h*> mit eingebunden wird.

```
dev = makedev(majornr, minornr);
```

Wenn durch die Angabe von *modus* keine Gerätedatei eingerichtet wird, dann wird die Angabe von *dev* ignoriert.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird −1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

mknod() kann nicht ausgeführt werden, d.h. die neue Datei wird nicht eingerichtet, wenn einer der folgenden Fehler auftritt:

[EACCES] Keine Durchsuchberechtigung für eine Komponente des Pfadnamenanfangs.

[EEXIST] Die angegebene Datei existiert bereits.

[EFAULT] *pfad* verweist auf einen unzulässigen Speicherbereich (Zeiger nicht oder falsch versorgt).

[ENOENT] Eine Komponente des Pfadnamen-Anfangs ist unbekannt.

[ENOSPC] Das Dateiverzeichnis, in dem die Datei eingerichtet werden soll, kann nicht erweitert werden.

[ENOTDIR] Eine Komponente von *pfad* ist kein Dateiverzeichnis.

[EPERM] Die effektive Benutzernummer des Prozesses ist nicht die des Systemverwalters und der Dateityp ist nicht FIFO.

[EROFS] Das Dateiverzeichnis, in dem die Datei eingerichtet werden soll, befindet sich in einem schreibgeschützten Dateisystem.

HINWEIS

Im Unterschied zu dem Kommando *mkdir* trägt die Funktion *mknod()* beim Einrichten eines Dateiverzeichnisses die beiden Verweise auf das Verzeichnis selbst und das übergeordnete Dateiverzeichnis nicht ein.
Es ist sehr ratsam, diese Verweise mit *link()* einzurichten, da Sie sonst mit dem neu erzeugten Dateiverzeichnis nicht vernünftig arbeiten können.

PORTABILITÄT

Die Funktion *mknod()* ist im X/Open-Standard nicht mehr enthalten. Anwendungen sollten statt dessen die Funktionen *mkdir()* und *mkfifo()* verwenden.

BEISPIEL

Folgendes Programm kann nur unter der Kennung des Systemverwalters laufen!

Es richtet für die Benutzerin *sissi* ein neues Dateiverzeichnis *frust* ein, wobei Gruppe und Andere das Verzeichnis lesen und durchsuchen dürfen.
Beachten Sie, daß, wie oben erwähnt, mit *link()* Verweise auf das neue Verzeichnis selbst und auf das übergeordnete Dateiverzeichnis eingetragen werden.

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

#define ACCESS (S_IFDIR | S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH)

main()
{
   if (mknod("usr/sissi/frust", ACCESS, 0) == 0)
   {
      printf("korrekt\n");
      link("usr/sissi/frust", "usr/sissi/frust/.");
      link("usr/sissi", "usr/sissi/frust/..");
   }
   else
   {
      perror("mknod");
      printf("Fehler\n");
   }
}
```

SIEHE AUCH

chmod(), *exec*, *mkdir()*, *mkfifo()*, *pipe()*, *stat()*, *umask()*, *<sys/stat.h>*, *<sys/types.h>*.

NAME

mktemp - make a unique filename
Eindeutigen Dateinamen erzeugen

DEFINITION

```
char *mktemp (schablone)
char *schablone;
```

BESCHREIBUNG

mktemp() erzeugt aus der Zeichenkette, auf die *schablone* zeigt, einen eindeutigen Dateinamen und liefert die Adresse von *schablone* zurück. Die Zeichenkette in *schablone* sollte sechs **X** am Ende haben; diese werden von *mktemp()* durch einen Buchstaben und die Prozeßnummer des aktuellen Prozesses ersetzt. Der Buchstabe wird so gewählt, daß der neue Dateiname eindeutig ist, d.h. mit keinem existierenden Dateinamen übereinstimmt.

HINWEIS

Es kann vorkommen, daß keine Buchstaben mehr frei sind; in diesem Fall wird ein Nullzeiger zurückgeliefert.

PORTABILITÄT

Die Funktion *mktemp()* ist im X/Open-Standard nicht mehr enthalten. Anwendungen sollten statt dessen die Funktion *tmpnam()* verwenden.

BEISPIEL

Erzeugen einer Datei mit Dateinamen "*temp/test*prozeßnummer". Anschließend wird die Eingabe von *stdin* in die neue Datei geschrieben:

```
#include <stdio.h>

char  *mktemp();
char  vorlage[] = "temp/testXXXXXX";
FILE  *fp;
int   c;

main()
{
   printf("%s\n",mktemp(vorlage));
                                  /* Anlegen der neuen Datei und
                                     Oeffnen zum Schreiben */

   fp = fopen(vorlage,"w");

   while ((c=getc(stdin))!=EOF)
      putc(c,fp);

   fclose(fp);
   exit(0);
}
```

SIEHE AUCH

getpid(), *tmpfile()*, *tmpnam()*.

NAME

mktime - convert broken-down time into time since the Epoch
aufgeschlüsselte Zeit in Zeit seit Epochenwert umwandeln

DEFINITION

```
#include <time.h>

time_t mktime (timeptr)
struct tm *timeptr;
```

BESCHREIBUNG

Die Funktion *mktime()* konvertiert die Zeit, die als lokale Zeit in der Struktur, auf die *timeptr* zeigt, aufgeschlüsselt dargestellt wird, in die Zeit seit dem Epochenwert. Dabei verwendet sie dieselbe Codierung wie sie von der Funktion *time()* geliefert wird. Die Original-Werte der Komponenten *tm_wday* und *tm_yday* aus der Struktur werden ignoriert und die Original-Werte der anderen Komponenten sind nicht auf die Bereiche, die im Eintrag *<time.h>* beschrieben werden, beschränkt. Der Bereich [0, 61] für *tm_sec* erlaubt die gelegentliche Schalt-Sekunde oder doppelte Schalt-Sekunde.

Die lokale Zeitzonen-Information wird gesetzt, als ob *mktime()* die Funktion *tzset()* aufrufen würde.

Bei erfolgreicher Beendigung werden die Werte der Komponenten *tm_wday* und *tm_yday* aus der Struktur richtig gesetzt und die anderen Komponenten werden so gesetzt, daß sie die angegebene Zeit seit dem Epochenwert darstellen, wobei ihre Werte an die oben erwähnten Bereiche angepaßt werden; der endgültige Wert für *tm_mday* wird nicht gesetzt, bevor *tm_mon* und *tm_year* bestimmt worden sind.

ERGEBNIS

Die Funktion *mktime()* liefert die angegebene Zeit seit dem Epochenwert, codiert als ein Wert vom Typ *time_t*. Falls die Zeit seit dem Epochenwert nicht dargestellt werden kann, liefert die Funktion den Wert (*time_t*) -1.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Welcher Wochentag ist der 4. Juli 2001?

```
#include <stdio.h>
#include <time.h>

static char *wday[] = {
   "Sonntag", "Montag", "Dienstag", "Mittwoch",
   "Donnerstag", "Freitag", "Samstag", "-unbekannt-"
};
   struct tm time_str;

   main(){
      time_str.tm_year = 2001 - 1900;
      time_str.tm_mon = 7 - 1;
      time_str.tm_mday = 4;
      time_str.tm_hour = 0;
      time_str.tm_min = 0;
      time_str.tm_sec = 1;
      time_str.tm_isdst = -1;
      if (mktime(&time_str) == -1)
            time_str.tm_wday = 7;
      printf("%s\n", wday[time_str.tm_wday]);
}
```

PORTABILITÄT

Die Funktion *mktime()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

time(), *tzset()*, *<time.h>*.

NAME

modf - decompose floating-point number
Gleitkommazahl aufspalten

DEFINITION

```
#include <math.h>

double modf (value, iptr)
double value, *iptr;
```

BESCHREIBUNG

Die Funktion *modf()* spaltet das Argument *value* in einen ganzzahligen und einem gebrochenen Anteil, jeden mit demselben Vorzeichen wie das Argument. Sie legt den ganzzahligen Anteil als ein *double* in dem Objekt ab, auf das *iptr* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *modf()* den gebrochenen Anteil von *value*.

Wenn *value* ein NaN ist, dann wird ein NaN zurückgeliefert.

Anderfalls kann *errno* gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktion *modf()* kann unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von *value* ist ein NaN.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *modf()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *modf()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Zerlegung der Zahl -456.789:

```
#include <math.h>

main()
{  double x, g;

   x = modf(-456.789,&g);
   printf("Bruchteil: %g \nGanzteil: %g\n",x,g);
}
```

SIEHE AUCH

frexp(), *isnan()*, *ldexp()*, *matherr()*, *<math.h>*.

NAME

mount - mount file system
Dateisystem einhängen

DEFINITION

```
int mount (ger, dvz, modus)
char *ger, *dvz;
int modus;
```

BESCHREIBUNG

Mit *mount()* können Sie ein Dateisystem in ein bestehendes Dateiverzeichnis ein Dateisystem einhängen, das sich auf einem blockorientierten Datenträger befindet.

Nur für den Systemverwalter!

Sobald das Dateisystem eingehängt ist, wird es wie ein Unterbaum behandelt. Prozesse können nun auf Dateien im eingehängten Dateisystem zugreifen, ohne berücksichtigen zu müssen, daß es ein eingehängtes Dateisystem ist. Lediglich Verweise über Dateisystemgrenzen durch *link()* sind nicht gestattet, da diese Funktion das Dateisystem einer Datei überprüft.

mount() darf nur unter der effektiven Benutzernummer eines Prozesses mit besonderen Rechten aufgerufen werden (Systemverwalter).

Mit dem Argument *ger* geben Sie den Pfadnamen der Gerätedatei an, auf der sich das einzuhängende Dateisystem befindet. Z.B. `/dev/sd1g`, falls sich das Dateisystem auf einer Festplatte befindet.

Mit dem Argument *dvz* geben Sie den Pfadnamen des Dateiverzeichnisses an, in das das Dateisystem eingehängt werden soll. Das Dateiverzeichnis muß vorhanden sein und sollte leer sein. Falls es nicht leer ist, kann auf den ursrünglichen Inhalt nicht zugegriffen werden, solange darin ein Dateisystem eingehängt ist. *dvz* ist nach dem Einhängen das Wurzel-Dateiverzeichnis des eingehängten Dateisystems; alle Pfadnamen, die explizit oder implizit *dvz* enthalten, beziehen sich auf das eingehängte Dateisystem.

Das niedrigste Bit des Arguments *modus* steuert die Schreibberechtigung für das eingehängte Dateisystem: der Wert 1 bedeutet "kein Schreibzugriff", andere Werte bedeuten, daß der Schreibzugriff entsprechend den Schutzbits der einzelnen Dateien erlaubt ist.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird –1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *mount()* schlägt fehl, wenn gilt:

[EBUSY]
Bei *dvz* wird gerade ein Dateisystem eingehängt, oder *dvz* ist aktuelles Dateiverzeichnis eines Benutzers, oder es ist auf andere Weise belegt.
Das mit *ger* bezeichnete Gerät wird gerade eingehängt.
Keine Einträge mehr in der Einhängtabelle.

[EFAULT]
ger oder *dvz* verweist auf einen Namen außerhalb des dem Prozeß zugewiesenen Adreßraums.

[ENOENT]
Eine der angegebenen Dateien ist unbekannt.

[ENOTBLK]
ger bezeichnet kein blockorientiertes Gerät.

[ENOTDIR]
Eine Komponente von *pfad* ist kein Dateiverzeichnis.
dvz ist kein Dateiverzeichnis.

[ENXIO]
Das mit *ger* bezeichnete Gerät ist unbekannt.

[EPERM]
Der Prozeß besitzt nicht die nötigen, besonderen Rechte.

HINWEIS

Dateisysteme auf physikalisch schreibgeschützten Geräten oder auf Magnetplatte müssen ohne Schreiberlaubnis einbgehängt werden. Selbst wenn Sie nicht in das Dateisystem schreiben, würde ein Fehler auftreten, wenn vom Betriebssystem die Zugriffszeiten auf den neuesten Stand gebracht werden.

Statt mit der Funktion *mount()* ein Dateisystem einzuhängen, sollten Sie das Kommando */etc/mount* benutzen, wie in *SINIX Systemverwaltung* beschrieben. Wenn ein Dateisystem eingehängt wird, müssen Systemtabellen entsprechend korrigiert werden; das Kommando *mount* berücksichtigt dies automatisch.

Die Kommandos */etc/mount* und */etc/umount* tragen alle aktuell angehängten Dateisysteme in der Datei */etc/mtab* ein. Wenn Sie das Kommando */etc/mount* ohne Argument aufrufen, wird der Inhalt dieser Datei ausgegeben. Diese Datei hat rein informativen Charakter und wird von *mount()* und *umount()* nicht verwendet. D.h. ein Kommando */etc/mount* trägt ein eingehängtes Dateisystem in der *mtab* Datei ein, wird das eingehängte Dateisystem durch die Funktion *umount()* wieder abgesetzt, bleibt der Eintrag in der Datei *mtab* bestehen, obwohl das Dateisystem nicht (mehr) eingehängt ist. Umgekehrt erscheint ein mittels der Funktion *mount()* eingehängtes Dateisystem nicht in der Datei *mtab*, das Kommando */etc/mount* scheitert.

PORTABILITÄT

Die Funktion *mount()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

umount().

NAME

mrand48 - generate uniformly distributed pseudo-random signed long integers
Gleichverteilte, vorzeichenbehaftete Pseudo-Zufallszahlen des Typs long erzeugen

DEFINITION

```
long mrand48 ( )
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *mrand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

msgctl - message control operations
Nachrichtenwarteschlangen, Kontrolloperationen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

BESCHREIBUNG

Die Funktion *msgctl()* bietet Operationen für die Nachrichtenkontrolle, wie durch *cmd* angegeben. Die folgenden Werte für *cmd* und die dazugehörenden Nachrichten-Kontrolloperationen sind:

IPC_STAT
: Die aktuellen Werte aller Elemente der *msqid* zugeordneten Datenstruktur werden in die Struktur eingetragen, auf die mit *buf* verwiesen wird. Der Inhalt dieser Struktur wird in *<sys/msg.h>* definiert.

IPC_SET
: Die Werte folgender Elemente der *msqid* zugeordneten Datenstruktur des Typs *msgqid_ds* werden auf die entsprechenden Werte aus der Struktur gesetzt, auf die *buf* zeigt:

  msg_perm.uid
  msg_perm.gid
  msg_perm.mode
  msg_qbytes

  IPC_SET kann nur von einem Prozeß mit besonderen Rechten ausgeführt werden, oder einem Prozeß, dessen effektive Benutzernummer gleich dem Wert von *msg_perm.cuid* oder *msg_perm.uid* in der *msqid_ds*-Struktur ist, die *msqid* zugeordnet ist. Nur ein Prozeß mit besonderen Rechten kann *msg_qbytes* erhöhen.

IPC_RMID
: Gelöscht werden die mit *msqid* angegebene Warteschlangenkennzahl sowie die Warteschlange samt zugeordneter Datenstruktur. IPC_RMID kann nur von einem Prozeß ausgeführt werden, der übr besondere Rechte verfügt

oder dessen effektive Benutzernummer mit *msg_perm.cuid* bzw. *msg_perm.uid* in der *msquid_ds*-Struktur zu *msqid* übereinstimmt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *msgctl()* den Wert 0; andernfalls liefert sie den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *msgctl()* schlägt fehl, wenn gilt:

[EACCES] Das Argument *cmd* ist IPC_STAT und der aufrufende Prozeß besitzt nicht die Leseerlaubnis

[EINVAL] Der Wert von *msqid* ist keine gültige Warteschlangenkennzahl oder der Wert von *cmd* ist kein gültiges Kommando.

[EPERM] Das Argument *cmd* ist IPC_RMID oder IPC_SET und die effektive Benutzernummer des aufrufenden Prozesses ist nicht gleich der eines Prozesses mit besonderen Rechten und nicht gleich dem Wert von *msg_perm.cuid* oder *msg_perm.uid* in der *msqid* zugeordneten Datenstruktur.

[EPERM] Das Argument *cmd* ist IPC_SET, ein Versuch wurde gemacht, den Wert von *msg_qbytes* zu erhöhen und die effektive Benutzernummer des aufrufenden Prozesses besitzt keine besonderen Rechte.

[ENOSYS] Diese Funktionalität wird vom der Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *msgctl()* ist im X/Open-Standard (Ausgabe 3) als optional definiert.Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

BEISPIEL

Siehe das Beispiel unter *msgget()*.

SIEHE AUCH

msgget(), *msgrcv()*, *msgsnd()*, *<sys/ipc.h>*, *<sys/msg.h>*, *<sys/types.h>*, *Abschnitt 2.6*.

NAME

msgget - get message queue
Nachrichtenwarteschlange einrichten

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key, msgflg)
key_t key;
int msgflg;
```

BESCHREIBUNG

Die Funktion *msgget()* liefert die Warteschlangenkennzahl die dem Argument *key* zugeordnet ist. Eine Warteschlangenkennzahl, die zugehörige Warteschlange und Datenstruktur (siehe auch *<sys/msg.h>*) werden für das Argument *key* dann erzeugt, wenn eine der folgenden Bedingungen erfüllt ist:

— Das Argument *key* ist gleich IPC_PRIVATE.

— Das Argument *key* besitzt noch keine zugeordnete Warteschlangenkennzahl und *(msgflg & IPC_CREAT)* ist ungleich 0.

Bei der Erzeugung wird die der neuen Warteschlangenkennzahl zugeordnete Datenstruktur wir folgt initialisiert:

— *msg_perm.cuid*, *msg_perm.uid*, *msg_perm.cgid* und *msg_perm.gid* werden gleich der effektiven Benutzer- bzw. Gruppennummer des aufrufenden Prozesses gesetzt;

— Die niederwertigsten 9 Bit von *msg_perm.mode* werden gleich den niederwertigsten 9 Bit von *msgflg* gesetzt;

— *msg_qnum*, *msg_lspid*, *msg_lrpid*, *msg_stime* und *msg_rtime* werden gleich 0 gesetzt;

— *msg_ctime* wird gleich der aktuellen Zeit gesetzt;

— *msg_qbytes* wird gleich der durch das System festgelegten Grenze gesetzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *msgget()* eine nichtnegative ganze Zahl, eine Warteschlangenkennzahl; andernfalls liefert sie den Wert -1 und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *msgget()* schlägt fehl, wenn gilt:

[EACCES] Es existiert eine Warteschlangenkennzahl für das Argument *key*, aber die in den niederwertigsten 9 Bit von *msgflg* angegebenen Zugriffsrechte werden nicht erteilt, siehe auch *Abschnitt 2.6*.

[EEXIST] Es existiert eine Warteschlangenkennzahl für das Argument *key*, aber der Wert von *((msgflg & IPC_CREAT) && (msgflg & IPC_EXCL))* ist ungleich 0.

[ENOENT] Es existiert keine Warteschlangenkennzahl für das Argument *key* und *(msgflg & IPC_CREAT)* ist gleich 0.

[ENOSPC] Es soll eine Warteschlangenkennzahl erzeugt werden, aber die durch das System festgelegte Grenze für die Maximalzahl der erlaubten Warteschlangenkennzahlen würde dadurch überschritten werden.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *msgget()* ist im X/Open-Standard (Ausgabe 3) als optional definiert.Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

BEISPIEL

Zwei Prozesse, die nicht miteinander verwandt sind, machen sich gegenseitig ihre Prozeßnummern bekannt. Schreiben Sie die beiden Programme und geben Sie den beiden ausführbaren Programmen den Namen z.B. *server* und *client*.

Das Programm *server* richtet eine Nachrichtenwarteschlange ein, und wartet dann - in einer Endlosschleife - darauf, daß das (irgendein) Programm *client* seine Prozeßnummer als Nachricht schickt. Daraufhin schickt das Programm *server* seine Prozeßnummer an das Programm *client*. Rufen Sie dann zuerst den *server*-Prozeß im Hintergrund auf, dann den *client*-Prozeß. Die beiden Prozesse müssen gleichzeitig laufen, deshalb werden sie als Hintergrundprozesse aufgerufen. Dies hat gleichzeitig den Vorteil, daß das Betriebssystem die zugehörigen Prozeßnummern ausgibt. Damit ist leicht zu sehen, daß die Prozeßnummern übergeben werden.

Die Nachrichtenwarteschlange (message queue) wird gemultiplext. Dadurch kann die Nachrichtenwarteschlange mehrfach ausgenutzt werden; ein *server* kann mehrere *clients* bedienen: Mehrere *clients* können gleichzeitig ihre Nachricht an den *server* schicken, und der *server* verteilt seine Nachrichten an die verschiedenen *clients*.

Programm server (in der Datei *server.c*)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSGKEY 75

struct msgform {
   long mtype;
   char mtext[100];
} msg;

int msgid;

main()
{  int i, pid, *pint;
   extern cleanup();

   printf("server: anfang\n");

   signal(SIGHUP, SIG_IGN);
   if (signal(SIGINT, SIG_IGN) != SIG_IGN)
      signal (SIGINT, cleanup);
   if (signal(SIGQUIT, SIG_IGN) != SIG_IGN)
      signal (SIGQUIT, cleanup);
   signal (SIGTERM, cleanup);

   msgid = msgget(MSGKEY, 0666 | IPC_CREAT);

   while (1)
   {  msgrcv(msgid, &msg, 100, 1, 0);
         /* empfange alle Nachrichten vom Typ 1 */
      pint = (int *) msg.mtext;
      pid = *pint;
      printf("server: empfange von pid %d\n", pid);

      msg.mtype = pid;  /* verschicke Nachricht vom Typ pid */
      *pint = getpid();
      msgsnd(msgid, &msg, sizeof(int), 0);
   }
}

cleanup()
{  printf("server.cleanup: laeuft\n");
   msgctl(msgid, IPC_RMID, 0);
   exit();
}
```

Programm client (in der Datei *client.c*)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSGKEY 75

struct msgform {
   long mtype;
   char mtext[100];
};

main()
{  struct msgform msg;
   int msgid, pid, *pint;

   printf("client: anfang\n");

   msgid = msgget(MSGKEY, 0666);

   pid = getpid();
   pint = (int *) msg.mtext;
   *pint = pid;
      /* pid wird in Nachrichtentext kopiert */
   msg.mtype = 1;
      /* verschicke Nachricht an server vom Typ 1 */

   msgsnd(msgid, &msg, sizeof(int), 0);
   msgrcv(msgid, &msg, 100, pid, 0);
      /* empfange nur Nachrichten vom Typ pid */
   printf("client: empfange von pid %d\n", *pint);
   printf("client: ende\n");
}
```

SIEHE AUCH

msgctl(), *msgrcv()*, *msgsnd()*, *<sys/ipc.h>*, *<sys/msg.h>*, *<sys/types.h>*, *Abschnitt 2.6.*

NAME

msgrcv - message receive operation
Nachricht empfangen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv(msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
void *msgp;
size_t msgsz;
long msgtyp;
int msgflg;
```

BESCHREIBUNG

Die Funktion *msgrcv()* liest eine Nachricht aus der Warteschlange, der die durch *msqid* angegebene Warteschlangenkennzahl zugeordnet ist und legt diese in dem vom Benutzer definierten Puffer ab, auf den *msgp* zeigt.

Das Argument *msgp* zeigt auf einen vom Benutzer definierten Puffer, der zunächst eine Komponente des Typs *long* für den Nachrichtentyp und dann einen Datenbereich für die Datenbytes des Nachricht enthalten muß. Die nachstehende Struktur ist ein Beispiel dafür, wie dieser vom Benutzer definierte Puffer aussehen könnte:

```
struct    mymsg {
   long   mtype;        /* Nachrichtentyp */
   char   mtext[SIZE];  /* Datenbytes */
}
```

Die Strukturkomponente *mtype* ist der Nachrichtentyp der empfangenen Nachricht, wie durch den sendenden Prozeß angegeben.

Die Strukturkomponente *mtext* ist der Text der Nachricht (Hier in der Größe SIZE).

Das Argument *msgsz* gibt die Größe von *mtext* in Bytes an. Die empfangene Nachricht wird, wenn sie länger und *(msgflg & MSG_NOERROR)* ungleich 0 ist, auf *msgsz* Bytes gekürzt. Der abgeschnittene Teil der Nachricht geht verloren und dem aufrufenden Prozeß wird dies nicht mitgeteilt.

Das Argument *msgtyp* gibt den Typ der geforderten Nachricht wie folgt an:

wenn *msgtyp* gleich 0 ist, dann wird die erste Nachricht in der Nachrichtenwarteschlange empfangen;

wenn *msgtyp* größer als 0 ist, dann wird die erste Nachricht des Typs *msgtyp* empfangen;

wenn *msgtyp* kleiner als 0 ist, dann wird die erste Nachricht kleiner oder gleich dem Absolutwert von *msgtyp* empfangen.

Das Argument *msgflg* gibt an, welche Aktion ausgeführt werden soll, wenn sich keine Nachricht des geforderten Typs in der Warteschlange befindet. Folgende Aktionen sind möglich:

Wenn *(msgflg & IPC_NOWAIT)* ungleich 0 ist, dann kehrt die Funktion sofort mit dem Ergebnis -1 zum aufrufenden Prozeß zurück und *errno* ist gleich [ENOMSG] gesetzt.

Wenn *(msgflg & IPC_NOWAIT)* gleich 0 ist, dann unterbricht der aufrufende Prozeß seine Ausführung bis eines der folgenden Ereignisse eintritt:

Eine Nachricht des geforderten Typs wird in die Warteschlange eingetragen.

Die Warteschlangenkennzahl *msqid* wird aus dem System entfernt; wenn dies geschieht, dann wird *errno* gleich [EIDRM] gesetzt und das Ergebnis -1 wird zurückgeliefert.

Der aufrufende Prozeß empfängt ein abzufangendes Signal; in diesem Fall wird die Nachricht nicht empfangen und der Prozeß setzt seine Ausführung so fort, wie dies unter *sigaction()* beschrieben wird.

Bei erfolgreicher Beendigung werden die folgenden Aktionen auf der *msqid* zugeordneten Datenstruktur ausgeführt:

msg_qnum wird um 1 vermindert

msg_lrpid wird gleich der Prozeßnummer des aufrufenden Prozesses gesetzt

msg_rtime wird auf die aktuelle Zeit gesetzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *msgrcv()* einen Wert, der gleich der Anzahl der im Puffer *mtext* abgelegten Bytes ist.

Andernfalls wird keine Nachricht empfangen, *msgrcv()* liefert den Wert -1 und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *msgrcv()* schlägt fehl, wenn gilt:

[E2BIG] Der Wert von *mtext* ist größer als *msgsz* und *(msgflg & MSG_NOERROR)* ist gleich 0.

[EACCES] Der aufrufende Prozeß erhält keine Erlaubnis für diese Operation.

[EIDRM] Die Warteschlangenkennzahl *msqid* wurde aus dem System entfernt.

[EINTR] Die Funktion *msgrcv()* wurde durch ein Signal unterbrochen.

[EINVAL] *msqid* ist keine gültige Warteschlangenkennzahl; oder der Wert von *msgsz* ist kleiner als 0.

[ENOMSG] Die Warteschlange enthält keine Nachricht des geforderten Typs und *(msgtyp & IPC_NOWAIT)* ist ungleich 0.

[ENOSYS] Die Funktionalität wird vom dieser Implementierung nicht unterstützt.

HINWEIS

Der Wert des Arguments *msgp* sollte in den Typ (*void* *) umgewandelt werden.

PORTABILITÄT

Die Funktion *msgrcv()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

msgctl(), *msgget()*, *msgsnd()*, *sigaction()*, *<sys/ipc.h>*, *<sys/msg.h>*, *<sys/types.h>*, *Abschnitt 2.6*.

NAME

msgsnd - message send operation
Nachricht senden

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(msqid, msgp, msgsz, msgflg)
int msqid;
void *msgp;
size_t msgsz;
int msgflg;
```

BESCHREIBUNG

Die Funktion *msgsnd()* wird verwendet, um eine Nachricht an die Warteschlange zu senden, die durch die Warteschlangenkennzahl *msqid* angegeben wird.

Das Argument *msgp* zeigt auf einen vom Benutzer definierten Puffer, der zunächst eine Komponente des Typs *long* für den Nachrichtentyp und dann einen Datenbereich für die Datenbytes der Nachricht enthalten muß. Die nachstehende Struktur ist ein Beispiel dafür, wie dieser vom Benutzer definierte Puffer aussehen könnte:

```
struct    mymsg {
   long   mtype;        /* message type */
   char   mtext[SIZE];  /* message text */
}
```

Die Strukturkomponente *mtype* ist ein von 0 verschiedener Wert vom Typ *long*, der vom empfangenden Prozeß zur Nachrichtenauswahl verwendet werden kann.

Die Strukturkomponente *mtext* ist irgendein Text der Länge *msgsz* Bytes. Das Argument *msgsz* kann von 0 bis zu einer vom System festgelegten Grenze reichen.

Das Argument *msgflg* gibt an, welche Aktion ausgeführt werden soll, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

Die Anzahl der Bytes in der Warteschlange ist bereits gleich *msg_qbytes*, siehe auch *<sys/msg.h>*.

Die Gesamtzahl der Nachrichten in allen Warteschlangen des Systems ist bereits gleich der durch das System festgelegten Grenze.

Folgende Aktionen können dann ausgeführt werden:

Wenn *(msgflg & IPC_NOWAIT)* ungleich 0 ist, dann wird keine Nachricht gesendet und die Funktion kehrt sofort zum aufrufenden Prozeß zurück.

Wenn *(msgflg & IPC_NOWAIT)* gleich 0 ist, dann unterbricht der aufrufende Prozeß seine Ausführung bis eines der folgenden Ereignisse eintritt:

Die Bedingung, die für die Unterbrechung verantwortlich war, existiert nicht mehr; in diesem Fall wird die Nachricht gesendet.

Die Warteschlangenkennzahl *msqid* wird aus dem System entfernt; wenn dies geschieht, dann wird *errno* gleich [EIDRM] gesetzt und das Ergebnis -1 wird zurückgeliefert.

Der aufrufende Prozeß empfängt ein abzufangendes Signal; in diesem Fall wird die Nachricht nicht gesendet und der Prozeß setzt seine Ausführung so fort, wie dies unter *sigaction()* beschrieben wird.

Bei erfolgreicher Beendigung werden die folgenden Aktionen auf der *msqid* zugeordneten Datenstruktur ausgeführt:

msg_qnum wird um 1 erhöht

msg_lspid wird gleich der Prozeßnummer des aufrufenden Prozesses gesetzt

msg_stime wird auf die aktuelle Zeit gesetzt.

ERGEBNIS

Bei Erfolg liefert die Funktion *msgsnd()* den Wert 0. Anderfalls wird keine Nachricht gesendet, *msgsnd()* liefert den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *msgsnd()* schlägt fehl, wenn gilt:

[EACCES] Der aufrufende Prtozeß erhält keine Erlaubnis für diese Operation.

[EAGAIN] Die Nachricht kann aus einem der oben genannten Gründe nicht gesendet werden und *(msgflg & IPC_NOWAIT)* ist ungleich 0.

[EIDRM] Die Warteschlangenkennzahl *msqid* wurde aus dem System entfernt.

[EINTR] Die Funktion *msgsnd()* wurde durch ein Signal unterbrochen.

[EINVAL] *msqid* ist keine gültige Warteschlangenkennzahl der Wert von *mtype* ist kleiner als 0, oder der Wert von *msgsz* ist kleiner als 0 oder größer als die vom System festgelgte Grenze.

[ENOSYS] Diese Funktion wird von der aktuellen Implementierung nicht unterstützt.

HINWEIS

Der Wert des Arguments *msgp* sollte in den Typ (*void* *) umgewandelt werden.

PORTABILITÄT

Die Funktion *msgsnd()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

msgctl(), *msgget()*, *msgrcv()*, *sigaction()*, *<sys/ipc.h>*, *<sys/msg.h>*, *<sys/types.h>*, *Abschnitt 2.6.*

NAME

nice - change priority of a process
Prozeßpriorität ändern

DEFINITION

```
int nice (incr)
int incr;
```

BESCHREIBUNG

Die Funktion *nice()* addiert den Wert von *incr* auf den Prioritätswert des aufrufenden Prozesses. Ein Prioritätswert ist eine nichtnegative Zahl bei der ein höherer Wert eine niedrigeren Prozessor-Priorität resultiert.

Ein maximaler Prioritätswert von 2*{NZERO}-1 und ein minimaler Prioritätswert von 0 werden durch das System festgelegt. Anforderungen für Werte oberhalb oder unterhalb dieser Grenzen verursachen, daß der Prioritätswert auf die entsprechende Grenze gesetzt wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *nice()* den neuen Prioritätswert abzüglich {NZERO}. Andernfalls wird der Wert -1 zurückgeliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *nice()* schlägt fehl, wenn gilt:

[EPERM] *incr* ist negativ oder größer als 2*{NZERO}-1 und der aufrufende Prozeß besitzt keine besonderen Rechte.

HINWEIS

Da bei Erfolg alle Ergebnisse möglich sind, sollte eine Anwendung, die auf Fehlersituationen hin überprüfen will, *errno* vor dem Aufruf von *nice()* gleich 0 setzen und dann *nice()* aufrufen. Wenn -1 zurückgeliefert wird, dann sollte sie prüfen, ob *errno* ungleich 0 ist.

PORTABILITÄT

Die Funktion *nice()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<limits.h>.

NAME

nl_codesize - number of simple and multibyte characters retrieval
Zeichensatz analysieren

DEFINITION

```
int nl_codesize(nspc,nmbc)
int *nspc, *nmbc;
```

BESCHREIBUNG

Wenn *nspc* und *nmbc* keine Nullzeiger sind, dann bestimmt die Funktion *nl_codesize()* die Anzahl der einfachen und der Multibyte-Zeichen des aktuellen Zeichensatzes. Einfache Zeichen bestehen aus einem Byte, Multibyte-Zeichen setzen sich aus mehreren Bytes zusammen (2 bis {NL_N_MAX} Zeichen). Die Funktion speichert die ermittelten Werte an den Adressen *nspc* und *nmbc* ab.

Die Bestimmung erfolgt gemäß der internationalen Umgebung des Programms (Kategorie LC_CTYPE), wenn zuvor die Funktion *setlocale()* erfolgreich aufgerufen wurde. Ansonsten richtet sich die Bestimmung nach der Umgebungsvariablen LANG. Ist diese nicht definiert, dann finden die Regeln des US-ASCII-Zeichensatzes Anwendung.

ERGEBNIS

Wenn die Anzahl der einfachen und Multibyte-Zeichen nicht bestimmt werden kann oder *nspc* bzw. *nmbc* ein Nullzeiger ist, dann liefert die Funktion den Wert –1. Andernfalls liefert die Funktion *nl_codesize()* den Wert 0.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *nl_codesize()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

SIEHE AUCH

setlocale(), *mbtowc()*, *Abschnitt 2.5*.

NAME

nl_langinfo - language information
Landessprachen-Information

DEFINITION

```
#include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo (item)
nl_item item;
```

BESCHREIBUNG

Die Funktion *nl_langinfo()* liefert einen Zeiger auf eine Zeichenkette, die Informationen entsprechend der jeweiligen Landessprache oder der anwenderspezifischen Umgebung enhält, die in der internationalen Umgebung des Programms definiert ist (siehe auch *<langinfo.h>*). Die verfügbaren Konstanten und Werte für *item* sind in *<langinfo.h>* definiert. Zum Beispiel:

nl_langinfo (ABDAY_1)

liefert einen Zeiger auf die Zeichenkette "Dom", wenn die Landessprache Portugiesisch, und "Sun", wenn die Landessprache Englisch ist.

ERGEBNIS

In einer Umgebung, in der keine *langinfo*-Daten definiert sind, liefert *nl_langinfo()* einen Zeiger auf die entsprechende Umgebung für die Programmiersprache C. In allen Umgebungen liefert *nl_langinfo()* einen Zeiger auf eine leere Zeichenkette, wenn *item* ungültig ist.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Der Vektor, auf den das Ergebnis zeigt, sollte vom Programm nicht verändert werden, aber weitere Aufrufe von *nl_langinfo()* können ihn ändern. Zusätzlich können auch Aufrufe der Funktion *setlocale()* mit einer Kategorie, die der von *item* entspricht (siehe auch *<langinfo.h>*), oder mit der Kategorie LC_ALL diesen Vektor überschreiben.

PORTABILITÄT

Die Funktion *nl_langinfo()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel im *Abschnitt 2.5 (Funktion yesdata())*.

SIEHE AUCH

setlocale(), *<langinfo.h>*, *<nl_types.h>*, *Abschnitt 2.5*.

NAME

nl_settype, nl_istype - character classification for NLS
Zeichenklassifikation (NLS)

DEFINITION

```
#include <ctype.h>
#include <nl_types.h>

nl_type nl_settype(class)
nl_class class;

int nl_istype(c, type)
int c;
nl_type type;
```

BESCHREIBUNG

Die Funktionen *nl_istype()* und *nl_settype()* bieten zusätzliche Möglichkeiten der Zeichenklassifikation, insbesondere für Zeichensätze mit Multibyte-Zeichen (Zeichen, die aus mehreren Bytes bestehen).

Die Funktion *nl_istype()* prüft, ob das Zeichen *c* in der Zeichenklasse *type* enthalten ist.

Die Funktion *nl_settype()* liefert einen Wert, der als zweites Argument der Funktion *nl_istype()* verwendet werden kann. Ihr Argument *class* ist eine generische Zeichenklasse, für die Zeichensatz-Information ermittelt werden soll.

Die Funktion *nl_settype()* bestimmt den Wert von *class* gemäß den Regeln des in der internationalen Umgebung des Programms definierten Zeichensatzes (Kategorie LC_CTYPE). In einer Umgebung, wo diese Information nicht verfügbar ist, werden die Zeichen nach den Regeln des US-ASCII Zeichensatzes klassifiziert.

Die symbolischen Konstanten für die Werte von *class* und die Werte für *nl_class* sind in der Include-Datei *<nl_types.h>* definiert.

ERGEBNIS

Die Funktion *nl_istype()* liefert genau dann den Wert 0, wenn *c* nicht in der Klasse *type* enthalten ist. Ansonsten liefert sie einen Wert ungleich 0.

Die Funktion *nl_settype()* liefert die zum Argument *class* passende Klasse. Wenn *class* eine Zeichenklasse ist, die im aktuellen Zeichensatz nicht unterstützt wird, dann liefert die Funktion einen Wert, der einen nachfolgenden Aufruf von *nl_istype()* fehlschlagen läßt.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Beispiel überprüft, ob alle Zeichen einer Zeichenkette gültige Zeichen im aktuellen zeichensatz sind:

```
#include <nl_types.h>
#include <ctype.h>

int verify(char *s)
{
  nl_type type = nl_settype(ILLEGAL);
  int valid = 1;

  while ( *s  &&  valid )
      valid = !(nl_istype(*s++,type));

  return valid;
}
```

SIEHE AUCH

<nl_types.h>, *Abschnitt 2.5*.

NAME

nlist - get entry from symbol table
Einträge aus Symboltabelle ermitteln

DEFINITION

```
#include <nlist.h>

int nlist (dateiname, nl)
char *dateiname
struct nlist *nl
```

BESCHREIBUNG

nlist() durchsucht die Symboltabelle in der ausführbaren Datei, auf deren Namen *dateiname* zeigt, nach bestimmten Werten und legt diese in dem Vektor ab, auf den *nl* zeigt. Dieser Vektor besteht aus Strukturen, die jeweils den symbolischen Namen einer Variablen, ihren Typ und ihren Wert enthalten. Die letzte Struktur enthält anstelle eines Namens eine leere Zeichenkette. Wird beim Durchsuchen der Symboltabelle nach Variablennamen ein gesuchter Name gefunden, so werden Typ und Wert in den entsprechenden Feldern der dazugehörigen Struktur eingetragen. Das Typ-Feld wird auf 0 gesetzt, wenn bei der Übersetzung der Datei nicht der -g Schalter gesetzt war. Wird der gesuchte Name nicht gefunden, so werden beide Einträge auf 0 gesetzt.
Mit dieser Funktion kann man aus der Symboltabelle des Systems in der Datei */sinix* die aktuellen Systemadressen abfragen.
Weitere Informationen zur Struktur der Symboltabelle finden Sie unter *<a.out.h>*.

ERGEBNIS

Falls die Datei nicht gelesen werden kann oder keine gültige Symboltabelle enthält, werden alle Einträge in dem *nl*-Vektor auf 0 gesetzt.

Im Fehlerfall liefert *nlist()* den Wert −1 zurück, ansonsten 0.

HINWEIS

Aus Kompatibilitätsgründen schließt *<a.out.h>* die Datei *<nlist.h>* automatisch mit ein. Wenn *<a.out.h>* jedoch nur wegen der Symboltabelle benötigt wird, so wird von einem Einbinden von *<a.out.h>* abgeraten. Ein Einbinden von *<a.out.h>* erfordert möglicherweise eine nachfolgende Zeile "#undef n_name".

SIEHE AUCH

<a.out.h>.

NAME

nrand48 - generate uniformly distributed pseudo-random non-negative long integers
Gleichmäßig verteilte, nichtnegative ganzzahlige Pseudo-Zufallszahlen erzeugen

DEFINITION

```
long nrand48 (xsubi)
unsigned short xsubi[3];
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *nrand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

open - open a file
Datei öffnen

DEFINITION

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

int open (path, oflag , ... )
char *path;
int oflag;
```

BESCHREIBUNG

Die Funktion *open* stellt eine Verbindung zwischen einer Datei und einer Dateikennzahl her. Sie erzeugt eine offene Dateibeschreibung, die auf eine Datei verweist und eine Dateikennzahl, die auf diese offene Dateibeschreibung verweist. Die Dateikennzahl wird von anderen Ein- und Ausgabefunktionen genutzt, um auf diese Datei zu verweisen. Das Argument *path* zeigt auf einen Pfadnamen, der die Datei bezeichnet.

Die Funktion *open()* liefert eine Dateikennzahl für die genannte Datei, welche die kleinste, noch nicht geöffnete Dateikennzahl des Prozesses ist. Die offene Dateibeschreibung ist neu, daher teilt diese Dateikennzahl sie nicht mit anderen Prozessen im System. Die neue Dateikennzahl ist so eingerichtet, daß sie bei einem Aufruf einer der *exec*-Funktionen offen bleibt (siehe auch *fcntl()*).

Die aktuelle Dateiposition wird auf den Anfang der Datei gesetzt.

Das System-Dateistatusbyte und die Zugriffsart werden entsprechend dem Wert von *oflag* gesetzt.

Die Werte für *oflag* werden durch bitweises ODER aus der folgenden Liste erzeugt, die in *<fcntl.h>* definiert ist. Anwendungen müssen genau einen der unten angegebenen ersten drei Werte (Zugriffsarten) im Wert von *oflag* angeben:

O_RDONLY Nur zum Lesen öffnen

O_WRONLY Nur zum Schreiben öffnen

O_RDWR Zum Lesen und Schreiben öffnen. Das Ergebnis ist nicht definiert, wenn dieses Kennzeichen für eine FIFO angewendet wird.

Jede Kombination der folgenden zusätzlichen Werte kann benutzt werden:

O_APPEND
Wenn dieses Bit gesetzt ist, dann wird die Position in der Datei vor jedem Schreiben auf das Ende der Datei gesetzt.

O_CREAT
Wenn die Datei existiert, dann hat dieses Bit keine Wirkung, außer gemäß der Bemerkung unten unter O_EXCL. Andernfalls wird die Datei erzeugt; die Benutzernummer der Datei wird gleich der effektiven Benutzernummer des Prozesses und die Gruppennummer der Datei wird gleich der effektiven Gruppennummer des Prozesses oder der Gruppennummer des übergeordneten Dateiverzeichnisses der Datei gesetzt. Die Zugriffsrechte der Datei (siehe auch *<sys/stat.h>*) werden gleich dem Wert des dritten Arguments gesetzt, das als vom Typ *mode_t* aufgefaßt wird und dann folgendermaßen verändert: die einzelnen Bits werden mit dem Komplement der Schutzbitmaske des Prozesses bitweise UND-verknüpft (siehe auch *umask()*). Das heißt, alle Bits in den Zugriffsrechten, die in der Schutzbitmaske gesetzt sind, werden gelöscht. Das dritte Argument beeinflußt nicht, ob die Datei zum Lesen, zum Schreiben oder zum Lesen und Schreiben geöffnet wird.

O_EXCL
Wenn O_CREAT und O_EXCL gesetzt sind, dann schlägt *open()* fehl, wenn die Datei exisitiert. Die Prüfung, ob die Datei existiert und deren Erzeugung, falls dies nicht der Fall ist, sind atomar im Hinblick auf andere Prozesse, die *open()* für den selben Pfad mit den Bits O_EXCL und O_CREAT ausführen.

O_NDELAY
Wird in Zukunft nicht mehr unterstützt. Anwendungen sollten statt dessen O_NONBLOCK benutzern (s.u.).

O_NOCTTY
Wenn dieses Kennzeichen gesetzt ist, und *path* eine Datensichtstation bezeichnet, dann sorgt *open()* dafür, daß diese Datensichtstation nicht das kontrollierende Terminal des Prozesses wird.

O_NONBLOCK
Wenn eine FIFO zum Lesen oder zum Schreiben geöffnet werden soll (O_RDONLY oder O_WRONLY):

Wenn O_NONBLOCK gesetzt ist:
Ein *open()* zum Lesen kehrt ohne Verzögerung zurück.
Ein *open()* zum Schreiben liefert nur dann einen Fehler, wenn kein Prozeß diese Datei zur Zeit zum Lesen geöffnet hat.

Wenn O_NONBLOCK nicht gesetzt ist:
Ein *open()* zum Lesen wartet, bis ein Prozeß die Datei zum Schreiben öffnet.
Ein *open()* zum Schreiben wartet, bis ein Prozeß die Datei zum Lesen öffnet.

Wenn eine Gerätedatei für ein block- oder zeichenorientiertes Gerät geöffnet wird, die nichtwartendes Öffnen unterstützt:

Wenn O_NONBLOCK gesetzt ist:
Die Funktion *open()* kehrt zurück, ohne darauf zu warten, daß das Gerät fertig oder verfügbar wird. Das nachfolgende Verhalten des Gerätes ist gerätespezifisch.

Wenn O_NONBLOCK nicht gesetzt ist:
Die Funktion *open()* wartet, bis das Gerät fertig oder verfügbar ist, bevor sie zurückkehrt.

Andernfalls ist das Verhalten von O_NONBLOCK nicht festgelegt.

O_SYNC
Wenn O_SYNC für eine normale Datei gesetzt ist, dann verursacht dies, daß ein Schreibzugriff auf diese Datei den Prozeß solange anhält, bis die Daten an die zugrundeliegende Hardware übergeben wurden.

O_TRUNC
Wenn die Datei existiert, eine normale Datei ist und erfolgreich mit O_RDWR oder O_WRONLY geöffnet wird, dann wird ihre Länge auf 0 gekürzt und Eigentümer und Zugriffsrechte bleiben unverändert. Dies hat keine Wirkung für FIFO-Gerätedateien oder Dateiverzeichnisse. Die Wirkung für andere Dateiarten ist nicht definiert, da sie von von vielen Faktoren abhängig ist. Das Ergebnis bei einer Verwendung von O_TRUNC zusammen mit O_RDONLY ist undefiniert.

Wenn O_CREAT gesetzt ist und die Datei vorher nicht existierte, dann markiert die Funktion *open()* im Erfolgsfall die Felder *st_atime*, *st_ctime* und *st_mtime* der Datei und die Felder *st_ctime* und *st_mtime* des übergeordneten Dateiverzeichnisses zum Ändern.

Wenn O_TRUNC gesetzt ist und die Datei vorher bereits existierte, dann markiert die Funktion *open()* im Erfolgsfall die Felder *st_ctime* und *st_mtime* der Datei zum Ändern.

ERGEBNIS

Bei erfolgreicher Beendigung öffnet die Funktion die Datei und liefert eine nichtnegative ganze Zahl, welche die kleinste nicht benutzte Dateikennzahl repräsentiert. Andernfalls wird der Wert -1 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen. Wenn die Funktion den Wert -1 liefert, werden keine Dateien erzeugt oder verändert.

FEHLER

Die Funktion *open()* schlägt fehl, wenn gilt:

[EACCES] Keine Durchsuchberechtigung für eine Komponente des Pfadnamen-Anfangs oder die Datei existiert und die durch *oflag* angegebenen Zugriffsrechte werden nicht erteilt. Oder die Datei ist nicht vorhanden und das Dateiverzeichnis, in dem die Datei eingerichtet werden soll, erlaubt keinen Schreibzugriff. Oder O_TRUNC ist angegeben worden und die Schreiberlaubnis für die Datei wird verweigert.

[EEXIST] O_CREAT und O_EXCL sind gesetzt und die angegebene Datei existiert bereits.

[EINTR] Während der Funktion wurde ein Signal abgefangen.

[EISDIR] Die angegebene Datei ist ein Dateiverzeichnis und *oflag* enthält O_WRONLY oder O_RDWR.

[EMFILE] Für den Prozeß sind derzeit bereits {OPEN_MAX} Dateikennzahlen offen.

[ENAMETOOLONG] Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENFILE] Die Dateitabelle des Systems ist voll.

[ENOENT] O_CREAT ist nicht gesetzt und die angegebene Datei existiert nicht; oder O_CREAT ist gesetzt und entweder der Pfadnamen-Anfang existiert nicht oder das Argument *path* zeigt auf die leere Zeichenkette.

[ENOSPC] Das Dateiverzeichnis oder Dateisystem, das die neue Datei enthalten würde, kann nicht vergrößert werden, die Datei existiert nicht und O_CREAT ist mit angegeben worden.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

[ENXIO] Die angegebene Datei ist eine Gerätedatei für ein block- oder zeichenorientiertes Gerät und das dieser Gerätedatei zugeordnete Gerät existiert nicht. O_NONBLOCK ist gesetzt, die angegebene Datei ist ein FIFO, O_WRONLY ist gesetzt und kein Prozeß hat die Datei zum Lesen geöffnet.

[EROFS] Die angegebene Datei befindet sich auf einem, nur zum Lesen eingehängten Dateisystem und eines der Bits O_WRONLY, O_RDWR, O_CREAT (wenn die Datei nicht existiert) oder O_TRUNC ist im Argument *oflag* mit angegeben worden.

Die Funktion *open()* kann fehlschlagen, wenn gilt:

[EINVAL] Der Wert des Arguments *oflag* ist ungültig.

[ETXTBSY] Die Datei ist eine reine Prozedurdatei (gemeinsamer Text) die gerade ausgeführt wird, und *oflag* ist O_WRONLY oder O_RDWR.

Unter SINIX V5.22 kann zusätzlich zu den im XPG3 [6] definierten Fehlern auch noch der folgende Fehler auftreten:

[EFAULT] Es wurde eine ungültige Adresse als Argument übergeben.

HINWEIS

O_NDELAY ist zurückgezogen und durch O_NONBLOCK ersetzt worden. O_NONBLOCK unterscheidet sich von O_NDELAY darin, daß es möglich ist, zwischen der Dateiende und einer Fehlerbedingung zu unterscheiden (siehe auch *read()* und *write()*).

PORTABILITÄT

Die Funktion *open()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Im Beispiel wird die Datei *jim_knopf* angelegt (falls sie noch nicht existiert) und zum Lesen und Schreiben geöffnet. Die zugewiesene Dateikennzahl ist *dk1*. Danach wird die Datei ein zweitesmal mit der Dateikennzahl *dk2* zum Lesen geöffnet. Das Programm liest dann von der Standardeingabe und schreibt die eingelesenen Zeichen über die Dateikennzahl *dk1* in die Datei *jim_knopf*. Danach werden alle Zeichen bis zum ersten Doppelpunkt über *dk1* aus der Datei gelesen und auf Standardausgabe geschrieben. Über *dk2* werden dann nochmals alle Zeichen bis zum Dateiende gelesen und auf Standardausgabe geschrieben. Anschließend wird der erste Lesevorgang nach dem Doppelpunkt fortgesetzt.

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

extern int errno;
extern char *sys_errlist[ ];

main()
{
   int dk1, dk2;              /* Dateikennzahlen */
   char buf[BUFSIZ];          /* Definition von BUFSIZ in <stdio.h> */
   char c;
   int n;

   if((dk1 = open("jim_knopf", O_RDWR | O_APPEND | O_CREAT, 0666)) == -1)
   {  /* falls open() scheitert */
      fprintf(stderr,"Fehler [error %d] beim ersten open: %s\n",
         errno, sys_errlist[errno]);
      exit(1);
   }

   if((dk2 = open("jim_knopf", O_RDONLY)) == -1)
   {  /* falls open() scheitert */
      fprintf(stderr,"Fehler [error %d] beim zweiten open: %s\n",
         errno, sys_errlist[errno]);
      exit(1);
   }

   while((n = read(0, buf, sizeof(buf))) > 0 )
      /* lesen von Standardeingabe */
      write(dk1, buf, n);
         /* gelesene Zeichen werden nach jim_knopf geschrieben */

   lseek(dk1, (off_t) 0, SEEK_SET);
      /* Schreib/Lesezeiger von dk1 auf Dateianfang setzen */

   while((n = read(dk1, &c, 1)) > 0 && (c != ':'))
      /* ueber dk1 lesen bis zum ersten Doppelpunkt */
      write(1, &c, n);
         /* Ausgabe der gelesenen Zeichen auf Standardausgabe */

   while((n = read(dk2, buf, sizeof(buf))) > 0)
      /* ueber dk2 lesen bis Dateiende */
      write(1, buf, n);
         /* Ausgabe der gelesenen Zeichen auf Standardausgabe */

   while((n = read(dk1, buf, sizeof(buf))) > 0)
      /* ueber dk1 weiterlesen bis Dateiende */
      write(1, buf, n);
         /* Ausgabe der gelesenen Zeichen auf Standardausgabe */
}
```

SIEHE AUCH

chmod(), close(), creat(), dup(), fcntl(), lseek(), read(), umask(), write(), <fcntl.h>, <sys/stat.h>, <sys/types.h>.

NAME

opendir - open directory
Dateiverzeichnis öffnen

DEFINITION

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

DIR *opendir(dirname)
char *dirname;
```

BESCHREIBUNG

Die Funktion *opendir()* öffnet einen Dateiverzeichnisstrom, entsprechend dem durch das Argument *dirname* angegebenen Dateiverzeichnis. Der Dateiverzeichnisstrom wird auf den ersten Eintrag positioniert. Der Datentyp *DIR*, der in *<dirent.h>* definiert ist, repräsentiert einen *Dateiverzeichnisstrom*, der eine geordnete Folge aller Dateiverzeichnis-Einträge in einem speziellen Dateiverzeichnis ist. Der Datentyp DIR ist unter SINIX durch eine Dateikennzahl implementiert. Daher können Anwendungen höchstens {OPEN_MAX} Dateien und Dateiverzeichnisse öffnen. Da eine Dateikennzahl verwendet wird, wird das Kennzeichen FD_CLOEXEC für diese Dateikennzahl gesetzt (siehe auch *<fcntl.h>*).

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *opendir()* einen Zeiger auf ein Objekt vom Typ *DIR*. Andernfalls wird der Nullzeiger zurückgeliefert und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *opendir()* schlägt fehl, wenn gilt:

[EACCES] Die Durchsuchberechtigung für eine Komponente von *dirname* oder die Leseberechtigung für *dirname* wird nicht erteilt.

[EMFILE] Für den Prozeß sind derzeit zuviele Dateikennzahlen offen.

[ENAMETOOLONG]
Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENFILE] Die Dateitabelle des Systems ist voll.

[ENOENT] Das Argument *dirname* zeigt auf den Namen einer Datei die nicht existiert oder auf die leere Zeichenkette.

[ENOTDIR] Eine Komponente von *dirname* ist kein Dateiverzeichnis.

HINWEIS

Die Funktion *opendir()* sollte in Verbindung mit *readdir()*, *closedir()* und *rewinddir()* verwendet werden, um den Inhalt eines Dateiverzeichnisses zu untersuchen (siehe auch Abschnitt BEISPIEL unter *readdir()*). Diese Methode wird aus Portabilitätsgründen empfohlen.

PORTABILITÄT

Die Funktion *opendir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

closedir(), *readdir()*, *rewinddir()*, *<dirent.h>*, *<sys/types.h>*.

NAME

fpathconf, pathconf - get configurable pathname variables
konfigurierbare Pfadnamen-Variablen ermitteln

DEFINITION

```
#include <unistd.h>

long pathconf (path, name)
char *path;
int name;

long fpathconf (fildes, name)
int fildes, name;
```

BESCHREIBUNG

Die Funktionen *pathconf()* und *fpathconf()* stellen eine Methode für eine Anwendung zur Verfügung, um den aktuellen Wert einer konfigurierbaren Systemgrenze oder Option (variabel) zu bestimmen, die einer Datei oder einem Dateiverzeichnis zugeordnet ist.

Für *pathconf()* zeigt das Argument *path* auf den Pfadnamen einer Datei. Für *fpathconf()* ist das Argument *fildes* die Dateikennzahl einer offenen Datei.

Das Argument *name* repräsentiert die Variable, die abgefragt werden soll, relativ zu dieser Datei. SINIX unterstützt alle die Variablen, die in der folgenden Tabelle aufgeführt sind. Andere X/Open-kompatible Implementierungen können weitere Variablen unterstützen. Die Variablen in der folgenden Tabelle entstammen den Dateien *<limits.h>* oder *<unistd.h>*. Die symbolischen Konstanten sind definiert in *<unistd.h>*. Sie sind die entsprechenden Werte für das Argument *name*:

Variable	Wert von name	Bemerkungen
LINK_MAX	_PC_LINK_MAX	1
MAX_CANON	_PC_MAX_CANON	2
MAX_INPUT	_PC_MAX_INPUT	2
NAME_MAX	_PC_NAME_MAX	3,4
PATH_MAX	_PC_PATH_MAX	4,5
PIPE_BUF	_PC_PIPE_BUF	6
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2

Bemerkungen:

1 Wenn *path* oder *fildes* auf ein Dateiverzeichnis verweist, dann bezieht sich der Rückgabewert auf das Dateiverzeichnis selbst.

2 Das Verhalten der Funktion ist undefiniert, falls *path* oder *fildes* nicht auf eine Gerätedatei für eine Datensichtstation verweist.

3 Falls *path* oder *fildes* auf ein Dateiverzeichnis verweist, bezieht sich der Rückgabewert auf Dateinamen im Dateiverzeichnis.

4 Das Verhalten ist undefiniert, wenn *path* oder *fildes* nicht auf ein Dateiverzeichnis verweist.

5 Wenn *path* oder *fildes* auf ein Dateiverzeichnis verweist, dann ist der Rückgabewert die maximale Länge eines relativen Pfadnamens wenn das angegebene Dateiverzeichnis das aktuelle Dateiverzeichnis ist.

6 Wenn *path* auf eine FIFO verweist, oder *fildes* auf eine Pipe oder FIFO verweist, dann bezieht sich der Rückgabewert auf das referenzierte Objekt. Wenn *path* oder *fildes* auf ein Dateiverzeichnis verweist, dann bezieht sich der Rückgabewert auf irgendeine existierende oder innerhalb des Dateiverzeichnisses erzeugte FIFO. Falls *path* oder *fildes* auf irgendeinen anderen Dateityp verweist, ist das Verhalten undefiniert.

7 Wenn *path* oder *fildes* auf ein Dateiverzeichnis verweist, dann bezieht sich der Rückgabewert auf irgendwelche, in diesem Standard definierten Dateien, die keine Dateiverzeichnisse sind und innerhalb des Dateiverzeichnisses existieren oder erzeugt werden können.

ERGEBNIS

Falls *name* einen ungültigen Wert hat liefern sowohl *pathconf()* als auch *fpathconf()* als Ergebnis -1;

Falls die Variable zu *name* keine Grenze für *path* oder die Dateikennzahl hat, liefert sowohl die Funktion *pathconf()* als auch die Funktion *fpathconf()* das Ergebnis -1 ohne die Variable *errno* zu ändern. Die Funktionen *pathconf()* und *fpathconf()* liefern den Wert −1 wenn gilt:

- die Implementierung muß *path* oder *fildes* benutzen, um den Wert von *name* zu bestimmen und die Implementierung unterstützt die Zuordnung von *name* zu der durch *path* oder *fildes* angegebenen Datei nicht,
- oder der Prozeß besitzt nicht die entsprechenden Rechte, um die durch *path* oder *fildes* angegebene Datei zu überprüfen,
- oder *path* existiert nicht,
- oder *fildes* ist keine gültige Dateikennzahl.

Andernfalls geben die Funktionen *pathconf()* bzw. *fpathconf()* den aktuellen Variablenwert für die Datei oder das Dateiverzeichnis zurück, ohne *errno* zu verändern. Der zurückgegebene Wert ist nicht eingeschränkter, als der entsprechende Wert, der in der Anwendung verfügbar wäre, wenn sie mit *<limits.h>* oder *<unistd.h>* der jeweiligen Implementierung übersetzt worden wären.

FEHLER

Die Funktion *pathconf()* schlägt fehl wenn gilt:

[EINVAL] Der Wert von *name* ist ungültig. Oder die Implementierung unterstützt die Zuordnung von *name* zur angegebenen Datei nicht.

Die Funktion *pathconf()* kann fehlschlagen wenn gilt:

[EACCES] Das Suchrecht für eine Komponente des Pfadnamens ist nicht vorhanden.

[ENAMETOOLONG] Die Länge der Zeichenkette *path* überschreitet den Wert {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Die angegebene Datei existiert nicht oder das Argument *path* zeigt auf eine leere Zeichenkette.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

Die Funktion *fpathconf()* schlägt fehl wenn gilt:

[EINVAL] Der Wert von *name* ist ungültig. Oder die Implementierung unterstützt die Zuordnung von *name* zur angegebenen Datei nicht.

Die Funktion *fpathconf()* kann fehlschlagen wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sysconf(), *<limits.h>*, *<unistd.h>*.

NAME

pause - suspend process until signal is received
Prozeß anhalten, bis Signal eintrifft

DEFINITION

```
int pause()
```

BESCHREIBUNG

Die Funktion *pause()* unterbricht den aufrufenden Prozß bis zur Zustellung eines Signal, dessen Aktion entweder die Ausführung einer Signalbehandlungsfunktion oder die Prozeßbeendigung ist.

Wenn die Aktion die Beendigung des Prozesses ist, dann kehrt die Funktion *pause()* nicht zurück.

Wenn die Aktion die Ausführung einer Signalbehandlungsfunktion ist, dann kehrt die Funktion *pause()* zurück, nachdem die Signalbehandlungsfunktion zurückkehrt.

ERGEBNIS

Da die Funktion *pause()* die Prozeßausführung für unbestimmte Zeit anhält, gibt es kein Ergebnis für die erfolgreiche Beendigung. Der Wert -1 wird zurückgegeben und *errno* wird gesetzt, um einen Fehler anzuzeigen.

FEHLER

Die Funktion *pause()* schlägt fehl, wenn gilt:

[EINTR] Während der Funktion wurde ein Signal abgefangen und die Kontrolle wurde von der Signalbehandlungsfunktion zurückgegeben.

PORTABILITÄT

Die Funktion *pause()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Programm simuliert die Funktionsweise von *sleep()* mit Hilfe von *pause()*. Das übersetzte Programm rufen Sie auf wie folgt:

a.out n

wobei n eine Zahl ist, mit der Sie angeben, wie lange das Programm "schlafen" soll.

```
#include <stdio.h>
#include <ctype.h>
#include <setjmp.h>
#include <signal.h>

static jmp_buf _jump;


static int catch(sig)
int sig;
{
   longjmp(_jump, 1);
}


int main(argc, argv)
int argc;
char *argv[ ];
{  unsigned ret, msek, sleep2();

   if (argv[1] == NULL)
   {  printf("\n Bitte als Argument von a.out eine(!) Zahl angeben!\n");
      exit(1);
   }

   if ( (msek= (unsigned) atoi(argv[1])) <= 0 )
      /* msek ist Zahl der Sekunden, die geschlafen werden soll */
   {  printf("\n Bitte als Argument von a.out eine Zahl>0 angeben!\n");
      exit(1);
   }

   ret = sleep2(msek);
   printf("\nIch habe %d Sekunden geschlafen.\n", ret);
}

unsigned sleep2(sek)
unsigned sek;
{  unsigned rsek;
   int (*sigfkt) ();

   rsek=alarm(sek);
      /*rsek ist Restzeit in der Alarmuhr */
   sigfkt = signal(SIGALRM, catch);

   if (setjmp(_jump) == 0)
   {  /* Einstellen des Sprung-Puffers erfolgt */
      pause();
      /* wird nicht durchlaufen: */
      return(0);
   }
   else
   {  /* von der Funktion catch() zurueckgekehrt */
      signal(SIGALRM, sigfkt);
         /* Alte Signalbehandlung wieder herstellen */

      /* Wenn noch Rest-Alarmzeit vorhanden, diese wieder einstellen */
      if (rsek > sek)
      {  sek = rsek;
         alarm (sek);
      }
      else
         alarm (1);
      return(sek);
   }
}
```

SIEHE AUCH

sigsuspend().

NAME

pclose - close a pipe stream to oder from a process
Pipe schließen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *pclose()* schließt den Datenstrom *stream* und wartet auf die Beendigung des zugeordneten Prozesses.

ERGEBNIS

Die Funktion *pclose()* liefert den Endestatus des zugeordneten Prozesses. Die Funktion *pclose()* liefert den Wert -1, wenn *stream* nicht durch ein *popen()* erzeugt wurde.

FEHLER

Die Funktion *pclose()* schlägt fehl, wenn gilt:

[ECHILD] Der Endestatus des Sohnprozesses konnte nicht ermittelt werden.

HINWEIS

Ein mit *popen()* geöffneter Datenstrom sollte mit *pclose()* geschlossen werden.

PORTABILITÄT

Die Funktion *pclose()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Die meisten anderen existierenden Implementierungen von *pclose()* verwenden *wait()*, um den Endestatus des Sohnprozesses zu ermitteln. Aus diesem Grund kann unter solchen Implementierungen der Status eines anderen Prozesses, der mit *fork()* oder einem weiteren *popen()* gestartet wurde, geliefert werden.

BEISPIEL

Siehe das Beispiel unter *popen()*.

SIEHE AUCH

fork(), *popen()*, *wait()*, *<stdio.h>*.

NAME

perror - write error messages to standard error
Fehlermeldung ausgeben

DEFINITION

```
#include <stdio.h>

void perror (s)
char *s;
```

BESCHREIBUNG

Die Funktion *perror()* bildet die Fehlernummer in der externen Variablen *errno* auf eine von der jeweiligen Landessprache abhängige Fehlermeldung ab, die wie folgt auf den Datenstrom für die Standard-Fehlerausgabe geschrieben wird: zuerst wird, wenn *s* nicht der Nullzeiger und das Zeichen, auf das *s* zeigt nicht das Nullbyte ist, die Zeichenkette ausgegeben, auf die *s* zeigt. Darauf folgt ein Doppenpunkt und ein Leerzeichen; dann wird eine Fehlermeldung, gefolgt von einem Neue-Zeile-Zeichen ausgegeben. Die Inhalte der Fehlermeldungen sind dieselben, die von der Funktion *strerror()* mit dem Argument *errno* ausgegeben werden.

Die Sprache für die Fehlermeldungen, die von *perror()* geschrieben werden, richtet sich unter SINIX nach der Umgebungsvariablen LANG.

ERGEBNIS

Die Funktion *perror()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler für diese Funktion definiert.

PORTABILITÄT

Die Funktion *perror()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Nicht alle X/Open-kompatiblen Implementierungen geben landessprachenspezifische Fehlermeldungen aus!

SIEHE AUCH

strerror(), *<stdio.h>*.

NAME

pipe create an interprocess channel
Pipe erzeugen

DEFINITION

```
int pipe (fildes)
int fildes[2];
```

BESCHREIBUNG

Die Funktion *pipe()* erzeugt eine Pipe und trägt zwei Dateikennzahlen, die auf die offenen Dateibeschreibungen für die Lese- bzw. Schreibseite der Datei verweisen, in die Argumente *fildes[0]* und *fildes[1]* ein. Diese beiden ganzzahligen Werte sind die beiden zum Zeitpunkt des Aufrufs der Funktion *pipe()* niedrigsten verfügbaren. Das Bit O_NONBLOCK ist für keine der beiden Dateikennzahlen gesetzt (die Funktion *fcntl()* kann verwendet werden, um das Bit O_NONBLOCK zu setzen).

Daten können dann über die Dateikennzahl *fildes[1]* geschrieben und über die Dateikennzahl *fildes[0]* gelesen werden. Ein Lesevorgang über die Dateikennzahl *fildes[0]* greift auf die Daten zu, die über die Dateikennzahl *fildes[1]* geschrieben wurden, und zwar nach der Methode first-in-first-out.

Ein Prozeß hat die Pipe zum Lesen geöffnet, wenn er die Dateikennzahl besitzt, die auf die Leseseite der Pipe verweist, d.h. *fildes[0]* (entsprechend zum Schreiben bei der Schreibseite, *fildes[1]*).

Bei erfolgreicher Beendigung markiert die Funktion *pipe()* die Felder *st_atime*, *st_ctime* und *st_mtime* der Pipe zum Ändern.

Das Bit FD_CLOEXEC ist für keine der beiden Dateikennzahlen gesetzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird der Wert -1 zurückgeliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *pipe()* schlägt fehl, wenn gilt:

[EMFILE] Für den Prozeß sind derzeit bereits {OPEN_MAX}-2 Dateikennzahlen offen.

[ENFILE] Die Anzahl der gleichzeitig geöffneten Dateien im System würde eine systemabhängige Grenze überschreiten.

PORTABILITÄT

Die Funktion *pipe()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Eine Einweg-Prozeßverbindung vom Vater zum Sohn baut man wie folgt auf:

1. Der Vaterprozeß erzeugt eine Pipe durch den Aufruf *pipe*(datkz).
2. Der Vaterprozeß erzeugt einen Sohnprozeß mittels *fork*(2).
3. Der Vaterprozeß schließt die Leseseite der Pipe durch den Aufruf *close*(datkz[0]).
4. Der Sohnprozeß schließt die Schreibseite der Pipe durch den Aufruf *close*(datkz[1]).

In folgendem Programm ist dieses Schema ausgeführt: Dabei liest der Vater von der Standardeingabe und schreibt die Zeichen in die Pipe. Der Sohn liest die Zeichen aus der Pipe, quittiert deren Empfang mit "Zeichen empfangen" und gibt das - sofern es druckbar ist auf die Standardausgabe.

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

int     datkz[2];
FILE    *inpipe, *outpipe, *fdopen();
int     sohn;
   /* Sohnprozess */
int     ch;
int     status;
  /* fuer wait */
struct stat puffer;

main()
{
   if (pipe(datkz) != 0)
      /* Pipe konnte nicht eroeffnet werden */
      exit(1);

   /* Information ueber die Pipe ausgeben: */

   fstat(datkz[1],&puffer);
   printf("Benutzernummer: %d\n",puffer.st_uid);
   printf("Groesse in Bytes: %ld\n",puffer.st_size);
   printf("Indexnummer: %d\n",puffer.st_ino);

   switch(sohn=fork())
   {  /* Fehler */
      case -1: exit(1);
         break;

      /* Sohn liest aus Pipe, quittiert den Empfang: */
      case  0: if((inpipe = fdopen(datkz[0],"r")) == NULL)
         /* Dateizeiger fuer Leseseite einrichten */
         /* falls Fehler: */
         exit(1);
         fclose(stdin);
            /* stdin wird nicht mehr benoetigt */
         close(datkz[1]);
            /* Sohn schliesst die Schreibseite */
         setbuf(inpipe,NULL);
            /* Pufferung umgehen, damit der Effekt */
            /*  deutlicher wird                    */
         while((ch=getc(inpipe)) != EOF)
            /* (Sobald der Vater die Schreibseite   */
            /*   schliesst, empfaengt der Sohn EOF) */
         {
            printf("\nZeichen empfangen: ");
            putchar(ch);
         }

         fclose(inpipe);
         exit(1);
```

```
        /* Vater schreibt auf Pipe: */
        default: if((outpipe= fdopen(datkz[1],"w")) == NULL)
           /* Dateizeiger fuer Schreibseite einrichten */
           /* falls Fehler: */
           exit(1);

           fclose(stdout);
              /* stdout wird nicht mehr benoetigt */
           close(datkz[0]);
              /* Leseseite schliessen */
           setbuf(outpipe,NULL);
              /* Vater soll auf die Pipe schreiben, */
              /*  Pufferung ausschalten             */

           while((ch=getchar()) != EOF)
              putc(ch,outpipe);
           fclose(outpipe);
              /* (Sohn erkennt EOF und beendet sich auch) */
           wait(&status);
    }
}
```

SIEHE AUCH

fcntl(), *read()*, *write()*.

NAME

popen - initiate pipe stream to oder from a process
Pipe zu einem Prozeß einrichten

DEFINITION

```
#include <stdio.h>

FILE *popen (command, type)
char *command, *type;
```

BESCHREIBUNG

Die Argumente von *popen()* sind Zeiger auf Zeichenketten, die mit dem Nullbyte abgeschlossen sind, und die eine Aufrufzeile und eine Ein-/Ausgabe-Art, entweder "r" für Lesen oder "w" für Schreiben, enthalten. Die Funktion *popen()* erzeugt eine Pipe zwischen dem aufrufenden Prozeß und dem auszuführenden Kommando *command*. Das Ergebnis ist ein Zeiger auf einen Datenstrom, der es erlaubt (wenn *type* gleich "w" ist), auf die Standardeingabe des Kommandos zu schreiben, indem auf diesen Datenstrom geschrieben wird; und der es erlaubt (wenn *type* gleich "r" ist), von der Standardausgabe des Kommandos zu lesen, indem von diesem Datenstrom gelesen wird. Wenn *command* nicht ausgeführt werden kann, schlägt ein Schreiben oder Lesen fehl.

Weil offene Dateien gemeinsam genutzt werden, kann ein Kommando des Typs "r" als Eingabefilter und ein Kommando des Typs "w" als Ausgabefilter verwendet werden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *popen()* einen Zeiger auf einen Datenstrom, andernfalls liefert *popen()* den Nullzeiger, wenn Dateien oder Prozesse nicht erzeugt werden können.

HINWEIS

Gepuffertes Lesen vor dem Öffnen eines Eingabefilters kann die Standardeingabe für diesen Filter falsch positioniert darstellen. Ähnliche Probleme mit einem Ausgabefilter können durch vorsichtiges Leeren von Puffern, z.B. mit *fflush()* vermieden werden.

PORTABILITÄT

Die Funktion *popen()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Ein Text wird von der Standardeingabe gelesen und in die Datei *datei* geschrieben, wobei Großbuchstaben in Kleinbuchstaben verwandelt werden (Eingabefilter):

```
#include <stdio.h>

main()
{
   FILE *in,*popen(),*dz;
   int  c;

   dz =fopen("datei","w");

                                /* Pipe wird zum Lesen geoeffnet */
                                /* der aufrufende Prozess liest seine Ein-
                                   gabe von der Standardausgabe des Kom-
                                   mandos tr */
   in = popen("tr \"[A-Z]\" \"[a-z]\"","r");
                                /* Im sie-Universum sind '[' und ']'
                                   nicht notwendig */

   while((c=getc(in)) != EOF)
      fprintf(dz,"%c",c);

   pclose(in);
   fclose(dz);
}
```

2. Beispiel:
Eine besonders nützliche Anwendung der Kommandos *popen()* und *pclose()* besteht in der direkten Ausgabe auf den Drucker. Da Druckausgaben unter SINIX gespoolt werden, ist es nicht ratsam, direkt auf die Gerätedatei des Druckers zu schreiben. Statt dessen sollte eine Druckerausgabe immer über das Spoolsystem, d.h. über das Kommando *lpr* erfolgen. Dies kann mit der Funktion *popen()* erreicht werden. Das nachfolgende Beispiel öffnet mit *popen()* eine Pipe zum Kommando *lpr* und schreibt dann alle Eingaben des Benutzers auf den Drucker (gespoolt):

```
#include <stdio.h>

int main()
{  int    ch;
   FILE  *printer;

   if ((printer=popen("lpr","w")) == NULL)
   {  fprintf(stderr,
            "Pipe zum Druckerkommando kann nicht geoeffnet werden\n");
      return 1;
   }
   else
   {  printf("Text fuer die Druckerausgabe eingeben:\n")
      while ((ch=getchar()) != EOF)
         fputc(ch, printer);
      pclose(printer);
   }
   return 0;
}
```

SIEHE AUCH

pclose(), *pipe()*, *system()*, *<stdio.h>*.

NAME

pow - power function
Potenzierfunktion

DEFINITION

```
#include <math.h>

double pow (x, y)
double x, y;
```

BESCHREIBUNG

Die Funktion *pow()* berechnet den Wert von x zur Potenz y, x^y. Wenn x gleich 0 ist, dann muß y positiv sein. Wenn x negativ ist, dann muß y ein ganzzahliger Wert sein.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *pow()* den Wert x zur Potenz y.

Wenn x gleich 0.0 und y gleich 0.0 ist, dann wird der Wert 1.0 zurückgeliefert.

Wenn x oder y ein NaN ist, dann wird ein NaN zurückgeliefert.

Wenn x negativ ist und y kein ganzzahliger Wert, dann wird der Wert 0.0 zurückgeliefert und *errno* ist gleich [EDOM] gesetzt.

Wenn x gleich 0.0 und y negativ ist, dann wird -HUGE_VAL zurückgeliefert und *errno* ist gleich [EDOM] gesetzt.

Wenn das Ergebnis einen Über- oder Unterlauf verursachen würde, dann wird HUGE_VAL zurückgeliefert und *errno* ist gleich [ERANGE] gesetzt.

Andernfalls wird entweder *errno* gesetzt, um den Fehler anzuzeigen oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktion *pow()* schlägt fehl, wenn gilt:

[EDOM] Der Wert von x ist negativ und y ist nicht ganzzahlig, x ist gleich 0.0 und y ist negativ.

[ERANGE] Der zurückgelieferte Wert würde einen Über- oder Unterlauf verursachen.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *pow()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis gleich HUGE_VAL oder ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *pow()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Berechne x^y für eingelesene Argumente x und y:

```
#include <math.h>

main()
{
   double x,y,z;

   scanf("%F%F",&x,&y);

   z=pow(x,y);
   perror("pow");

   printf("%g**%g : %g\n", x,y,z);
}
```

SIEHE AUCH

exp(), *isnan()*, *matherr()*, *<math.h>*.

NAME

fprintf, printf, sprintf - print formatted output
Formate Ausgabe

DEFINITION

```
#include <stdio.h>

int printf (format, ...)
char *format;

int fprintf (stream, format, ...)
FILE *stream;
char *format;

int sprintf (s, format, ...)
char *s, *format;
```

BESCHREIBUNG

Die Funktion *printf()* schreibt Ausgaben auf die Standardausgabe *stdout*. Die Funktion *fprintf()* schreibt Ausgaben auf den genannten Ausgabestrom *stream*. Die Funktion *sprintf()* schreibt Ausgaben, gefolgt vom Nullbyte '\0', in aufeinanderfolgende Bytes, beginnend an der Adresse *s*; es liegt in der Verantwortung des Benutzers, daß genügend Platz für die Ausgabe verfügbar ist.

Jede dieser Funktionen konvertiert, formatiert und gibt ihre Argumente unter der Kontrolle von *format* aus. *format* ist dabei eine Zeichenkette, die zwei Arten von Objekten enthalten kann: normale Zeichen, die einfach in den Ausgabedatenstrom kopiert werden, sowie Umwandlungsanweisungen, von denen jede dafür sorgt, daß keines oder mehr der Argumente geholt wird. Die Ergebnisse sind undefiniert, wenn weniger Argumente als in *format* festgelegt, übergeben werden. Wenn *format* bereits vollständig abgearbeitet wurde, aber noch weitere Argumente verbleiben, dann werden die überflüssigen Argumente einfach ignoriert.

Die Umwandlung kann auf das *n*-te Argument der Argumentliste anstelle des nächsten, unbenutzten Arguments angewendet werden. In diesem Fall wird das Umwandlungszeichen % (siehe auch unten) durch die Sequenz *%ziffer$*, wobei *ziffer* eine Dezimalzahl *n* aus dem Bereich [1, {NL_ARGMAX}] ist, welche die Position des Arguments in der Argumentliste angibt. Diese Eigenschaft erlaubt die Definition von Formatzeichenketten, die Argumente entsprechend einer speziellen Sprache auswählen (siehe auch BEISPIEL).

Alle Formen von *printf()* erlauben das Einfügen eines landessprach-spezifischen Dezimalpunkts bzw. Exponentenzeichens in die Ausgabezeichenkette. Der Dezimalpunkt und das Exponentenzeichen werden von den *langinfo*-Daten in der internationalen Umgebung des Progamms definiert (Kategorie LC_NUMERIC). In der Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, in der Dezimalpunkt und Exponentenzeichen nicht definiert sind, gilt die Voreinstellung '.' bzw. 'e' oder 'E'.

Jede Umwandlungsanweisung wird entweder vom Zeichen % oder von der Sequenz *%ziffer$* eingeleitet; darauf folgen die nachfolgend angegebenen Daten:

Keines oder mehr Kennzeichen, die die Bedeutung der Umwandlungsanweisung verändern.

Eine optionale Dezimalzahl, die eine minimale Feldbreite angibt. Wenn der umgewandelte Wert aus weniger Zeichen als der Feldbreite besteht, dann wird links bis zur Feldbreite aufgefüllt (bzw. rechts, wenn das Kennzeichen '-' für linksbündige Ausrichtung angegeben wurde, siehe unten).

Eine Genauigkeit, die angibt, wieviele Ziffern mindestens für die Umwandlungen *d*, *i*, *o*, *u*, *x* oder *X* erscheinen sollen, wieviele Ziffern nach dem Dezimalpunkt für die Umwandlungen *e*, *E* und *f* erscheinen sollen, wieviele signifikante Stellen bei den Umwandlungen *g* und *G* vorhanden sind oder wieviele Zeichen maximal aus der Zeichenkette für die Umwandlung *s* ausgegeben werden sollen. Die Genauigkeit hat die Form '.', gefolgt von einer Zeichenkette aus dezimalen Ziffern, wobei keine Ziffer als 0 gewertet wird.

Ein optionales *h*, das angibt, daß ein folgendes *d*, *i*, *o*, *u*, *x* oder *X* auf ein Argument vom Typ *short int* oder *unsigned short int* angewendet werden soll (das Argument ist entsprechend der ganzzahligen Erweiterung erweitert worden und sein Wert wird vor der Ausgabe in ein *short int* oder *unsigned short int* umgewandelt); ein optionales *h*, welches angibt, daß ein nachfolgendes *n* auf einen Zeiger auf ein Argument vom Typ *short int* angewendet werden soll; ein optionales *l* (ein kleines L), welches angibt, daß ein folgendes *d*, *i*, *o*, *u*, *x* oder *X* auf ein Argument vom Typ *long int* oder *unsigned long int* angewendet werden soll; ein optionales *l*, welches angibt, daß ein nachfolgendes *n* auf einen Zeiger auf ein Argument vom Typ *long int* angewendet werden soll.

Wenn *h* oder *l* mit irgendeinem anderen Umwandlungszeichen auftritt, dann ist das Verhalten undefiniert.

Portabilitäts-Hinweis:
Andere X/Open-kompatible Systeme, die den Typ *long double* unterstützen, können zusätzlich zu *l* auch ein optionales *L* zulassen, welches angibt, daß ein folgendes *e*, *E*, *f*, *g* oder *G* auf ein Argument vom Typ *long double* angewendet werden soll.

Ein Zeichen, daß den Typ der durchzuführenden Umwandlung angibt.

In Formatzeichenketten, die die Form % für eine Umwandlungsanweisung enthalten, kann eine Feldbreite oder Genauigkeit durch das Zeichen * anstelle einer Ziffernfolge angegeben werden. In diesem Fall wird ein ganzzahliges Argument die Feldbreite oder Genauigkeit festlegen. Das umzuwandelnde Argument wird nicht eher geholt, als bis das entsprechende Umwandlungszeichen gefunden ist, so daß die Argumente, die Feldbreite und Genauigkeit angeben, *vor* dem umzuwandelnden Argument, sofern vorhanden, erscheinen müssen.

In Formatzeichenketten, welche die Form *%ziffer$* der Umwandlungsanweisung enthalten, kann eine Feldbreite oder Genauigkeit durch die Sequenz **ziffer$* angegeben werden, wobei *ziffer* eine Dezimalzahl aus dem Intervall [1, {NL_ARGMAX}] ist, welche die Position einer Ganzzahl in der Argumentliste angibt, die ihrerseits die Feldbreite oder Genauigkeit angibt, z.B.:

printf ("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);

format kann entweder numerierte oder nichtnumerierte Argument-Angaben enthalten, d.h. entweder *%ziffer$* und **ziffer$* oder % und *, aber nicht beides. Die Ergebnisse einer Vermischung von numerierten und unnumerierten Argument-Angaben in einer *format*-Zeichenkette sind undefiniert.

Die Verwendung numerierter Argument-Angaben für das Argument *N* erfordert, daß alle führenden Argumente, vom ersten bis zum (*N*-1)., in der Formatzeichenkette angegeben werden.

Die Kennzeichen und Ihre Bedeutung sind:

'-' Das Ergebnis der Umwandlung wird linksbündig innerhalb des Felds ausgerichtet.

'+' Das Ergebnis einer vorzeichenbehafteten Umwandlung beginnt immer mit einem Vorzeichen ('+' oder '-').

' ' (Leerzeichen) Wenn das erste Zeichen einer vorzeichenbehafteten Umwandlung kein Vorzeichen ist, dann wird dem Resultat ein Leerzeichen vorangestellt. Dies bedeutet, daß das Leerzeichen ignoriert wird, wenn das Leerzeichen und das Zeichen '+' beide angegeben werden.

'#' Dieses Kennzeichen gibt an, daß der umzuwandelnde Wert in einer "anderen Form" darzustellen ist. Für *c*, *d*, *i*, *s* und *u* hat dieses Kennzeichen keine Wirkung. Für die Umwandlung *o* wird die Genauigkeit in der Form erhöht, daß die erste Ziffer des Ergebnisses die Ziffer '0' ist. Für *x* (oder *X*) wird einem Resultat ungleich 0 die Sequenz *"0x"* (oder *"0X"*) vorangestellt. Für *e*, *E*, *f*, *g* oder *G* enthält das Ergebnis immer einen Dezimalpunkt, auch wenn keine weiteren Ziffern folgen (normalerweise erscheint ein Dezimalpunkt nur dann im Ergebnis, wenn er von einer Ziffer gefolgt wird). Für *g* und *G* werden abschließende Nullen nicht aus dem Ergebnis entfernt, wie sonst üblich.

'0' Für *d*, *i*, *o*, *u*, *x*, *X*, *e*, *E*, *f*, *g* und *G* werden führende Nullen, nach einer vorhandenen Anzeige von Vorzeichen oder Basis, verwendet, um bis zur Feldbreite aufzufüllen; es wird nicht mit Leerzeichen aufgefüllt. Wenn sowohl die Kennzeichen *0* als auch - angegeben werden, dann wird das Kennzeichen *0* ignoriert. Für *d*, *i*, *o*, *u*, *x* und *X* wird, wenn eine Genauigkeit angegeben ist, das Kennzeichen *0* ignoriert. Für andere Umwandlungen ist das Verhalten undefiniert.

Die Umwandlungszeichen und ihre Bedeutung sind:

d, i, o, u, x, X
: Das Argument vom Typ *int* wird in eine vorzeichenbehaftete Ganzzahl (*d* oder *i*), eine vorzeichenlose Oktal- (*o*), Dezimal- (*u*) oder Hexadezimalzahl (*h* oder *H*) umgewandelt; die Buchstaben *abcdef* werden für die Umwandlung *x*, die Buchstaben *ABCDEF* für die Umwandlung *X* verwendet. Die Genauigkeit gibt die Minimale Anzahl von Ziffern an, die ausgegeben werden sollen; wenn der umgewandelte Wert durch weniger Ziffern dargestellt werden kann, wird er mit führenden Nullen ergänzt. Die Voreinstellung für die Genauigkeit ist 1. Das Ergebnis einer Umwandlung des Werts 0 mit einer ausdrücklich genannten Genauigkeit von 0 liefert kein Zeichen.

f
: Das Argument vom Typ *double* wird in die dezimale Schreibweise der Art *[-]ddd.ddd* umgewandelt, wobei die Anzahl der Ziffern nach dem Dezimalpunkt gleich der angegebenen Genauigkeit ist. Ist keine Genauigkeit angegeben, so wird diese mit dem Wert 6 angenommen; wenn die Genauigkeit gleich 0 ist, dann wird kein Dezimalpunkt ausgegeben. Wenn der Dezimalpunkt ausgegeben wird, dann wird zuvor mindestens eine Ziffer ausgegeben. Der Wert wird auf die entsprechende Zahl von Ziffern gerundet.

e, E
: Das Argument vom Typ *double* wird in die Form *[-]d.ddde+-dd* umgewandelt, wobei genau eine Ziffer vor dem Dezimalpunkt ausgegeben wird (die ungleich 0 ist, wenn das Argument ungleich 0 ist), und wobei die Anzahl der Nachkommastellen gleich der Genauigkeit ist; wenn die Genauigkeit fehlt, dann wird hier der Wert 6 angenommen; wenn die Genauigkeit gleich 0 ist, dann wird kein Dezimalpunkt ausgegeben. Der Wert wird auf die entsprechende Zahl von Ziffern gerundet. Das Umwandlungszeichen *E* erzeugt eine Zahl mit *E* anstelle von *e* für die Anzeige des Exponenten. Der Exponent enthält immer mindestens zwei Ziffern. Wenn der Wert gleich 0 ist, dann ist der Exponent gleich 0.

g, G Das Argument vom Typ *double* wird in der Form von *f* oder *e* umgewandelt (bzw. in der Form *E* für das Umwandlungszeichen *G*), wobei die Genauigkeit die Anzahl der signifikanten Stellen angibt. Wenn die angegebene Genauigkeit gleich 0 ist, dann wird der Wert 1 angenommen. Die Form hängt vom umgewandelten Wert ab; die Form *e* wird nur dann verwendet, wenn der Exponent, einer solchen Umwandlung kleiner als -4 oder größer gleich der Genauigkeit ist. Abschließende Nullen werden vom gebrochenen Teil des Ergebnisses entfernt; ein Dezimalpunkt erscheint nur dann, wenn er von einer Ziffer gefolgt wird.

c Das Argument vom Typ *int* wird in den Typ *unsigned char* umgewandelt, das resultierende Zeichen wird geschrieben.

s Das Argument ist vom Typ Zeiger auf *char*. Zeichen aus dem Vektor werden bis zum abschließenden Nullbyte geschrieben (ausschließlich); wenn die Genauigkeit angegeben ist, dann werden nicht mehr als diese Anzahl von Zeichen geschrieben. Wird die Genauigkeit nicht angegeben, oder ist diese größer als die Länge des Vektors, dann enthält der Vektor das Nullbyte.

p Das Argument ist ein Zeiger auf *void*. Der Wert des Zeigers wird in eine Folge von abdruckbaren Zeichen umgewandelt; unter SINIX ist dies die hexadezimale Darstellung der Adresse.

n Das Argument ist ein Zeiger auf ein *int*, in welches die Anzahl der bisher von diesem Aufruf einer der *printf()*-Funktionen geschriebenen Zeichen eingetragen wird. Es wird kein Argument umgewandelt.

% Es wird das Zeichen % ausgegeben; kein Argument wird umgewandelt.

Wenn das Zeichen nach % oder nach der Sequenz *%ziffer$* kein gültiges Umwandlungszeichen ist, dann ist das Ergebnis der Umwandlung undefiniert.

In keinem Fall verursacht eine nicht existierende oder zu kleine Feldbreite das Abschneiden eines Feldes; wenn das Ergebnis einer Umwandlung breiter als die Feldbreite ist, dann wird das Feld einfach erweitert, um die Ausgabe aufzunehmen. Zeichen die von *printf()* und *fprintf()* erzeugt werden, werden ausgegeben, als ob *putc()* aufgerufen werden würde.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *printf()* oder *fprintf()* und der nächsten erfolgreichen Beendigung eines Aufrufs von *fflush()* oder *fclose()* für denselben Datenstrom oder einem Aufruf von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Jede Funktion liefert die Anzahl der übertragenen Zeichen (ausschließlich des Nullbytes bei *sprintf()*), oder einen negativen Wert bei Auftreten eines Ausgabefehlers.

FEHLER

Für die Bedingungen, unter denen die Funktionen *printf()* und *fprintf()* fehlschlagen, siehe unter *fputc()*. Zusätzlich können die Funktionen *printf()*, *fprintf()*, und *sprintf()* in anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EINVAL] Es gibt nicht genügend Argumente.

HINWEIS

In Formatzeichenketten, die die Form mit % für die Umwandlungs-Anweisungen enthalten, wird jedes Argument der Argumentliste genau einmal verwendet.

In Formatzeichenketten, die die Form mit *%ziffer$* für die Umwandlungs-Anweisungen enthalten, kann jedes numerierte Argument der Argumentliste sooft verwendet werden, wie nötig.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die gängigsten Formate von *printf()* erklären sich durch ihre Anwendung in den übrigen Beispielprogrammen von selbst. Nachstehend finden Sie einige weitere Formatangaben einschließlich ihrer Wirkung aufgelistet. Zur Verdeutlichung ist das umgewandelte Ergebnis in >< eingerahmt.

Formatangabe	Argument(e)	Ergebnis
%.6s	"Konstanz"	>Konsta<
%10.5s	"Konstanz"	> Konst<
%-10.5s	"Konstanz"	>Konst <
%15.15s	"Konstanz"	> Konstanz<
%*.*s	20,7,"Konstanz"	> Konstan<
%-*.*s	15,10,"Konstanz"	>Konstanz <
%8d	721932	> 721932<
%-8d	721932	>721932 <
%+8d	+721932	> +721932<
%+8d	-721932	> -721932<
%*.*f	7,2,27.31928	> 27.32<
%-*.*f	3,2,27.31928	>27.32<
%-0*.*f	1,12,19.84	>19.84000000000<
%08.*f	2,10.6000000	>00010.60<
%-0*.*g	1,12,19.84	>19.84<
%e	1712.1961	>1.71220e+03<
%.10e	1712.1961	>1.71220e+03<
%10.10e	1712.1961	>1.7121961000e+03<

#pi soll bis zu 5 Stellen nach dem Dezimalpunkt ausgegeben werden:

```
printf("pi = %.5f", 4 * atan(1.0));
```

Es soll ein Datum mit Zeitangabe im Format Sunday, July 3, 10:02 ausgegeben werden, wobei *weekday* und *month* Zeiger auf mit Nullzeichen abgeschlossene Zeichenketten sind:

```
printf("%s, %s %d, %d:%.2d",
        weekday, month, day, hour, min);
```

Um das jeweilige, landessprachliche Datum- und Zeitformat auszugeben, kann die folgende Anweisung verwendet werden:

```
printf (format, weekday, month, day, hour, min);
```

Für amerikansche Zwecke könnte *format* ein Zeiger auf folgende Zeichenkette sein:

"%1$s, %2$s %3$d, %4$d:%5$.2d\n"

wodurch die folgende Meldung erzeugt wird:

Sunday, July 3, 10:02

wogegen für die deutsche Schreibweise *format* auf die folgende Zeichenkette zeigen könnte:

"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"

mit der Ausgabe:

Sonntag, 3. Juli, 10:02

SIEHE AUCH

putc(), *scanf()*, *setlocale()*, *<stdio.h>*, *Abschnitt 2.5*.

NAME

profil - execution time profile
Zeitprofil der Prozeßausführung

DEFINITION

```
void profil (puf, pufgr, dist, skal)
char *puf;
int pufgr, dist, skal;
```

BESCHREIBUNG

profil() erstellt ein Histogramm der Prozeßausführung, ein Prozeßprofil. Damit können Sie feststellen, wo das Programm "seine Zeit verbringt", und so den Programmablauf gegebenenfalls beschleunigen.

Das Argument *puf* zeigt auf einen Speicherbereich, dessen Länge mit *pufgr* (in Bytes) angegeben wird. Nach einem *profil()*-Aufruf wird der Befehlszähler des Benutzerprogramms bei jedem Takt ({CLOCK_TCK}-mal pro Sekunde) abgefragt. Von diesem Wert wird *dist* subtrahiert und das Ergebnis mit *skal* multipliziert. Entspricht der so gewonnene Wert einem Eintrag in *puf*, so wird dieser hochgezählt. Ein "Eintrag" ist hier als eine Bytefolge in der Länge *sizeof(short)* definiert.

Die Skalierung *skal* wird als vorzeichenlose Festkomma-Mantisse mit Komma ganz links interpretiert: 0177777 (oktal) ergibt ein eins-zu-eins Abbild von Befehlszähler auf Einträge in *puf*; 077777 (oktal) bildet je ein Befehlswort-Paar zusammen ab. 02 (oktal) bildet alle Befehle auf den Anfang von *puf* ab (wodurch eine nicht unterbrechende interne Uhr entsteht).

Die Ausführung von *profil()* wird in folgenden Fällen deaktiviert:

- Angabe von 0 oder 1 für *skal*
- Ausführung eines *exec*-Aufrufs
- wenn eine Aktualisierung in *puf* zu einem Speicherfehler führen würde.

Durch die Angabe von 0 für *pufgr* wird der *profil()*-Aufruf unwirksam. Nach einem *fork()*-Aufruf läuft *profil()* sowohl im Sohn- als auch im Vaterprozeß weiter.

ERGEBNIS

Die funktion *profil()* liefert kein Ergebnis.

HINWEIS

profil() wird nur bei der Entwicklung eines Anwendungsprogramms eingesetzt, um die Programmausführung zu untersuchen.

Normalerweise sollten Sie deshalb für Performance-Untersuchungen den Schalter *-p* beim Kommando *cc* verwenden. Dann wird eine Datei *mon.out* erstellt, die Sie mit *prof* auswerten können. *prof* gibt Ihnen eine Tabelle der CPU-Zeiten aus, die das Programm verbraucht hat.

PORTABILITÄT

Die Funktion *profil()* ist im X/OPEN-Standard nicht enthalten.

SIEHE AUCH

exec, *fork()*.

NAME

ptrace - process trace
Prozeßüberwachung

DEFINITION

```
int ptrace (auftrag, pid, adr, daten);
int auftrag, pid, daten;
/* Der Datentyp von adr ist von auftrag abhängig */
```

BESCHREIBUNG

Mit Hilfe der Funktion *ptrace()* kann ein Vaterprozeß die Ausführung eines Sohnprozesses überwachen. Hauptanwendungsgebiet ist die Fehlersuche mit Hilfe von Haltepunkten. Der Sohnprozeß läuft so lange normal ab, bis er durch ein Signal gestoppt wird (siehe auch *signal()*). Das Anhalten des Sohnprozesses wird dem Vaterprozeß über die *wait()*-Funktion mitgeteilt. Während sich der Sohnprozeß im Haltezustand befindet, kann der Vaterprozeß das "Speicherabbild" des Sohnprozesses mit Hilfe von *ptrace()* überprüfen und Änderungen vornehmen. Er kann den Sohnprozeß auch beenden oder das Signal, durch das der Sohn angehalten wurde, ignorieren und ihn weiterlaufen lassen.

Der Datentyp des Arguments *adr* ist abhängig von *auftrag*:

auftrag	— Datentyp *adr*	
1	int *zeiger	/* Zeiger auf eine Adresse im Sohnprozeß */
2	int *zeiger	/* Zeiger auf eine Adresse im Sohnprozeß */
3	int *zeiger	/* Zeiger auf eine Adresse im USER Bereich */
4	int *zeiger	/* Zeiger auf eine Adresse im Sohnprozeß */
5	int *zeiger	/* Zeiger auf eine Adresse im Sohnprozeß */
6	int *zeiger	/* Zeiger auf eine Adresse im USER Bereich des Sohnprozesses im System */
7	/* siehe Beschreibung von *auftrag* 7 */	
8	/* entfällt */	
9	/* siehe Beschreibung von *auftrag* 9 */	

Das Argument *auftrag* bestimmt, welche Aktion *ptrace()* durchführen soll, und kann einen der folgenden Werte annehmen:

0

Dieser *auftrag* muß vom Sohnprozeß abgesetzt werden, wenn er durch den Vaterprozeß überwacht werden soll. Damit wird das Trace Flag des Sohnes gesetzt. Das Trace Flag veranlaßt, daß der Sohnprozeß bei Empfang eines Signals in den Haltezustand versetzt wird, und nicht in den in *funk* angegebenen Zustand (s. *signal()*). Die Argumente *pid*, *adr* und *daten* werden ignoriert und es wird kein Ergebnis zurückgeliefert. Wenn der Vaterprozeß auf einen *ptrace()*-Aufruf nicht reagiert, kann dies zu eigenartigen Ergebnissen führen.

Die übrigen *auftrag*-Werte können nur vom Vaterprozeß verwendet werden. Dabei ist *pid* immer die Prozeßnummer des Sohnprozesses. Alle *aufträge* können erst dann abgesetzt werden, wenn sich der Sohnprozeß im Haltezustand befindet.

1, 2

Diese *aufträge* bewirken, daß das an der Stelle *adr* im Adreßraum des Sohnprozesses gespeicherte Wort gelesen und an den Vaterprozeß übergeben wird. Sind Befehlsbereich (Textbereich) und Datenbereich getrennt angelegt, so wird bei *auftrag* 1 ein Wort aus dem Befehlsbereich und bei *auftrag* 2 ein Wort aus dem Datenbereich gelesen. Sind Befehls- und Datenbereiche nicht getrennt angelegt, liefern beide *aufträge* die gleichen Ergebnisse. Das Argument *daten* wird ignoriert. Diese *aufträge* können nicht ausgeführt werden, wenn *adr* nicht die Startadresse eines Wortes ist. In diesem Fall wird −1 an den Vaterprozeß übergeben und sein *errno* wird auf [EIO] gesetzt.

3

Dieser *auftrag* bewirkt, daß das an der Stelle *adr* im Benutzerbereich (*user area* - uarea) für den Sohnprozeß im Adreßraum des Systems gespeicherte Wort gelesen und an den Vaterprozeß übergeben wird. Der Benutzerbereich existiert für jeden Prozeß und enthält Verwaltungsinformationen über den Prozeß. Der Benutzerbereich gehört zum virtuellen Adreßraum des Betriebssystems. Das Argument *daten* wird ignoriert. Dieser *auftrag* kann nicht ausgeführt werden, wenn *adr* nicht die Startadresse eines Wortes ist oder außerhalb des Benutzerbereichs liegt. In diesem Fall wird −1 an den Vaterprozeß übergeben und sein *errno* wird auf [EIO] gesetzt.

4, 5

Diese *aufträge* bewirken, daß der mit *daten* angegebene Wert an der Stelle *adr* in den Adreßraum des Sohnprozesses geschrieben wird. Sind Befehls- und Datenbereiche getrennt angelegt, so wird bei *auftrag* 4 ein Wort in den Befehlsbereich und bei *auftrag* 5 ein Wort in den Datenbereich geschrieben. Sind Befehls- und Datenbereiche nicht getrennt angelegt, liefern beide *aufträge* die gleichen Ergebnisse. Bei erfolgreicher Ausführung wird der in den Adreßraum geschriebene Wert an den Vater zurückgeliefert. Diese *aufträge* können nicht ausgeführt werden, wenn *adr* eine Stelle in einem reinen Prozedurbereich ist und dieser gerade von einem anderen Prozeß benutzt wird, oder wenn *adr* nicht die Startadresse eines Wortes ist. In diesen Fällen wird −1 an den Vaterprozeß geliefert und seine *errno* auf [EIO] gesetzt.

6

Mit diesem *auftrag* kann man bestimmte Einträge in den Benutzerbereich (*user area*) für den Sohnprozeß schreiben. Mit *daten* gibt man den zu schreibenden Wert an und mit *adr* die Stelle, an die er geschrieben werden soll. Mögliche Einträge sind:

- die Mehrzweckregister
- das Gleitpunkt-Zustandsregister und acht Gleitpunktregister
- bestimmte Bits des Prozessorstatus

7

Mit diesem *auftrag* wird der Sohnprozeß wieder gestartet. Hat *daten* den Wert 0, werden alle ausstehenden Signale einschließlich des Signals, durch das der Sohnprozeß gestoppt wurde, vor dem Wiederanlauf zurückgesetzt. Ist *daten* eine gültige Signalnummer, so reagiert der Sohnprozeß, als habe er dieses Signal empfangen, alle anderen ausstehenden Signale werden zurückgesetzt. Bei diesem *auftrag* muß *adr* den Wert 1 haben. Bei erfolgreicher Ausführung wird der Wert von *daten* dem Vater übergeben. Dieser *auftrag* kann nicht ausgeführt werden, wenn *daten* weder den Wert 0 hat noch eine gültige Signalnummer ist. In diesen Fällen wird −1 an den Vaterprozeß zurückgeliefert und sein *errno* wird auf [EIO] gesetzt.

8

Mit diesem *auftrag* wird der Sohnprozeß wie bei *exit*(2) beendet.

9
Dieser *auftrag* setzt das Trace Bit im Prozessorstatus des Sohnprozesses und führt danach die unter 7 beschriebenen Schritte durch. Das Trace Bit bewirkt eine Unterbrechung nach jedem ausgeführten Maschinenbefehl. Damit kann der Sohnprozeß wirksam Schritt für Schritt überwacht werden.

Um ein unberechtigtes Eindringen in das System via *ptrace()* zu verhindern, wird bei nachfolgenden *exec()* Aufrufen das s-Bit (set-user-id Bit) nicht ausgewertet. Setzt ein mit *ptrace()* bearbeiteter Prozeß einen *exec*-Aufruf ab, wird er mit dem Signal SIGTRAP angehalten, bevor der erste Befehl des neuen Speicherabbilds ausgeführt wird.

ERGEBNIS

Kann der Aufruf nicht ausgeführt werden, wird –1 geliefert und *errno* wird wie oben gesetzt. Die Ergebnisse bei erfolgreicher Ausführung sind abhängig vom *auftrag* (s.o.).

FEHLER

ptrace kann im allgemeinen nicht ausgeführt werden, wenn einer der folgenden Fehler auftrtritt:

[EIO]
auftrag hat einen ungültigen Wert (gültige Werte s.o.).

[ESRCH]
pid bezeichnet entweder einen nicht existierenden Sohnprozeß oder der Sohnprozeß hat keinen *ptrace* mit *auftrag* 0 ausgeführt.

HINWEIS

- *ptrace()* sollte nicht in Anwendungen, sondern nur in Programmen zur Softwarefehlersuche verwendet werden.
- *ptrace()* ist extrem hardwareabhängig.
- Die Bedeutung der im Abschnitt BESCHREIBUNG verwendeten Begriffe "Wort", "Stelle", "Startadresse" usw. ist hardwareabhängig.

PORTABILITÄT

Die Funktion *ptrace()* ist im X/OPEN-Standard nicht enthalten.

BEISPIEL

Im Beispielprogramm wird die Funktion *wait()* mehrfach aufgerufen, obwohl es nur einen Sohnprozeß gibt. Der Systemaufruf wird deshalb so benutzt, weil der Vaterprozeß jedesmal warten soll, bis der Sohnprozeß durch ein internes Signal gestoppt wird. Der Vaterprozeß soll nicht in seiner ganzen Länge parallel zum Sohnprozeß ablaufen. Diese mehrfachen Aufrufe von *wait()* beziehen sich hier also alle auf ein und denselben Sohnprozeß.

```
#include <stdio.h>
#include <signal.h>

main()
{
   int status, x,pnr,adr,i,inh;

   adr = 1;

   printf("Hier ist der Vaterprozess vor der Verzweigung\n");
   pnr = fork();

   if (pnr == -1)
   {
      printf("Sohnprozess nicht erzeugt\n");
      exit(1);
   }

   if (pnr == 0)
   {                                /* Hier im Sohnprozeß */
      ptrace(0);
      printf("Hier im Sohnprozess!!!\n");
      printf("Ergebnis ist %d\n",256/16);
      kill(getpid(), SIGINT);
                                    /* Hier wird durch Signal aus
                                       dem Sohn gesprungen */

      printf("Hier zum 2.Mal im Sohnprozess\n");
      kill(getpid(), SIGINT);
                                    /* Hier wird zum 2. Mal rausgesprungen */

      printf("Hier zum 3.Mal im Sohnprozess\n");

      for(i=0; i<5; i++)
         printf("HALLO ");
         printf("Hallo\n");
         kill(getpid(), SIGINT);
                                    /* Hier wird zum 3.Mal rausgesprungen */

      printf("Sohnschluss\n");
                                    /* Dieses Print-Kommando wird nicht mehr
                                       ausgeführt, da zu diesem Zeitpunkt das
                                       Tracing vom Vater bereits beendet ist */
   }
```

```
    else
    {                                   /* Hier im Vaterprozeß */
       while(wait(&status) != pnr);

       printf("Hier im Vater nach erstem Sprung aus Sohnprozess\n");
       ptrace(7,pnr,adr,0);

       while(wait(&status) != pnr);
                                        /* Vater soll warten, bis Sohn stoppt! */

       printf("Hier im Vater nach zweitem Sprung aus Sohnprozess\n");

       for(i=0;i<8;i=i+2)
       {
        inh = ptrace(1,pnr,i);
        printf("inhalt von adr ist %x\n",inh);
       }
       ptrace(7,pnr,adr,0);

       while(wait(&status)!= pnr);

       printf("Hier im Vater nach drittem Sprung aus Sohnprozess\n");
       ptrace(8);                       /* Beendigung des Tracing */

       printf("Hier im Vaterprozess nach Tracing\n");
    }
}
```

SIEHE AUCH

exec, *signal()*, *wait()*.

NAME

putc - put character on a stream
Zeichen auf Datenstrom ausgeben

DEFINITION

```
#include <stdio.h>

int putc (c, stream)
int c;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *putc()* ist äquivalent zu *fputc()*, außer daß sie, wenn sie wie unter SINIX als Makro implementiert ist, *c* und *stream* mehr als einmal auswerten kann. Daher sollten diese Argumente niemals Ausdrücke mit Seiteneffekten sein.

ERGEBNIS

Siehe unter *fputc()*.

FEHLER

Siehe unter *fputc()*.

HINWEIS

Da *putc()* als Makro implementiert sein kann, kann diese Funktion Argumente *c* oder *stream* mit Seiteneffekten falsch behandeln. Insbesondere *putc(c, *f++)* wird normalerweise unkorrekt arbeiten. Statt dessen sollte die Funktion *fputc()* benutzt werden.

PORTABILITÄT

Die Funktion *putc()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Text von der Standardeingabe zeichenweise in *datei* schreiben:

```
#include <stdio.h>
main()
{  FILE    *fp;
   int      c;
   if ((fp=fopen("datei","w")) == NULL)
      exit(1);
   while ((c=getchar()) != EOF)
      putc(c,fp);
   fclose(fp);
}
```

SIEHE AUCH

fputc(), *<stdio.h>*.

NAME

putchar - put character on stdout stream
Zeichen auf Standardausgabe schreiben

DEFINITION

```
#include <stdio.h>

int putchar (c)
int c;
```

BESCHREIBUNG

Der Funktionsaufruf *putchar(c)* ist äquivalent zu *putc(c, stdout)*.

ERGEBNIS

Siehe unter *fputc()*.

PORTABILITÄT

Die Funktion *putchar()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Text zeichenweise von Standardeingabe lesen und auf Standardausgabe schreiben:

```
#include <stdio.h>

main()
{  int c;

   while ((c=getchar()) != EOF)
      putchar(c);
}
```

SIEHE AUCH

putc(), *<stdio.h>*.

NAME

putenv - change or add value to environment
Umgebungsvariablen ändern oder hinzufügen

DEFINITION

```
int putenv (string)
char *string;
```

BESCHREIBUNG

Die Funktion *putenv()* verwendet das Argument *string*, um die Werte von Umgebungsvariablen zu setzen. Das Argument *string* sollte auf eine Zeichenkette der Form "*name=wert*" zeigen. Die Funktion *putenv()* setzt den Wert der Umgebungsvariablen *name* gleich *wert*, indem sie eine existierende Variable verändert oder eine neue erzeugt. In jedem Fall wird die Zeichenkette *string* Teil der Umgebung, so daß eine Änderung der Zeichenkette die Umgebung ändern würde. Der von *string* belegte Platz wird nicht länger verwendet, wenn eine neue Zeichenkette, die *name* definiert, an *putenv()* übergeben wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *putenv()* den Wert 0. Andernfalls liefert sie einen Wert ungleich 0 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *putenv()* kann fehlschlagen, wenn gilt:

[ENOMEM] Es steht nicht genügend Speicherplatz zur Verfügung.

HINWEIS

Die Funktion *putenv()* manipuliert die Umgebung, auf die *environ* zeigt, und kann in Verbindung mit *getenv()* verwendet werden.

Diese Routine kann *malloc()* verwenden, um die Umgebung zu vergrößern.

Eine mögliche Fehlerquelle ist der Aufruf von *putenv()* mit einer automatischen Variablen als Argument und einer anschließenden Rückkehr von der aufrufenden Funktion, während *string* noch immer Teil der Umgebung ist.

PORTABILITÄT

Die Funktion *putenv()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

exec, *getenv()*, *malloc()*.

NAME

putpwent - write password file entry
Eintrag für die Kennwortdatei schreiben

DEFINITION

```
#include <stdio.h>
#include <pwd.h>

int putpwent (p, d)
struct passwd *p;
FILE *d;
```

BESCHREIBUNG

putpwent() ist das Pendant zu *getpwent()*. Unter Verwendung eines Zeigers auf eine von *getpwent()* (oder *getpwnam()*) eingerichtete *passwd*-Struktur schreibt *putpwent()* eine Zeile im Format der */etc/passwd*-Datei in die Datei *d*.

ERGEBNIS

putpwent() liefert im Fehlerfall einen Wert ungleich 0, ansonsten 0.

HINWEIS

Um Inkonsistenzen zu vermeiden, sollten Sie einen neuen Benutzer nur über das entsprechende *admin*-Menü eintragen.

PORTABILITÄT

Die Funktion *putpwent()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

getpwent(), *getpwnam()*.

NAME

puts - put a string on a standard output
Zeichenkette auf Standardausgabe schreiben

DEFINITION

```
#include <stdio.h>

int puts (s)
char *s;
```

BESCHREIBUNG

Die Funktion *puts()* schreibt die Zeichenkette, auf die *s* zeigt, gefolgt von einem Neue-Zeile-Zeichen, auf die Standardausgabe *stdout*. Das abschließende Nullbyte wird nicht geschrieben.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *puts()* und der nächsten erfolgreichen Beendigung eines Aufrufs von *fflush()* oder *fclose()* für denselben Datenstrom oder eines Aufrufs von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *puts()* eine nichtnegative Zahl. Andernfalls liefert sie den Wert EOF, setzt das Fehlerkennzeichen für den Datenstrom und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fputc()*.

HINWEIS

Die Funktion *puts()* fügt ein Neue-Zeile-Zeichen an, während die Funktion *fputs()* dies nicht macht.

PORTABILITÄT

Die Funktion *puts()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel unter *fputs()*.

SIEHE AUCH

fputs(), *fopen()*, *putc()*, *stdio*, *<stdio.h>*.

NAME

putw - put a word on a stream
Maschinenwort auf Datenstrom ausgeben

DEFINITION

```
#include <stdio.h>

int putw (w, stream)
int w;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *putw()* schreibt das Maschinenwort (dh. das *int*) *w* auf den Ausgabestrom *stream* (an die Position, auf die der Positionszeiger der Datei, falls definiert, zeigt). Die Größe eines Maschinenworts ist die Größe einer ganzen Zahl vom Typ *int* und ist von Rechner zu Rechner verschieden. Die Funktion *putw()* nimmt weder an, noch verursacht sie, daß in der Datei eine bestimmte Ausrichtung vorliegt.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *putw()* und der nächsten erfolgreichen Beendigung eines Aufrufs von *fflush()* oder *fclose()* für denselben Datenstrom oder eines Aufrufs von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *putw()* den Wert 0. Anderfalls wird ein Wert ungleich 0 zurückgeliefert, das Fehlerkennzeichen für den Datenstrom gesetzt und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fputc()*.

HINWEIS

Aufgrund der möglichen Unterschiede in Maschinenwortgröße und der Anordnung der Bytes sind Dateien, die mit *putw()* geschrieben wurden, rechneabhängig und sollten nicht mit *getw()* auf einem anderen Rechner gelesen werden.

PORTABILITÄT

Die Funktion *putw()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Datei *dat* wortweise auf Datei *aus* übertragen: Falls die Datei *dat* nicht ein ganzzahliges Vielfaches von Maschinenwörtern enthält, kann *getw* die Datei nicht bis zum Ende lesen. In diesem Fall gibt *putw* die Datei auch nicht komplett aus.

```
#include <stdio.h>

FILE        *dz, *dz1;
int         w;

main()
{
   dz = fopen("dat","r");
   dz1 = fopen("aus","w");

   while(!feof(dz) && !ferror(dz) && !ferror(dz1))
   {
        w = getw(dz);
        if(!feof(dz) && !ferror(dz))
           putw(w,dz1);
   }

   fclose(dz); fclose(dz1);
}
```

SIEHE AUCH

fopen(), *fwrite()*, *getw()*, *<stdio.h>*.

NAME

qsort - sort a table of data
Vektor sortieren

DEFINITION

```
#include <stdlib.h>

void qsort (base, nel, width, compar)
void *base;
size_t nel, width;
int (*compar)();
```

BESCHREIBUNG

Die Funktion *qsort()* sortiert den Vektor von *nel* Elementen, auf dessen erstes Element *base* zeigt. Die Größe jedes Objekts in Bytes wird durch das Argument *width* angegeben.

Der Inhalt des Vektors wird in aufsteigender Reihenfolge gemäß einer Vergleichsfunktion sortiert. Das Argument *compar* ist ein Zeiger auf die Vergleichsfunktion, die mit zwei Argumenten aufgerufen wird, die auf zwei zu vergleichende Elemente zeigen. Die Funktion muß eine ganze Zahl zurückliefern die kleiner, gleich oder größer als 0 ist, je nachdem, ob des erste Argument entsprechend als kleiner, gleich oder größer dem zweiten Argument angenommen werden soll. Wenn zwei Elemente nach dieser Funktion gleich sind, dann ist deren Reihenfolge im sortierten Vektor unbestimmt.

ERGEBNIS

Die Funktion *qsort()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Vergleichsfunktion muß nicht jedes Byte vergleichen, so daß auch zusätzliche Daten zu den Vergleichswerten in den Elementen enthalten sein können.

PORTABILITÄT

Die Funktion *qsort()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Sortieren eines Zahlenfeldes und Ausgabe auf Standardausgabe:

```
#include <stdlib.h>

int vergl (a,b)
int *a, *b;
{
   return (*a - *b);
}

main()
{
   int j;
   static int feld[ ] = {7,4,2,1,54,9,2,3,1,23};

   for (j=0; j < (sizeof(feld)/sizeof(int)); j++)
      printf("%d ", feld[j]);
   printf ("\n");

   qsort ((void *) feld, (size_t) (sizeof(feld)/sizeof(int)),
          (size_t) sizeof(int), vergl);

   for (j=0; j < (sizeof(feld)/sizeof(int)); j++)
      printf("%d ", feld[j]);
   printf("\n");
}
```

SIEHE AUCH

<stdlib.h>.

NAME

rand - pseudo-random number generator
Pseudo-Zufallszahlengenerator

DEFINITION

```
#include <stdlib.h>

int rand()
```

BESCHREIBUNG

Die Funktion *rand()* verwendet einen multiplikativen, kongruenten Zufallszahlengenerator der Periode 2^{32}, der fortlaufende Pseudo-Zufallszahlen im Bereich 0 bis {RAND_MAX} erzeugt.

Die Funktion *srand()* kann aufgerufen werden, um die Reihe der zurückgelieferten Pseudo-Zufallszahlen zu ändern. Wenn *srand()* dann mit demselben Startwert aufgerufen wird, dann wird die Reihe der Pseudo-Zufallszahlen wiederholt. Wenn *rand()* vor jedem Aufruf von *srand()* aufgerufen wird, dann wird dieselbe Reihe erzeugt, als ob *srand()* zuerst mit einem Startwert von 1 aufgerufen worden wäre.

Die Implementierung verhält sich so, als ob keine in diesem Handbuch definierte Funktion *rand()* aufrufen würde.

ERGEBNIS

Die Funktion *rand()* liefert die nächste Pseudo-Zufallszahl in der Reihe.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Funktion *drand48()* stellt einen erheblich komplizierteren Zufallszahlengenerator zur Verfügung.

Die folgenden Funktionen definieren ein Paar von Funktionen, die zu Anwendungen hinzugefügt werden sollten, die sicherstellen wollen, daß dieselbe Sequenz auf verschiedenen Rechnern generiert wird.

```
static unsigned long int next = 1L;

int rand() /* RAND_MAX wird angenommen als: 32767 */
{
   next = next * 1103515245L + 12345L;
   return((unsigned int)(next/65536L) % 32768L);
}

void srand(seed)
unsigned int seed;
{
   next = seed;
}
```

PORTABILITÄT

Die Funktion *rand()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

drand48(), *srand()*, *<stdlib.h>*, *<limits.h>*.

NAME

read read from file
Aus Datei lesen

DEFINITION

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

BESCHREIBUNG

Die Funktion *read()* versucht, *nbyte* Bytes aus der Datei, die der offenen Dateikennzahl *fildes* zugeordnet ist, in den Puffer zu lesen, auf den *buf* zeigt.

Wenn *nbyte* gleich 0 ist, dann liefert die Funktion *read()* den Wert 0 und hat keine anderen Ergebnisse.

In Dateien, die ein Positionieren unterstützen (z.B. normale Dateien), beginnt die Funktion *read()* immer an einer Stelle, die durch die aktuelle Dateiposition, die *fildes* zugeordnet ist, bestimmt wird. Die Dateiposition wird um die Anzahl der tatsächlich gelesenen Bytes erhöht.

In Dateien, die ein Positionieren nicht unterstützen, wie z.B. Datensichtstationen, wird immer an der aktuellen Position gelesen. Der Wert für die Dateiposition, die solch einer Datei zugeordnet ist, ist undefiniert.

Bei erfolgreicher Beendigung liefert die Funktion *read()* die Anzahl der tatsächlich gelesenen und in den Puffer eingetragenen Bytes. Diese Anzahl ist niemals größer als *nbyte*. Der zurückgelieferte Wert kann kleiner als *nbyte* sein, wenn die Anzahl der in der Datei verbleibenden Bytes kleiner als *nbyte* ist, wenn die Funktion *read()* durch ein Signal unterbrochen wurde oder wenn die Datei eine Pipe oder FIFO ist und weniger als *nbyte* Bytes sofort zum Lesen verfügbar sind. So kann zum Beispiel ein *read()* für eine Datei, die einer Datensichtstation zugeordnet ist, genau eine Eingabezeile liefern.

Wenn ein *read()*-Aufruf von einem Signal unterbrochen wird, bevor er Daten lesen kann, dann liefert er den Wert -1, wobei *errno* gleich [EINTR] gesetzt ist.

Wenn ein *read()*-Aufruf von einem Signal unterbrochen wird, nachdem er erfolgreich einige Daten lesen konnte, dann liefert er entweder den Wert -1, wobei *errno* gleich [EINTR] gesetzt ist, oder er liefert die Anzahl der gelesenen Bytes. Ein *read()*-Aufruf

für eine Pipe oder FIFO kehrt niemals mit dem Wert [EINTR] für *errno* zurück, wenn er Daten übertragen konnte.

Nach dem aktuellen Dateiende findet keine Datenübertragung statt. Wenn die Anfangsposition am oder nach dem Dateiende liegt, dann wird der Wert 0 zurückgeliefert. Wenn die Datei eine Gerätedatei ist, dann ist das Ergebnis nachfolgender *read()*-Aufrufe geräteabhängig.

Beim Versuch, von einer leeren Pipe oder FIFO zu lesen, geschieht folgendes:

- Wenn kein Prozeß die Pipe zum Schreiben geöffnet hat, dann liefert *read()* den Wert 0, um das Dateiende anzuzeigen.
- Wenn ein Prozeß die Pipe zum Schreiben geöffnet hat und das O_NONBLOCK-Bit gesetzt ist, dann liefert *read()* den Wert -1 und besetzt *errno* mit [EAGAIN].
- Wenn ein Prozeß die Pipe zum Schreiben geöfnet hat und das O_NONBLOCK-Bit ist nicht gesetzt, dann blockiert *read()* solange, bis Daten geschrieben, oder bis die Pipe von allen Prozessen geschlossen wird, die diese zum Schreiben geöffnet hatten.

Beim Versuch, von einer Datei zu lesen, die keine Pipe oder FIFO ist, die nichtblockierendes Lesen unterstützt und für die zur Zeit keine Daten verfügbar sind, geschieht folgendes:

- Wenn O_NONBLOCK gesetzt ist, dann liefert *read()* den Wert -1 und besetzt *errno* mit [EAGAIN].
- Wenn O_NONBLOCK nicht gesetzt ist, dann blockiert *read()* solange, bis Daten verfügbar werden.
- Die Verwendung des O_NONBLOCK-Bits hat keine Wirkung, wenn Daten verfügbar sind.

Die Funktion *read()* liest Daten, die vorher in eine Datei geschrieben wurden. Wenn irgendein Bereich einer Datei vor dem Dateiende nicht geschrieben wurde, dann liefert die Funktion *read()* Bytes mit dem Wert 0. Die Funktion *lseek()* zum Beispiel erlaubt es, hinter das Ende der in dieser Datei existierenden Daten zu positionieren. Werden später Daten an diese Position geschrieben, dann liefern nachfolgende Leseoperationen in der Lücke zwischen dem ehemaligen Dateiende und den neu geschriebenen Daten solange Nullbytes, bis Daten in die Lücke geschrieben werden.

Bei erfolgreicher Beendigung markiert die Funktion *read()* das Feld *st_atime* der Datei zum Ändern.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion eine ganze Zahl, welche die Anzahl der tatsächlich gelesenen Bytes angibt. Andernfalls wird der Wert -1 zurückgeliefert, *errno* gesetzt, um den Fehler anzuzeigen und der Inhalt des Puffers, auf den *buf* zeigt, ist unbestimmt.

FEHLER

Die Funktion *read()* schlägt fehl, wenn gilt:

[EAGAIN] Das Bit O_NONBLOCK ist für die Dateikennzahl gesetzt und der Prozeß würde durch die Leseoperation angehalten werden.

[EBADF] Das Argument *fildes* ist keine gültige, zum Lesen geöffnete Dateikennzahl.

[EINTR] Die Funktion wurde durch ein Signal unterbrochen und es wurden entweder keine Daten übertragen oder die Implementierung meldet keine teilweise Übertagung für diese Datei.

[EIO] Die Implementierung unterstützt Auftragskontrolle, der Prozeß ist ein Mitglied einer Hintergrund-Prozeßgruppe und versucht von seinem kontrollierenden Terminal zu lesen, er ignoriert oder blockiert das Signal SIGTTIN oder die Prozeßgruppe ist verwaist. Dieser Fehler kann auch aus anderen implementierungsabhängigen Gründen erzeugt werden.

Die Funktion *read()* kann fehlschlagen, wenn gilt:

[ENXIO] Eine Anforderung für ein nicht existierendes Gerät wurde gemacht oder die Anforderung lag außerhalb der Fähigkeiten des Geräts.

Unter SINIX kann zusätzlich zu den im XPG3 [6] definierten Fehlern noch der folgende Fehler auftreten:

[EFAULT] Das Argument *buf* ist eine ungültige Adresse.

HINWEIS

Die hier beschriebene O_NONBLOCK-Behandlung unterscheidet sich von der Behandlung von O_NDELAY, das bisher verfügbar war wurde. O_NDELAY wurde zurückgezogen, da es durch O_NONBLOCK ersetzt wurde. O_NONBLOCK unterscheidet sich von O_NDELAY darin, daß es jetzt möglich ist, zwischen dem Ende der Datei und einer Fehlerbedingung zu unterscheiden. Neue Anwendungen sollten nur O_NONBLOCK verwenden.

PORTABILITÄT

Die Funktion *read()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Wenn der Wert von *nbyte* größer als {INT_MAX} ist, dann gilt dies unter SINIX als Fehler. Andere X/Open-kompatible Systeme können sich in diesem Fall anders verhalten.

BEISPIEL

Das folgende Programm liest aus einer Datei mit 1000 Bytes in Stücken zu je 512 Bytes. Beim ersten Zugriff liefert *read()* als Ergebnis 512 zurück.
Beim zweiten Zugriff müßte *read* den Wert 0, also Datei-Ende zurückliefern. Das bedeutete, daß auch *nread* gleich 0 wäre und für *write()* nicht mehr benutzt werden könnte, um die Zahl der zu schreibenden Bytes anzugeben.
Jedoch wird das Dateiende im letzten Leseschritt separat geliefert. Im vorletzten Leseschritt liest *read()* nur soviele Bytes, wie noch in der Datei stehen und gibt deren Zahl als Ergebnis weiter.

Das Programm gibt die Zahl der Leseschritte minus eins (!) aus und wieviele Bytes jeweils gelesen wurden.

```
#include <fcntl.h>
#include <stdio.h>
#define BUFSIZE 512

int main (argc, argv)
int argc;
char **argv;
{
   extern int errno;
   extern char *sys_errlist[];
   extern int sys_nerr;
   void perror();

   int fromfd, tofd, nread, count;
   char buf[BUFSIZE];
   count=0;

   if((fromfd=open(argv[1], O_RDONLY))==-1)
      perror(argv[1]);

   if((tofd=open(argv[2], O_CREAT | O_WRONLY, 0777))==-1)
      perror(argv[2]);

   while((nread=read(fromfd, buf, sizeof(buf)))>0)
   {
      count++;
      if(write(tofd, buf, nread)!=nread)
         perror("write");
      printf("Bei Schleife Nr. %d: %d Bytes\n", count, nread);
   }

   if(nread==-1)
      perror("read");

   if(close(fromfd)==-1||close(tofd)==-1)
      perror("close");

   printf("%s->%s: %d Schleifen\n",argv[1],argv[2], count);
}
```

SIEHE AUCH

fcntl(), *lseek()*, *open()*, *pipe()*, *Abschnitt 2.4, Allgemeine Terminalschnittstelle.*

NAME

readdir - read directory
Dateiverzeichniseintrag lesen

DEFINITION

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

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

BESCHREIBUNG

Der Datentyp *DIR*, der in der Include-Datei *<dirent.h>* definiert ist, repräsentiert einen Dateiverzeichnisstrom, der eine geordnete Folge aller Dateiverzeichniseinträge eines speziellen Dateiverzeichnisses ist. Dateiverzeichniseinträge repräsentieren Dateien; Dateien können gleichzeitig mit der Ausführung der Funktion *readdir()* aus einem Dateiverzeichnis entfernt bzw. zu einem Dateiverzeichnis hinzugefügt werden.

Die Funktion *readdir()* liefert einen Zeiger auf den nächsten, nichtleeren Dateiverzeichniseintrag in dem Dateiverzeichnisstrom, der durch das Argument *dirp* angegeben wird und positioniert den Dateiverzeichnisstrom auf den nächsten Eintrag. Sobald sie das Ende des Dateiverzeichnisstroms erreicht, liefert sie den Nullzeiger. Die Struktur *dirent*, die in der Include-Datei *<dirent.h>* definiert ist, beschreibt einen Dateiverzeichniseintrag.

Die Funktion *readdir()* liefert keine Dateiverzeichniseinträge, die leere Namen enthalten. Sofern Einträge für . und .. existieren, wird genau ein Eintrag für . und einer für .. zurückgeliefert; andernfalls werden diese Einträge nicht zurückgeliefert.

Der von *readdir()* zurückgelieferte Zeiger zeigt auf Daten, die von einem weiteren Aufruf von *readdir()* für denselben Dateiverzeichnisstrom überschrieben werden können. Diese Daten werden von einem weiteren Aufruf von *readdir()* für einen anderen Dateiverzeichnisstrom nicht überschrieben.

Wenn nach dem letzten Aufruf von *opendir()* oder *rewinddir()* eine Datei aus dem Dateiverzeichnis entfernt, bzw. zu diesem hinzugefügt wurde, dann ist es unbestimmt, ob ein nachfolgender Aufruf von *readdir()* einen Eintrag für diese Datei liefert.

Die Funktion *readdir()* kann mehrere Dateiverzeichniseinträge bei einer einzelnen Leseoperation zwischenspeichern; die Funktion *readdir()* markiert das Feld *st_atime* des Dateiverzeichnisses jedesmal, wenn das Dateiverzeichnis wirklich gelesen wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *readdir()* einen Zeiger auf ein Objekt vom Typ *struct dirent*. Wenn ein Fehler auftritt, dann wird der Nullzeiger zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen. Wenn das Ende des Dateiverzeichnisses erreicht wird, dann wird der Nullzeiger zurückgeliefert und *errno* wird nicht verändert.

FEHLER

Die Funktion *readdir()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *dirp* zeigt nicht auf einen offenen Dateiverzeichnisstrom.

BEISPIEL

Das folgende Programmstück durchsucht das aktuelle Dateiverzeichnis nach dem Eintrag *name*:

```
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL)
        if (strcmp(dp->d_name, name) == 0) {
                closedir(dirp);
                return FOUND;
        }
closedir(dirp);
return NOT_FOUND;
```

HINWEIS

Die Funktion *readdir()* sollte in Verbindung mit *opendir()*, *closedir()* und *rewinddir()* verwendet werden, um den Inhalt des Dateiverzeichnisses zu untersuchen. Da *readdir()* den Nullzeiger sowohl am Ende des Dateiverzeichnisses als auch bei Fehler liefert, sollte eine Anwendung, die Fehlersituationen überprüfen will, *errno* gleich 0 setzen, danach *readdir()* aufrufen, dann den Wert von *errno* prüfen und wenn dieser ungleich 0 ist, das Auftreten eines Fehlers annehmen.

PORTABILITÄT

Die Funktion *readdir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

closedir(), *opendir()*, *rewinddir()*, *<dirent.h>*, *<sys/types.h>*.

NAME

realloc - memory reallocator
Speicherblockgröße verändern

DEFINITION

```
#include <stdlib.h>

void *realloc (ptr, size)
void *ptr;
size_t size;
```

BESCHREIBUNG

Die Funktion *realloc()* verändert die Größe des Speicherblocks, auf den *ptr* zeigt, auf die Größe, die durch *size* angegeben wird. Der Inhalt des Objekts bleibt bis zu der kleineren der alten und neuen Größe unverändert. Wenn die neue Größe des Objekts ein Verschieben des Objekts erfordert, dann wird der Speicherplatz für den alten Bereich des Objekts freigegeben. Wenn die neue Größe größer als die alte ist, dann ist der Inhalt des neuen Teilbereichs des Objekts undefiniert. Wenn *size* gleich 0 ist, dann wird das angegebene Objekt freigegeben. Wenn der benötigte Platz nicht reserviert werden kann, dann bleibt das Objekt unverändert.

Wenn *ptr* gleich dem Nullzeiger ist, dann verhält sich die Funktion *realloc()* genauso wie die Funktion *malloc()* für die angegebene Größe.

Wenn *ptr* kein Zeiger ist, der vorher durch die Funktionen *calloc()*, *malloc()*, oder *realloc()* geliefert wurde, oder wenn der Speicherplatz vorher bereits durch die Funktionen *free()* oder *realloc()* freigegeben wurde, dann ist das Verhalten undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *realloc()* einen Zeiger auf den (möglicherweise verschobenen) reservierten Speicherblock. Wenn nicht genügend Speicherplatz zur Verfügung steht, dann liefert *realloc()* den Nullzeiger und besetzt *errno* mit [ENOMEM].

FEHLER

Die Funktion *realloc()* schlägt fehl, wenn gilt:

[ENOMEM] Es steht nicht genügend Speicherplatz zur Verfügung.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<malloc.h>* anstelle von *<stdlib.h>*. Die Verwendung von *<malloc.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft zurückgezogen wird.

PORTABILITÄT

Die Funktion *realloc()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

calloc(), *free()*, *malloc()*, *<stdlib.h>*.

NAME

regexp, advance, compile, step, loc1, loc2, locs
- regular expression compile und match routines
Routinen für das Übersetzen und Erkennen regulärer Ausdrücke

DEFINITION

```
#define INIT declarations
#define GETC() getc code
#define PEEK() peek code
#define UNGETC() ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step(string, expbuf)
char *string, *expbuf;

int advance(string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;
```

BESCHREIBUNG

Diese Funktionen dienen zum Erkennen von Mustern in Programmen, die reguläre Ausdrücke verarbeiten. Sie werden in der Include-Datei <*regexp.h*> definiert.

Die Funktionen *step()* und *advance()* führen Vergleiche durch, bei denen eine Zeichenkette sowie ein übersetzter regulärer Ausdruck als Eingabe vorhanden sind.

Die Funktion *compile()* verarbeitet als Eingabe reguläre Ausdrücke und erzeugt einen übersetzten Ausdruck der mit *step()* oder *advance()* weiterverarbeitet werden kann.

In einem Programm müssen vor der Anweisung #*include* <*regexp.h*> die folgenden Makros vom Benutzer selbst definiert werden. Diese Makros werden von der Funktion *compile()* benutzt. Mit den Makros GETC(), PEEKC() und UNGETC() werden reguläre Ausdrücke verarbeitet, die *compile()* als Eingabe dienen.

GETC() Dieses Makro liefert den Wert des nächsten Zeichens im regulären Ausdruck. Wiederholte, aufeinanderfolgende Aufrufe von GETC() sollten aufeinanderfolgende Zeichen des regulären Ausdrucks ausgeben.

PEEKC() Dieses Makro liefert das nächste Zeichen im regulären Ausdruck. Wiederholte, unmittelbar aufeinanderfolgende Aufrufe von PEEKC() sollten immer dasselbe Zeichen liefern, das zudem mit dem nächsten, von GETC() gelieferten Zeichen identisch sein sollte.

UNGETC(c) Dieses Makro bewirkt, daß beim nächsten Aufruf von GETC() und PEEKC() das Argument *c* ausgegeben wird. Es wird auf keinen Fall mehr als ein Zeichen benötigt, das in die Eingabe zurückgeschoben wird, und dieses Zeichen ist in jedem Fall das letzte von GETC() eingelesene Zeichen. Der Wert des Makros *UNGETC(c)* wird immer ignoriert.

RETURN(ptr) Dieses Makro wird bei einer normalen Beendigung der Funktion *compile()* benutzt. Der Wert des Arguments *ptr* ist ein Zeiger auf das Zeichen, das auf das letzte Zeichen des übersetzten regulären Ausdrucks folgt. Dieses Makro ist bei Programmen hilfreich, die Speicherbereiche verwalten.

ERROR (val) Dieses Makro entspricht der unnormalen Beendigung der Funktion *compile()* Das Argument *val* ist eine Fehlernummer (Bedeutung der einzelnen Rückgabewerte siehe unter FEHLER). Dieser Aufruf sollte niemals zurückkehren.

Die Syntax der Funktion *compile()* ist die folgende:

```
compile(instring, expbuf, endbuf, eof)
```

Der erste Parameter *instring* wird niemals direkt von der Funktion *compile()* benutzt, ist aber nützlich für Programme, die verschiedene Zeiger auf Eingabezeichen übergeben. Er wird manchmal in den Deklarationen zu *INIT* verwendet (siehe auch unten).

Programme, die Funktionen aufrufen, um Zeichen einzugeben, oder die Zeichen aus einem externen Vektor verarbeiten, können hier den Wert ((*char**)0) übergeben.

Der nächste Parameter *expbuf* ist ein Zeiger auf *char*. Er zeigt auf den Ort, an dem der übersetzte reguläre Ausdruck abgelegt werden soll.

Der Parameter *endbuf* entspricht der Adresse des ersten Zeichens hinter dem Ende des Vektors, in den der übersetzte reguläre Ausdruck eingetragen werden soll. Wenn der übersetze Ausdruck länger als (endbuf-expbuf) Bytes ist, dann wird ein Aufruf von *ERROR(50)* ausgeführt.

Der Parameter *eof* ist dasjenige Zeichen, welches das Ende des regulären Ausdrucks markiert. Für das Kommando *ed* zum Beispiel ist diese Zeichen normalerweise das Zeichen '/'.

Jesed Programm, das eine #*include*-Anweisung für <*regexp.h*> enthält, muß auch eine #*define*-Anweisung für *INIT* enthalten. Dieses Makro wird für spezielle Deklarationen und Initialisierungen verwendet. In den meisten Fällen wird es verwendet, um eine *register*-Variable auf den Anfang des regulären Ausdrucks zeigen zu lassen, so daß diese *register*-Variable in den Deklarationen von *GETC()*, *PEEKC()* und *UNGETC()* verwendet werden kann. Anderfalls könnte es benutzt werden, um externe Variablen zu deklarieren, die von *GETC()*, *PEEKC()* und *UNGETC()* benutzt werden könnten. Siehe auch im Abschnitt BEISPIEL unten.

Der erste Parameter der Funktion *step()* ist ein Zeiger auf eine Zeichenkette, die gegen einen regulären Ausdruck geprüft werden soll. Diese Zeichenkette sollte mit dem Nullbyte abgeschlossen sein.

Der zweite Parameter *expbuf* ist der übersetzte reguläre Ausdruck, der von einem Aufruf der Funktion *compile()* geliefert wurde.

Die Funktion *step()* liefert einen Wert ungleich 0, wenn eine Teilfolge von *string* zu dem regulären Ausdruck *expbuf* paßt und sie liefert den Wert 0, wenn es keine Übereistimmung gibt. Wenn es eine Übereinstimmung gibt, dann werden zwei externe Zeiger als Seiteneffekt des Aufrufs von *step()* gesetzt. Die Variable *loc1* zeigt dann auf das erste, zum regulären Ausdruck passende Zeichen; die Variable *loc2* zeigt auf das Zeichen nach dem letzten Zeichen, das zum regulären Ausdruck paßt.

Wenn also die gesamte Eingabezeichenkette zum regulären Ausdruck paßt, so zeigt *loc1* auf das erste Zeichen von *string* und *loc2* zeigt auf das Nullbyte am Ende von *string*.

Die Funktion *advance()* liefert einen Wert ungleich 0, wenn die erste Teilfolge von *string* zum regulären Ausdruck in *expbuf* paßt. Wenn eine Übereinstimmung vorliegt, dann wird als Seiteneffekt ein externer Zeiger auf *char* gesetzt: *loc2*. Die Variable *loc2* zeigt auf das erste Zeichen in *string*, das sich hinter dem letzen passenden Zeichen befindet.

Wenn *advance()* auf ein Zeichen * oder auf die Zeichenkette \{\} im regulären Ausdruck trifft, so setzt sie ihren Zeiger hinter die größtmögliche, dazu passende Zeichenkette und ruft sich selbst rekursiv auf, um den Rest der Zeichenkette mit dem Rest des regulären Ausdrucks zu vergleichen. Liegt keine Übereinstimmung vor, so wird geprüft, ob das gesuchte Muster bereits im vorher erkannten Teilstring enthalten ist, indem jeweils um eine Stelle zurückgerückt wird, bis eine Übereinstimmung festgestellt oder zu der Stelle in der Zeichenkette gelangt wird, die anfangs zu dem * oder der Zeichenkette \{ \} paßte. In manchen Fällen ist es wünschenswert, daß *advance()* das Zurückgehen vor Erreichen dieser Stelle beendet. Wenn der externe Zeiger *locs* zu irgendeinem Zeitpunkt während des Zurückgehens identisch ist mit dieser Stelle in der Zeichenkette, so beendet *advance()* die Schleife und gibt den Wert 0 zurück.

Die externen Variablen *circf*, *sed* und *nbra* sind reserviert.

ERGEBNIS

Die Funktion *compile()* verwendet das Makro *RETURN()* bei Erfolg und das Makro *ERROR()* bei Mißerfolg (siehe oben). Die Funktionen *step()* und *advance()* liefern einen Wert ungleich 0 bei einer Übereinstimmung und den Wert 0, falls es keine Übereinstimmung gibt.

FEHLER

11	Bereich zu groß
16	Ungültige Zahl
25	*Zahl* liegt nicht im zulässigen Bereich
36	Unzulässiges oder fehlendes Trennzeichen
41	Keine Suchfolge gespeichert
43	Unterschiedliche Zahl von \(und \)
43	Zu viele \(
44	Zwischen \{ und \} stehen mehr als zwei Zahlen
45	} nach \ erwartet
46	Die erste in \{ \} eingeschlossene Zahl ist größer als die zweite
49	unterschiedliche Zahl von [und]
50	Regulärer Ausdruck zu groß

HINWEIS

Wenn das Programm mit der Option -DINTL oder der Präprozessor-Anweisung #*define INTL* übersetzt wird, dann verarbeiten diese Funktionen auch internationalisierte reguläre Ausdrücke.

Wird das Programm mit der Option -DEXTENDED oder der Präprozessor-Anweisung #*define EXTENDED* übersetzt, so verarbeiten diese Funktionen erweiterte reguläre Ausdrücke. Auch eine Kombination der beiden Optionen bzw. Präprozessor-Anweisungen ist zulässig, um erweiterte internationalisierte reguläre Ausdrücke zu verarbeiten.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

Portabilitätshinweis:
Die internationalisierten regulären Ausdrücke werden von den Funktionen nach dem X/Open-Standard noch nicht unterstützt.

BEISPIEL

Das folgende Beispiel zeigt, wie in einem Anwendungsprogramm die Makros und Funktionsaufrufe zur Behandlung von regulären Ausdrücken definiert werden könnten:

```
#define INIT        register char *sp=instring
#define GETC()      (*sp++)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regerr()

#include <regexp.h>

main(argc,argv)
int argc;
char **argv;
{  char expbuf[ESIZE], linebuf[LINESIZE];
   ...
   /* Uebersetzen eines regulaeren Ausdrucks */
   (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
   ...
   if (step(linebuf, expbuf))
      /* regulaerer Ausdruck paßt zu Teil von linebuf */
   ...
}
```

SIEHE AUCH

Kommando ed, setlocale(), <regexp.h>, Abschnitt 2.5.

NAME

remove- remove files
Dateien löschen

DEFINITION

```
#include <stdio.h>

int remove (path)
char *path;
```

BESCHREIBUNG

Wenn *path* kein Dateiverzeichnis bezeichnet, verhält sich *remove(path)* äquivalent zu *unlink(path)*.

Wenn *path* ein Dateiverzeichnis ist, verhält sich *remove(path)* äquivalent zu *rmdir(path)*.

ERGEBNIS

Siehe bei den Funktionen *rmdir()* oder *unlink()*.

FEHLER

Siehe bei den Funktionen *rmdir()* oder *unlink()*.

PORTABILITÄT

Die Funktion *remove()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

rmdir(), *unlink()*, *<stdio.h>*.

NAME

rename - rename a file
Datei umbenennen

DEFINITION

```
#include <stdio.h>

int rename (old, new)
char *old, *new;
```

BESCHREIBUNG

Die Funktion *rename()* ändert den Namen einer Datei. Das Argument *old* zeigt auf den Pfadnamen der Datei, die umbenannt werden soll. Das Argument *new* zeigt auf den neuen Pfadnamen der Datei.

Wenn sowohl das Argument *old* als auch das Arguement *new* auf dieselbe existierende Datei verweisen, dann kehrt die Funktion *rename()* erfolgreich zurück und führt keine weitere Aktion aus.

Falls das Argument *old* auf einen Pfadnamen einer Datei zeigt, die kein Dateiverzeichnis ist, darf das Argument *new* nicht auf den Pfadnamen eines Dateiverzeichnisses zeigen. Falls der Verweis existiert, der durch das Argument *new* angegeben wird, so wird er entfernt und *old* wird in *new* umbenannt. Sowohl für das Dateiverzeichnis, das *old* enthält, als auch für das Dateiverzeichnis, das *new* enthält wird die Schreiberlaubniss benötigt.

Wenn das Argument *old* auf den Pfadnamen eines Dateiverzeichnisses zeigt, dann darf das Argument *new* nicht auf den Pfadnamen einer Datei zeigen, die kein Dateiverzeichnis ist. Wenn das Dateiverzeichnis existiert, das durch *new* angegeben wird, dann wird es entfernt und *old* wird in *new* umbenannt. Wenn *new* ein existierendes Dateiverzeichnis angibt, dann muß dieses ein leeres Dateiverzeichnis sein.

Der Pfadnamen-Anfang von *new* darf nicht identisch sein mit *old*. Die Schreiberlaubniss wird für das Dateiverzeichnis, das *old* enthält und für das Dateiverzeichnis, das *new* enthält benötigt.

Wenn das Argument *old* auf den Pfadnamen eines Dateiverzeichnisses zeigt, dann kann die Schreiberlaubniss für das durch *old* angegebene Dateiverzeichnis benötigt werden und, falls es exisitiert, für das Dateiverzeichnis, das durch *new* angegeben wird.

Falls der Verweis existiert, der durch das Argument *new* angegeben wird und der Verweiszähler der Datei durch das Entfernen dieser Datei gleich 0 wird, und falls außerdem kein Prozeß diese Datei geöffnet hat, so wird der Platz freigegeben, der durch diese Datei belegt wird und auf die Datei kann nicht länger zugegriffen werden. Falls einer oder mehrere Prozesse die Datei geöffnet haben, während der letzte Verweis entfernt wird, so wird der Verweis entfernt, bevor die Funktion *rename()* zurückkehrt aber die Entfernung der Datei wird aufgeschoben, bis alle Referenzen auf diese Datei geschlossen sind.

Bei erfolgreicher Beendigung markiert die Funktion *rename()* die Felder *st_ctime* und *st_mtime* des übergeordneten Dateiverzeichnisses jeder der beiden Dateien.

ERGEBNIS

Bei erfolgreicher Beendigung liefert *rename()* den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und die Variable *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *rename()* schlägt fehl, wenn gilt:

[EACCES] Für eine Komponente einer der Pfadnamen-Anfänge existiert kein Suchrecht oder für eines der Dateiverzeichnisse, die *old* oder *new* enthalten, existiert kein Schreibrecht; oder das Schreibrecht für eines der Dateiverzeichnisse auf die Argumente *old* oder *new* zeigen wird benötigt, ist aber nicht vorhanden.

[EBUSY] Eines der Dateiverzeichnisse die durch *old* oder *new* angegeben werden, wird zur Zeit durch das System oder einen anderen Prozeß verwendet und die Implementierung nimmt dies als einen Fehler an.

[ENOTEMPTY] oder [EEXIST]
Der Verweis, der durch *new* angegeben wird, ist ein nichtleeres Dateiverzeichnis.

[EINVAL] Der Dateiverzeichnis-Pfadname *new* enthält einen Pfadnamen-Anfang, der das Dateiverzeichnis *old* bezeichnet.

[EISDIR] Das Argument *new* zeigt auf ein Dateiverzeichnis und das Argument *old* zeigt auf eine Datei, die kein Dateiverzeichnis ist.

[ENAMETOOLONG]
Die Länge des Arguments *old* oder des Arguments *new* überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX} während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Der Verweis, der durch *old* bezeichnet wird, existiert nicht oder *old* bzw. *new* zeigt auf eine leere Zeichenkette.

[ENOSPC] Das Dateiverzeichnis, das *new* enthalten würde, kann nicht erweitert werden.

[ENOTDIR] Eine Komponente in einem der Pfadnamen-Anfänge ist kein Dateiverzeichnis; oder das Argument *old* bezeichnet ein Dateiverzeichnis und das Argument *new* bezeichnet eine Datei, die kein Dateiverzeichnis ist.

[EROFS] Die verlangte Operation muß in einem Dateiverzeichnis schreiben, das sich in einem nur zum Lesen eingehängten Dateisystems befindet.

[EXDEV] Die durch *new* und *old* bezeichneten Verweise befinden sich in verschiedenen Dateisystemen.

Die Funktion *rename()* kann fehlschlagen, wenn gilt:

[ETXTBSY] Die umzubenennende Datei ist eine reine Prozedur-Datei (gemeinsamer Text), die gerade ausgeführt wird.

PORTABILITÄT

Die Funktion *rename()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Andere X/Open-kompatible Systeme können Verweise zwischen Dateisystemen zulassen.

SIEHE AUCH

link(), *rmdir()*, *unlink()*, *<stdio.h>*.

NAME

rewind - reset file position indicator in a stream
Dateiposition auf Anfang setzen

DEFINITION

```
#include <stdio.h>

void rewind (stream)
FILE *stream;
```

BESCHREIBUNG

Der Aufruf
rewind(stream);
ist äquivalent zu
(void) fseek(stream, 0L, SEEK_SET);
außer daß *rewind()* auch das Fehlerkennzeichen löscht.

ERGEBNIS

Die Funktion *rewind()* hat kein Ergebnis.

FEHLER

Siehe unter *fseek()* mit Ausnahme des Fehlers EINVAL, der nicht auftreten kann.

HINWEIS

Da *rewind()* kein Ergebnis liefert, sollte eine Anwendung, die Fehler erkennen will, zuerst *errno* gleich 0 setzen, dann *rewind()* aufrufen und dann, wenn *errno* ungleich 0 ist, einen Fehler annehmen.

PORTABILITÄT

Die Funktion *rewind()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die ersten 10 Zeichen überlesen, den Rest der Datei ausgeben, dann die ersten 10 Zeichen ausgeben:

```
#include <stdio.h>

main()
{  extern int errno;
   FILE *dz;
   int c,i;

   dz = fopen("datei","r");
   fseek(dz,10L,SEEK_SET);     /* Die ersten 10 Zeichen ueberlesen */

   while((c=getc(dz)) != EOF)
      putc(c,stdout);

   errno=0;
   rewind(dz);                      /* Auf Dateianfang positionieren */
   if (errno)                       /* Fehlerueberpruefung */
      perror("rewind");
   else
      for (i=0; i<10; i++)
      {  c=getc(dz);
         putc(c,stdout);
      }
   fclose(dz);
}
```

SIEHE AUCH

fseek(), *<stdio.h>*.

NAME

rewinddir - resets position of directory stream to the beginning of a directory
Position in Dateiverzeichnisstrom auf Anfang des Dateiverzeichnisses setzen

DEFINITION

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

void rewinddir(dirp)
DIR *dirp;
```

BESCHREIBUNG

Die Funktion *rewinddir()* setzt die Position des Dateiverzeichnisstroms auf den *dirp* zeigt, auf den Anfang des Dateiverzeichnisses. Es veranlaßt den Dateiverzeichnisstrom auch, den aktuellen Zustand des entsprechenden Dateiverzeichnisses zu berücksichtigen, so wie dies ein Aufruf von *opendir()* machen würde. Wenn *dirp* nicht auf einen Dateiverzeichnisstrom zeigt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *rewinddir()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Funktion *rewinddir()* sollte in Verbindung mit *opendir()*, *readdir()* und *closedir()* verwendet werden, um den Inhalt eines Dateiverzeichnisses zu untersuchen. Diese Methode wird aus Portabilitätsgründen empfohlen.

PORTABILITÄT

Die Funktion *rewinddir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

closedir(), *opendir()*, *readdir()*, *<dirent.h>*, *<sys/types.h>*.

NAME

rmdir - remove a directory
Dateiverzeichnis löschen

DEFINITION

```
int rmdir (path)
char *path;
```

BESCHREIBUNG

Die Funktion *rmdir()* löscht ein Dateiverzeichnis, dessen Name durch *path* angegeben wird. Das Dateiverzeichnis wird nur dann gelöscht, wenn es ein leeres Dateiverzeichnis ist.

Wenn das Dateiverzeichnis das root-Dateiverzeichnis oder das aktuelle Dateiverzeichnis irgendeines Prozesses ist, ist die Wirkung von *rmdir()* implementierungs-abhängig.

Wenn der Verweiszähler des Dateiverzeichnisses gleich 0 wird und kein Prozeß das Dateiverzeichnis geöffnet hat, dann wird der Platz freigegeben, der vom Dateiverzeichnis belegt wird, und auf das Dateiverzeichnis kann nicht länger zugegriffen werden. Falls einer oder mehrere Prozesse das Dateiverzeichnis geöffnet haben, wenn der letzte Verweis entfernt wird, so werden die Einträge . und .., sofern vorhanden, entfernt, bevor die Funktion *rmdir()* zurückkehrt und es können keine neuen Einträge mehr in diesem Dateiverzeichnis vorgenommen werden; das Dateiverzeichnis wird jedoch nicht eher entfernt, bevor alle Referenzen auf das Dateiverzeichnis geschlossen worden sind.

Bei erfolgreicher Beendigung markiert die Funktion *rmdir()* die Felder *st_ctime* und *st_mtime* des übergeordneten Dateiverzeichnisses zum Ändern.

ERGEBNIS

Ein Rückkehrwert von 0 zeigt eine erfolgreiche Beendigung an. Ein Rückkehrwert von -1 zeigt an, daß ein Fehler aufgetreten ist, und daß die Fehlernummer in der Variablen *errno* abgelegt wurde.

FEHLER

Die Funktion *rmdir()* schlägt fehl, wenn gilt:

[EACCES] Für eine Komponente des Pfadnamen-Anfangs ist kein Suchrecht vorhanden oder das Schreibrecht für das übergeordnete Dateiverzeichnis des zu löschenden Dateiverzeichnisses ist nicht vorhanden.

[EBUSY] Das zu entfernende Dateiverzeichnis wird derzeit vom System oder einem anderen Prozeß benutzt und die Implementierung nimmt dies als einen Fehler an.

[EEXIST] oder [ENOTEMPTY]
Das Argument *path* bezeichnet ein Dateiverzeichnis, das nicht leer ist.

[ENAMETOOLONG]
Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Das Argument *path* bezeichnet ein nicht-existierendes Dateiverzeichnis oder zeigt auf eine leere Zeichenkette.

[ENOTDIR] Eine Komponente des Pfads ist kein Dateiverzeichnis.

[EROFS] Der zu löschende Dateiverzeichniseintrag befindet sich in einem nur zum Lesen eingehängten Dateisystem.

PORTABILITÄT

Die Funktion *rmdir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

mkdir(), *remove()*, *unlink()*.

NAME

sbrk - change data segment space allocation
Größe des Datensegments ändern

DEFINITION

```
char *sbrk (inkr)
int inkr;
```

BESCHREIBUNG

Die Funktionen *brk()* und *sbrk()* werden dazu verwendet, den Speicherplatz für das Datensegment eines Prozesses dynamisch zu ändern (s. *exec*).

Eine genauere Beschreibung der Funktion *sbreak()* finden Sie unter *brk()*.

PORTABILITÄT

Die Funktion *sbrk()* ist im X/OPEN-Standard nicht enthalten.

NAME

fscanf, scanf, sscanf - convert formatted input
Formatierte Eingabe umwandeln

DEFINITION

```
#include <stdio.h>

int scanf (format, ... )
char *format;

int fscanf (stream, format, ... )
FILE *stream;
char *format;

int sscanf (s, format, ... )
char *s, *format;
```

BESCHREIBUNG

Die Funktion *scanf()* liest von der Standardeingabe *stdin*. Die Funktion *fscanf()* liest aus dem angegeben Eingabestrom *stream*. Die Funktion *sscanf()* liest aus der Zeichenkette *s*. Jede Funktion liest Zeichen, interpretiert diese gemäß einem Format und speichert die Ergebnisse in ihren Argumenten ab. Jede dieser Funktionen erwartet als Argumente eine Kontrollzeichenkette *format*, wie unten beschrieben, und eine Menge von Zeiger-Argumenten, die angeben, wo die umgewandelte Eingabe abgelegt werden soll.

Die Umwandlung kann auf das *n*-te Argument der Argumentliste anstelle des nächsten, unbenutzten Arguments angewendet werden. In diesem Fall wird das Umwandlungszeichen % (siehe auch unten) durch die Sequenz *%ziffer$* ersetzt, wobei *ziffer* eine Dezimalzahl *n* aus dem Bereich [1, {NL_ARGMAX}] ist, die angibt, welche Position das Argument in der Argumentliste besitzt. Diese Eigenschaft erlaubt die Definition von Formatzeichenketten, die Argumente entsprechend einer speziellen Landessprache auswählen.

format kann beide Formen der Umwandlungsangabe enthalten, d.h. % oder *%digit$*, obwohl beide Formen nicht gleichzeitig innerhalb einer einzelnen *format*-Zeichenkette verwendet werden dürfen.

Alle Formen von *scanf()* erlauben das Erkennen eines landessprach-spezifischen Dezimalpunkts bzw. Exponentenzeichens in der Eingabezeichenkette. Der Dezimalpunkt und das Exponentenzeichen werden von den *langinfo*-Daten in der internationalen Umgebung des Programms

definiert (Kategorie LC_NUMERIC). In der Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, in der Dezimalpunkt und Exponentenzeichen nicht definiert sind, gilt die Voreinstellung '.' bzw. 'e' oder 'E'.

Das Format setzt sich aus keinen oder mehreren Anweisungen zusammen: ein oder mehrere Zwischenraumzeichen (wie durch die Funktion *isspace()* definiert), einfachen Zeichen ungleich % oder Umwandlungs-Anweisungen. Jede Umwandlungsanweisung wird entweder vom Zeichen % oder von der Sequenz *%ziffer$* eingeleitet.

Nach dem Zeichen % oder der Sequenz *%ziffer$* folgen die nachfolgend angegebenen Daten:

- Ein optionales Zeichen * zur Unterdrückung der Zuweisung.
- Eine optionale Ganzzahl, welche die maximale Feldbreite angibt.
- Ein optionales *h* oder *l* (kleines L), die die Größe des aufnehmenden Objekts angeben. Den Umwandlungszeichen *d*, *i* und *n* wird ein *h* vorangestellt, wenn das entsprechende Argument ein Zeiger auf *short int* und kein Zeiger auf *int* ist, oder ein *l*, wenn es ein Zeiger auf *long int* ist. Analog dazu wird den Umwandlungszeichen *o*, *u* und *x* ein *h* vorangestellt, wenn das entsprechende Argument ein Zeiger auf *unsigned short int* statt auf *unsigned int* ist, oder ein *l*, wenn es ein Zeiger auf *unsigned long int* ist. Schließlich wird den Umwandlungszeichen *e*, *f* und *g* ein *l* vorangestellt, wenn das entsprechende Argument ein Zeiger auf *double* anstelle eines Zeigers auf *float* ist. Wenn *h* oder *l* mit irgeneinem anderen Umwandlungszeichen auftritt, dann ist das Verhalten undefiniert.
- Ein Zeichen, das den Typ der durchzuführenden Umwandlung angibt. Die gültigen Umwandlungszeichen sind unten beschrieben.

Die Funktion *scanf()* führt jede Anweisung einzeln aus. Wenn eine Anweisung fehlschlägt, wie unten genauer erläutert, dann kehrt die Funktion zurück. Fehler werden als Eingabefehler bezeichnet, wenn Eingabezeichen fehlen, oder als Formatfehler, wenn unpassende Eingabezeichen auftreten.

Eine Anweisung, die aus einem Zwischenraumzeichen besteht, wird so ausgeführt, daß die Eingabe bis zum ersten Zeichen gelesen wird, das kein Zwischenraumzeichen ist und selbst nicht gelesen wird, oder bis keine Zeichen mehr gelesen werden

können.

Eine Anweisung, die aus einem normalen Zeichen besteht, wird so ausgeführt, daß die nächsten Zeichen aus der Eingabe gelesen werden. Wenn eines dieser Zeichen sich von dem Zeichen der Anweisung unterscheidet, so schlägt die Anweisung fehl und das unpassende und alle nachfolgenden Zeichen werden nicht gelesen.

Eine Anweisung, die eine Umwandlungsanweisung ist, definiert eine Menge von passenden Eingabefolgen, wie dies unten für jede einzelne Umwandlungsanweisung beschrieben wird. Eine Umwandlungsanweisung wird in den folgenden Schritten ausgeführt:

Die Eingabe von Zwischenraumzeichen wird überlesen, solange die Anweisung weder ein [, noch eines der Umwandlungszeichen *c* oder *n* enthält.

Ein Eingabeelement wird aus der Eingabe gelesen, solange die Anweisung nicht das Umwandlungszeichen *n* enthält. Ein Eingabeelement ist definiert als die längste Folge von Eingabezeichen (bis zu einer eventuell angegebenen maximalen Feldbreite), die ein Anfang einer passenden Folge ist. Das erste Zeichen nach einem Eingabeelement bleibt, sofern es vorhanden ist, ungelesen. Wenn die Länge des Eingabeelements gleich 0 ist, dann schlägt die Ausführung der Anweisung fehl; diese Bedingung bedeutet einen Formatfehler, solange nicht ein Fehler weitere Eingaben verhindert, was dann einen Eingabefehler bedeutet.

Außer im Fall des Umwandlungszeichens % wird das Eingabeelement, bzw. im Fall einer Umwandlungsanweisung *%n* die Anzahl der Eingabezeichen, umgewandelt in einen Datentyp, der dem Umwandlungszeichen entspricht. Wenn das Eingabeelement keine passende Folge ist, dann schlägt die Ausführung dieser Anweisung fehl: diese Bedingung ist ein Formatfehler. Wenn nicht die Unterdrückung der Zuweisung durch das Zeichen * angegeben wurde, dann wird das Ergebnis der Umwandlung in dem Objekt abgelegt, welches das erste auf *format* folgende Argument ist, in dem bisher noch kein Umwandlungsergebnis abgelegt wurde. Wenn dieses Objekt nicht den passenden Datentyp hat, oder wenn das Ergebnis der Umwandlung nicht in dem zur Verfügung stehenden Platz dargestellt werden kann, dann ist das Verhalten undefiniert.

Die folgenden Umwandlungszeichen sind zulässig:

d Liest eine dezimale Ganzzahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtod()* mit dem Wert 10 für das Argument *base* erwartet. Das entsprechende Argument sollte ein Zeiger auf *int* sein.

i Liest eine dezimale Ganzzahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtol()* mit dem Wert 0 für das Argument *base* erwartet. Das entsprechende Argument sollte vom Typ Zeiger auf *int* sein.

o Liest eine oktale Ganzzahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtol()* mit dem Wert 8 für das Argument *base* erwartet. Das entsprechende Argument sollte vom Typ Zeiger auf *unsigned* sein.

u Liest eine dezimale Ganzzahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtol()* mit dem Wert 10 für das Argument *base* erwartet. Das entsprechende Argument sollte vom Typ Zeiger auf *unsigned* sein.

x Liest eine hexadezimale Ganzzahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtol()* mit dem Wert 16 für das Argument *base* erwartet. Das entsprechende Argument sollte ein Zeiger auf *unsigned* sein.

e, f, g
Diese Umwandlungszeichen lesen eine Gleitkommazahl ein, die auch mit einem Vorzeichen versehen sein kann. Deren Format ist dasselbe, das auch die Funktion *strtod()* erwartet. Das entsprechende Argument sollte vom Typ Zeiger auf *float* sein.

s Liest eine Folge von Zeichen ein, die keine Zwischenraumzeichen sind. Das entsprechende Argument sollte ein Zeiger auf das erste Zeichen eines *char*-Vektors sein, der groß genug ist, die Folge und ein abschließendes Nullbyte aufzunehmen, das automatisch angefügt wird.

[Liest eine nichtleere Folge von Zeichen aus einer Menge von erwarteten Zeichen (der Eingabemenge). Das entsprechende Argument sollte ein Zeiger auf das erste Zeichen eines *char*-Vektors sein, der groß genug ist, die Folge und ein abschließendes Nullbyte aufzunehmen, das automatisch angefügt wird. Die Umwandlungsanweisung schließt alle folgenden Zeichen in der Zeichenkette *format* ein, einschließlich der zugehörenden rechten eckigen Klammer (]). Die Zeichen zwischen den Klammern stellen die Eingabemenge dar, solange nicht das erste Zeichen nach der linken Klammer das Zeichen ^ ist. In diesem Fall enthält die Eingabemenge alle Zeichen, die nicht in der Liste zwischen dem Zeichen ^ und der Klammer] aufgeführt sind. Als Sonderfall gilt, daß die rechte eckige Klammer in den beiden Fällen, in denen die Umwandlungsanweisung mit den Zeichenketten [] bzw. [^] beginnt, zur Liste der Zeichen gehört und erst die nächste rechteckige Klammer diejenige ist, welche die Umwandlungsanweisung abschließt. Wenn das Zeichen - in der Liste auftritt und nicht das erste Zeichen bzw. das zweite Zeichen nach dem Zeichen ^ und auch nicht das letzte Zeichen ist, dann ist das Verhalten undefiniert.

c Liest eine Folge von Zeichen, deren Anzahl durch die Feldbreite bzw. durch 1, wenn keine Feldbreite angegeben ist, bestimmt wird. Das entsprechende Argument sollte ein Zeiger auf das erste Zeichen eines *char*-Vektors sein, der groß genug ist, die Folge aufzunehmen. Ein abschließendes Nullbyte wird nicht angefügt. Das normale Überlesen von Zwischenraumzeichen wird in diesem Fall unterdrückt; um das nächste Zeichen zu lesen, das kein Zwischenraumzeichen ist, sollte "%1s" verwendet werden.

p Liest eine Menge von Folgen, die denen entsprechen sollten, die von der Umwandlungsanweisung "%p" der Funktion *printf()* erzeugt werden. Das entsprechende Argument sollte ein Zeiger auf einen Zeiger auf *void* sein. Die Interpretation des Eingabeelements ist jeweils implementierungs-abhängig; für ein Eingabeelement, das nicht früher während derselben Programmausführung umgewandelt wurde, ist das Verhalten der Umwandlungsanweisung *%p* undefiniert. Dies gilt insbesondere für Zeigerausgaben, die von anderen Systemen erzeugt worden sind.

n Es wird keine Eingabe verarbeitet. Das entsprechende Argument sollte ein Zeiger auf *int* sein, in das die bisher von diesem Aufruf gelesene Zahl der Eingabezeichen eingetragen wird. Die Ausführung einer Anweisung des Typs *%n* erhöht nicht den Zuweisungszähler, der bei Beendigung der Ausführung der Funktion zurückgeliefert wird.

% Liest ein einzelnes %. Dabei findet keine Umwandlung oder Zuweisung statt. Die vollständige Umwandlungsanweisung lautet *%%*.

Wenn ein Umwandlungszeichen ungültig ist, dann ist das Verhalten der Funktion *scanf()* undefiniert.

Die Umwandlungszeichen *E*, *G* und *X* sind ebenfalls gültig und verhalten sich genauso, wie die entsprechenden Umwandlungszeichen *e*, *g* und *x*.

Wenn das Dateiende während der Eingabe gefunden wird, dann wird die Umwandlung abgebrochen. Wenn das Dateiende auftritt, bevor irgenwelche, zur aktuellen Anweisung passenden Zeichen gelesen wurden (ungleich Zwischenraumzeichen, wo diese erlaubt sind), dann wird die Ausführung der aktuellen Anweisung mit einem Eingabefehler abgebrochen. Andernfalls wird, falls die Bearbeitung der aktuellen Anweisung nicht mit einem Formatfehler abbricht, die folgende Anweisung mit einem Eingabefehler abgebrochen, sofern diese vorhanden ist.

Wenn die Umwandlung wegen eines nicht passenden Zeichens abbricht, dann verbleibt dieses Zeichen ungelesen im Eingabestrom. Nachfolgende Zwischenraumzeichen (einschließlich der Neue-Zeile-Zeichen) bleiben ungelesen, solange dies nicht ein Anweisung macht. Der Erfolg des direkten Einlesens von Buchstaben und unterdrückten Zuweisungen kann nicht direkt bestimmt werden außer über die Anweisung *%n*.

Die Funktionen *scanf()* und *fscanf()* können das Feld *st_atime* der Datei zum Ändern markieren, die *stream* zugeordnet ist. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung einer der Funktionen *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* mit *stream* zum Ändern markiert, die Daten zurückliefert, die nicht zuvor von *ungetc()* bereitgestellt wurden.

ERGEBNIS

Diese Funktionen liefern die Anzahl der erfolgreich gelesenen und zugewiesenen Eingabeelemente; diese Zahl kann gleich 0 sein, wenn gleich zu Beginn ein nicht zur Formatzeichenkette passendes Eingabezeichen gefunden wird. Wenn die Eingabe vor dem ersten Konflikt zwischen Eingabezeichen und Formatzeichenkette oder vor der ersten Umwandlung endet, dann wird EOF zurückgeliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktionen *scanf()* und *fscanf()* schlagen fehl, wenn gilt:

[EAGAIN] Das O_NONBLOCK-Bit ist für die *stream* zugrundeliegende Dateikennzahl gesetzt und der Prozeß müßte warten.

[EBADF] Die *stream* zugrundeliegende Dateikennzahl ist keine gültige, zum Lesen offene Dateikennzahl.

[EINTR] Die Leseoperation wurde durch ein Signal unterbrochen und es wurden entweder keine Daten übertragen oder die Implementierung meldet keine teilweise Übertragung von Daten für diese Datei.

Die Funktionen *scanf()*, *fscanf()* und *sscanf()* können fehlschlagen, wenn gilt:

[ENXIO] Eine Anforderung für ein nichtexistierendes Gerät wurde gemacht, oder die Anforderung lag außerhalb der Fähigkeiten des Geräts.

Portabilitäts-Hinweis:
Unter anderen X/Open-kompatiblen Systemen, können die *scanf*-Funktionen auch noch in folgenden Fallen fehlschlagen:

[EINVAL] Es gibt nicht genügend Argumente.

[EIO] Wenn das System Auftragskontrolle unterstützt: der Prozeß ist ein Mitglied einer Hintergrundprozeßgruppe und versucht, von seinem kontrollierenden Terminal zu lesen, der Prozeß blockiert oder ignoriert das Signal SIGTTIN oder die Prozeßgruppe ist verweist.

BEISPIEL

Der Aufruf:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

mit der Eingabezeile:

```
25 54.32E-1 Hamster
```

weist an *n* den Wert 3, an *i* den Wert 25, an *x* den Wert 5.432 zu und *name* enthält die Zeichenkette "Hamster".

Der Aufruf:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0-9]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

weist den Wert 56 an *i* und 789.0 an *x* zu, überspringt 0123 und trägt die Zeichenkette "56" in *name* ein. Der nächste Aufruf der Funktion *getchar()* wird dann das Zeichen *'a'* liefern.

HINWEIS

In Formatzeichenketten, die die Form mit % für die Umwandlungs-Anweisungen enthalten, wird jedes Argument der Argumentliste genau einmal verwendet. In Formatzeichenketten, die die Form mit *%ziffer$* für die Umwandlungs-Anweisungen enthalten, kann jedes numerierte Argument der Argumentliste sooft verwendet werden, wie nötig.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
In anderen X/Open-kompatiblen Systemen, die den Datentyp *long double* unterstützen, kann den Umwandlungszeichen *e*, *f* und *g* ein *L* vorangestellt werden, wenn das entsprechende Argument ein Zeiger auf *long double* ist.

SIEHE AUCH

getc(), *printf()*, *setlocale()*, *strtod()*, *strtol()*, *<langinfo.h>*, *<stdio.h>*, *Abschnitt 2.5*.

NAME

seed48 - seed uniformly distributed pseudo-random non-negative long integer generator
Pseudo-Zufallszahlengenerator für gleichmäßig verteilte, nichtnegative Ganzzahhlen vom typ long initialisieren

DEFINITION

```
unsigned short *seed48 (seed16v)
unsigned short seed16v[3];
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *seed48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

seekdir - set position of directory stream
In Dateiverzeichnisstrom positionieren

DEFINITION

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

void seekdir (dirp, loc)
DIR *dirp;
long loc;
```

BESCHREIBUNG

Die Funktion *seekdir()* setzt die Position für die nächste Operation *readdir()* auf dem Dateiverzeichnisstrom, auf den *dirp* zeigt. Die Position wird auf die durch *loc* angegebene Position gesetzt. Der Wert von *loc* sollte von einem vorangegangenen Aufruf von *telldir()* zurückgeliefert worden sein. Die neue Position geht an die Position des Dateiverzeichnisstroms zurück, die diesem dann zugeordnet war, als die Operation *telldir()* ausgeführt wurde.

ERGEBNIS

Die Funktion *seekdir()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *seekdir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

opendir(), *readdir()*, *telldir()*, *<dirent.h>*, *<sys/types.h>*.

NAME

semctl - semaphore control operations
Semaphor-Kontrolloperationen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(semid, semnum, cmd, arg)
int semid, semnum, cmd;
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short *array;
        } arg;
```

BESCHREIBUNG

Die Funktion *semctl()* bietet eine Vielzahl von Operationen für die Steuerung von Semaphoren, die durch *cmd* angegeben werden.

Mit *cmd* werden die im folgenden aufgeführten Semaphor-Steuerungsoperationen angegeben, mit *semid* und *semnum* das Semaphor, für das die angegebene Operation ausgeführt werden soll. Die für die jeweilige Operation erforderlichen Zugriffsberechtigungen werden bei den entsprechenden Kommandos angegeben (siehe auch *Abschnitt 2.6*). Die symbolischen Namen für die Werte von *cmd* sind in der Include-Datei *<sys/sem.h>* definiert.

GETVAL
: Wert von *semval* liefern (siehe auch *<sys/sem.h>*). Leseberechtigung erforderlich.

SETVAL
: Wert von *semval* auf *arg.val* setzen. Nach erfolgreicher Durchführung dieses *cmd* ist der dem angegebenen Semaphor entsprechende *semadj*-Wert in allen Prozessen gelöscht. Änderungsberechtigung erforderlich (siehe auch *Abschnitt 2.6*).

GETPID
: Wert von *sempid* liefern. Leseberechtigung erforderlich.

GETNCNT
: Wert von *semncnt* liefern. Leseberechtigung erforderlich.

GETZCNT
Wert von *semzcnt* liefern. Leseberechtigung erforderlich.

Die folgenden Kommandos wirken auf jeden *semval* aus der Menge der zulässigen Semaphore.

GETALL
semvals abfragen und in den Vektor eintragen, auf den *arg.array* zeigt. Leseberechtigung erforderlich.

SETALL
semvals gemäß dem Vektor setzen, auf den *arg.array* zeigt. Nach erfolgreicher Ausführung dieses Kommandos sind die den angegebenen Semaphoren entsprechenden *semadj*-Werte in allen Prozessen gelöscht. Änderungsberechtigung erforderlich.

Die folgenden Kommandos sind außerdem verfügbar:

IPC_STAT
Die aktuellen Werte aller Elemente der *semid* zugeordneten Datenstruktur in die Struktur eintragen, auf die *arg.buf* zeigt. Der Inhalt dieser Struktur ist in *<sys/sem.h>* definiert. Leseberechtigung erforderlich.

IPC_SET
Die Werte der folgenden Elemente der *semid* zugeordneten Datenstruktur auf die entsprechenden Werte in der Struktur setzen, auf die *arg.buf* zeigt:

```
sem_perm.uid
sem_perm.gid
sem_perm.mode /* nur 9 niederwertige Bits */
```

Dieses Kommando kann nur von einem Prozeß ausgeführt werden, dessen effektive Benutzernummer die eines Prozesses mit besonderen Rechten ist oder mit *sem_perm.cuid* oder *sem_perm.uid* in der *semid* zugeordneten Datenstruktur übereinstimmt.

IPC_RMID
Die mit *semid* angegebene Semaphorkennzahl im System sowie die Semaphorenmenge samt zugeordneter Datenstruktur löschen. Dieses Kommando kann nur von einem Prozeß ausgeführt werden, dessen effektive Benutzernummer die eines Prozesses mit besonderen Rechten ist oder mit *sem_perm.cuid* oder *sem_perm.uid* in der *semid* zugeordneten Datenstruktur übereinstimmt.

ERGEBNIS

Bei erfolgreicher Ausführung liefert *semctl()* je nach angegebenem Kommando einen der folgenden Werte:

GETVAL Wert von *semval*.

GETPID Wert von *sempid*.

GETNCNT Wert von *semncnt*.

GETZCNT Wert von *semzcnt*.

Für alle Kommandos wird der Wert 0 zurückgeliefert.

Andernfalls liefert *semctl()* den Wert -1 und *errno* zeigt den Fehler an.

FEHLER

Die Funktion *semctl()* schlägt fehl, wenn gilt:

[EACCES] Der aufrufende Prozeß hat für das auszuführende Kommando nicht die erforderliche Zugriffsberechtigung siehe *Abschnitt 2.6*.

[EINVAL] *semid* ist keine gültige Semaphorkennzahl, *semnum* hat einen Wert kleiner 0 oder größer *sem_nsems* oder *cmd* ist kein gültiges Kommando.

[EPERM] Das Argument *cmd* ist gleich IPC_RMID oder IPC_SET und die effektive Benutzernummer des aufrufenden Prozesses nicht die eines Prozesses mit besonderen Rechten und stimmt nicht mit *sem_perm.cuid* oder *sem_perm.uid* in der *semid* zugeordneten Datenstruktur überein.

[ERANGE] Das Argument *cmd* ist gleich SETVAL oder SETALL und der Wert, auf den *semval* gesetzt werden soll, ist größer als der im System zulässige Höchstwert.

[ENOSYS] Diese Funktionalität wird vom dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *semctl()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

semget(), *semop()*, *<sys/ipc.h>*, *<sys/sem.h>*, *<sys/types.h>*.

NAME

semget - get set of semaphores
Semaphormenge anlegen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key, nsems, semflg)
key_t key;
int nsems, semflg;
```

BESCHREIBUNG

Die Funktion *semget()* liefert die *key* zugeordnete Semaphorkennzahl.

Es wird eine Semaphorkennzahl mit der dazugehörigen Datenstruktur *semid_ds* und der dazugehörigen Menge von *nsems* Semaphoren (siehe *<sys/sem.h>*) für das Argument *key* eingerichtet wenn eine der folgenden Bedingungen zutrifft:

- Das Argument *key* hat den Wert IPC_PRIVATE.
- Für das Argument *key* wurde noch keine Semaphorkennzahl eingerichtet und (*semflg & IPC_CREAT*) ist ungleich 0.

Beim Einrichten der neuen Semaphorkennzahl wird die dazugehörige Datenstruktur *semid_ds* wie folgt initialisiert:

- in der Zugriffsberechtigungs-Struktur werden für *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid* und *sem_perm.gid* die effektive Benutzernummer und die effektive Gruppennummer des aufrufenden Prozesses eingetragen;
- in die 9 niederwertigen Bits von *sem_perm.mode* werden die 9 niederwertigen Bits von *semflg* eingetragen;
- für die Variable *sem_nsems* wird der Wert von *nsems* eingetragen;
- die Variable *sem_otime* wird auf 0 gesetzt und in *sem_ctime* wird die aktuelle Zeit eingetragen;
- die den einzelnen Semaphoren zugeordneten Datenstrukturen werden nicht initialisiert. Die Funktion *semctl* mit dem Kommando SETVAL oder SETALL kann dazu verwendet werden, die einzelnen Semaphore zu initialisieren.

ERGEBNIS

Bei erfolgreicher Ausführung liefert *semget()* eine nicht negative ganze Zahl, eine Semaphorkennzahl, zurück. Andernfalls wird –1 geliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *semget()* schlägt fehl, wenn gilt:

[EACCES] Für *key* existiert bereits eine Semaphorkennzahl, aber die in den 9 niederwertigen Bits von *semflg* angegebene Berechtigung wurde nicht erteilt.

[EEXIST] Für das Argument *key* existiert eine Semaphorkennzahl aber ((*semflg & IPC_CREAT*) && (*semflg & IPC_EXCL*)) ist ungleich 0.

[EINVAL] Der Wert von *nsems* ist entweder kleiner gleich 0 oder größer als der vom System festgelegte Höchstwert, oder es existiert bereits eine Semaphorkennzahl für das Argument *key*, aber die dazugehörige Semaphorenmenge enthält weniger als *nsems* Semaphore und *nsems* ist ungleich 0.

[ENOENT] Für das Argument *key* existiert keine Semaphorkennzahl und (*semflg & IPC_CREAT*) ist gleich 0.

[ENOSPC] Es soll eine Semaphorkennzahl eingerichtet werden aber dadurch wird die Maximalzahl von im System insgesamt zulässigen Semaphoren überschritten.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *semget()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

semctl(), *semop()*, *<sys/ipc.h>*, *<sys/sem.h>*, *<sys/types.h>*.

NAME

semop - semaphore operations
Semaphor-Operationen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(semid, sops, nsops)
int semid;
struct sembuf *sops;
unsigned nsops;
```

BESCHREIBUNG

Die Funktion *semop()* ermöglicht die automatische Ausführung einer Liste mit vom Benutzer definierten Semaphoroperationen bezogen auf die Semaphormenge mit der im Argument *semid* angegebenen Semaphorkennzahl.

Das Argument *sops* zeigt auf eine vom Benutzer definierte Liste von Strukturen für Semaphoroperationen.

Das Argument *nsops* gibt die Anzahl an Strukturen in dem Vektor an.

Jede *sembuf*-Struktur enthält folgende Komponenten:

```
short   sem_num;      /* Semaphornummer */
short   sem_op;       /* Semaphoroperation */
short   sem_flg;      /* Operationsschalter */
```

Jede mit *sem_op* definierte Semaphoroperation wird für das mit *semid* und *sem_num* angebene Semaphor ausgeführt.

Die Variable *sem_op* definiert eine der folgenden drei Semaphoroperationen:

1. Ist *sem_op* eine negative ganze Zahl und der aufrufende Prozeß besitzt das Änderungsrecht, so tritt einer der folgenden Fälle ein:

   – Ist *semval* (siehe *<sys/sem.h>*) größer oder gleich dem Absolutbetrag von *sem_op*, so wird der Absolutbetrag von *sem_op* von *semval* subtrahiert. Ist (*sem_flg & SEM_UNDO*) ungleich 0, wird der Absolutbetrag von *sem_op* zum *semadj*-Wert des aufrufenden Prozesses für das angegebene Semaphor addiert (siehe *exit()*).

- Ist *semval* kleiner als der Absolutbetrag von *sem_op* und (*sem_flg & IPC_NOWAIT*) ist ungleich 0, so kehrt *semop()* sofort zurück.
- Ist *semval* kleiner als der Absolutbetrag von *sem_op* und (*sem_flg & IPC_NOWAIT*) ist gleich 0, so erhöht *semop()* den Wert von *semncnt* des angegebenen Semaphors um 1 und der aufrufende Prozeß wird angehalten, bis eine der folgenden Bedingungen eintritt:

  Der Wert von *semval* wird größer oder gleich dem Absolutbetrag von *sem_op*. Wenn dies eintritt, dann wird der *semncnt*-Wert des angegebenen Semaphors um 1 vermindert, der Absolutbetrag von *sem_op* wird von *semval* subtrahiert und, falls (*sem_flg & SEM_UNDO*) ungleich 0 ist, wird der Absolutbetrag von *sem_op* zum *semadj*-Wert des aufrufenden Prozesses für das angegebene Semaphor addiert.

  Die Kennzahl *semid*, für die der aufrufende Prozeß auf eine Operation wartet, wird im System gelöscht (siehe *semctl()*). In diesem Fall wird *errno* auf [EIDRM] gesetzt und der Wert -1 zurückgeliefert.

  Der aufrufende Prozeß empfängt ein abzufangendes Signal. In diesem Fall wird der *semncnt*-Wert des angegebenen Semaphors um 1 vermindert und der aufrufende Prozeß setzt seine Ausführung in der Weise fort, wie dies bei der Funktion *sigaction()* beschrieben ist.

2. Ist *sem_op* eine positive ganze Zahl und der aufrufende Prozeß besitzt das Änderungsrecht, so wird der Wert von *sem_op* zum *semval*-Wert addiert, und, falls (*sem_flg & SEM_UNDO*) ungleich 0 ist, vom *semadj*-Wert des aufrufenden Prozesses für das angegebene Semaphor subtrahiert.
3. Ist *sem_op* gleich 0 und dem aufrufenden Prozeß wurde Leseberechtigung erteilt, so tritt einer der folgenden Fälle ein:
   - Ist *semval* gleich 0, so kehrt die Funktion *semop()* sofort zurück.

- Ist sowohl *semval* als auch (*sem_flg & IPC_NOWAIT*) ungleich 0, so kehrt die Funktion *semop()* sofort zurück.
- Ist *semval* ungleich 0 und (*sem_flg & IPC_NOWAIT*) gleich 0, so erhöht die Funktion *semop()* den Wert von *semzcnt* des angegebenen Semaphors um 1 und der aufrufende Prozeß wird angehalten, bis eins der folgenden Ereignisse eingetritt:

  semval nimmt den Wert 0 an. Dann wird der Wert von *semzcnt* des angegebenen Semaphors um 1 vermindert.

  Die Kennzahl *semid* des Semaphors, für das der aufrufende Prozeß auf eine Operation wartet, wird im System gelöscht. In diesem Fall wird *errno* auf [EIDRM] gesetzt und der Wert -1 zurückgeliefert.

  Der aufrufende Prozeß empfängt ein abzufangendes Signal. In diesem Fall wird der Wert von *semzcnt* des angegebenen Semaphors um 1 vermindert und der aufrufende setzt seine Ausführung in der Weise fort, wie dies bei der Funktion *sigaction()* beschrieben ist.

Bei erfolgreicher Ausführung wird der Wert von *sempid* für alle in dem Vektor, auf den *sops* zeigt, enthaltenen Semaphore gleich der Prozeßnummer des aufrufenden Prozesses gesetzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *semop()* den Wert 0 zurück. In allen anderen Fällen wird der Wert -1 geliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *semop()* schlägt fehl, wenn gilt:

[E2BIG] Der Wert von *nsops* ist größer als das vom System festgelegte Maximum.

[EACCES] Der Prozeß hat für das auszuführende Kommando nicht die erforderliche Zugriffsberechtigung siehe *Abschnitt 2.6*.

[EAGAIN] Diese Operation würde dazu führen, daß der aufrufende Prozeß angehalten wird, (*sem_flg & IPC_NOWAIT*) ist jedoch ungleich 0.

[EFBIG] Der Wert von *sem_num* ist kleiner 0 oder größer als oder gleich der Anzahl von Semaphoren in der mit *semid* bezeichneten Semaphorenmenge.

[EIDRM] Die Semaphorkennzahl *semid* wurde im System gelöscht.

[EINTR] Die Funktion *semop()* wurde durch ein Signal unterbrochen.

[EINVAL] Der Wert von *semid* ist keine gültige Semaphorkennzahl oder die Anzahl der einzelnen Semaphore, für die der aufrufende Prozeß ein SEM_UNDO anfordert, würde die vom System festgelegte Höchstgrenze überschreiten.

[ENOSPC] Die zulässige Höchstzahl an Prozessen, die ein SEM_UNDO anfordern dürfen, würde überschritten.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

[ERANGE] Eine Operation würde dazu führen, daß ein *semval* oder ein *semadj* den systemspezifischen Maximalwert überschreiten würde.

PORTABILITÄT

Die Funktion *semop()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

exec, *exit()*, *fork()*, *semctl()*, *semget()*, *<sys/ipc.h>*, *<sys/sem.h>*, *<sys/types.h>*.

NAME

setbuf - assign buffering to a stream
Puffer an Datenstrom zuweisen

DEFINITION

```
#include <stdio.h>
void setbuf (stream, buf)
FILE *stream;
char *buf;
```

BESCHREIBUNG

Außer, daß er kein Ergebnis liefert, ist der Funktionsaufruf

```
setbuf(stream, buf)
```

äquivalent zu

```
setvbuf(stream, buf, _IOFBF, BUFSIZ)
```

wenn *buf* nicht der Nullzeiger ist, oder zu

```
setvbuf(stream, buf, _IONBF, BUFSIZ)
```

wenn *buf* gleich dem Nullzeiger ist.

ERGEBNIS

Die Funktion *setbuf()* hat kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Eine häufige Fehlerquelle ist es, den Platz für den Puffer als "automatische" Variable in einem Programmblock zu definieren und es dann zu versäumen, den Datenstrom noch im selben Block zu schließen.

Die Bereitstellung eines Puffers der Größe *size* mit *setbuf()* bedeutet nicht notwendigerweise, daß auch alle *size* Bytes wirklich für den Pufferbereich verwendet werden.

PORTABILITÄT

Die Funktion *setbuf()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die Pipe wird auf ungepuffert gesetzt. Damit erscheint nach Drücken der return-Taste sofort die gemäß dem Kommando *tr* veränderte Eingabe am Bildschirm:

```
#include <stdio.h>

main()
{
   FILE *out,*popen();
   int c;

   /* Pipe zwischen Prozess und tr-Kommando */
   /*  einrichten                           */
   /* Prozess schreibt auf die              */
   /*  Standardeingabe des Kommandos        */
   out = popen("tr \"a-z\" \"A-Z\"","W");

                                  /* Ausgabe auf die Pipe ungepuffer
   setbuf(out,NULL);
   while((c=getchar()) != EOF)
      putc(c,out);
   pclose(out);
}
```

Wenn Sie im obigen Programm den *setbuf()*-Aufruf weglassen oder kommentieren, dann wird die Ausgabe auf *out* gepuffert, d.h. erst wenn der Puffer vollgeschrieben ist, erfolgt eine Ausgabe auf den Bildschirm.

SIEHE AUCH

fopen(), *setvbuf()*, *<stdio.h>*.

NAME

setcfadent, setcfgrent, setcfprent, setcftyent
- set to start of structure entry
auf den Anfang des Struktureintrags positionieren

DEFINITION

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

void setcfadent()

void setcfgrent()

void setcfprent()

void setcftyent()
```

BESCHREIBUNG

Dieser Funktionen gehören zu einer Gruppe von Funktionen zur Druckerbeschreibung.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getcfadent()-getcftynam()*.

PORTABILITÄT

Alle vier Funktionen sind im X/Open-Standard nicht enthalten.

NAME

setgid - set group ID
Gruppennummer setzen

DEFINITION

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

int setgid (gid)
gid_t gid;
```

BESCHREIBUNG

Wenn der Prozeß besondere Rechte hat, dann setzt die Funktion *setgid()* die reale, die effektive und die gesicherte Gruppennummer gleich *gid*.

Wenn der Prozeß keine besonderen Rechte hat, aber *gid* gleich der realen oder der gesicherten Gruppennummer ist, dann setzt die Funktion *setgid()* die effektive Gruppennummer gleich *gid*, während die reale und gesicherte Gruppenummer unverändert bleiben.

Vorhandene weitere Gruppennummern des aufrufenden Prozesses bleiben unverändert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird der Wert -1 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *setgid()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert des Arguments *gid* ist ungültig und wird nicht unterstützt.

[EPERM] Der Prozeß besitzt keine besonderen Rechte und *gid* entspricht weder der realen noch der gesicherten Gruppennummer.

PORTABILITÄT

Die Funktion *setgid()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispiel zeigt ein Programmstück, mit dem die effektive Gruppennummer neu gesetzt wird. Abgefragt wird die reale Gruppennummer und dann wird die effektive Gruppennummer auf den gleichen Wert gesetzt. Ein solches Programm ist nur sinnvoll, wenn die effektive und reale Gruppennummer unterschiedlich sind und wenn während des Programmablaufs die effektive Gruppenummer geändert werden soll, so daß sie der realen Gruppennummer entspricht.

```
extern int errno;

main()
{
   int i;

   .
   .
   printf("reale Gruppennummer: %d\n",getgid());
   printf("effektive Gruppennummer: %d\n",getegid());

   i = setgid(getgid());

   if (i==0)
      printf ("Gruppennummer neu gesetzt: re Gr %d , eff Gr %d \n",
                                          getgid(), getegid());
   else
   {
      printf ("Nix passiert\n");
      printf("errno = %d\n",errno);
   }
   .
   .
}
```

SIEHE AUCH

exec, *getgid()*, *setuid()*, *<sys/types.h>*.

NAME

setgrent - rewind position in group database
auf Datcianfang der Gruppendatei positionieren

DEFINITION

```
#include <grp.h>
#include <stdio.h>

void setgrent ()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie die Gruppendatei */etc/group* bearbeiten können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getgrent()*.

PORTABILITÄT

Die Funktion *setgrent()* ist im X/Open-Standard nicht enthalten.

NAME

setjmp - set jump point for a non-local goto
Sprungmarke für nichtlokalen Sprung setzen

DEFINITION

```
#include <setjmp.h>

int setjmp (env)
jmp_buf env;
```

BESCHREIBUNG

Das Makro *setjmp()* sichert die Aufrufumgebung im Argument *env* für eine spätere Verwendung durch die Funktion *longjmp()*.

ERGEBNIS

Wenn die Rückkehr von einem direkten Aufruf erfolgt, dann liefert das Makro *setjmp()* den Wert 0. Wenn die Rückkehr von einem Aufruf der Funktion *longjmp()* erfolgt, dann liefert das Makro *setjmp()* einen Wert ungleich 0.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Wenn sich der Aufruf von *longjmp()* in einer anderen Funktion befindet, als der entsprechende Aufruf von *setjmp()*, dann können alle Variablen der Speicherklasse *register* unvorhersehbare Werte haben.

Das Makro *sigsetjmp()* ist nützlich bei der Behandlung von Fehlern und Signalen, die in einem Unterprogramm auf einer tieferen Ebene des Programms auftreten.

Ein Aufruf des Makros *setjmp()* sollte nur in folgenden Zusammenhängen auftreten:

- als vollständiger Kontrollausdruck einer Auswahl- oder Schleifenanweisung, z.B:
  ```
  if (setjmp(env))
      ...
  ```
- als ein Operand einer Vergleichsoperators, wobei der andere Operand ein konstanter ganzzahliger Ausdruck, und der Gesamtausdruck der vollständige Kontrollausdruck einer Auswahl- oder Schleifenanweisung ist, z.B:
  ```
  if (setjmp(env)==0)
      ...
  ```
- als Operand des einstelligen Operators !, wobei der Gesamtausdruck der vollständige Kontrollausdruck einer Auswahl- oder Schleifenanweisung ist, z.B:
  ```
  if (!setjmp(env))
      ...
  ```
- als vollständiger Ausdruck einer Ausdrucksanweisung (ggf. umgewandelt in den Typ (*void*), z.B:
  ```
  (void) setjmp(env);
  ```

PORTABILITÄT

Die Funktion *setjmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel unter *longjmp()*.

SIEHE AUCH

longjmp(), *sigsetjmp()*, *<setjmp.h>*.

NAME

setkey- set encoding key
Schlüssel einstellen

DEFINITION

```
void setkey (key)
char *key;
```

BESCHREIBUNG

Die Funktion *setkey()* bietet einen (ziemlich einfachen) Zugriff auf einen Verschlüsselungsalgorithmus. Das Argument von *setkey()* ist ein Vektor von Zeichen der Länge 64, der nur Zeichen mit den Werten 0 und 1 enthält. Wenn diese Zeichenkette in Gruppen von 8 Zeichen aufgeteilt wird, dann wird das niederwertigste "Bit" in jeder Gruppe ignoriert. Dies ergibt einen 56-Bit-Schlüssel, der vom Algorithmus verwendet wird. Dieser Schlüssel wird vom Algorithmus verwendet, um die Zeichenkette *block* in der Funktion *encrypt()* zu verschlüsseln.

ERGEBNIS

Es wird kein Ergebnis geliefert.

FEHLER

Die Funktion *setkey()* schlägt fehl, wenn gilt:

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *setkey()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

BEISPIEL

```
#include <stdio.h>

char schl[ ] =
{
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
};

char feld[ ] =
{
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,
   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
};

main()
{  void pr();

   setkey(schl); pr();

   encrypt(feld,0); pr();
   encrypt(feld,1); pr();
}

void pr()
{  register int i;

   for(i=0; i<sizeof(feld); i++)
      printf("%d",feld[i]);
   putchar('\n');
}
```

SIEHE AUCH

crypt(), *encrypt()*.

NAME

setlocale - set program locale
internationale Umgebung für Programm setzen

DEFINITION

```
#include <locale.h>

char *setlocale (category, locale)
int category;
char *locale;
```

BESCHREIBUNG

Die Funktion *setlocale()* wählt den passenden Teil der internationalen Umgebung des Programms, so wie er vom Argument *category* angegeben wird, und kann verwendet werden, um die internationale Umgebung des Programms zu wechseln oder abzufragen. Der Wert *LC_ALL* für *category* steht für die vollständige internationale Umgebung des Programms; andere Werte für *category* sprechen nur einen Teil der internationalen Umgebung des Programms an:

LC_COLLATE beeinflußt das Verhalten von regulären Ausdrücken und der NLS-Vergleichsfunktionen für Zeichenketten.

LC_CTYPE beeinflußt das Verhalten von regulären Ausdrücken und der Funktionen zur Zeichenklassifikation und -umwandlung.

LC_MONETARY beeinflußt das Verhalten von Funktionen, die Geldbeträge behandeln.

LC_NUMERIC beeinflußt den Dezimalpunkt für Funktionen zur formatierten Ein- und Ausgabe und die Zeichenketten-Umwandlungsfunktionen.

LC_TIME beeinflußt das Verhalten der Zeit-Umwandlungsfunktionen.

Das Verhalten der Sprach-Informationsfunktion *nl_langinfo()* wird ebenfalls durch die Einstellungen von *category* beeinflußt.

Das Argument *locale* ist ein Zeiger auf eine Zeichenkette, die die benötigten Einstellungen für *category* enthält. Die Inhalte dieser Zeichenketten sind abhängig von der Shellvariablen INTLINFO und den installierten NLS-Datenbasen.

Zusätzlich sind die folgenden vordefinierten Werte für alle Einstellungen von *category* definiert:

"C" spezifiziert die minimale Umgebung für die Programmiersprache C. Falls *setlocale()* nicht aufgerufen wird, ist die internationale Umgebung für "C" Standard. Das operationelle Verhalten innerhalb des internationalen Umgebung für "C" ist für jede Schnittstellen-Funktion einzeln definiert.

"" spezifiziert eine implementations-definierte landessprchliche Umgebung. Für SINIX-Systeme, den *IEEE Standard 1003.1-1988* und andere X/Open-Systeme entspricht dies dem Wert der zugehörigen Umgebungsvariablen LC_* bzw. LANG.

NULL wird verwendet, um die Funktion *setlocale()* anzuweisen, die aktuelle internationale Umgebung abzufragen und den Namen von *locale* zurückzugeben.

Internationalisierte Programme müssen die Funktion *setlocale()* aufrufen, um die Arbeit mit einer speziellen Sprache einzuleiten. Dies kann durch einen Aufruf von *setlocale()* geschehen, der wie folgt aussieht:

```
setlocale (LC_ALL, "");
```

Dieser Aufruf verwendet die Einstellungen der NLS Umgebungs-Variablen, um die internationale Umgebung des Programms zu initialisieren.

ERGEBNIS

Falls für *locale* eine Zeiger auf eine Zeichenkette angegeben wird und die Auswahl durchgeführt werden kann, liefert die Funktion *setlocale()* die Zeichenkette, die mit der der angegebenen *category* für die neue internationale Umgebung verbunden ist. Falls die Auswahl nicht ausgeführt werden kann, so liefert die Funktion *setlocale()* den Nullzeiger und die internationale Umgebung des Programms wird nicht verändert.

Ein Nullzeiger für *locale* veranlaßt die Funktion *setlocale()*, die Zeichenkette zurückzugeben, die der *category* der aktuellen internationalen Umgebung des Programms zugeordnet ist. Die internationale Umgebung des Programms wird dabei nicht verändert.

Die von der Funktion *setlocale()* zurückgelieferte Zeichenkette sieht derart aus, daß ein nachfolgender Aufruf mit dieser Zeichenkette und der zugehörigen *category* diesen Teil der internationalen Umgebung des Programms wiederherstellt. Die zurückgelieferte Zeichenkette wird nicht durch das Programm verändert, kann aber durch einen nachfolgenden Aufruf von *setlocale()* überschrieben werden.

HINWEIS

Die folgenden Programmanweisungen zeigen, wie ein Programm die internationale Umgebung für eine Sprache initialisieren kann, während die internationale Umgebung für eine Programm teilweise verändert wird, so daß reguläre Ausdrücke und Zeichenketten-Operationen auf einen fremdsprachigen Text angewendet werden können.

```
setlocale (LC_ALL, "De"); setlocale (LC_COLLATE,
"Fr@dict");
```

PORTABILITÄT

Die Funktion *setlocale()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel im *Abschnitt 2.5 (Funktion yesdata())*.

SIEHE AUCH

isalpha(), *isascii()*, *nl_langinfo()*, *printf()*, *scanf()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxrfm()*, *strtod()*, *<langinfo.h>*, *<locale.h>*, *Abschnitt 2.5*.

NAME

setpdjbent, setpdprent
- set to start of POOLDAT entry
auf den Anfang des Struktureintrags positionieren

DEFINITION

```
#include <lpr.h>

void setpdjbent()

void setpdprent()
```

BESCHREIBUNG

Diese Funktionen gehören zu einer Gruppe von Funktionen, mit denen Sie Auftragsdateien für Drucker durchsuchen können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getpdjbent()-getpdprent()*.

PORTABILITÄT

Beide Funktionen sind im X/Open-Standard nicht enthalten.

NAME

setpgid - set process group ID for Job Control
Prozessgruppennummer für Auftragskontrolle setzen

DEFINITION

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

int setpgid (pid, pgid)
pid_t pid, pgid;
```

BESCHREIBUNG

Falls {_POSIX_JOB_CONTROL} definiert ist:

Die Funktion *setpgid()* wird benutzt, um sich entweder einer existierenden Prozeßgruppe anzuschließen, oder um eine neue Prozeßgruppe innerhalb der Sitzung des aufrufenden Prozesses zu erzeugen. Die Prozeßgruppennummer des Prozeßgruppenchefs ändert sich nicht. Bei erfolgreicher Beendigung wird die Prozeßgruppennummer des Prozesses mit der Prozeßnummer, die zu *pid* paßt, auf *pgid* gesetzt. Als ein Spezialfall wird, wenn *pid* gleich 0 ist, die Prozeßnummer des aufrufenden Prozesses verwendet. Ebenso wird, wenn *pgid* gleich 0 ist, die Prozeßnummer des angegebenen Prozesses benutzt.

Andernfalls:

Entweder unterstützt die Implementierung die Funktion *setpgid()* so wie oben beschrieben, oder die Funktion *setpgid()* schlägt fehl.

ERGEBNIS

Bei erfolgreicher Beendigung liefert *setpgid()* der Wert 0. Andernfalls wird der Wert -1 zurückgegeben und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *setpgid()* schlägt fehl, wenn gilt:

[EACCES] Der Wert des Arguments *pid* entspricht der Prozeßnummer eines Sohnprozesses des aufrufenden Prozesses und der Sohnprozeß hat erfolgreich eine der *exec*-Funktionen aufgerufen.

[EINVAL] Der Wert des Arguments *pgid* ist kleiner als 0 oder ein Wert, der von der Implementierung nicht unterstützt wird.

[ENOSYS] Die Funktion *setpgid()* wird von der aktuellen Implementierung nicht unterstützt.

[EPERM] Der Prozeß, der durch *pid* angegeben wird ist ein Sitzungsführer.
Der Wert des Arguments *pid* entspricht der Prozeßnummer eines Sohnprozesses des aufrufenden Prozesses und der Sohnprozeß ist nicht in derselben Sitzung wie der aufrufende Prozeß.
Der Wert des Arguments *pgid* ist gültig, aber entspricht nicht der Prozeßnummer des Prozesses, der durch das Argument *pid* angesprochen wird und es gibt keinen Prozeß mit einer Prozeßgruppennummer die dem Wert von *pgid* in derselben Sitzung wie der aufrufende Prozeß entspricht.

[ESRCH] Der Wert des Arguments *pid* entspricht nicht der Prozeßnummer des aufrufenden Prozesses oder eines Sohnprozesses des aufrufenden Prozesses.

PORTABILITÄT

Die Funktion *setpgid()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

exec, *getpgrp()*, *setsid()*, *tcsetpgrp()*, *<sys/types.h>*.

NAME

setpgrp - set process group ID
Prozeßgruppennummer setzen

DEFINITION

```
int setpgrp ()
```

BESCHREIBUNG

setpgrp() definiert die Prozeßnummer des aufrufenden Prozesses als seine Prozeßgruppennummer. Die neue Prozeßgruppennummer wird zurückgeliefert.

Gehört der Prozeß zu einer Prozeßgruppe, so wird er durch *setpgrp()* aus dieser Gruppe herausgelöst, es sei denn, er ist Prozeßgruppenführer.

ERGEBNIS

setpgrp() liefert die neue Prozeßgruppennummer.

PORTABILITÄT

Die Funktion *setpgrp()* ist im X/Open-Standard nicht mehr enthalten. Anwendungen sollten die neue Funktion *setsid()* verwenden.

BEISPIEL

Im Beispiel wird die Prozeßnummer des aufrufenden Prozesses als Prozeßgruppennummer gesetzt. Mit der Funktion *system()* wird das Kommando *ps* mit dem Schalter *-lg* aufgerufen, so daß Sie über laufende Prozesse und deren Prozeßnummern informiert sind.

```
main()
{
   int i, j;

   system("ps -lg");
   j = getpgrp();
   printf("Prozessgruppennummer: %d\n", j);
   i = setpgrp();
   printf("Jetzt wurde Prozessgruppennummer neu gesetzt.\n");
   j = getpgrp();
   printf("Neue Prozessgruppennummer: %d\n", j);
   system("ps -lg");
   printf("Rueckgabewert von setprg: %d\n", i);
   system("ps -lg");
}
```

SIEHE AUCH

setsid().

NAME

setpwent - rewind index into the password file
auf Dateianfang der Kennwortdatei positionieren

DEFINITION

```
#include <pwd.h>
#include <stdio.h>

void setpwent ()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie die Kennwortdatei */etc/passwd* bearbeiten können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getpwent()*.

PORTABILITÄT

Die Funktion *setpwent()* ist im X/Open-Standard nicht definiert.

NAME

setsid - create session and set process group ID
Sitzung erzeugen und Prozessgruppennummer setzen

DEFINITION

```
#include <sys/types.h>
pid_t setsid()
```

BESCHREIBUNG

Die Funktion *setsid()* erzeugt eine neue Sitzung, falls der aufrufende Prozeß kein Prozeßgruppenchef ist. Nach der Rückkehr dieser Funktion ist der aufrufende Prozeß der Sitzungsführer dieser neuen Sitzung und der Prozeßgruppenchef einer neuen Prozeßgruppe. Außerdem besitzt der Prozeß kein kontrollierendes Terminal. Die Prozeßgruppennummer des aufrufenden Prozesses wird gleich der Prozeßnummer des aufrufenden Prozesses gesetzt. Der aufrufende Prozeß ist der einzige Prozeß in der neuen Prozeßgruppe und der einzige Prozeß in der neuen Sitzung.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *setsid()* die Prozeßgruppennummer des aufrufenden Prozesses. Andernfalls liefert sie (*pid_t*) -1 und die Variable *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *setsid()* schlägt fehl wenn gilt:

[EPERM] Der aufrufende Prozeß ist bereits ein Prozeßgruppenchef oder die Prozeßgruppennummer eines anderen Prozesses als des aufrufenden stimmt mit der Prozeßnummer des aufrufenden Prozesses überein.

PORTABILITÄT

Die Funktion *setsid()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

setpgid(), *<sys/types.h>*.

NAME

setuid - set user ID
Benutzernummer setzen

DEFINITION

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

int setuid (uid)
uid_t uid;
```

BESCHREIBUNG

Wenn der Prozeß besondere Rechte hat, dann setzt die Funktion *setuid()* die reale, die effektive und die gesicherte Benutzernummer gleich *uid*.

Wenn der Prozeß keine besonderen Rechte hat, aber *uid* gleich der realen oder der gesicherten Benutzernummer ist, dann setzt die Funktion *setuid()* die effektive Benutzernummer gleich *uid*. Die reale und die gesicherte Benutzernummer bleiben unverändert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird der Wert −1 zurückgegeben und *errno* besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *setuid()* schlägt fehl, liefert den Wert −1 und setzt *errno* auf den entsprechenden Wert, wenn mindestens eine der folgenden Bedingungen wahr ist:

[EINVAL] Der Wert des Arguments *uid* ist nicht gültig und wird nicht unterstützt.

[EPERM] Der Prozeß hat keine besonderen Rechte und *uid* entspricht auch nicht seiner realen oder gesicherten Benutzernummer.

HINWEIS

Eine besonders häufige Anwendung der Funktion *setuid()* ist die Aufgabe von nicht mehr benötigten Rechten in Programmen mit gesetztem s-Bit für den Eigentümer (insbesondere *root*). Solche Programme benötigen die durch das s-Bit gewährten Rechte oft nur für ganz bestimmte Aufgaben. Werden die Rechte nicht mehr benötigt, so können sie durch einen Aufruf der Form

```
erg = setuid(getuid());
```

wieder aufgegeben werden.

PORTABILITÄT

Die Funktion *setuid()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispiel zeigt ein Programmstück, mit dem die effektive Benutzernummer neu gesetzt wird. Abgefragt wird die reale Benutzernummer und dann wird die effektive Benutzernummer auf den gleichen Wert gesetzt. Ein solches Programm ist nur sinnvoll, wenn die effektive und reale Benutzernummer unterschiedlich sind und wenn während des Programmablaufs die effektive Benutzernummer geändert werden soll, so daß sie der realen Benutzernummer entspricht.

```
extern int errno;

main()
{
    int i;

    .
    .
    printf("reale Benutzernummer: %d\n",getuid());
    printf("effektive Benutzernummer: %d\n",geteuid());

    i = setuid(getuid());

    if (i==0)
       printf ("Benutzernummer neu gesetzt\n");
    else
    {
       printf ("Nix passiert\n");
       printf("errno = %d\n",errno);
    }

    .
    .
}
```

SIEHE AUCH

exec, *geteuid()*, *getuid()*, *setgid()*, *<sys/types.h>*.

NAME

setutent - rewind utmp file
Auf Anfang der utmp-Datei positionieren

DEFINITION

```
#include <utmp.h>

void setutent()
```

BESCHREIBUNG

Die Funktion *setutent()* gehört zu einer Gruppe von Funktionen, die Einträge in der Datei */etc/utmp* bearbeiten. Sie positioniert auf den Anfang der *utmp*-Datei. Eine ausführliche Beschreibung finden Sie unter *getutent()*.

PORTABILITÄT

Die Funktion *setutent()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

NAME

setvbuf - assign buffering to a stream
Pufferung für Datenstrom zuordnen

DEFINITION

```
#include <stdio.h>

int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type;
size_t size;
```

BESCHREIBUNG

Die Funktion *setvbuf()* kann verwendet werden, nachdem der Datenstrom *stream* einer offenen Datei zugeordnet wurde aber bevor irgeneine andere Aktion auf dem Datenstrom ausgeführt wurde. Das Argument *type* bestimmt folgendermaßen, wie *stream* gepuffert werden soll:

- _IOFBF sorgt für vollständige Pufferung der Ein- und Ausgaben
- _IOLBF sorgt für zeilenweise Pufferung
- _IONBF sorgt für ungepufferte Ein- und Ausgabe

Wenn *buf* nicht der Nullzeiger ist, dann kann der Vektor, auf den *buf* zeigt; anstelle eines von der Funktion *setvbuf()* reservierten Puffers verwendet werden. Das Argument *size* gibt die Größe des Vektors an. Der Inhalt des Vektors ist zu jeder Zeit unbestimmt.

Wenn ein Datenstrom nicht gepuffert wird, dann ist beabsichtigt, daß die Zeichen so bald wie möglich von der Quelle geholt oder zum Ziel zugestellt werden. Andernfalls können die Zeichen gesammelt und als Block zu oder von der Rechnerumgebung übertragen werden. Wenn ein Datenstrom vollständig gepuffert wird, dann sollen die Zeichen als Block zwischen Programm und Rechnerumgebung übertragen werden, wenn der Puffer gefüllt ist. Wenn ein Datenstrom zeilenweise gepuffert wird, dann sollen die Zeichen als Block zwischen Programm und Rechnerumgebung übertragen werden, wenn ein Neue-Zeile-Zeichen auftritt.

Zusätzlich sollen die Zeichen als Block zwischen Programm und Rechnerumgebung übertragen werden, wenn der Puffer gefüllt ist, wenn Eingaben von einem ungepufferten Datenstrom angefordert werden oder wenn Eingaben von einem zeilenweise gepufferten Datenstrom angefordert werden, der die Übertragung von Daten aus der Rechnerumgebung benötigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *setvbuf()* den Wert 0. Andernfalls wird ein Wert ungleich 0 zurückgeliefert, wenn ein ungültiger Wert für *type* angegeben wurde, oder die Anforderung nicht ausgeführt werden kann.

FEHLER

Die Funktion *setvbuf()* kann fehlschlagen, wenn gilt:

[EBADF] Die dem Datenstrom *stream* zugrundeliegende Dateikennzahl ist ungültig.

HINWEIS

Eine häufige Fehlerquelle ist die Reservierung des Puffers als eine "automatische" Variable in einem Programmblock, und dann das Versäumnis, den Datenstrom im selben Block zu schließen.

PORTABILITÄT

Die Funktion *setvbuf()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fopen(), *setbuf()*, *<stdio.h>*.

NAME

shmat - shared memory attach operation
Gemeinsamen Speicherbereich anhängen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat(shmid, shmaddr, shmflg)
int shmid;
char *shmaddr
int shmflag;
```

BESCHREIBUNG

Die Funktion *shmat()* hängt das mit der shared-memory-Speicherkennzahl *shmid* bezeichnete, gemeinsam benutzte Speichersegment an das Datensegment des aufrufenden Prozesses an. An welcher Stelle das Segment angehängt wird, richtet sich nach folgenden Kriterien:

- Ist *shmaddr* gleich 0, so wird das Segment an der ersten vom System festgestellten freien Adresse angehängt.
- Sind *shmaddr* und (*shmflg & SHM_RND*) ungleich 0, so wird das Segment an der mit (*shmaddr-(shmaddr % SHMLBA*)) angegebenen Adresse angehängt (Das Zeichen % ist der Modulo-Operator der Sprache C).
- Ist *shmaddr* ungleich 0 und (*shmflg & SHM_RND*) gleich 0, so wird das Segment an der mit *shmaddr* angegebenen Adresse angehängt.
- Das Segment wird zum Lesen angehängt, wenn (*shmflg & SHM_RDONLY*) ungleich 0 ist und der aufrufende Prozeß Leseberechtigung hat. Andernfalls, wenn (*shmflg & SHM_RDONLY*) gleich 0 ist und der aufrufende Prozeß Schreib- und Leseberechtigung hat, wird das Segment zum Lesen und Schreiben angehängt.

Die folgenden symbolischen Namen sind in der Include-Datei <*sys/shm.h*> definiert:

Name	Beschreibung
SHMLBA	Vielfaches der Adresse der Segmentuntergrenze
SHM_RDONLY	Anhängen nur zum Lesen (sonst: Lesen und Schreiben)
SHM_RND	Anhängeadresse aufrunden

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *shmat()* die Startadresse des Datensegments für das angehängt gemeinsame Speichersegment. Andernfalls wird das gemeinsame Speichersegment nicht angehängt, *shmat()* liefert den Wert -1 und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *shmat()* schlägt fehl, wenn gilt:

[EACCES] Dem aufrufenden Prozeß werden die für die Operation benötigten Zugriffsrechte verweigert. Siehe *Abschnitt 2.6.*

[EINVAL] Der Wert von *shmid* ist keine gültige shared-memory-Speicherkennzahl; oder der Wert von *shmaddr* ist ungleich 0 und der Wert von *(shmaddr - (shmaddr % SHMLBA))* ist eine unzulässige Adresse für das Anhängen von gemeinsamem Speicher; oder der Wert von *shmaddr* ist ungleich 0, *(shmflg & SHM_RND)* ist gleich 0 und der Wert von *shmaddr* ist eine unzulässige Adresse für das Anhangen von gemeinsamem Speicher.

[EMFILE] Die Anzahl der beim aufrufenden Prozeß angehängten gemeinsamen Speichersegmente würde die systemspezifische Grenze überschreiten.

[ENOMEM] Der verfügbare Datenspeicher ist nicht groß genug, um das gemeinsame Speichersegment unterzubringen.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *shmat()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funtion unterstützen müssen.

SIEHE AUCH

exec, exit(), fork(), shmctl(), shmdt(), shmget(), <sys/ipc.h>, <sys/shm.h>, <sys/types.h>.

NAME

shmctl - shared memory control operations
Kontrolloperationen für gemeinsame Speicherbereiche

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```

BESCHREIBUNG

Die Funktion *shmctl()* bietet eine Vielzahl von Steuerungsoperationen für gemeinsame Speicherbereiche ("shared memory"), die mit *cmd* angegeben werden. Die folgenden Werte für *cmd* sind verfügbar:

IPC_STAT
: Aktuelle Werte aller Komponenten der *shmid* zugeordneten Datenstruktur *shmid_ds* in die Struktur eintragen, auf *buf* zeigt. Der Aufbau der Struktur wird in *<sys/shm.h>* definiert.

IPC_SET
: Die Werte folgender Komponenten der *shmid* zugeordneten Datenstruktur *shmid_ds* auf die entsprechenden Werte aus der Struktur setzen, auf die *buf* zeigt:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode    /* nur 9 niederwertige Bits */
```

IPC_SET kann nur von einem Prozeß ausgeführt werden, dessen effektive Benutzernummer gleich der eines Prozesses mit besonderen Rechten oder gleich dem Wert von *shm_perm.cuid* bzw. *shm_perm.uid* in der *shmid* zugeordneten Datenstruktur *shmid_ds*.

IPC_RMID
: Die mit *shmid* angegebene shared-memory-Speicherkennzahl im System sowie das dazugehörige Speichersegment und die dazugehörige Datenstruktur *shmid_ds* löschen. IPC_RMID kann nur von einem Prozeß ausgeführt werden, dessen effektive Benutzernummer gleich der eines Prozesses mit besonderen Rechten oder gleich dem Wert von *shm_perm.cuid* bzw. *shm_perm.uid* in der *shmid* zugeordneten Datenstruktur *shmid_ds*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *shmctl()* den Wert 0; andernfalls liefert sie den Wert -1 und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *shmctl()* schlägt fehl, wenn gilt:

[EACCES] Das Argument *cmd* ist gleich IPC_STAT und der aufrufende Prozeß keine Leseberechtigung (siehe *Abschnitt 2.6*).

[EINVAL] Der Wert von *shmid* ist keine gültige shared-memory-Speicherkennzahl, oder der Wert von *cmd* ist kein gültiges Kommando.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

[EPERM] Das Argument *cmd* ist gleich IPC_RMID oder IPC_SET und die effektive Benutzernummer des aufrufenden Prozesses nicht die eines Prozesses mit besonderen Rechten und stimmt nicht mit *shm_perm.cuid* oder *shm_perm.uid* in der *shmid* zugeordneten Datenstruktur überein.

PORTABILITÄT

Die Funktion *shmctl()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

shmat(), *shmdt()*, *shmget()*, *<sys/ipc.h>*, *<sys/shm.h>*, *<sys/types.h>*.

NAME

shmdt - shared memory detach operation
Gemeinsamen Speicherbereich abhängen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt(shmaddr)
char *shmaddr
```

BESCHREIBUNG

Die Funktion *shmdt()* hängt das gemeinsame Speichersegment vom Datensegment des aufrufenden Prozesses ab, das sich an der durch *shmaddr* angegebenen Adresse befindet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *shmdt()* den Wert 0. Andernfalls wird das gemeinsame Speichersegment nicht abgehängt, *shmdt()* liefert den Wert -1 und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *shmdt()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert von *shmaddr* ist nicht die Datensegment-Startadresse eines gemeinsamen Speichersegments.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *shmdt()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktionen unterstützen müssen.

SIEHE AUCH

exec, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*, *<sys/ipc.h>*, *<sys/shm.h>*, *<sys/types.h>*.

NAME

shmget - get shared memory segment
Gemeinsames Speichersegment anlegen

DEFINITION

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key, size, shmflg)
key_t key;
int size, shmflg;
```

BESCHREIBUNG

Die Funktion *shmget()* liefert die *key* zugeordnete shared-memory-Speicherkennzahl.

Es wird eine shared-memory-Speicherkennzahl mit dazugehöriger Datenstruktur und das dazugehörige gemeinsame Speichersegment in einer Größe von mindestens *size* Bytes (siehe *<sys/shm.h>*) für *key* eingerichtet, wenn eine der folgenden Bedingungen zutrifft:

- Das Argument *key* hat den Wert IPC_PRIVATE.
- Das Argument *key* besitzt noch keine ihm zugeordnete shared-memory-Speicherkennzahl und (*shmflg & IPC_CREAT*) ist ungleich 0.

Beim Einrichten der neuen shared-memory-Speicherkennzahl wird die dazugehörige Datenstruktur wie folgt initialisiert:

- Die Werte von *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid* und *shm_perm.gid* werden gleich der effektiven Benutzer- bzw. Gruppennummer des aufrufenden Prozesses gesetzt.
- Die 9 niederwertigen Bits von *shm_perm.mode* werden gleich den 9 niederwertigen Bits von *shmflg* gesetzt. Das Argument *shm_segsz* wird auf den Wert von *size* gesetzt.
- Die Werte von *shm_lpid*, *shm_nattch*, *shm_atime* und *shm_dtime* werden auf 0 gesetzt.
- Für *shm_ctime* wird die aktuelle Zeit eingetragen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *shmget()* eine nichtnegative ganze Zahl zurück, die sogenannte shared-memory-Speicherkennzahl.

Andernfalls wird der Wert –1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *shmget()* schlägt fehl, wenn gilt:

[EACCES] Es existiert eine shared-memory-Speicherkennzahl für das Argument *key*, aber die in den 9 niederwertigen Bits von *shmflg* angegebene Berechtigung wurde nicht erteilt.

[EEXIST] Für *key* existiert eine shared-memory-Speicherkennzahl, aber ((*shmflg & IPC_CREAT) && (shmflg & IPC_EXCL*)) ist ungleich 0.

[EINVAL] Der Wert von *size* ist kleiner als die systemspezifische Mindestgröße oder größer als die systemspezifische Maximalgröße.
Oder für das Argument *key* existiert bereits eine shared-memory-Speicherkennzahl aber die Größe des ihr zugeordneten Segments ist kleiner als *size* und *size* ist ungleich 0.

[ENOENT] Für das Argument *key* existiert keine shared-memory-Speicherkennzahl und (*shmflg & IPC_CREAT*) ist gleich 0.

[ENOMEM] Es sollen eine shared-memory-Speicherkennzahl und das zugehörige gemeinsame Speichersegment eingerichtet werden, diese Anforderung kann jedoch wegen physikalischem Speicherplatzmangel nicht erfüllt werden.

[ENOSPC] Es soll eine shared-memory-Speicherkennzahl eingerichtet werden, dadurch würde jedoch die systemspezifische Höchstzahl an im System insgesamt zulässigen shared-memory-Speicherkennzahlen überschritten.

[ENOSYS] Die Funktionalität wird unter dieser Implementierung nicht unterstützt.

PORTABILITÄT

Die Funktion *shmget()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht alle Implementierungen diese Funktion unterstützen müssen.

SIEHE AUCH

shmat(), *shmctl()*, *shmdt()*, *<sys/ipc.h>*, *<sys/shm.h>*, *<sys/types.h>*.

NAME

sigaction - examine and change signal action
Signal-Aktionen untersuchen und ändern

DEFINITION

```
#include <signal.h>

int sigaction (sig, act, oact)
int sig;
struct sigaction *act, *oact;
```

BESCHREIBUNG

Die Funktion *sigaction()* erlaubt es dem aufrufenden Prozeß, die mit einem bestimmten Signal verbundene Aktion zu untersuchen oder festzulegen. Das Argument *sig* spezifiziert das Signal; mögliche Werte sind in der Datei <*signal.h*> definiert.

Die Struktur *sigaction*, die zur Beschreibung der zu wählenden Aktion benutzt wird, ist in der Datei <*signal.h*> definiert und beinhaltet mindestens die folgenden Komponenten:

Komponenten-Typ	Name	Beschreibung
void(*)()	sa_handler	SIG_DFL, SIG_IGN oder ein Zeiger auf eine Funktion
sigset_t	sa_mask	Zusätzliche Menge von Signalen, die während der Abarbeitung der Signalbehandlungs-Funktion blockiert werden sollen
int	sa_flags	Spezielle Kennzeichen um das Verhalten des Signals zu beeinflußen

Falls das Argument *act* ungleich NULL ist, so zeigt es auf eine Struktur, welche die dem speziellen Signal zugeordnete Aktion beschreibt. Falls das Argument *oact* ungleich NULL ist, so wird die vorher dem Signal zugeordnete Aktion in dem Bereich abgespeichert, auf den *oact* zeigt. Falls das Argument *act* gleich NULL ist, bleibt die Signalbehandlung unverändert; daher kann dieser Aufruf verwendet werden, um die aktuelle Behandlung für ein gegebenes Signal abzufragen. Die Komponente *sa_handler* der Struktur *sigaction* identifiziert die Aktion, die dem angegebenen Signal zugeordnet ist. Wenn die Komponente *sa_handler* eine Signalbehandlungs-Funktion spezifiziert, dann gibt die Komponente *sa_mask* eine Reihe von Signalen an, die vor Aufruf der Signalbehandlungs-Funktion zu der Signal-Maske des Prozesses hinzugefügt werden. Die Signale SIGKILL und

SIGSTOP werden durch diesen Mechanismus nicht zur Signalmaske des Prozesses hinzugefügt; diese Einschränkung wird vom System erzwungen, ohne daß ein Fehler angezeigt wird.

Die Komponente *sa_flags* kann verwendet werden, um die das Verhalten des angegebenen Signals zu Verändern.

Das folgende, in der Datei <*signal.h*> definierte Kennzeichen-Bit kann in *sa_flags* gesetzt werden:

SA_NOCLDSTOP Verhindert, daß SIGCHLD generiert wird, wenn ein Sohnprozeß anhält.

Wenn *sig* gleich SIGCHLD ist, SA_NOCLDSTOP nicht in *sa_flags* gesetzt ist dann wird das Signal SIGCHLD jedesmal an den aufrufenden Prozeß gesendet, wenn irgeneiner seiner Sohn-Prozesse anhält. Wenn *sig* gleich SIGCHLD ist und SA_NOCLDSTOP in *sa_flags* gesetzt ist, dann wird kein SIGCHLD-Signal auf diese Weise generiert.

Wenn ein Signal durch eine mit der Funktion *sigaction()* installierte Signalbehandlungs-Funktion abgefangen wird, dann wird eine neue Signalmaske berechnet und für die Dauer der Signalbehandlungs-Funktion installiert (bzw. bis entweder die Funktion *sigprocmask()* oder die Funktion *sigsuspend()* aufgerufen wird). Diese Maske wird aus der Vereinigung der aktuellen Signalmaske und dem Wert aus *sa_mask* für das gesendete Signal gebildet, einschließlich des gesendeten Signals selbst. Falls die benutzerdefinierte Signalbehandlungsroutine normal beendet wird, wird die ursprüngliche Maske wiederhergestellt.

Sobald eine Aktion für ein spezielles Signal installiert worden ist, bleibt es installiert, bis eine andere Aktion explizit verlangt wird (durch einen weiteren Aufruf der Funktion *sigaction()*),. oder bis eine der *exec*-Funktionen aufgerufen wird.

Wenn die vorherige Aktion für *sig* durch die Funktion *signal()* installiert wurde, die im Standard *Draft ANSI X3.159 Programming Language C* definiert ist, sind die Komponenten der Struktur auf die *oact* zeigt nicht spezifiziert und in der speziellen Komponente *oact->sv_handler* befindet sich nicht notwendig derselbe Wert, der vorher der Funktion *signal()* übergeben wurde. Trotzdem wird, wenn ein Zeiger auf dieselbe Struktur oder einer Kopie davon über das Argument *act* an einen nachfolgenden Aufruf der Funktion *sigaction()* übergeben wird, die Signalbehandlung so sein, als ob der ursprüngliche Aufruf der Funktion *signal()* wiederholt worden wäre.

Wenn die Funktion *sigaction()* fehlschlägt, dann wird keine neue Signalbehandlungs-Funktion installiert.

Man sagt, ein Signal wird für einen Prozeß *generiert* (oder an einen Prozeß *geschickt*), wenn das Ereignis, welches das Signal auslöst, erstmalig eintritt. Beispiele für solche Ereignisse schließen die Erkennung von Hardware-Fehlern, das Ablaufen von Timern und Bildschirm-Aktivitäten ebenso ein, wie den Aufruf der Funktion *kill()*. Unter bestimmten Umständen generiert ein Ereignis Signale für mehrere Prozesse.

Für jeden Prozeß ist vom System eine eigene Aktion als Antwort auf jedes einzelne Signal definiert (siehe *Signal-Aktionen*). Ein Signal nennt man *zugestellt* an einen Prozeß, wenn die entsprechende Aktion für den Prozeß und das Signal gestartet wird.

In der Zeit zwischen der Generierung und der Zustellung eines Signals heißt ein Signal *anstehend*. Normalerweise kann dieses Zeitintervall nicht von einer Anwendung erkannt werden. Dennoch kann ein Signal von der Zustellung an einen Prozess abgehalten, es kann *blockiert* werden. Wenn die Aktion, die einem blockierten Signal zugeordnet ist irgendeine andere Aktion als das Ignorieren des Signals ist und das Signal für den Prozeß generiert wurde, dann bleibt das Signal solange anstehend bis entweder die Blockierung aufgehoben wird oder bis die diesem Signal zugeordnete Aktion gleich *Ignorieren* gesetzt wird. Wenn die einem blockierten Signal zugeordnete Aktion das Ignorieren des Signals ist und das Signal für den Prozeß generiert wurde, dann ist es nicht festgelegt, ob das Signal sofort nach der Generierung aufgegeben wird oder ob es anstehend bleibt.

Jeder Prozeß besitzt eine *Signalmaske*, die diejenigen Signale definiert, die derzeit vor der Zustellung an diesen Prozeß blockiert werden. Die Signalmaske eines Prozesses wird von dessen Vaterprozeß initialisiert. Die Funktionen *sigaction()*, *sigprocmask()* und *sigsuspend()* kontrollieren die Manipulation dieser Signalmaske.

Die Entscheidung, welche Aktion als Antwort auf ein Signal ausgeführt wird, wird zu dem Zeitpunkt getroffen, zu dem das Signal zugestellt wird. Dabei können auch nach dem Zeitpunkt der Generierung beliebige Änderungen vorgenommen werden. Diese Entscheidung ist unabhängig von dem Weg, auf dem ein Signal ursprünglich generiert wurde. Falls ein bereits anstehendes Signal generiert wird, ist es undefiniert, ob dieses Signal mehr als einmal zugestellt wird. Die Reihenfolge, in der

mehrere gleichzeitig anstehende Signale an einen Prozeß zugestellt werden, ist nicht festgelegt.

Wenn ein Halte-Signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) für einen Prozeß generiert wird, so wird ein evtl. anstehendes Signal des Typs SIGCONT aufgegeben. Umgekehrt werden, sobald ein Signal des Typs SIGCONT für einen Prozeß generiert wird, alle noch ausstehenden Halte-Signale für diesen Prozeß aufgegeben. Wenn SIGCONT für einen Prozeß generiert wird, der angehalten ist, so wird dieser Prozeß fortgesetzt, auch wenn das Signal SIGCONT blockiert ist oder ignoriert wird. Wenn das Signal SIGCONT blockiert ist und nicht ignoriert wird, dann bleibt es anstehend bis es entweder freigegeben wird oder bis ein ein Halte-Signal für den Prozeß generiert wird.

Signal-Aktionen

Es gibt drei Arten von Aktionen, die einem Signal zugeordnet werden können: SIG_DFL, SIG_IGN oder ein Zeiger auf eine Funktion. Zu Anfang, d.h vor Eintritt in die Funktion *main()*, sind alle Signale auf SIG_DFL oder SIG_IGN gesetzt (siehe unter *exec*). Die folgenden Aktionen, werden durch diese Werte beschrieben:

SIG_DFL - signal-spezifische Standard-Aktion

- Die Standard-Aktionen für die in diesem Handbuch definierten Signale werden in den vorhergehenden Tabellen beschrieben.
- Wenn die Standard-Aktion das Anhalten des Prozesses ist, dann wird die Ausführung des Prozesses zeitweilig unterbrochen. Wenn ein Prozeß anhält, dann wird ein Signal des Typs SIGCHLD für dessen Vater-Prozeß generiert, solange dieser nicht das Flag SA_NOCLDSTOP gesetzt hat. Solange ein Prozeß angehalten ist, werden alle weiteren, Signale, die an diesen Prozeß gesendet werden nicht mehr zugestellt, bis der Prozeß fortgesetzt wird. Eine Ausnahme bildet das Signal SIGKILL, das den empfangenden Prozeß immer abbricht. Einem Prozeß, der Miglied in einer verwaisten Prozeßgruppe ist, ist es nicht erlaubt, als Antwort auf eines der Signale SIGTSTP, SIGTTIN oder SIGTTOU anzuhalten. In den Fällen, in denen die Zustellung eines dieser Signale einen solchen Prozeß anhalten würden, wird dieses Signal aufgegeben.
- Wenn die Signal-Aktion für ein anstehendes Signal, dessen Standard-Aktion das Ignorieren dieses Signals ist, auf SIG_DFL gesetzt wird (z.B. SIGCHLD), so wird das anstehende Signal aufgegeben, gleichgültig ob es blockiert ist oder nicht.

SIG_IGN - Signal ignorieren

- Die Zustellung des Signals hat keine Wirkung auf den Prozeß. Das Verhalten eines Prozesses ist undefiniert, nachdem dieser eines der Signale SIGFPE, SIGILL oder SIGSEGV ignoriert hat, sofern diese Signale nicht durch die Funktion *kill()* oder die Funktion *raise()* generiert wurde, die im *Draft ANSI X3.159 Programming Language C* definiert sind.
- Das System erlaubt die Aktion SIG_IGN nicht für die Signale SIGKILL oder SIGSTOP. Wenn die Signal-Aktion für ein anstehendes Signal auf SIG_IGN gesetzt wird, dann wird das anstehende Signal aufgegeben, gleichgültig ob es blockiert ist oder nicht.
- Wenn ein Prozeß die Aktion für das Signal SIGCHLD auf SIG_IGN setzt, so ist das Verhalten nicht festgelegt.

Zeiger auf Funktion - Signal abfangen

- Bei der Zustellung eines Signals hat der empfangende Prozeß eine das Signal abfangende Funktion an einer festgelegten Adresse auszuführen. Nach der Rückkehr aus der Signalbehandlungs-Funktion nimmt der Prozeß die Ausführung an dem Punkt wieder auf, an dem er unterbrochen wurde.
- Die Signalbehandlungs-Funktion wird wie ein C-Funktion in der folgenden Art und Weise aufgerufen:

```
void func (signo)
int signo;
```

wobei *func* die angegebene Signalbehandlungs-Funktion ist und *signo* die Nummer des Signals, das zugestellt wird.

- Das Verhalten eines Prozesses ist dann undefiniert, wenn er normal von einer Fehlerbehandlungs-Funktion für eines der Signale SIGFPE, SIGILL oder SIGSEGV zurückkehrt, das nicht von der Funktion *kill()* oder der Funktion *raise()* generiert wurde, die im *Draft ANSI X3.159 Programming Language C* definiert sind.
- Das System verbietet es einem Prozeß, die Signale SIGKILL und SIGSTOP abzufangen.
- Wenn ein Prozeß eine Signalbehandlungs-Funktion für das Signal SIGCHLD einführt, während er einen beendeten Sohnprozeß besitzt, auf den er nicht wartet, dann ist nicht festgelegt, ob ein Signal des Typs SIGCHLD generiert wird, um diesen Sohn-Prozeß anzuzeigen.

Wirkung der Signale auf andere Funktionen Signale beeinflußen das Verhalten von zahlreichen in diesem Handbuch definierten Funktionen, wenn sie einem Prozeß zugestellt werden, der gerade eine dieser Funktionen ausführt. Falls die Aktion zu diesem

Signal das Abbrechen des Prozesses ist, dann wird der Prozeß abgebrochen und die Funktion kehrt nicht zurück. Wenn die Aktion zu diesem Signal das Anhalten des Prozesses ist, dann wird der Prozeß angehalten bis er fortgesetzt oder abgebrochen wird. Die Generierung eines SIGCONT-Signals für den Prozeß verursacht eine Fortsetzung des Prozesses an dem Punkt, an dem der Prozeß angehalten wurde. Wenn die Aktion zu diesem Signal das Aufrufen einer Signalbehandlungs-Funktion ist, dann wird die Signalbehandlungs-Funktion aufgerufen; in diesem Fall wird die ursprüngliche Funktion von dem Signal *unterbrochen*. Falls die Signalbehandlungs-Funktion eine *return*-Anweisung ausführt, dann ist das Verhalten der der unterbrochenen Funktion genauso, wie es für diese Funktion beschrieben ist. Signale, die ignoriert werden, beeinflußen das Verhalten keiner Funktion; Signale, die blockiert werden, haben keinen Einfluß auf das Verhalten irgendeiner Funktion bis sie zugestellt sind.

Wenn Signalbehandlungs-Funktionen asynchron zur Prozeßausführung aufgerufen werden, dann ist das Verhalten einiger in diesem Handbuch definierter Funktionen nicht definiert, falls diese aus einer Signalbehandlungs-Funktion heraus aufgerufen werden. Die folgende Tabelle definiert eine Reihe von Funktionen für die der *IEEE Standard 1003.1-1988* vorschreibt, daß sie entweder reentrant oder nicht durch Signale unterbrechbar sind. Daher können diese ohne Einschränkung von Anwendungen aus Signalbehandlungs-Routinen heraus aufgerufen werden:

access()	getegid()	rmdir()	tcflush()
alarm()	geteuid()	setgid()	tcgetattr()
cfgetispeed()	getgid()	setpgid()	tcgetpgrp()
cfgetospeed()	getgroups()	setsid()	tcsendbreak()
cfsetispeed()	getpgrp()	setuid()	tcsetattr()
cfsetospeed()	getpid()	sigaction()	tcsetpgrp()
chdir()	getppid()	sigaddset()	time()
chmod()	getuid()	sigdelset()	times()
chown()	kill()	sigemptyset()	umask()
close()	link()	sigfillset()	uname()
creat()	lseek()	sigismember()	unlink()
dup2()	mkdir()	sigpending()	ustat()
dup()	mkfifo()	sigprocmask()	utime()
execle()	open()	sigsuspend()	wait()
execve()	pathconf()	sleep()	waitpid()
_exit()	pause()	stat()	write()
fcntl()	pipe()	sysconf()	
fork()	read()	tcdrain()	

fstat() rename() tcflow()

Zusätzlich schreibt der *Draft ANSI X3.159 Programming Language C* für die folgenden Funktionen vor, das diese entweder reentrant sein müssen oder nicht durch Signale unterbrechbar sein dürfen:

signal()

X/Open bestimmt, daß die folgenden Funktionen entweder reentrant sein müssen oder nicht durch Signale unterbrechbar sein dürfen:

abort() exit()
chroot() longjmp()

Alle Funktionen, die nicht in den obigen Tabellen aufgeführt sind, gelten als unsicher in Bezug auf Signale. Falls eine unsichere Funktion durch eine Signalbehandlungs-Funktion unterbrochen wird, die dann irgendeine unsichere Funktion aufruft, so ist ihr Verhalten undefiniert.

Das Verhalten von unsicheren Funktionen, so wie in diesem Abschnitt definiert, ist undefiniert, wenn sie von Signalbehandlungs-Funktionen aus unter bestimmten Umständen aufgerufen werden. Das Verhalten von reentranten Funktionen, wie in diesem Abschnitt definiert, ist so, wie es in diesem Handbuch beschrieben wird, ungeachtet eines Aufrufs aus einer Signalbehandlungs-Funktion heraus. Dies ist die einzige beabsichtigte Bedeutung der Bemerkung, daß reentrante Funktionen ohne Einschränkung in Signalbehandlungs-Funktionen verwendet werden können. Anwendungen müssen dennoch alle Wirkungen solcher Funktionen berücksichtigen die sich auf Dinge wie Datenstrukturen, Dateien und Prozeß-Zustände beziehen. Insbesondere müssen die Autoren von Anwendungen die Einschränkungen von Interaktionen beachten, die sich bei der Unterbrechung der Funktion *sleep()* ergeben und die Interaktionen zwischen mehreren Kennzahlen für eine Dateibeschreibung. Die Tatsache, daß irgendeine bestimmte Funktion als reentrant aufgeführt ist, bedeutet nicht notwendigerweise, daß der Aufruf dieser Funktion aus einer Signalbehandlungs-Funktion heraus empfehlenswert ist.

Um Fehler zu vermeiden, die sich aus der Unterbrechung von nicht reentranten Funktionsaufrufen ergeben, sollten Anwendungen die Aufrufe solcher Funktionen entweder durch das Blockieren der entsprechenden Signale oder durch die Verwendung von programmatischen Semaphoren schützen. Dieses Handbuch spricht die allgemeineren Probleme der

Synchronisation des Zugriffs auf gemeinsam genutzte Datenstrukturen nicht an. Beachten Sie insbesondere, daß auch die "sicheren" Funktionen die globale Variable *errno* verändern können; die Signalbehandlungs-Funktion kann deren Wert sichern und wiederherstellen wollen. Selbstverständlich treffen dieselben Prinzipien auch auf reentrante Anwendungs-Funktionen und asynchronen Datenzugriff zu. Beachten sie, daß *siglongjmp()* nicht in der Liste der reentranten Funktionen enthalten ist. Dies kommt daher, daß der Code, der nach *siglongjmp()* ausgeführt wird, beliebige unsichere Funktionen aufrufen kann, mit denselben Gefahren, die beim Aufruf dieser unsicheren Funktionen direkt aus der Signalbehandlungs-Funktion auftreten. Anwendungen, die *siglongjmp()* aus Signalbehandlungs-Funktionen heraus verwenden, benötigen einen rigerosen Schutz, um portabel zu sein. Viele andere der Funktionen, die nicht in der Liste aufgeführt sind, sind traditionell so implementiert, daß sie die C-Funktionen *malloc()* oder *free()* aus der Standard-I/O-Bibliothek verwenden; beide verwenden Datenstrukturen traditionell in einer nicht-reentranten Weise. Weil jede Kombination verschiedener Funktionen, die eine gemeinsame Datenstruktur verwenden, Probleme bei der reentranten Verwendung verursachen können, definiert dieses Handbuch nicht das Verhalten, falls irgeneine unsichere Funktion aus einer Signalbehandlungs-Funktion heraus aufgerufen wird, die eine unsichere Funktion unterbricht.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigaction()* den Wert 0. Andernfalls wird der Wert -1 geliefert, die Variable *errno* gesetzt, um den Fehler anzuzeigen und keine neue Signalbehandlungs-Funktion installiert.

FEHLER

Die Funktion *sigaction()* schlägt fehl, wenn gilt:

[EINVAL] Das Argument *sig* ist keine gültige Signalnummer oder es wird versucht, eine Aktion für ein Signal einzuführen, das nicht abgefangen oder ignoriert werden kann.

HINWEIS

Die Funktion *sigaction()* löst die Funktion *signal()* ab, und sollte vorzugsweise verwendet werden. Insbesondere sollten *sigaction()* und *signal()* nicht im selben Prozeß verwendet werden um dasselbe Signal zu kontrollieren.

PORTABILITÄT

Die Funktion *sigaction()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Nicht alle X/Open-kompatible Implementierungen unterstützen das Signal SIGCHLD.

SIEHE AUCH

kill(), *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*, *sigismember()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

sigaddset - add a signal to a signal set
Signal zu einer Menge von Signalen hinzufügen

DEFINITION

```
#include <signal.h>

int sigaddset (set, signo)
sigset_t *set;
int signo;
```

BESCHREIBUNG

Die Funktion *sigaddset()* fügt ein einzelnes Signal, das durch *signo* gegeben ist, zu der Signalmenge hinzu, auf die *set* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigaddset()* den Wert 0. Andernfalls liefert sie den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *sigaddset()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert von *signo* ist eine ungültige oder nicht unterstützte Signalnummer.

HINWEIS

Anwendungen sollten vor jeder anderen Verwendung eines Objekts vom Typ *sigset_t* entweder die Funktion *sigemptyset()* oder die Funktion *sigfillset()* wenigstens einmal für dieses Objekt aufrufen. Wenn solch ein Objekt nicht auf diese Weise initialisiert wird, dennoch aber als Argument für eine der Funktionen *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*, *sigpending()* oder *sigprocmask()* verwendet wird, so sind die Ergebnisse undefiniert.

PORTABILITÄT

Die Funktion *sigaddset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

sigdelset - delete a signal from a signal set
Signal aus Signalmenge löschen

DEFINITION

```
#include <signal.h>

int sigdelset (set, signo)
sigset_t *set;
int signo;
```

BESCHREIBUNG

Die Funktion *sigdelset()* löscht ein einzelnes Signal, das durch *signo* angegeben wird, aus der Signalmenge, auf die *set* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigdelset()* den Wert 0. Andernfalls liefert die Funktion *sigdelset()* den Wert -1 und besetzt die Variable *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *sigdelset()* schlägt fehl, wenn gilt:

[EINVAL] Das Argument *signo* ist keine gültige oder eine nicht unterstützte Signalnummer.

HINWEIS

Anwendungen sollten vor jeder anderen Verwendung eines Objekts vom Typ *sigset_t* entweder die Funktion *sigemptyset()* oder die Funktion *sigfillset()* wenigstens einmal für dieses Objekt aufrufen. Wenn solch ein Objekt nicht auf diese Weise initialisiert wird, dennoch aber als Argument für eine der Funktionen *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*, *sigpending()* oder *sigprocmask()* verwendet wird, so sind die Ergebnisse undefiniert.

PORTABILITÄT

Die Funktion *sigdelset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), *sigaddset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

sigemptyset - initialise and empty a signal set
Signalmenge initialisieren und leer setzen

DEFINITION

```
#include <signal.h>

int sigemptyset (set)
sigset_t *set;
```

BESCHREIBUNG

Die Funktion *sigemptyset()* initialisiert die Signalmenge, auf die *set* zeigt, so daß keines der in diesem Handbuch definierten Signale enthalten ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigemptyset()* den Wert 0. Andernfalls liefert die Funktion *sigemptyset()* den Wert -1.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Anwendungen sollten vor jeder anderen Verwendung eines Objekts des Typs *sigset_t* eine der Funktionen *sigemptyset*() oder *sigfillset*() wenigstens einmal aufrufen.

PORTABILITÄT

Die Funktion *sigemptyset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

sigfillset - initialise and fill a signal set
Signalmenge initialisieren und füllen

DEFINITION

```
#include <signal.h>

int sigfillset (set)
sigset_t *set;
```

BESCHREIBUNG

Die Funktion *sigfillset()* initialisiert die Signalmenge, auf die *set* zeigt, so daß alle in diesem Handbuch definierten Signale enthalten sind.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigfillset()* den Wert 0. Andernfalls liefert die Funktion *sigfillset()* den Wert -1.

FEHLER

Es sind keine Fehler definiert

HINWEIS

Anwendungen sollten vor jeder anderen Verwendung eines Objekts des Typs *sigset_t* wenigstens einmal *sigemptyset*() oder *sigfillset*() aufrufen.

PORTABILITÄT

Die Funktion *sigfillset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

sigismember - test for a signal in a signal set
Signalmenge auf Signal prüfen

DEFINITION

```
#include <signal.h>

int sigismember (set, signo)
sigset_t *set;
int signo;
```

BESCHREIBUNG

Die Funktion *sigismember()* prüft, ob das durch *signo* angegebene Signal in der Menge enthalten ist, auf die *set* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigismember()* den Wert 1, falls das angegebene Signal in der angegebenen Menge enthalten ist, bzw. den Wert 0, wenn das Signal nicht enthalten ist. Andernfalls liefert die Funktion *sigismember()* den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *sigismember()* schlägt fehl, wenn gilt:

[EINVAL] Das Argument *signo* ist eine ungültige oder aber eine nicht unterstützte Signalnummer.

HINWEIS

Anwendungen sollten vor jeder anderen Verwendung eines Objekts vom Typ *sigset_t* entweder die Funktion *sigemptyset()* oder die Funktion *sigfillset()* wenigstens einmal für dieses Objekt aufrufen. Wenn solch ein Objekt nicht auf diese Weise initialisiert wird, dennoch aber als Argument für eine der Funktionen *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*, *sigpending()* oder *sigprocmask()* verwendet wird, so sind die Ergebnisse undefiniert.

PORTABILITÄT

Die Funktion *sigismember()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *<signal.h>*.

NAME

siglongjmp - non-local goto with signal handling
nichtlokaler Sprung mit Signalbehandlung

DEFINITION

```
#include <setjmp.h>

void siglongjmp (env, val)
sigjmp_buf env;
int val;
```

BESCHREIBUNG

Die Funktion *siglongjmp()* stellt die vom im selben Prozeß zuletzt mit demselben Argument *sigjmp_buf* ausgeführten Makro *sigsetjmp()* gesicherte Umgebung wieder her. Falls es keine solche Ausführung des Makros *sigsetjmp()* gibt, oder die Funktion, in der dieses Makro aufgerufen wurde in der Zwischenzeit beendet wurde, so ist das Verhalten undefiniert.

Alle zugreifbaren Objekte besitzen dieselben Werte wie zu dem Zeitpunkt, als *siglongjmp()* aufgerufen wurde, mit der Ausnahme, daß die Werte von automatischen Objekten, die zwischen der Ausführung von *sigsetjmp()* und dem Aufruf von *siglongjmp()* geändert wurden, unbestimmt sind.

Da die Funktion *siglongjmp()* den normalen Funktionsaufruf- und -Rückkehrmechanismus verwendet, läüft sie auch im Zusammenhang mit Unterbrechungen, Signalen und den damit zusammenhängenden Funktionen korrekt ab. Trotzdem gilt, daß das Verhalten dieser Funktion undefiniert ist, wenn die Funktion *siglongjmp()* von einer verschachtelten Signalbehandlungs-Funktion aus aufgerufen wird (d.h. von einer Funktion die aufgrund eines Signals aus einer anderen Signalbehandlungs-Funktion heraus aufgerufen wurde).

Die Funktion *siglongjmp()* stellt die gesicherte Signalmaske wieder her, wenn und nur wenn das Argument *env* durch einen Aufruf von *sigsetjmp()* mit einem Argument *savemask* ungleich 0 initialisiert wurde.

ERGEBNIS

Nachdem *siglongjmp()* beendet ist, setzt die Ausführung des Programms so fort, als ob die zugehörige Ausführung des Makros *sigsetjmp()* soeben mit dem durch *val* angegebenen Wert beendet worden wäre. Die Funktion *siglongjmp()* kann das Makro *sigsetjmp()* nicht veranlassen, den Wert 0 zurückzugeben; wenn *val* gleich 0 ist, dann liefert das entsprechende Makro *sigsetjmp()* den Wert 1.

HINWEIS

Wenn *siglongjmp()* aufgerufen wird, obwohl *env* niemals durch einen Aufruf von *sigsetjmp()* besetzt wurde, oder wenn der letzte solche Aufruf innerhalb einer Funktion erfolgt ist, die seitdem beendet wurde, so ist das Verhalten undefiniert.

Wenn der Aufruf von *siglongjmp()* in einer anderen Funktion erfolgt, als der zugehörige Aufruf von *sigsetjmp()*, können alle Variablen der Speicherklasse *register* unvorhersehbare Werte haben.

Der Unterschied zwischen *setjmp()* oder *longjmp()* und *sigsetjmp()* oder *siglongjmp()* ist nur für solche Programme von Bedeutung, die die Funktionen *sigaction()*, *sigprocmask()* oder *sigsuspend()* verwenden.

PORTABILITÄT

Die Funktion *siglongjmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

longjmp(), *setjmp()*, *sigprocmask()*, *sigsetjmp()*, *sigsuspend()*, *<setjmp.h>*.

NAME

signal - specify what to do upon receipt of a signal
Signalbehandlung definieren

DEFINITION

```
#include <signal.h>

void (*signal (sig, func))()
int sig;
void (*func)();
```

BESCHREIBUNG

Mit *signal()* kann angegeben werden, wie der aufrufende Prozeß auf den Empfang eines bestimmten Signals reagieren soll. Es stehen drei Reaktionen zur Auswahl: Prozeßabbruch, Ignorieren des Signals oder Signalbehandlung durch eine Funktion. Mit *sig* wird das Signal angegeben, auf das reagiert werden soll und mit *func* die gewünschte Reaktion.

Signale, die ein Prozeß empfängt, können auf verschiedene Arten erzeugt werden:

von außen:	— durch den Benutzer an der Datensichtstation, der eine Unterbrechungstaste (DEL, QUIT,...) drückt.
	— durch einen anderen Prozeß, der ein Signal schickt (Kommando *kill*, *kill()*).
von innen:	— durch den Prozeß selbst (*alarm()*).
	— durch Programmfehler (Adreßfehler, Anstoß eines ungültigen Befehls, Division durch Null,...).

Für das Argument *sig* kann jeder der folgenden Werte mit Ausnahme von SIGKILL angegeben werden:

SIGHUP	Verbindung zur Datensichtstation unterbrochen
SIGINT	unterbrechen mit Interrupt-Taste
SIGQUIT *)	abbrechen mit Quit-Taste
SIGILL *)	unzulässiger Befehl (wird nach dem Abfangen nicht zurückgesetzt, s. HINWEIS)
SIGTRAP *)	Unterbrechung bei ptrace (wird nach dem Abfangen nicht zurückgesetzt, s. HINWEIS)
SIGFPE *)	Fehler bei Gleitkommaoperation

SIGKILL	unbedingter Prozeßabbruch (kann weder abgefangen noch ignoriert werden)
SIGSEGV *)	unerlaubter Segmentzugriff
SIGSYS *)	ungültiges Argument für einen Systemaufruf oder ungültiger Systemaufruf
SIGPIPE	Ausgabe auf eine Pipe, deren Leseseite geschlossen ist
SIGALRM	Alarmuhr abgelaufen
SIGTERM	Prozeßbeendigung durch *kill*
SIGCHLD	Beenden eines Sohnprozesses. Defaultbehandlung ist Ignorieren des Signals (SIG_DFL)
SIGUSR1	vom Benutzer verwendete Signale
SIGUSR2	vom Benutzer verwendete Signale

*) Standardmäßig wird auf diese Signale mit Prozeßabbruch reagiert. Siehe unten, SIG_DFL.

Aus Portabilitätsgründen sollten in Anwendungen nur die oben aufgeführten Signale verwendet werden.

Hat ein Prozeß nichts in Bezug auf ein Signal vereinbart, so wird bei Eintreffen des Signals der Prozeß abgebrochen, und der Vater des Prozesses erfährt vom Prozeßende.
Durch den Wert von *func* kann die Art der Reaktion auf das eintreffende Signal vereinbart werden. Es gibt drei Möglichkeiten auf ein Signal zu reagieren, dazu kann *func* einen der folgenden Werte annehmen: SIG_DFL, SIG_IGN oder eine Funktionsadresse. Die Werte von *funk* haben folgende Konsequenzen:

SIG_DFL
Bei Empfang von *sig* wird der empfangende Prozeß mit allen unter *exit()* aufgeführten Konsequenzen beendet. Ausnahme: Signal SIGCHLD (s.o.).
Ist *sig* eines der oben mit *) markierten Signale, wird u.U. zusätzlich ein Speicherabzug erzeugt.

SIG_IGN
Das Signal *sig* soll ignoriert werden.

Achtung
Das Signal SIGKILL kann nicht ignoriert werden.

Funktionsadresse
Bei Empfang von *sig* soll der empfangende Prozeß die Signalbehandlungsfunktion ausführen, auf die mit *func* verwiesen wird. Als einziges Argument wird der Funktion die Signalnummer *sig* übergeben. Für von der Hardware

ausgehende Signale können der Funktion weitere Argumente übergeben werden. Nach Beendigung der Signalbehandlung wird der Prozeß an der Stelle fortgesetzt, an der er unterbrochen wurde, wenn nicht in *func exit()* aufgerufen wurde.
Das Signal SIGKILL kann nicht abgefangen werden.

Vor Eintritt in die Funktion wird *func* außer bei SIGILL oder SIGTRAP wieder auf SIG_DFL gesetzt.

signal() kann nicht erkennen, ob ihr Argument *func* eine gültige Adresse enthält; der Versuch, die Programmausführung an einer ungültigen Adresse fortzusetzen, liefert undefinierte Ergebnisse.

Nach Rücksprung aus der Signalbehandlungsfunktion wird der Prozeß an der Stelle fortgesetzt, an der er unterbrochen wurde.

Tritt ein abzufangendes Signal während eines der Systemaufrufe *read()*, *write()*, *open()* oder *ioctl()* an einem langsamen Gerät (z.B. einer Datenstation, nicht jedoch einer Datei), während eines *pause()*-Aufrufs oder eines *wait()*-Aufrufs ohne sofortigen Rücksprung auf, so wird die Signalbehandlungsfunktion ausgeführt; der unterbrochene Systemaufruf kann dann −1 an den aufrufenden Prozeß liefern und *errno* auf [EINTR] setzen.

Ein *signal()*-Aufruf setzt ein anstehendes *sig* zurück; dies gilt nicht für das SIGKILL- und das SIGTRAP-Signal.

ERGEBNIS

Bei erfolgreicher Ausführung liefert *signal()* den vorherigen Wert von *func* für das angegebene Signal *sig* zurück. In allen anderen Fällen wird der Wert (int (*)())−1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *signal()* schlägt fehl, wenn gilt:

[EINVAL]
signal() kann nicht ausgeführt werden, wenn *sig* eine ungültige Signalnummer oder SIGKILL ist.

HINWEIS

Die Funktion *signal()* wurde im X/Open-Standard (Ausgabe 3) durch *sigaction()* ersetzt. Die Funktion *signal()* sollte in neuen Anwendungen nicht mehr benutzt werden; statt dessen sollte *sigaction()* verwendet werde. Die Funktionen *sigaction()* und *signal()* sollten nicht im selben Prozeß verwendet werden, um dasselbe Signal zu kontrollieren.

Aus Portabilitätsgründen sollten in Anwendungen statt der Signalwerte nur ihre symbolischen Namen und nur die hier definierten Signale verwendet werden. Dabei ist es durchaus möglich, daß bestimmte Implementierungen weitere Signale verwenden.

Das Signal SIGSEGV wurde nur aus Gründen der Kompatibilität mit vorhandenen Anwendungen mit aufgeführt. In Programmen, die portabel sein sollen, sollte es nicht verwendet werden.

Werden Signale abgefangen, so wird *errno* u.U. durch Fehler, die in der Signalbehandlungsfunktion auftreten, verändert. Sein Wert ist daher unzuverlässig, wenn die Möglichkeit besteht, daß zwischen dem Setzen von *errno* und der Verwendung des Wertes ein Signal eintrifft.

Außer bei SIGILL und SIGTRAP wird bei allen Signalen nach einmaliger Ausführung von *func* die Signalbehandlung wieder auf SIG_DFL gesetzt.

Die von einer Datensichtstation erzeugten Signale werden an alle Prozesse weitergeleitet, die zu dieser Terminalprozeßgruppe gehören (es können auch Prozesse von diesem Terminal gestartet sein, die eigene Prozeßgruppen bilden, s. *setgrp()*).

Nach einem *fork()*-Aufruf übernimmt der Sohnprozeß die Signalbehandlung des Vaterprozesses.

Nach einem *exec*-Aufruf ist jede Signalbehandlung wieder auf die Voreinstellung zurückgesetzt.

In Signalbehandlungsfunktionen sollte die Funktion *sleep()* nicht verwendet werden, weil es dadurch eventuell zu rekursiven *sleep()* Aufrufen kommen könnte. Das Verhalten bei rekursivem Aufruf von *sleep()* ist aber undefiniert.

PORTABILITÄT

Die Funktion *signal()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Programm, das die Signale SIGINT (Drücken der Taste DEL) und SIGALRM abfängt:

```
#define PEEP 07
#include <signal.h>
#include <stdio.h>

/* Die Signalbehandlungsfunktion wird beim Druecken der Taste [DEL]   */
/*  (SIGINT) bzw. automatisch nach Ablauf einer Zeitspanne von einer */
/*  Sekunde (SIGALRM) ausgefuehrt.                                   */

catch(sig)
   /* Signalbehandlungsfunktion */
   int sig;
{
   putchar(PEEP);
      /* Piepst, wenn Signal eintrifft */

   if (sig == SIGALRM)
      /*  Signalbehandlung wird fuer SIGARLARM        */
      /*   neu aufgesetzt. Nochmaliges Druecken der   */
      /*   Taste [DEL] fuehrt dagegen zum             */
      /*   Programmabbruch                            */
      signal(SIGALRM,catch);
   alarm(1);
      /* Alarmuhr neu einstellen                      */
      /* Einstellen der Uhr erst nach dem Aufsetzen   */
      /*  der Signalbehandlung                        */
}

main()
{
   FILE *fp;
   int c, catch();

   signal(SIGINT,catch);
      /* Signalbehandlung einrichten */
   signal(SIGALRM,catch);

   alarm(1);
      /* Zeit einstellen, nach einer Sekunde */
      /*  Verzweigung zur Signalbehandlung   */

   fp = fopen("datei","r");
   while((c=getc(fp)) != EOF)
      putc(c,stdout);
   fclose(fp);
}
```

SIEHE AUCH

pause(), *sigaction()*, *<signal.h>*.

NAME

signgam - storage for sign of *gamma()*
Speicherplatz für das Vorzeichen von gamma()

DEFINITION

```
extern int signgam;
```

BESCHREIBUNG

Siehe unter *gamma()*.

PORTABILITÄT

Die Variable *signgam* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

sigpending - examine pending signals
anstehende Signale prüfen

DEFINITION

```
#include <signal.h>

int sigpending (set)
sigset_t *set;
```

BESCHREIBUNG

Die Funktion *sigpending()* speichert die Menge der Signale, deren Zustellung blockiert ist und die für den aufrufenden Prozess anstehen, in dem Objekt ab, auf das *set* zeigt

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigpending*() den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *sigpending()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaddset(), *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigprocmask()*, *<signal.h>*.

NAME

sigprocmask - examine and change blocked signals
Signalmaske prüfen und ändern

DEFINITION

```
#include <signal.h>

int sigprocmask (how, set, oset)
int how;
sigset_t *set, *oset;
```

BESCHREIBUNG

Die Funktion *sigprocmask()* erlaubt dem aufrufenden Prozeß, seine Signalmaske zu überprüfen und/oder zu ändern.

Wenn das Argument *set* ungleich dem Nullzeiger ist, dann zeigt es auf eine Signalmenge die verwendet wird, um die augenblicklich blockierte Menge zu ändern.

Das Argument *how* gibt an, auf welche Weise die Menge geändert werden soll. Es kann einen der folgenden Werte annehmen (siehe auch *<signal.h>*):

SIG_BLOCK Die Ergebnismenge besteht aus der Vereinigung der aktuellen und der durch *set* angegebenen Menge.

SIG_UNBLOCK Die Ergebnismenge besteht aus der Schnittmenge der aktuellen und des Komplements der durch *set* angegebenen Menge.

SIG_SETMASK Die Ergebnismenge entspricht der durch *set* angegebenen Menge.

Wenn das Argument *oset* kein Nullzeiger ist, dann wird die alte Maske in dem Bereich abgespeichert, auf den *oset* zeigt. Wenn *set* ein Nullzeiger ist, dann spielt der Wert des Arguments *how* keine Rolle und die Signalmaske des Prozesses bleibt unverändert; daher kann der Aufruf verwendet werden, um die derzeit blockierten Signale abzufragen.

Falls es irgendwelche anstehenden, nichtblockierten Signale nach einem Aufruf von *sigprocmask()* gibt, so wird wenigstens eins dieser Signale zugestellt, bevor der Aufruf von *sigprocmask()* zurückkehrt.

Es ist nicht möglich, diejenigen Signale zu blockieren, die nicht ignoriert werden können. Dies wird durch das System sichergestellt, ohne daß ein Fehler angezeigt wird.

Wenn eines der Signale SIGFPE, SIGILL oder SIGSEGV generiert wird, während es blockiert ist, so ist das Ergebnis undefiniert, außer das Signal wurde durch einen Aufruf der Funktion *kill()* generiert.

Wenn die Funktion *sigprocmask()* fehlschlägt, so wird die Signalmaske des Prozesses nicht geändert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sigprocmask*() den Wert 0. Anderfalls wird der Wert -1 zurückgeliefert, *errno* gesetzt, um den Fehler anzuzeigen und die Signalmaske des Prozesses nicht geändert.

FEHLER

Die Funktion *sigprocmask()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert des Arguments *how* ist ungleich einem der definierten Werte.

PORTABILITÄT

Die Funktion *sigprocmask()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), sigpending(), sigsuspend(), <signal.h>.

NAME

sigsetjmp - set jump point for a non-local goto
Sprungmarke für nichtlokalen Sprung setzen

DEFINITION

```
#include <setjmp.h>

int sigsetjmp (env, savemask)
sigjmp_buf env;
int savemask;
```

BESCHREIBUNG

Das Makro *sigsetjmp()* sichert seine Aufrufumgebung in sein Argument *env* für eine spätere Benutzung durch die Funktion *siglongjmp()*.

Wenn der Wert des Arguments *savemask* ungleich 0 ist, dann sichert das Makro auch die aktuelle Signalmaske des Prozesses als einen Teil der Aufruf-Umgebung.

ERGEBNIS

Wenn die Rückkehr von einer unmittelbaren Ausführung des Makros *sigsetjmp()* erfolgt, so liefert das Makro den Wert 0. Wenn die Rückkehr von einem Aufruf der Funktion *siglongjmp()* erfolgt, so liefert das Makro *sigsetjmp()* einen Wert ungleich 0 zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Wenn der Aufruf von *siglongjmp()* in einer anderen Funktion erfolgt, als der zugehörige Aufruf von *sigsetjmp()*, können alle Variablen der Speicherklasse *register* unvorhersehbare Werte haben.

Das Makro *sigsetjmp()* ist nützlich, um Fehler und Unterbrechungen in einer Low-Level-Unterfunktion eines Programms zu behandeln.

Der Unterschied zwischen *setjmp()* oder *longjmp()* und *sigsetjmp()* oder *siglongjmp()* ist nur für solche Programme von Bedeutung, die die Funktionen *sigaction()*, *sigprocmask()* oder *sigsuspend()* verwenden.

PORTABILITÄT

Die Funktion *sigsetjmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

siglongjmp(), *signal()*, *sigprocmask()*, *sigsuspend()*, *<setjmp.h>*.

NAME

sigsuspend - wait for a signal
auf ein Signal warten

DEFINITION

```
#include <signal.h>

int sigsuspend (sigmask)
sigset_t *sigmask;
```

BESCHREIBUNG

Die Funktion *sigsuspend()* ersetzt die derzeitige Signalmaske des Prozesses durch die Signalmenge, auf die *sigmask* zeigt, und suspendiert dann den Prozeß bis ein Signal zugestellt wird, dessen Aktion entweder die Ausführung einer Signalbehandlungs-Funktion oder der Abbruch des Prozesses ist.

Wenn die Aktion der Abbruch des Prozesses ist, dann kehrt die Funktion *sigsuspend*() niemals zurück. Wenn die Aktion die Ausführung einer Signalbehandlungs-Funktion ist, dann kehrt die Funktion nach Ende dieser Signalbehandlungs-Funktion zurück, wobei die Signalmaske so wiederhergestellt wird, wie sie vor dem Aufruf von *sigsuspend()* eingestellt war.

Es ist nicht möglich, Signale zu blockieren, die nicht ignoriert werden können (siehe <*signal.h*>). Dies wird durch das System sichergestellt, ohne daß ein Fehler angezeigt wird.

ERGEBNIS

Da *sigsuspend()* die Ausführung des Prozesses für unbestimmte Zeit unterbricht, gibt es für die erfolgreiche Beendigung keinen Ergebniswert. Wenn ein Fehler eintritt, dann wird ein Wert von -1 zurückgeliefert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *sigsuspend()* schlägt fehl, wenn gilt:

[EINTR] Ein Signal wurde vom aufrufenden Prozeß abgefangen und die Kontrolle wird von der Signalbehandlungs-Funktion zurückgegeben.

PORTABILITÄT

Die Funktion *sigsuspend()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

pause(), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *<signal.h>*.

NAME

sin - sine function
Sinus

DEFINITION

```
#include <math.h>

double sin (x)
double x;
```

BESCHREIBUNG

Die Funktion *sin()* berechnet den Sinus ihres Arguments *x*, das im Bogenmaß angegeben ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sin()* den Sinus von *x*.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Wenn *x* gleich +HUGE_VAL oder gleich -HUGE_VAL ist, dann wird der Wert 0.0 zurückgeliefert und *errno* ist gleich *ERANGE* gesetzt.

Andernfalls wird entweder *errno* gesetzt, um den Fehler anzuzeigen oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktion *sin()* schlägt fehl, wenn gilt:

[ERANGE] Der Wert von *x* ist gleich +HUGE_VAL oder gleich -HUGE_VAL.

[ERANGE] Die Größe von *x* ist derart, daß ein völliger oder teilweiser Verlust an Genauigkeit die Folge wäre.

Die Funktion *sin()* kann unter anderen X/OPen-kompatiblen Systemen auch fehlschlagen, wenn gilt:

[EDOM] Der Wert von *x* ist ein NaN, oder *x* ist +HUGE_VAL oder -HUGE_VAL.

HINWEIS

Die Funktion *sin()* kann ihre Genauigkeit einbüßen, wenn ihr Wert sehr von 0 verschieden ist.

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *sin()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, dann müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *sin()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die Sinuswerte im Bereich von −1.0 bis +1.0 ausgeben mit einer Schrittweite von 0.1:

```
#include <math.h>

main()
{
   double x;

   for (x = -1.0; x < 1.1; x = x+0.1)
      printf ("sin(%1.2f) = %.4f \n", x, sin(x));
}
```

SIEHE AUCH

asin(), *isnan()*, *matherr()*, *<math.h>*.

NAME

sinh - hyperbolic sine function
Sinus Hyperbolicus

DEFINITION

```
#include <math.h>

double sinh (x)
double x;
```

BESCHREIBUNG

Die Funktion *sinh()* berechnet den Sinus Hyperbolicus von *x*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sinh()* den Sinus Hyperbolicus von *x*.

Wenn das Ergebnis einen Überlauf verursachen würde, dann wird der Wert HUGE_VAL zurückgeliefert und *errno* ist gleich [ERANGE] gesetzt.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Anderfalls wird -HUGE_VAL, +HUGE_VAL oder ein NaN geliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktion *sinh()* schlägt fehl, wenn gilt:

[ERANGE] Das Ergebnis würde einen Überlauf verursachen oder *x* ist gleich +HUGE_VAL oder -HUGE_VAL.

Die Funktion *sinh()* kann fehlschlagen, wenn gilt:

[EDOM] Der Wert von *x* ist ein NaN.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *sinh()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, dann müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *sinh()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die Sinus-Hyperbolicus Werte aus dem Bereich von −1.0 bis 1.0 ausgeben. Schrittweite 0.1:

```
#include <math.h>

main()
{
   double x;

   for (x = -1.0; x < 1.1; x = x+0.1)
      printf ("sinh(%1.2f) = %.4f \n", x, sinh(x));
}
```

SIEHE AUCH

cosh(), *isnan()*, *matherr()*, *tanh()*, *<math.h>*.

NAME

sleep - suspend execution for an interval of time
Prozeß für festgesetzte Zeitspanne anhalten

DEFINITION

```
unsigned int sleep (seconds)
unsigned int seconds;
```

BESCHREIBUNG

Die Funktion *sleep()* sorgt dafür, daß der aktuelle Prozeß angehalten wird, bis entweder die durch das Argument *seconds* angegebene Zeit von Echtzeit-Sekunden vergangen sind, oder bis ein Signal an den aufrufenden Prozeß zugestellt wird, dessen Aktion eine Signalbehandlungsfunktion oder das Beenden des Prozesses ist. Die Anhaltezeit kann aufgrund der Verteilung anderer Aktivitäten des Systems länger als die angegebene Zeit sein.

Wenn während der Ausführung von *sleep()* ein Signal des Typs SIGALRM für den aufrufenden Prozeß generiert wird, und wenn das SIGALRM-Signal ignoriert oder blockiert wird, dann ist es unbestimmt, ob *sleep()* zurückkehrt, wenn das Signal bearbeitet wird. Wenn das Signal blockiert ist, dann ist es ebenfalls unbestimmt, ob es auch noch nach der Rückkehr der Funktion *sleep()* ansteht, oder ob es verworfen wird.

Wenn während der Ausführung von *sleep()* ein Signal des Typs SIGALRM für den aufrufenden Prozeß generiert wird, wenn dieses Signal nicht das Ergebnis eines vorhergehenden Aufrufs der Funktion *alarm()* ist und wenn das Signal SIGALRM nicht blockiert oder ignoriert wird, dann ist es unbestimmt, ob das Signal noch eine andere Wirkung hat, als die Funktion *sleep()* dazu zu bringen zurückzukehren.

Wenn eine Signalbehandlungsfunktion die Funktion *sleep()* unterbricht und entweder die Zeit, zu der ein Signal SIGALRM erzeugt werden soll, überprüft oder verändert oder die dem Signal SIGALRM zugeordnete Aktion verändert oder verändert, ob das Signal SIGALRM von der Zustellung blockiert werden soll, dann sind die Ergebnisse unbestimmt.

Wenn eine Signalbehandlungsfunktion die Funktion *sleep()* unterbricht und eine der Funktionen *siglongjmp()* oder *longjmp()* aufruft, um eine Umgebung wiederherzustellen, die vor dem Aufruf von *sleep()* gesichert wurde, dann sind sowohl die dem Signal SIGALRM zugeordnete Aktion als auch die Zeit, zu der dieses Signal ausgelöst werden soll, unbestimmt. Es ist ebenso

unbestimmt, ob das Signal SIGALRM blockiert wird, wenn nicht die Signalmaske des Prozesses als Teil der Umgebung auch wiederhergestellt wird. (siehe auch *sigsetjmp()*).

ERGEBNIS

Wenn die Funktion *sleep()* zurückkehrt, weil die eingestellte Zeit abgelaufen ist, dann wird der Wert 0 zurückgeliefert. Wenn die Funktion *sleep()* zurückkehrt, weil sie durch die Zustellung eines Signals vorzeitig beendet wurde, dann ist das Ergebnis die Zeit, die die Funktion eigentlich noch hätte schlafend verbringen müssen (d.h. die eingestellte Zeit minus der schlafend verbrachten Zeit) in Sekunden.

FEHLER

Die Funktion *sleep()* ist immer erfolgreich und kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

PORTABILITÄT

Die Funktion *sleep()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Programm simuliert die Funktionsweise von *sleep()* mit Hilfe von *pause()*:

```
#include <stdio.h>
#include <signal.h>

                /* Funktion zum Signal-Abfang */
abfang(sig)
   int sig;
{
   printf("Signal wurde abgefangen\n");
}


main()
{
   unsigned sek;
   int      (*speich)();

   printf("Prozess begonnen\n");

                /* ---------Simulationsanfang----------
                   Zunaechst wird die vorherige Signalbe-
                   handlung gespeichert */
   speich =signal(SIGALRM,abfang);
   if((sek=alarm(15)) >15)
      alarm(sek);
                /* Prozess auf Eis bis SIGALRM eintrifft */
   pause();
                /* Alte Signalbehandlung wieder einstellen */
   signal(SIGALRM,speich);
   if (sek>15)
      alarm(sek -15);
   else if(sek >0)
      alarm(1);
                /* ---------Simulationsende---------- */
   printf("Prozess beendet\n");
   sleep(15);
   printf("Nochmals 15 Sekunden gewartet!\n");
}
```

SIEHE AUCH

alarm(), *pause()*, *sigaction()*.

NAME

sprintf - print formatted output
Formatierte Ausgabe in Zeichenkette

DEFINITION

```
#include <stdio.h>

int sprintf (s, format, ... )
char *s, *format;
```

BESCHREIBUNG

Siehe unter *printf()*.

PORTABILITÄT

Die Funktion *sprintf()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Sie können *sprintf* z.B. dazu benutzen eine Zeichenkette zu kopieren. Damit kann man die Funktion *strncpy()* implementieren. Das Beispiel bei *strncpy()* sähe dann so aus:

```
#include <stdio.h>

main()
{
   int n;
   char *s2 = " Peter ...!";
   char s1[BUFSIZ];

   printf("der satz heisst: %s \nWieviele Zeichen kopieren ?", s2);
   scanf("%d",&n);

      /* Alternativ koennte an dieser Stelle */
      /*  folgender Aufruf stehen:           */
      /*  strncpy (s1, s2, n)                */
   sprintf(s1,"%.*s",n,s2);
   printf ("%s \n", s1);
}
```

NAME

sqrt - square root function
Wurzelfunktion

DEFINITION

```
#include <math.h>

double sqrt (x)
double x;
```

BESCHREIBUNG

Die Funktion *sqrt()* berechnet die Quadratwurzel von *x*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *sqrt()* die Quadratwurzel von *x*, #Wua*x*#Wue.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Andernfalls wird entweder 0 zurückgeliefert und *errno* wird besetzt, um den Fehler anzuzeigen, oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktion *sqrt()* schlägt fehl, wenn gilt:

[EDOM] Der Wert von *x* ist negativ.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *sqrt()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, dann müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *sqrt()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Berechne die Quadratwurzel für einen eingelesenen Wert *x*:

```
#include  <math.h>

int errno;

main()
{
   double x;

   scanf("%lf",&x);
   printf("Wurzel aus %g: %g \n",x,sqrt(x));
   printf("errno = %d\n",errno);
}
```

SIEHE AUCH

isnan(), *matherr()*, *<math.h>*.

NAME

srand - seed simple pseudo-random number generator
Pseudo-Zufallszahlengenerator initialisieren

DEFINITION

```
#include <stdlib.h>

void srand (seed)
unsigned seed;
```

BESCHREIBUNG

Die Funktion *srand()* verwendet ihr Argument als Startwert für eine neue Folge von Pseudo-Zufallszahlen, die von aufeinanderfolgenden Aufrufen von *rand()* geliefert werden. Wenn *srand()* danach mit demselben Startwert aufgerufen wird, dann wird die Folge von Pseudo-Zufallszahlen wiederholt. Wenn *rand()* aufgerufen wird, bevor ein Aufruf von *srand()* erfolgt ist, so wird dieselbe Folge generiert, als ob *srand()* zuerst mit dem Startwert 1 aufgerufen worden wäre.

Die Implementierung verhält sich so, als ob keine andere in diesem Handbuch definierte Funktion die Funktion *srand()* aufrufen würde.

ERGEBNIS

Die Funktion *srand()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *srand()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel unter *rand()*.

SIEHE AUCH

drand48(), *rand()*, *<stdlib.h>*.

NAME

srand48 - seed uniformly distributed double-precision pseudo-random number generator
Generator für gleichmäßig verteilte Pseudo-Zufallszahlen des Typs double initialisieren

DEFINITION

```
void srand48 (seedval)
long seedval;
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *srand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

sscanf - convert formatted input
Formatierte Eingabe aus Zeichenkette

DEFINITION

```
#include <stdio.h>

int sscanf (s, format, ...)
char *s, *format;
```

BESCHREIBUNG

Siehe unter *scanf()*.

PORTABILITÄT

Die Funktion *sscanf()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Die Daten aus der Zeichenkette *name* werden wie folgt zugewiesen:

```
i    : 25
x    : 341.0
p    : nasowa\0
```

Eine Zuweisung von 234 erfolgt wegen der %*d Angabe nicht.

```
#include <stdio.h>

main()
{
   int i;
   float x;
   char p[10];
   char *name = "25341     234   nasowas";

   sscanf(name,"%2d %f %*d %6s",&i,&x,p);
   printf("%2d %f %6s",i,x,p);
}
```

NAME

ssignal, gsignal - software signals
Software-Signale

DEFINITION

```
#include <signal.h>

int (*ssignal (sig, funk))()
int sig, (*funk)();

int gsignal (sig)
int sig;
```

BESCHREIBUNG

Mit *ssignal()* und *gsignal()* werden dem Benutzer softwaremäßige Funktionen analog zu *signal()* und *kill()* angeboten.

Diese Software-Signale sind in *signal()* aufgeführt. Mit *ssignal()* wird dem Software-Signal *sig* die Prozedur *funk* zugeordnet; *sig* wird durch den Aufruf von *gsignal()* ausgelöst. Das Auslösen eines Software-Signals hat zur Folge, daß die diesem Signal zugeordnete Funktion ausgeführt wird.

Das erste Argument von *ssignal()* ist eines der in *signal()* aufgeführten Signale, dem eine bestimmte Funktion zugeordnet werden soll. Das zweite Argument gibt diese Funktion an; dabei kann entweder der Name einer (benutzereigenen) Funktion oder eine der vorgegebenen Konstanten SIG_DFL (Standardbehandlung) oder SIG_IGN (Ignorieren) angegeben werden. War dem angegebenen Signal bereits eine Funktion zugeordnet, so wird diese von *ssignal()* zurückgeliefert; wenn nicht oder wenn eine ungültige Signalnummer angegeben wurde, so liefert *ssignal()* als Ergebnis SIG_DFL.

gsignal() löst das mit dem Argument *sig* angegebene Signal aus:

Ist *sig* eine Funktion zugeordnet, so wird *funk* auf SIG_DFL zurückgesetzt, bevor mit *sig* in die angegebene Funktion gesprungen wird. *gsignal()* liefert den von der Funktion übergebenen Wert zurück.

Wurde für *funk* SIG_IGN angegeben, so liefert *gsignal()* den Wert 1 zurück und führt keine spezielle Signalbehandlungsfunktion aus.

Wurde für *funk* SIG_DFL angegeben, so liefert *gsignal()* den Wert 0 zurück und führt keine spezielle Signalbehandlungsfunktion aus.

Hat *sig* einen unzulässigen Wert oder wurde *sig* zu keinem Zeitpunkt eine Funktion zugeordnet, so liefert *gsignal()* den Wert 0 zurück und führt keine spezielle Signalbehandlungsfunktion aus.

PORTABILITÄT

Die Funktionen *gsignal()* und *ssignal()* sind im X/Open-Standard nicht enthalten.

SIEHE AUCH

kill(), *sigaction()*, *signal()*.

NAME

stat - get file status
Dateistatus ermitteln

DEFINITION

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

int stat (path, buf)
char *path;
struct stat *buf;
```

BESCHREIBUNG

Das Argument *path* zeigt auf den Pfadnamen einer Datei. Die Lese-, Schreib- oder Ausführungserlaubnis der Datei wird nicht benötigt, aber alle Dateiverzeichnisse, die im Pfadnamen aufgeführt werden, müssen durchsuchbar sein.

Das Argument *buf* wird als Zeiger auf eine Struktur vom Typ *stat* aufgefaßt, so wie diese in der Include-Datei *<sys/stat.h>* definiert ist. In diese Struktur werden die Informationen zur Datei eingetragen.

Die Funktion *stat()* ändert alle zeitbezogenen Felder so, wie im Glossar unter *Dateizeiten-Änderung* beschrieben wird, bevor diese in die Struktur *stat* geschrieben werden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *stat()* schlägt fehl, wenn gilt:

[EACCES] Die Durchsuchberechtigung für eine Komponente des Pfadnamen-Anfangs wird verweigert.

[ENAMETOOLONG]
Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Die genannte Datei existiert nicht oder das Argument *path* zeigt auf eine leere Zeichenkette.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

Unter SINIX V5.22 kann zusätzlich zu den im XPG3 [6] definierten Fehlern auch noch der folgende Fehler auftreten:

[EFAULT] Es wurde eine ungültige Adresse als Argument übergeben.

PORTABILITÄT

Die Funktion *stat()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Eine Implementierung, die erweiterte Sicherheitskontrollen anbietet, kann unter implementierungs-abhängigen Bedingungen für das Fehlschlagen der Funktion *stat()* sorgen. Insbesondere kann das System die Existenz der durch *path* angegebenen Datei verleugnen.

BEISPIEL

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

extern int errno;
extern char *sys_errlist;
char *gtime();

main(argc,argv)
   int argc;
   char **argv;
{
   int mode;
   struct stat infos;  /* Ergebnisstruktur fuer Dateiinformation */

   if (stat(argv[1],&infos) == -1)
   {
      fprintf(stderr,"[error %d]: %s: %s\n", errno,
                     argv[1], sys_errlist[errno]);
      exit(1);
   }
                                    /* Überprüfung des Dateityps */
   mode = infos.st_mode & S_IFMT;
   switch(mode)
   {
      case S_IFDIR:
         printf("Dateiverzeichnis : %s\n",argv[1]);
         break;
      case S_IFREG:
         printf("normale Datei : %s\n",argv[1]);
         break;
      case S_IFFIFO:
         printf("FIFO Datei : %s\n",argv[1]);
         break;
      case S_IFBLK:
         printf("blockorientierte Datei : %s\n",argv[1]);
         break;
      case S_IFCHR:
         printf("normale Datei : %s\n",argv[1]);
         break;
      default:
         printf("\nDateityp nicht feststellbar; Programmabbruch!\n");
         exit(1);
   }

                          /* Aenderung der Schutzbitstellung unter
                             Verwendung der Konstanten S_ISUID
                             und S_IEXEC */
   chmod(argv[1],(S_ISUID | S_IEXEC));

                          /* Detaillierte Information ueber eine
                             Datei ausgeben */
   printf("benutzter Speicherplatz : %ld\n", infos.st_size);
   printf("letzte Aenderung  : %s", ctime(&infos.st_mtime));
}
```

SIEHE AUCH

fstat(), *<sys/stat.h>*, *<sys/types.h>*.

NAME

statfs - get file system information
Informationen zum Dateisystem ermitteln

DEFINITION

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

int statfs(path, buf, len, fstyp)
char  *path;
struct statfs *buf;
int len, fstyp;

int fstatfs(fildes, buf, len, fstyp)
int fildes;
struct statfs *buf;
int len, fstyp;
```

BESCHREIBUNG

Die Funktion *statfs()* liefert einen sogenannten *generischen Superblock*, der ein Dateisystem beschreibt. Diese Funktion kann Informationen für eingehängte wie auch für nicht eingehängte Dateisysteme ermitteln. In beiden Fällen ergeben sich geringe Unterschiede in der Verwendung der Funktion.

In allen Fällen ist *buf* ein Zeiger auf eine Struktur des Typs *struct statfs*, die von der Funktion mit Informationen gefüllt wird. *len* gibt dabei die Anzahl der Bytes an Informationen an, die in der Struktur zurückgeliefert werden sollen. Der Wert von *len* darf dabei nicht größer als *sizeof(struct statfs)* sein.

Die Struktur *statfs*, auf die *buf* zeigt, enthält die folgenden Komponenten:

```
short   f_fstyp;        /* Typ des Dateisystems          */
short   f_bsize;        /* Blockgröße                    */
short   f_frsize;       /* Fragment-Größe                */
long    blocks;         /* Gesamtzahl der Blöcke         */
long    bfree           /* Anzahl der freien Blöcke      */
long    f_files;        /* Gesamtzahl der Inodes         */
long    f_ffree;        /* Anzahl freier Inodes          */
char    f_fname[6];     /* Volume-Name                   */
char    f_fpack[6];     /* Paket-Name                    */
```

Wenn ein eingehängtes Dateisystem von der Funktion *statfs()* untersucht werden soll, dann muß das Argument *path* eine Datei in diesem Dateisystem bezeichnen. Da der Typ des Dateisystems in diesem Fall bekannt ist, muß das Argument *fstyp* gleich 0 sein. Für die Untersuchung eines nicht eingehängten Dateisystems muß *path* die Gerätedatei des Geräts angeben, auf dem sich das Dateisystem befindet und das Argument *fstyp* gibt

den Typ des Dateisystems an. In beiden Fällen werden Lese-, Schreib- und Ausführungserlaubnis für die angegebene Datei nicht benötigt, aber alle Dateiverzeichnis im Pfadnamenanfang der angegebenen Datei müssen durchsuchbar sein.

Die Funktion *fstatfs()* arbeitet analog zur Funktion *statfs()*, nur wird hier anstelle eines Dateinamens die Dateikennzahl einer offenen Dateibeschreibung benötigt.

ERGEBNIS

Bei Erfolg liefern die Funktionen *statfs()* und *fstatfs()* den Wert 0 und besetzen die Struktur *statfs*, auf die *buf* zeigt, mit den ermittelten Informationen. Im Fehlerfall liefern beide Funktionen den Wert −1 und besetzen *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktionen *stafs()* und *fstatfs()* schlagen fehl, wenn gilt:

[ENOTDIR]
Eine Komponente des Pfadnamenanfangs ist kein Dateiverzeichnis.

[ENOENT]
Die angegebene Datei existiert nicht.

[EACCESS]
Für eine Komponente des Pfandnamenanfangs existiert keine Durchsuchberechtigung.

[EFAULT]
buf oder *path* zeigen auf eine ungültige Adresse.

[EBADF]
fildes ist keine gültige, offene Dateikennzahl.

[EINVAL]
fstyp ist ein ungültiger Dateisystem-Typ; *path* ist keine blockorientierte Gerätedatei und *fstyp* ist ungleich 0 oder *len* ist negativ bzw. größer als *sizeof(struct statfs)*.

HINWEIS

Diese Funktionen lösen die Funktion *ustat()* ab und sollten von neuen Anwendungen anstelle dieser Funktion verwendet werden.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) nicht enthalten.

SIEHE AUCH

ustat().

NAME

stderr, stdin, stdout - standard I/O streams
Standard-Ein-/Ausgabeströme

DEFINITION

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

BESCHREIBUNG

Eine Datei mit zugeordneter Pufferung heißt ein Datenstrom (*stream*) und ist als Zeiger auf einen definierten Datentyp *FILE* deklariert. Die Funktion *fopen()* erzeugt bestimmte beschreibende Daten für einen Datenstrom und liefert einen Zeiger, mit dem dieser Datenstrom in allen weiteren Transaktionen gekennzeichnet wird. Normalerweise gibt es drei offene Datenströme mit konstanten Zeigern, die in der Include-Datei *<stdio.h>* deklariert sind, und die den offenen Standarddateien zugeordnet sind.

Zum Zeitpunkt des Programmstarts gibt es drei Datenströme, die vordefiniert sind, und die nicht explizit geöffnet werden müssen:

Standardeingabe	(für das Lesen konventioneller Eingaben),
Standardausgabe	(für das Schreiben konventioneller Ausgaben) und
Standardfehler-ausgabe	(für die Ausgabe von Diagnose- und Fehlermeldungen)

Wenn die Standardfehlerausgabe geöffnet ist, dann ist sie nicht vollständig gepuffert (siehe auch *setvbuf()*). Die Standardeingabe und die Standardausgabe sind genau dann vollständig gepuffert, wenn sichergestellt werden kann, daß diese Datenströme keinem interaktiven Gerät zugeordnet sind.

Die folgenden symbolischen Werte in *<unistd.h>* definieren die Dateikennzahlen, die den Datenströmen *stdin*, *stdout* und *stderr* zugeordnet werden, wenn die Anwendung gestartet wird:

STDIN_FILENO Dateikennzahl für Standardeingabe, *stdin*. Sie hat den Wert 0.

STDOUT_FILENO Dateikennzahl für Standardausgabe, *stdout*. Sie hat den Wert 1.

STDERR_FILENO Dateikennzahl für Standardfehlerausgabe, *stderr*. Sie hat den Wert 2.

PORTABILITÄT

Die Variablen *stdin*, *stdout* und *stderr* sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fclose(), *feof()*, *ferror()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *gets()*, *popen()*, *printf()*, *putc()*, *puts()*, *read()*, *scanf()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vprintf()*, *<stdio.h>*, *<unistd.h>*.

NAME

step - pattern match with regular expressions
Mustervergleich mit regulärem Ausdruck

DEFINITION

```
int step (string, expbuf)
char *string, *expbuf;
```

BESCHREIBUNG

Siehe unter *regexp*.

PORTABILITÄT

Die Funktion *step()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

stime - set time
Systemuhr stellen

DEFINITION

```
int stime (sekzg)
long *sekzg;
```

BESCHREIBUNG

Mit der Funktion *stime()* stellt man die Systemuhr (Uhrzeit und Datum).

Diese Funktion darf nur von einem Prozeß mit besonderen Rechten aufgerufen werden.

Das Argument *sekzg* zeigt auf eine Zeitangabe, die die seit 1. Januar 1970, 00:00:00 Uhr GMT vergangene Zeit in Sekunden angibt.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird –1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *stime()* schlägt fehl, wenn gilt:

[EPERM]
Die effektive Benutzernummer des aufrufenden Prozesses ist nicht die eines Prozesses mit besonderen Rechten, oder die Zeit kann nicht verändert werden.

HINWEIS

Da SINIX-Systeme eine batteriegepufferte Uhr haben, sollte die Uhr nicht verstellt werden.

PORTABILITÄT

Die Funktion *stime()* ist im X/Open-Standard nicht mehr enthalten.

SIEHE AUCH

time(), *<time.h>*.

NAME

strcat - concatenate two strings
Zeichenkette anfügen

DEFINITION

```
#include <string.h>

char *strcat (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strcat()* fügt eine Kopie der Zeichenkette, auf die *s2* zeigt, einschließlich des abschließenden Nullbytes, an das Ende der Zeichenkette an, auf die *s1* zeigt. Das erste Zeichen von *s2* überschreibt das Nullbyte am Ende von *s1*. Wenn das Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strcat()* die Adresse *s1*. Andernfalls liefert *strcat()* den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strcat()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strncat(), *<string.h>*.

NAME

strchr - string scanning operation
Zeichenkette durchsuchen

DEFINITION

```
#include <string.h>

char *strchr (s, c)
char *s;
int c;
```

BESCHREIBUNG

Die Funktion *strchr()* findet das erste Auftreten von *c* (umgewandelt in ein *char*) in der Zeichenkette, auf die *s* zeigt. Das abschließende Nullbyte wird als Teil der Zeichenkette angenommen.

ERGEBNIS

Bei ihrer Rückkehr liefert die Funktion *strchr()* einen Zeiger auf das Zeichen, oder den Nullzeiger, wenn das Zeichen nicht gefunden wurde.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Da der Nullzeiger sowohl bei Fehler geliefert wird als auch dann, wenn das Zeichen nicht gefunden wird, sollte eine Anwendung, die Fehlersituationen überprüfen will, *errno* vor dem Aufruf von *strchr()* gleich 0 setzen, Wenn *errno* nach der Rückkehr ungleich 0 ist, dann ist ein Fehler aufgetreten.

PORTABILITÄT

Die Funktion *strchr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strrchr(), *<string.h>*.

NAME

strcmp - compare two strings
Zeichenketten vergleichen

DEFINITION

```
#include <string.h>

int strcmp (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strcmp()* vergleicht die Zeichenkette, auf die *s1* zeigt mit der Zeichenkette, auf die *s2* zeigt. Das Vorzeichen eines Ergebnisses ungleich 0, das von *strcmp()* zurückgeliefert wird, hängt vom Vorzeichen der Differenz zwischen dem ersten Paar von Zeichen (die beide als *unsigned char* interpretiert werden), die in den beiden verglichenen Objekten unterschiedlich sind.

ERGEBNIS

Bei der Rückkehr liefert *strcmp()* eine ganze Zahl, die größer als, gleich oder kleiner als 0 ist, je nachdem, ob die Zeichenkette, auf die *s1* zeigt, größer, gleich oder kleiner der Zeichenkette ist, auf die *s2* zeigt.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strcmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strncmp(), *<string.h>*.

NAME

strcoll - string comparison using collating information
Zeichenketten gemäß Sortierreihenfolge vergleichen

DEFINITION

```
#include <string.h>

int strcoll (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strcoll()* vergleicht die Zeichenkette, auf die *s1* zeigt mit der Zeichenkette, auf die *s2* zeigt, wobei beide entsprechend der Kategorie *LC_COLLATE* der aktuellen internationalen Umgebung interpretiert werden.

Das Vorzeichen eines Ergebnisses ungleich 0, das von der Funktion *strcoll()* geliefert wird, ist durch die relative Ordnung innerhalb der Sortierreihenfolge des ersten Paars von Zeichen bestimmt, die sich in den verglichenen Objekten unterscheiden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strcoll()* eine ganze Zahl, die größer, gleich oder kleiner 0 ist, je nachdem ob die Zeichenkette, auf die *s1* zeigt, größer, gleich oder kleiner der Zeichenkette ist, auf die *s2* zeigt, wenn beide entsprechende der aktuellen internationalen Umgebung interpretiert werden. Im Fehlerfall besetzt *strcoll()* die Variable *errno*, aber es gibt kein Ergebnis, das für die Anzeige eines Fehlers reserviert ist.

FEHLER

Die Funktion *strcoll()* kann fehlschlagen, wenn gilt:

[EINVAL] Die Argumente *s1* oder *s2* enthalten Zeichen, die außerhalb des Definitionsbereichs der Sortierreihenfolge liegen.

HINWEIS

Da kein Ergebnis für die Anzeige eines Fehlers reserviert ist, sollte eine Anwendung, die Fehlersituationen überprüfen will, *errno* vor dem Aufruf von *strcoll()* gleich 0 setzen, Wenn *errno* nach der Rückkehr ungleich 0 ist, dann ist ein Fehler aufgetreten.

Die Funktionen *strxfrm()* und *strcmp()* sollten verwendet werden, um große Listen zu sortieren.

PORTABILITÄT

Die Funktion *strcoll()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strcmp(), *strxfrm()*, *<string.h>*.

NAME

strcpy - copy a string
Zeichenkette kopieren

DEFINITION

```
#include <string.h>

char *strcpy (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strcpy()* kopiert die Zeichenkette, auf die *s2* zeigt, einschließlich des abschließenden Nullbytes in den Vektor, auf den *s1* zeigt. Wenn das Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strcpy()* die Adresse *s1*. Anderfalls liefert *strcpy()* den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Kopieren von Zeichen wird in unterschiedlichen Implementierungen unterschiedlich durchgeführt. Daher kann eine Überlappung in portablen Anwendungen zu Überraschungen führen.

PORTABILITÄT

Die Funktion *strcpy()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Folgendes Programm gibt die aktuellen Inhalte von *s1* und *s2* aus, ruft dann *strcpy()* auf und gibt nochmal beide Inhalte aus:

```
main()
{
   char *strcpy();
   char *s1 = "silvia";
   char *s2 = "peter";

   printf("Inhalt s1: %s\nInhalt s2: %s\n", s1, s2);

   strcpy(s1,s2);                  /* s2 nach s1 kopieren */

   printf("Nach strcpy:\nInhalt s1: %s\nInhalt s2: %s\n", s1, s2);
   exit(0);
}
```

SIEHE AUCH

strncpy(), *<string.h>*.

NAME

strdup - duplicate string
Zeichenkette duplizieren

DEFINITION

```
#include <string.h>

char *strdup(string)
const char *string;
```

BESCHREIBUNG

Die Funktion *strdup()* liefert einen Zeiger auf eine neue Zeichenkette, die das Duplikat der als Argument *string* übergebenen Zeichenkette ist. Der Speicherplatz für die neue Zeichenkette, einschließlich des abschließenden Nullbytes, wird mit der Funktion *malloc()* reserviert.

ERGEBNIS

Bei Erfolg liefert die Funktion *strdup()* den Zeiger auf die neue Zeichenkette. Wenn der Platz für die neue Zeichenkette nicht reserviert werden kann, dann liefert die Funktion *strdup()* den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strdup()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Beispiel liest eine Zeichenkette von Standardeingabe, dupliziert diese und gibt, wenn kein Fehler aufgetreten ist, die beiden Zeichenketten sowie deren Adressen aus.

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

main()
{  char buffer[70], *ptr;

   if (gets(buffer) != NULL)
      if ((ptr=strdup(buffer)) != NULL)
      {  printf("buffer (%p): \"%s\"\n", buffer, buffer);
         printf("ptr (%p):    \"%s\"\n", ptr, ptr);
      }
      else
         fprintf(stderr, "strdup() fehlgeschlagen!\n");
   else
      fprintf(stderr, "gets() fehlgschlagen!\n");
}
```

SIEHE AUCH

malloc().

NAME

strerror - error message strings
Fehlermeldungstexte

DEFINITION

```
#include <string.h>

char *strerror (errnum)
int errnum;
```

BESCHREIBUNG

Die Funktion *strerror()* bildet die Fehlernummer in *errnum* auf einen sprachabhängigen Fehlermeldungstext ab und liefert einen Zeiger auf diesen. Die gelieferte Zeichenkette wird vom Programm nicht verändert, kann aber durch einen nachfolgenden Aufruf der Funktion *strerror()* überschrieben werden.
Die Sprache für die von *strerror()* gelieferten Fehlermeldungstexte richtet sich unter SINIX nach der Umgebungsvariablen LANG (siehe auch PORTABILITÄT).

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strerror()* einen Zeiger auf den erzeugten Fehlermeldungstext. Kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

FEHLER

Die Funktion *strerror()* kann fehlschlagen, wenn gilt:

[EINVAL] Der Wert von *errnum* ist keine gültige Fehlermeldungsnummer.

HINWEIS

Da kein Ergebnis für die Anzeige eines Fehlers reserviert ist, sollte eine Anweisung, die Fehlersituationen überprüfen will, *errno* gleich 0 setzen, dann *strerror()* aufrufen, dann *errno* prüfen und wenn es ungleich 0 ist, einen Fehler annehmen.

PORTABILITÄT

Die Funktion *strerror()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Die Sprache für die von *strerror()* gelieferten Fehlermeldungstexte ist implementierungs-abhängig. Unter SINIX richtet sich die Sprache nach der Umgebungsvariablen LANG.

SIEHE AUCH

<string.h>.

NAME

strftime - convert date and time to string
Datum und Zeit in eine Zeichenkette umwandeln

DEFINITION

```
#include <time.h>

size_t strftime (s, maxsize, format, timptr)
char *s, *format;
size_t maxsize;
struct tm *timptr;
```

BESCHREIBUNG

Die Funktion *strftime()* trägt Zeichen in den Vektor auf den *s* zeigt ein, so wie dies durch die Zeichenkette, auf die *format* zeigt, vorgegeben wird. Die Zeichenkette *format* besteht aus 0 oder mehr Umwandlungsanweisungen und einfachen Zeichen. Eine Umwandlungsanweisung besteht aus dem Zeichen '%', gefolgt von einem weiteren Zeichen, welches das Verhalten der Umwandlungs-Beschreibung festlegt. Alle einfachen Zeichen, einschließlich des abschließenden Zeichens '\0', werden ohne Änderung in den Vektor kopiert. Wenn das Kopieren zwischen überlappenden Objekten erfolgt, dann ist das Verhalten undefiniert. Nicht mehr als *maxsize* Zeichen werden in den Vektor eingetragen. Jede Umwandlungsanweisung wird, wie in der nachfolgenden Liste beschrieben, durch die passenden Zeichen ersetzt. Die passenden Zeichen werden durch die internationale Umgebung des Programms bestimmt, und durch die Werte in der Struktur, auf die *timptr* zeigt.

Die örtliche Zeitzonen-Information wird verwendet, als ob *strftime()* die Funktion *tzset()* aufrufen würde.

%a wird durch den abgekürzten Wochentag-Namen der internationalen Umgebung ersetzt.

%A wird durch den vollen Wochentag-Namen der internationalen Umgebung ersetzt.

%b wird durch den abgekürzten Monatsnamen der internationalen Umgebung ersetzt.

%B wird durch den vollen Monatsnamen der internationalen Umgebung ersetzt.

%c wird durch die passende Darstellung von Datum und Zeit der internationalen Umgebung ersetzt.

%d wird durch den Tag des Monats als ganze Zahl im Bereich [01, 31] ersetzt.

%D wird durch das Datum ersetzt (%m/%d/%y).

%h wird durch den abgekürzten Monatsnamen der internationalen Umgebung ersetzt.
%H wird ersetzt durch die Stunde als ganze Zahl im Bereich [00, 23] (24-Stunden-Uhr).
%I wird ersetzt durch die Stunde als ganze Zahl im Bereich [01, 12] (12-Stunden-Uhr).
%j wird durch den Tag des Jahres als ganze Zahl im Bereich [001, 366] ersetzt.
%m wird ersetzt durch den Monat als ganze Zahl im Bereich [01, 12].
%M wird ersetzt durch die Minute als ganze Zahl im Bereich [00, 59].
%n wird durch ein Neue-Zeile-Zeichen '\n' ersetzt.
%p wird durch das örtliche Äquivalent zu AM oder PM ersetzt.
%r wird ersetzt durch die Zeit in Vormittag/Nachmittag-Notation entsprechend den britischen/amerikanischen Konventionen (%I:%M:%S\e [AM|PM]).
%S wird ersetzt durch die Sekunde als ganze Zahl im Bereich [00, 61].
%t wird ersetzt durch ein Tabulator-Zeichen '\t'.
%T wird durch die Zeit ersetzt (%H:%M:%S).
%U wird ersetzt durch die Wochennummer des Jahres als ganze Zahl im Bereich [00,53], wobei der Sonntag als erster Tag der Woche gilt.
%w wird ersetzt durch den Wochentag als ganze Zahl im Bereich [0,6], wobei Sonntag gleich 0 ist.
%W wird ersetzt durch die Wochennummer des Jahres als ganze Zahl im Bereich [00,53], wobei der Montag als erster Tag der Woche gilt.
%x wird ersetzt durch die passende örtliche Datums-Darstellung.
%X wird ersetzt durch die passende örtliche Zeit-Darstellung.
%y wird ersetzt durch das Jahr als ganze Zahl im Bereich [00,99] (ohne Jahrhundert).
%Y wird ersetzt durch das Jahr als ganze Zahl (einschließlich Jahrhundert).
%Z wird ersetzt durch den Namen der Zeitzone oder durch nichts, falls keine Zeitzone existiert.
%% wird durch das Zeichen '%' ersetzt.

Für Anweisungen, die nicht in dieser Liste aufgeführt sind, ist das Verhalten undefiniert.

ERGEBNIS

Wenn die Gesamtzahl der Ergebnis-Zeichen einschließlich des abschließenden Nullbytes nicht größer als *maxsize* ist, dann liefert die die Funktion *strftime()* die Anzahl der Zeichen, die in den Vektor *s* geschrieben wurden ohne das abschließende Zeichen '\0'. Andernfalls wird der Wert *(size_t) 0* geliefert und der Inhalt des Vektors ist unbestimmt.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Der Bereich für die Werte von %S ist [00,61] statt [00,59] um die gelegentliche Schaltsekunde und die seltenere doppelte Schaltsekunde zu erlauben.

Einige der Anweisungen sind zu anderen gleich. Diese wurden aufgenommen, um die Kompatibilität mit den Funktionen *nl_cxtime()* und *nl_ascxtime()* zu gewährleisten, die in Ausgabe 2 des XPG definiert wurden.

PORTABILITÄT

Die Funktion *strftime()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

ctime(), *setlocale()*, *tzset()*, *<time.h>*.

NAME

strlen - get string length
Länge einer Zeichenkette ermitteln

DEFINITION

```
#include <string.h>

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

BESCHREIBUNG

Die Funktion *strlen()* berechnet die Anzahl der Zeichen in der Zeichenkette *s* ohne das abschließende Nullbyte.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strlen()* die Anzahl der Zeichen in der Zeichenkette *s*.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strlen()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<string.h>.

NAME

strncat - concatenate part of two strings
Zeichenkette teilweise anfügen

DEFINITION

```
#include <string.h>

char *strncat (s1, s2, n)
char *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *strncat()* fügt nicht mehr als *n* Zeichen aus dem Vektor *s2* an des Ende der Zeichenkette *s1* an. Ein Nullbyte und nachfolgende Zeichen werden nicht angefügt. Das erste Zeichen aus *s2* überschreibt das Nullbyte am Ende von *s1*. An das Ergebnis wird immer ein abschließendes Nullbyte angehängt. Wenn ein Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strncat()* die Adresse *s1*.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strncat()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strcat(), *<string.h>*.

NAME

strncmp - compare part of two strings
Zeichenketten teilweise vergleichen

DEFINITION

```
#include <string.h>

int strncmp (s1, s2, n)
char *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *strncmp()* vergleicht nicht mehr als *n* Zeichen aus dem Vektor *s1* mit dem Vektor *s2*. Zeichen, die auf das Nullbyte folgen, werden nicht verglichen.

Das Vorzeichen eines Ergebnisses ungleich 0, das von *strcmp()* zurückgeliefert wird, hängt vom Vorzeichen der Differenz zwischen dem ersten Paar von Zeichen (die beide als *unsigned char* interpretiert werden), die in den beiden verglichenen Objekten unterschiedlich sind.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strncmp()* eine ganze Zahl, die größer, gleich oder kleiner 0 ist, je nachdem, ob die, möglicherweise mit dem Nullbyte abgeschlossene Zeichenkette *s1* größer, gleich oder kleiner der möglicherweise mit dem Nullbyte abgeschlossene Zeichenkette ist, auf die *s2* zeigt.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strncmp()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

In folgendem Rate-Programm wird *strncmp* dazu benutzt die lexikographische Ordnung zweier Zeichenketten zu bestimmen:

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

main()
{
   int i,n,wert;
   char s[BUFSIZ],wort[BUFSIZ];

   printf("Bitte geben sie das zu ratende Wort ein:");

   system("stty -echo ");          /* Shellkommando-Aufruf :
                                    Bildschirmausgabe unterdrücken */
   scanf("%s",wort);

   system("stty  echo ");          /* Shellkommando-Aufruf :
                                    Bildschirmausgabe normal */
   n =strlen(wort);
   printf("\nDas eingegebene Wort hat %d Buchstaben \n",n);
   i=0;
   do
   {
      i++;
      printf("Ihr Versuch: ");
      scanf("%s",s);
      if (strlen(s) > n)
      {
       printf("Ihre Eingabe ist zu lang!\n");
       continue;
      }
      wert = strncmp(s,wort,n);    /* wert wird das Ergebnis von
                                    strncmp() zugewiesen */
      if (wert > 0)
         printf("%s ist lexikographisch groesser. \n",s);
      else if (wert < 0)
          printf("%s ist lexikographisch kleiner.\n",s);
   }
   while (wert != 0);
   printf("Richtig, das Wort hiess: %s\n",wort);
   printf("Sie haben %d Versuche gebraucht.\n",i);
}
```

SIEHE AUCH

strcmp(), *<string.h>*.

NAME

strncpy - copy part of a string
Zeichenkette teilweise kopieren

DEFINITION

```
#include <string.h>

char *strncpy (s1, s2, n)
char *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *strncpy()* kopiert nicht mehr als *n* Zeichen aus dem Vektor *s2* in den Vektor *s1*. auf den *s1* zeigt. Zeichen, die auf das Nullbyte folgen, werden nicht kopiert. Wenn das Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

Wenn der Vektor *s2* eine Zeichenkette kürzer als *n* ist, dann werden solange Nullbytes an das Ende der Kopie in den Vektor *s1* geschrieben, bis *n* Zeichen geschrieben sind.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strncpy()* die Adresse *s1*.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Kopieren von Zeichen wird in unterschiedlichen Implementierungen unterschiedlich durchgeführt. Daher kann eine Überlappung zu Überraschungen führen.

Wenn in den ersten *n* Zeichen des Vektors *s2* kein Nullbyte auftritt, dann ist das Ergebnis nicht mit dem Nullbyte abgeschlossen.

PORTABILITÄT

Die Funktion *strncpy()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strcpy(), *<string.h>*.

NAME

strpbrk - scan string for character
Zeichenkette auf Zeichen untersuchen

DEFINITION

```
#include <string.h>

char *strpbrk (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strpbrk()* ermittelt das erste Auftreten eines beliebigen Zeichens aus der Zeichenkette *s2* in der Zeichenkette *s1*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strpbrk()* einen Zeiger auf das Zeichen, bzw. den Nullzeiger, wenn kein Zeichen aus *s2* in *s1* auftritt.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strpbrk()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strchr(), *strrchr()*, *<string.h>*.

NAME

strrchr - string scanning operation
Zeichenkette durchsuchen

DEFINITION

```
#include <string.h>

char *strrchr (s, c)
char *s;
int c;
```

BESCHREIBUNG

Die Funktion *strrchr()* findet das letzte Auftreten von *c*, umgewandelt in den Typ *char*, in der Zeichenkette *s*. Das abschließende Nullbyte wird als Teil der Zeichenkette aufgefaßt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strrchr()* einen Zeiger auf das Zeichen, bzw. den Nullzieger, wenn das Zeichen *c* nicht in *s1* auftritt.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strrchr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strchr(), *<string.h>*.

NAME

strspn - get length of substring
Länge der Teilzeichenkette ermitteln

DEFINITION

```
#include <string.h>

size_t strspn (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strspn()* berechnet die Länge der maximalen Teilzeichenkette am Anfang der Zeichenkette *s1*, die ausschließlich aus Zeichen aus der Zeichenkette *s2* besteht.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strspn()* die Länge der Teilzeichenkette.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strspn()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strcspn(), *<string.h>*.

NAME

strstr - find substring
Teilzeichenkette suchen

DEFINITION

```
#include <string.h>

char *strstr (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Die Funktion *strstr()* findet das erste Auftreten der Zeichenkette in der Zeichenkette *s2* (ohne das abschließende Nullbyte) in der Zeichenkette *s1*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strstr()* einen Zeiger auf die gefundene Zeichenkette, bzw. den Nullzeiger, wenn die Zeichenkette nicht gefunden wird.

Wenn *s2* auf eine Zeichenkette der Länge 0 zeigt, dann liefert die Funktion die Adresse *s1*.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strstr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

strchr(), *<string.h>*.

NAME

strtod - convert string to double-precision number
Zeichenkette in Gleitkommazahl doppelter Genauigkeit umwandeln

DEFINITION

```
#include <stdlib.h>

double strtod (str, ptr)
char *str, **ptr;
```

BESCHREIBUNG

Die Funktion *strtod()* wandelt den Anfang der Zeichenkette *str* in eine *double*-Darstellung um. Zuerst zerlegt sie die Eingabe-Zeichenkette in drei Teile: eine erste, möglicherweise leere Folge von Zwischenraumzeichen (wie durch die Funktion *isspace()* definiert), eine Arbeitsfolge von Zeichen, die als Gleitkommakonstante interpretiert wird und schließlich eine abschließende Zeichenkette von nicht erkannten Zeichen, die das abschließende Nullbyte der Eingabezeichenkette einschließt. Danach versucht sie, die Arbeitsfolge in eine Gleitpunktzahl umzuwandeln und liefert schließlich das Ergebnis.

Die erwartete Form der Arbeitszeichenfolge ist: ein optionales Vorzeichen '+' oder '-', danach eine Folge von Ziffern, die optional den Dezimalpunkt enthalten dürfen und schließlich einen optionalen Exponententeil. Ein Exponententeil besteht aus dem Zeichen 'e' oder 'E', gefolgt von einem optionalen Vorzeichen und dann einer oder mehreren Dezimalziffern. Die Arbeitsfolge ist definiert als die längste Teilfolge der Eingabezeichenkette, die mit einem Zeichen beginnt, das kein Zwischenraumzeichen ist, und die ein Anfang der Unterfolge einer Folge ist, die der erwarteten Form entspricht. Die Arbeitsfolge enthält keine Zeichen, wenn die Eingabezeichenkette leer ist, bzw. nur aus Zwischenraumzeichen besteht, oder wenn das erste Zeichen, das kein Zwischenraumzeichen ist, ein anderes Zeichen als ein Vorzeichen, eine Ziffer oder der Dezimalpunkt ist.

Wenn die Arbeitsfolge die erwartete Form hat, dann wird diese Zeichenkette, beginnend mit der ersten Ziffer oder dem Dezimalpunkt (je nachdem, was eher auftritt), als eine Gleitkommakonstante interpretiert, außer daß der Dezimalpunkt, wenn weder er noch ein Exponententeil erscheint, als hinter der letzen Ziffer stehend angenommen wird. Wenn die Arbeitsfolge mit dem Vorzeichen '-' beginnt, dann wird das Ergebnis der

Umwandlung mit -1.0 multipliziert. Ein Zeiger auf die abschließende Zeichenkette wird in dem Objekt abgelegt, auf das *ptr* zeigt, vorausgesetzt, *ptr* ist nicht der Nullzeiger.

Der Dezimalpunkt wird durch die *langinfo*-Daten in der internationalen Umgebung des Programms definiert (Kategorie *LC_NUMERIC*). In der Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, in der dieses Zeichen nicht definiert ist, wird als Dezimalpunkt der Punkt '.' angenommen.

In anderen internationalen Umgebungen als der für die Sprache C, können andere, sprachabhängige Arbeitsfolgen akzeptiert werden.

Wenn die Arbeitsfolge leer ist oder nicht der erwarteten Form entspricht, dann wird keine Umwandlung durchgeführt. Der Wert von *str* wird dann in dem Objekt abgelegt, auf das *ptr*, zeigt, vorausgesetzt *ptr* ist nicht der Nullzeiger.

ERGEBNIS

Die Funktion *strtod()* liefert, falls vorhanden, den umgewandelten Wert. Wenn keine Umwandlung durchgeführt werden konnte, dann wird der Wert 0.0 zurückgeliefert.

Wenn der korrekte Wert einen Überlauf verursachen würde, dann wird +HUGE_VAL oder -HUGE_VAL zurückgeliefert, je nach dem Vorzeichen des Ergebnisses.

Wenn der korrekte Wert einen Unterlauf verursachen würde, dann wird der Wert 0.0 zurückgeliefert.

FEHLER

Die Funktion *strtod()* schlägt fehl, wenn gilt:

[ERANGE] Der Wert des Ergebnisses würde einen Über- oder Unterlauf verursachen.

HINWEIS

Da der Wert 0.0 sowohl bei Fehlern, als auch als gültiges Ergebnis bei Erfolg zurückgegeben werden kann, sollte eine Anwendung, die auf Fehlersituationen überprüfen will, *errno* gleich 0 setzen, dann *strtod()* aufrufen, dann *errno* prüfen und wenn diese Variable ungleich 0 ist, einen Fehler annehmen.

PORTABILITÄT

Die Funktion *strtod()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel unter *atof()*.

SIEHE AUCH

isspace(), *scanf()*, *setlocale()*, *strtol()*, *<stdlib.h>*, *Abschnitt 2.5*.

NAME

strtok - split string into tokens
Zeichenkette in Teile aufspalten

DEFINITION

```
#include <string.h>

char *strtok (s1, s2)
char *s1, *s2;
```

BESCHREIBUNG

Eine Folge von Aufrufen der Funktion *strtok()* teilt die Zeichenkette *s1* in eine Folge von Text-Zeichenketten auf, von denen jede durch ein Zeichen aus der Trennzeichenkette *s2* begrenzt wird. Der erste Aufruf in der Folge hat *s1* alsdas erste Argument und wird gefolgt von weiteren Aufrufen, mit dem Nullzeiger als erstem Argument. Die Trennzeichenkette *s2* kann von Aufruf zu Aufruf verschieden sein.

Der erste Aufruf in der Folge durchsucht *s1* nach dem ersten Zeichen, das nicht in der aktuellen Trennzeichenkette *s2* enthalten ist. Wird ein solches Zeichen nicht gefunden, dann sind keine Text-Zeichenketten in der Zeichenkette *s1* enthalten und die Funktion *strtok()* liefert den Nullzeiger. Wird ein solches Zeichen gefunden, dann ist des der Anfang der ersten Text-Zeichenkette.

Danach sucht die Funktion *strtok()* nach einem Zeichen, das in der aktuellen Trennzeichenkette enthalten ist. Wenn kein solches Zeichen gefunden wird, dann erstreckt sich die Text-Zeichenkette bis zum Ende von Zeichenkette *s1* und nachfolgede Suchen nach weiteren Text-Zeichenketten liefern den Nullzeiger. Wird solch ein Zeichen gefunden, dann wird es durch ein Nullbyte überschrieben, welches die aktuelle Text-Zeichenkette beendet. Die Funktion *strtok()* speichert die Adresse des nächsten Zeichens, von der aus die nächste Suche nach einer Text-Zeichenkette gestartet wird.

Jeder nachfolgende Aufruf mit dem Nullzeiger als erstem Argument beginnt seine Suche an der gesicherten Adresse und verhält sich sonst so, wie oben beschrieben.

Die Implementierung verhält sich so, als ob keine Bibliotheksfunktion die Funktion *strtok()* aufriefe.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strtok()* einen Zeiger auf das erste Zeichen einer Text-Zeichenkette. Andernfalls, wenn es keine Text-Zeichenkette gibt, liefert *strtok()* den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *strtok()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Programm ist ein einfaches Beispiel für *strtok()* (ohne Fehlerbehandlung).

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

char trenn[ ] = "-,;:.";

main()
{
   char string [512];
   register char *pc;

   while (scanf("%s", string) != 0)
   {
      printf("%s\n", strtok(string, trenn));
      while (pc = strtok((char*)0, trenn));
         printf("%s\n", pc);
   }
exit(0);
}
```

SIEHE AUCH

<string.h>.

NAME

strtol - convert string to long integer
Zeichenkette in ganze Zahl vom Typ long umwandeln

DEFINITION

```
#include <stdlib.h>

long strtol (str, ptr, base)
char *str, **ptr;
int base;
```

BESCHREIBUNG

Die Funktion *strtol()* wandelt den Anfang der Zeichenkette *str* in eine *long int*-Darstellung um. Zuerst unterteilt sie dazu die Eingabezeichenkette in drei Teile: eine, möglicherweise leere, Anfangsfolge von Zwischenraumzeichen (wie von der Funktion *isspace()* definiert), eine Arbeitsfolge, die als ganze Zahl gemäß einer Basis interpretiert wird, die durch den Wert von *base* bestimmt ist, und eine abschließende Zeichenkette von einem oder mehreren nicht erkannten Zeichen, die das abschließende Nullbyte mit einschließt. Danach versucht die Funktion die Arbeitsfolge umzuwandeln und liefert das Ergebnis zurück.

Wenn der Wert von *base* gleich 0 ist, dann ist die erwartete Form der Arbeitsfolge die einer ganzzahligen Konstanten mit einem optional vorangestellten '+' oder '-', aber ohne einen Suffix. Wenn der Wert von *base* zwischen 2 und 36 liegt, dann ist die erwartete Form der Arbeitsfolge eine Folge von Buchstaben und Ziffern, die eine ganze Zahl zu der Basis darstellt, die durch *base* angegeben wird. Auch diese Folgen können mit einem optionalen Vorzeichen '+' oder '-' beginnen, und auch dieser Folgen enthalten keinen Suffix. Den Buchstaben von *a* (oder *A*) bis *z* (oder *Z*) werden die Werte 10 bis 35 zugeordnet; nur Buchstaben, deren zugeordnete Werte kleiner als *base* sind, sind zugelassen. Wenn der Wert von *base* gleich 16 ist, dann können die Zeichen *0x* oder *0X* der Folge von Buchstaben und Ziffern vorangestellt sein (nach einem möglichen Vorzeichen).

Die Arbeitsfolge ist definiert als die längste Teilzeichenkette der Eingabezeichenkette, die mit dem ersten Zeichen beginnt, das kein Zwischenraumzeichen ist, und die der Anfang einer Folge der erwarteten Form ist. Die Arbeitsfolge enthält keine Zeichen, wenn die Eingabezeichenkette leer ist oder nur aus Zwischenraumzeichen besteht, oder wenn das erste Zeichen, das kein Zwischenraumzeichen ist, ungleich einem Vorzeichen oder einem erlaubten Buchstaben oder einer Ziffer ist.

Wenn die Arbeitsfolge die erwartete Form hat und der Wert von *base* gleich 0 ist, dann wird die Zeichenkette, beginnend mit der ersten Ziffer, als eine ganze Zahl interpretiert. Wenn die Arbeitsfolge die erwartete Form hat und der Wert von *base* zwischen 2 und 36 liegt, dann wird dieser Wert als Basis für die Umwandlung verwendet, wobei jedem Buchstaben der oben genannte Wert zugeordnet wird. Wenn die Arbeitsfolge mit einem Minuszeichen beginnt, dann wird das Ergebnis mit -1 multipliziert. Ein Zeiger auf die abschließende Zeichenkette wird in dem Objekt abgelegt, auf das *ptr* zeigt, vorausgesetzt, *ptr* ist nicht der Nullzeiger.

In einer anderen internationalen Umgebung als der der Sprache C können weitere sprachabhängige Folgen von Arbeitsformen akzeptiert werden.

Wenn die Arbeitsfolge leer ist oder nicht die erwartete Form hat, dann wird keine Umwandlung durchgeführt und der Wert von *str* wird in dem Objekt abgelegt, auf das *ptr* zeigt, vorausgesetzt, *ptr* ist nicht der Nullzeiger.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *strtol()* den umgewandelten Wert, sofern vorhanden. Wenn der korrekte Wert einen Überlauf verursachen würde, dann wird unter SINIX der Wert HUGE_VAL geliefert und *errno* ist gleich ERANGE gesetzt (siehe auch PORTABILITÄT). Für alle anderen Fehler wird der Wert 0 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *strtol()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert von *base* wird nicht unterstützt.

[ERANGE] Das Ergebnis würde einen Überlauf verursachen.

HINWEIS

Da der Wert 0 sowohl bei Fehlern, als auch als gültiges Ergebnis bei Erfolg zurückgegeben werden kann, sollte eine Anwendung, die auf Fehlersituationen überprüfen will, *errno* gleich 0 setzen, dann *strtol()* aufrufen, dann *errno* prüfen und wenn diese Variable ungleich 0 ist, einen Fehler annehmen.

PORTABILITÄT

Die Funktion *strtol()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Wenn der korrekte Wert des Ergebnisses einen Überlauf verursachen würde, dann ist der zurückgelieferte Wert implementierungsabhängig und *errno* muß nicht unbedingt gleich ERANGE gesetzt sein.

Die Fehler [EINVAL] und [ERANGE] müssen für diese Funktion nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden.

SIEHE AUCH

isalpha(), *scanf()*, *strtod()*, *<stdlib.h>*.

NAME

strxfrm - string transformation
Umwandlung von zeichenketten

DEFINITION

```
#include <string.h>

size_t strxfrm (s1, s2, n)
char *s1, *s2;
size_t n;
```

BESCHREIBUNG

Die Funktion *strxfrm()* wandelt die Zeichenkette *s2* um und schreibt die Ergebniszeichenkette in den Vektor *s1*. Die Umwandlung erfolgt derart, daß eine Anwendung der Funktion *strcmp()* auf die beiden umgewandelten Zeichenketten einen Wert größer, gleich oder kleiner 0 liefert, entsprechend dem Resultat der Funktion *strcoll()*, die auf die beiden Originalzeichenketten angewendet wird. Es werden nicht mehr als *n* Zeichen in den Ergebnisvektor *s1* eingetragen, einschließlich des abschließenden Nullbytes. Wenn *n* gleich 0 ist, dann kann *s1* ein Nullzeiger sein. Wenn ein Kopieren zwischen überlappenden Objekten stattfindet, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *strxfrm()* liefert die Länge der umgewandelten Zeichenkette ohne das abschließende Nullbyte. Wenn der zurückgegebene Wert größer oder gleich *n* ist, dann ist der Inhalt des Vektors, auf den *s1* zeigt, unbestimmt.

Im Fehlerfall liefert die Funktion *strxfrm()* den Wert (*size_t*) -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *strxfrm()* schlägt fehl, wenn gilt:

[EINVAL] Die Argumente *s1* oder *s2* enthalten Zeichen, die außerhalb des Bereichs der Sortierreihenfolge sind.

HINWEIS

Die Umwandlungsfunktion arbeitet derart, daß zwei umgewandelte Zeichenketten durch die Funktionen *memcmp()* oder *strcmp()* sortiert werden können, wobei dies gemäß der Sortierreihenfolge-Information in der jeweiligen internationalen Umgebung des Programms erfolgt (Kategorie *LC_COLLATE*).

PORTABILITÄT

Die Funktion *strxfrm()* ist im X/Open-Standard (Ausgabe 3) definiert.

Portabilitäts-Hinweis:
Der Fehler [EINVAL] muß nicht unter allen X/Open-kompatiblen Implementierungen auftreten.

SIEHE AUCH

strcmp(), *strcoll()*, *<string.h>*.

NAME

swab - swap bytes
Bytes vertauschen

DEFINITION

```
void swab (src, dest, nbytes)
char *src, *dest;
int nbytes;
```

BESCHREIBUNG

Die Funktion *swab()* kopiert *nbytes* Bytes, auf die *src* zeigt, in das Objekt, auf das *dest* zeigt, wobei benachbarte Bytes vertauscht werden. Das Argument *nbytes* sollte gerade und nicht negativ sein. Wenn *nbytes* ungerade und positiv ist, dann kopiert und vertauscht *swab() nbytes-1* Bytes und die Anordnung des letzten Bytes ist undefiniert. Wenn *nbytes* negativ ist, dann macht *swab()* nichts.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Kopieren von Zeichen wird von Implementierung zu Implementierung unterschiedlich durchgeführt. Daher kann eine Überlappung in portablen Programmen für Überraschungen sorgen.

PORTABILITÄT

Die Funktion *swab()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispiel zeigt, wie swab() die benachbarten Bytes vertauscht:

```
char a[ ] = "uHcs!hF12X";
char b[ ] = "              ";

main()
{
   swab(a,b,10);
   printf( "%s\n%s\n",a,b);
}
```

NAME

sync - synchronise file system
Superblock aktualisieren

DEFINITION

```
void sync ( )
```

BESCHREIBUNG

sync() schreibt alle Informationen, die ein Dateisystem aktualisieren, aus den Systempuffern auf Festplatte. Hierzu gehören geänderte Superblöcke und Indexeinträge sowie verzögerte Block-Ein/Ausgaben.

Die Ausgabe ist beim Rücksprung von *sync()* nicht notwendigerweise bereits beendet.

Der Begriff Systempuffer bezeichnet Puffer, die dem Benutzer nicht zugänglich sind und nur der Leistungssteigerung dienen. Er bezieht sich nicht auf die von den Funktionen der Standard-Ein- und Ausgabe verwendeten Puffer.

HINWEIS

Der *sync()*-Aufruf sollte in normalen Anwenderprogrammen nicht verwendet werden, sondern nur bei Programmen, die das Dateisystem überprüfen, wie z.B. */etc/fsck*.

PORTABILITÄT

Die Funktion *sync()* ist im X/Open-Standard nicht mehr enthalten. Anwendungen sollten statt dessen die Funktion *fsync()* verwenden oder Dateien mit gesetztem O_SYNC-Bit öffnen. Diese beiden Verfahren sorgen für eine Synchronisation von Ausgaben, die im Gegensatz zu *sync()* dateibezogen ist.

SIEHE AUCH

fcntl(), *fsync()*, *open()*, *write()*.

NAME

sysconf - get configurable system variables
konfigurierbare Systemvariablen prüfen

DEFINITION

```
#include <unistd.h>

long sysconf (name)
int name;
```

BESCHREIBUNG

Die Funktion *sysconf()* stellt eine Methode für Anwendungen dar, die augenblicklichen Werte einer konfigurierbaren Systemgrenze oder -option zu bestimmen.

Das Argument *name* repräsentiert die Systemvariable, die überprüft werden soll. Die folgende Tabelle zeigt die Mindestmenge der Systemvariablen aus *<limits.h>*, *<unistd.h>* oder *<time.h>* (für {CLK_TCK}) die von *sysconf()* zurückgegeben werden können, sowie die symbolischen Konstanten, die in *<unistd.h>* definiert sind, und die die entsprechenden Werte besitzen, die für *name* verwendet werden:

Variable	*Wert von name*
ARG_MAX	_SC_ARG_MAX
CHILD_MAX	_SC_CHILD_MAX
CLK_TCK	_SC_CLK_TCK
NGROUPS_MAX	_SC_NGROUPS_MAX
OPEN_MAX	_SC_OPEN_MAX
PASS_MAX	_SC_PASS_MAX
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
_POSIX_SAVED_IDS	_SC_SAVED_IDS
_POSIX_VERSION	_SC_VERSION

Der Wert von {CLK_TCK} kann variabel sein und man sollte nicht annehmen, daß {CLK_TCK} eine Konstante zur Übersetzungszeit ist. Der Wert von {CLK_TCK} ist derselbe den *sysconf(_SC_CLK_TCK)* liefert.

ERGEBNIS

Wenn *name* ein ungültiger Wert ist, dann liefert *sysconf()* den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen. Wenn die Variable, die *name* entspricht, nicht im System definiert ist, dann liefert *sysconf()* den Wert -1, ohne die Variable *errno* zu verändern.

Andernfalls liefert die Funktion *sysconf()* den augenblicklichen Wert der Variablen für das System. Der zurückgegebene Wert ist nicht mehr eingeschränkt als der entsprechende Wert in der Anwendung, wenn diese mit *<limits.h>* oder *<unistd.h>* der jeweiligen Implementierung übersetzt worden wäre. Der Wert ändert sich während der Lebensdauer des aufrufenden Prozesses nicht.

FEHLER

Die Funktion *sysconf()* schlägt fehl, wenn gilt:

[EINVAL] Der Wert des Arguments *name* ist ungültig.

HINWEIS

Da alle Ergebniswerte im Erfolgsfall erlaubt sind, sollte eine Anwendung, die Fehlersituationen überprüfen will, die Variable *errno* auf 0 setzen, danach *sysconf()* aufrufen und prüfen, falls die Funktion -1 zurückgibt, ob *errno* ungleich 0 ist.

PORTABILITÄT

Die Funktion *sysconf()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

pathconf(), *<limits.h>*, *<unistd.h>*.

NAME

system - issue a command
Kommando ausführen

DEFINITION

```
#include <stdio.h>

int system (string)
char *string;
```

BESCHREIBUNG

Die Funktion *system()* sorgt dafür, daß das Argument *string* als Eingabe an einen Kommando-Interpreter übergeben wird, der zum Kommando *sh* kompatibel ist.

Die Ausführung des aktuellen Prozesses wird unterbrochen, bis sich der Kommando-Interpreter beendet und der Ende-Status zurückgeliefert wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *system()* den Ende-Status des Kommando-Interpreter-Prozesses. Andernfalls liefert sie eine negative Zahl und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *system()* schlägt fehl, wenn gilt:

[EAGAIN] Die systemspezifische Grenze für die Maximalzahl gleichzeitig ausgeführter Prozesse für das System oder eine einzelne Benutzernummer ({CHILD_MAX}) würde überschritten werden.

[EINTR] Die Funktion *system()* wurde durch ein Signal unterbrochen.

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

PORTABILITÄT

Die Funktion *system()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das als erstes Argument übergebene Kommando wird ausgeführt und der Rückgabestatus ausgegeben:

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

   i=system(argv[1]);
   if (i<0)
      printf("Fehler beim system()-Aufruf, i = %d\n", i);
   else
      if (system > 0)
         printf("Fehler im Kommando '%s', i = %d\n", argv[1], i);
      else
         printf("system()-Aufruf korrekt ausgefuehrt, i = %d\n",i);
   return i;
}
```

SIEHE AUCH

exec, *pipe()*, *wait()*, *<signal.h>*, *<stdio.h>*, *Kommando s.*

NAME

tan - tangent function
Tangens

DEFINITION

```
#include <math.h>

double tan (x)
double x;
```

BESCHREIBUNG

Die Funktion *tan()* berechnet den Tanges ihres Arguments *x*, das im Bogenmaß angegeben wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *tan()* den Tangens von *x*.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Andernfalls wird entweder *errno* gesetzt, um den Fehler anzuzeigen oder ein NaN wird zurückgeliefert.

FEHLER

Die Funktion *tan()* schlägt fehl, wenn gilt:

[ERANGE] Das Ergebnis würde einen Über- oder Unterlauf verursachen oder die Größe von *x* ist derart, daß ein völliger oder teilweiser Verlust von Signifikanz eintreten würde.

Die Funktion *tan()* kann unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von *x* ist ein NaN.

HINWEIS

Die Funktion *tan()* kann an Genauigkeit einbüßen, wenn ihr Argument sehr von 0.0 verschieden ist.

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *tan()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

```
cc programm.c -lm
```

PORTABILITÄT

Die Funktion *tan()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Siehe das Beispiel unter *sin()*.

SIEHE AUCH

atan(), *isnan()*, *matherr()*, *<math.h>*.

NAME

tanh - hyperbolic tangent function
Tangens Hyperbolicus

DEFINITION

```
#include <math.h>

double tanh (x)
double x;
```

BESCHREIBUNG

Die Funktion *tanh()* berechnet den Tangens Hyperbolicus von *x*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *tanh()* den Tangens Hyperbolicus von *x*.

Wenn *x* ein NaN ist, dann wird ein NaN zurückgeliefert.

Andernfalls wird entweder 0.0 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen, oder ein NaN wird zurückgeliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tanh()* kann unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von *x* ist ein NaN.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf von *tanh()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Die Funktion *tanh()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Programm gibt den Tangens Hyperbolicus einer eingelesenen Zahl aus:

```
#include <math.h>

main()
{
   double x;

   printf("Bitte Zahl eingeben:  ");
   scanf("%lf",&x);
   printf("Der Tangens hyperbolicus von %g ist %g \n", x, tanh(x));
}
```

SIEHE AUCH

isnan(), *matherr()*, *tan()*, *<math.h>*.

NAME

tcdrain - wait for transmission of output
Auf die Übertragung einer Ausgabe warten

DEFINITION

```
#include <termios.h>

int tcdrain (fildes)
int fildes;
```

BESCHREIBUNG

Die Funtkion *tcdrain()* wartet, bis alle Ausgaben auf das Objekt übertragen worden sind, das durch *fildes* angegeben wird. Das Argument *fildes* ist eine offene Dateikennzahl, verbunden mit einer Datensichtstation.

Wenn _POSIX_JOB_CONTROL definiert ist, dann verursachen Versuche eines Prozesses, der Mitglied einer Hintergrund-Prozeßgruppe ist, die Funktion *tcdrain()* mit einem *fildes* aufzurufen, der mit seinem kontrollierenden Terminal verbunden ist, daß das Signal SIGTTOU an die Prozeßgruppe geschickt wird. Wenn der aufrufende Prozeß SIGTTOU-Signale blockiert oder ignoriert, dann darf der Prozeß die Operation ausführen und es wird kein Signal SIGTTOU gesendet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird -1 geliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcdrain()* schlägt fehl, wenn gilt:

[EBADF]	Das Argument *fildes* ist keine gültige Dateikennzahl.
[EINTR]	Ein Signal hat die Funktion *tcdrain()* unterbrochen.
[ENOTTY]	Die Datei, die durch *fildes* beschrieben wird, ist keine Datensichtstation.

PORTABILITÄT

Die Funktion *tcdrain()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

tcflush(), <termios.h>, Abschnitt 2.4, Allgemeine Terminalschnittstelle.

NAME

tcflow - suspend or restart the transmission or reception of data
Übertragung oder Empfang von Daten anhalten oder neu starten

DEFINITION

```
#include <termios.h>

int tcflow (fildes, action)
int fildes, action;
```

BESCHREIBUNG

Die Funktion *tcflow()* hält die Übertragung oder den Empfang von Daten zu oder von dem Objekt an, auf das *fildes* verweist, je nachdem, welchen Wert *action* hat. Das Argument *fildes* ist eine offene Dateikennzahl, die einer Datensichtstation zugeordnet ist.

Wenn *action* gleich TCOOFF ist, dann wird die Ausgabe angehalten. Wenn *action* gleich TCOON ist, dann wird die Ausgabe neu gestartet. Wenn *action* gleich TCIOFF ist, dann wird die Eingabe durch Übertragung des STOP-Zeichens angehalten. Wenn *action* gleich TCION ist, dann wird die Eingabe durch Übertragung des START-Zeichens neu gestartet.

Der Standard beim Öffnen einer Datensichtgerätedatei ist, daß weder Eingabe noch Ausgabe angehalten sind.

Wenn _POSIX_JOB_CONTROL definiert ist, dann verursachen Versuche eines Prozesses, der Mitglied einer Hintergrund-Prozeßgruppe ist, die Funktion *tcflow()* mit einem *fildes* aufzurufen, der mit seinem kontrollierenden Terminal verbunden ist, daß das Signal SIGTTOU an die Prozeßgruppe geschickt wird. Wenn der aufrufende Prozeß SIGTTOU-Signale blockiert oder ignoriert, dann darf der Prozeß die Operation ausführen und es wird kein Signal SIGTTOU gesendet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird -1 geliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcflow()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[EINVAL] Das Argument *action* besitzt keinen unterstützten Wert.

[ENOTTY] Die Datei, die durch *fildes* beschrieben wird, ist keine Datensichtstation.

PORTABILITÄT

Die Funktion *tcflow()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

tcsendbreak(), <termios.h>, Abschnitt 2.4, Allgemeine Terminalschnittstelle.

NAME

tcflush - flush non-transmitted output data,
non-read input data or both
nichtgesendete Ausgabe- oder
nichtgelesene Eingabedaten aufgeben

DEFINITION

```
#include <termios.h>

int tcflush (fildes, queue_selector)
int fildes, queue_selector;
```

BESCHREIBUNG

Die Funktion *tcflush()* verwirft Daten, die auf das Objekt, auf das *fildes* zeigt (eine offene Dateikennzahl, verbunden mit einem Terminal) geschrieben, aber noch nicht übertragen wurden oder empfangene aber noch nicht gelesene Daten, abhängig vom Wert von *queue_selector*:

Wenn *queue_selector* gleich TCIFLUSH ist, dann werden empfangene, aber noch nicht gelesene Daten verworfen. Wenn *queue_selector* gleich is TCOFLUSH ist, dann verwirft sie geschriebene, aber noch nicht übertragene Daten. Wenn *queue_selector* gleich TCIOFLUSH ist, dann verwirft sie sowohl empfgangene und noch nicht gelesene, als auch geschriebene und noch nicht übertragene Daten.

Wenn _POSIX_JOB_CONTROL definiert ist, dann verursachen Versuche eines Prozesses, der Mitglied einer Hintergrund-Prozeßgruppe ist, die Funktion *tcflush()* mit einem *fildes* aufzurufen, der mit seinem kontrollierenden Terminal verbunden ist, daß das Signal SIGTTOU an die Prozeßgruppe geschickt wird. Wenn der aufrufende Prozeß SIGTTOU-Signale blockiert oder ignoriert, dann darf der Prozeß die Operation ausführen und es wird kein Signal SIGTTOU gesendet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird -1 geliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcflush()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[EINVAL] Das Argument *queue_selector* besitzt keinen unterstützten Wert.

[ENOTTY] Die Datei, die durch *fildes* beschrieben wird, ist keine Datensichtstation.

PORTABILITÄT

Die Funktion *tcflush()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

tcdrain(), <termios.h>, Abschnitt 2.4, Allgemeine Terminalschnittstelle.

NAME

tcgetattr - get the parameters associated with the terminal
der Datensichtstation zugeordnete Parameter lesen

DEFINITION

```
#include <termios.h>

int tcgetattr (fildes, termios_p)
int fildes;
struct termios *termios_p;
```

BESCHREIBUNG

Die Funktion *tcgetattr()* liest die Attribute, die der *fildes* zugeordneten Datensichtstation und legt sie in der Struktur vom Typ *termios* ab, auf die *termios_p* zeigt. Das Argument *fildes* ist eine offene Dateikennzahl, die einer Datensichtstation zugeordnet ist.

Das Argument *termios_p* ist ein Zeiger auf eine Struktur vom Typ *termios*.

Wenn die spezielle Datensichtstation keine aufgespaltenen Baudraten unterstützt, so ist die Eingabe-Baudsrate, die in der Struktur des Typs *termios* abgelegt wird, gleich 0.

Jeder Prozeß darf die Funktion *tcgetattr()* ausführen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und die Variable *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcgetattr()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[ENOTTY] Die Datei, die zu *fildes* gehört, ist keine Datensichtstation.

PORTABILITÄT

Die Funktion *tcgetattr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

tcsetattr(), *<termios.h>*, *Abschnitt 2.4, Allgemeine Terminalschnittstelle.*

NAME

tcgetpgrp - get foreground process group ID
Vordergrund-Prozeßgruppennummer lesen

DEFINITION

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

pid_t tcgetpgrp (fildes)
int fildes;
```

BESCHREIBUNG

Falls {_POSIX_JOB_CONTROL} definiert ist:

Die Funktion *tcgetpgrp()* liefert den Wert der Prozeßgruppennummer der Vordergrund-Prozeßgruppe, die mit der Datensichtstation verbunden ist.

Die Funktion *tcgetpgrp()* kann von einem Prozeß aufgerufen werden, der Mitglied einer Hintergrund-Prozeßgruppe ist; trotzdem kann die Information nachträglich von einem Prozeß geändert werden, der Mitglied einer Vordergrund-Prozeßgruppe ist.

Andernfalls:

Entweder, die Implementierung unterstützt die Funktion *tcgetpgrp()* so wie oben beschrieben, oder der Aufruf von *tcgetpgrp()* schlägt fehl.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *tcgetpgrp()* den Wert der Prozeßgruppennummer der zur Datensichtstation gehörenden Vordergrund-Prozeßgruppe. Andernfalls wird der Wert -1 zurückgegeben und die Variable *errno* besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcgetpgrp()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[ENOSYS] Die Implementierung unterstützt diese Funktionalität nicht.

[ENOTTY] Der aufrufende Prozeß besitzt kein kontrollierendes Terminal, oder die Datei ist nicht das kontrollierende Terminal.

PORTABILITÄT

Die Funktion *tcgetpgrp()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

setsid(), *setpgid()*, *tcsetpgrp()*, *<sys/types.h>*.

NAME

tcsendbreak - send a "break" for a specific duration
Pause (Füllzeichen) für eine bestimmte Dauer senden

DEFINITION

```
#include <termios.h>

int tcsendbreak (fildes, duration)
int fildes, duration;
```

BESCHREIBUNG

Wenn die Datensichtstation asynchrone serielle Datenübertragung verwendet, dann verursacht die Funktion *tcsendbreak()* die Übertragung eines andauernden Stroms von 0-Bits für eine bestimmte Dauer. Wenn *duration* gleich 0 ist, dann werden 0-Bits für wenigstens 0,25 Sekunden und höchstens 0,5 Sekunden übertragen. Wenn *duration* ungleich 0 ist, dann sendet die Funktion für eine implementierungs-abhängige Zeit 0-Bits.

Wenn die Datensichtstation keine asynchrone serielle Datenübertragung verwendet, dann ist es nicht festgelegt, ob die Funktion *tcsendbreak()* Daten sendet, um eine "break"-Bedingung zu generieren (so wie in der Implementierung definiert), oder ob sie zurückkehrt, ohne eine Aktion auszuführen.

Wenn _POSIX_JOB_CONTROL definiert ist, dann verursacht jeder Versuch eines Prozesses, der Mitglied einer Hintergrund-Prozeßgruppe ist, die Funktion *tcsendbreak()* für einen *fildes*, der mit dem kontrollierenden Terminal verbunden ist, daß der Prozeßgruppe das Signal SIGTTOU geschickt wird. Wenn der aufrufende Prozeß Signale des Typs SIGTTOU blockiert oder ignoriert, dann darf dieser Prozeß die Operation ausführen und kein Signal wird geschickt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgegeben und die Variable *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcsendbreak()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[ENOTTY] Die zu *fildes* gehörende Datei ist keine Datensichtstation.

PORTABILITÄT

Die Funktion *tcsendbrk()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<termios.h>, *Abschnitt 2.4, Allgemeine Terminalschnittstelle.*

NAME

tcsetattr - set the parameters associated with the terminal
einer Datensichtstation zugeordnete Attribute setzen

DEFINITION

```
#include <termios.h>

int tcsetattr (fildes, optional_actions, termios_p)
int fildes, optional_actions;
struct termios *termios_p;
```

BESCHREIBUNG

Die Funktion *tcsetattr()* setzt die Attribute, die der Datensichtstation zugeordnet sind, die durch die offene Dateikennzahl *fildes* angesprochen wird. Sie legt diese wie folgt in der Struktur vom Typ *termios* ab, auf die *termios_p* zeigt:

Wenn *optional_actions* gleich TCSANOW ist, dann wird die Änderung sofort vorgenommen. Wenn *optional_actions* gleich TCSADRAIN ist, dann werden die Änderungen vorgenommen, nachdem alle Ausgaben, die auf *fildes* geschrieben wurden, übertragen worden sind. Diese Funktion sollte verwendet werden, wenn Parameter geändert werden, die die Ausgabe beeinflußen. Wenn *optional_actions* gleich TCSAFLUSH ist, dann werden die Änderungen vorgenommen, nachdem alle Ausgaben, die auf *fildes* geschrieben wurden, übertragen worden sind, und alle Eingaben, die bis dahin empfangen, aber noch nicht gelesen wurden, werden verworfen bevor die Änderungen vorgenommen werden.

Wenn die Ausgabebaudrate, die in der Struktur des Typs *termios* abgelegt sind, auf die *termios_p* zeigt gleich der Baudrate 0 ist (B0), dann werden die Modem-Kontrolleitungen nicht länger angesprochen. Normalerweise unterbricht dies die Verbindung.

Wenn die Eingabebaudrate, die in der Struktur des Typs *termios* abgelegt ist, auf die *termios_p* zeigt, gleich 0 ist, dann wird die Eingabebaudrate, die in der Hardware gesetzt wird, gleich der Ausgabebaudrate sein, die in der Struktur vom Typ *termios* abgelegt ist.

Wenn _POSIX_JOB_CONTROL definiert ist, dann verursachen Versuche eines Prozesses, der ein Mitglied einer Hintergrund-Prozeßgruppe ist, die Funktion *tcsetattr()* mit dem *fildes* aufzurufen, der mit dem kontrollierenden Terminal verbunden ist, daß der Prozeßgruppe ein Signal des Typs SIGTTOU gesendet wird.

Wenn der aufrufende Prozeß Signale des Typs SIGTTOU blockiert oder ignoriert, dann darf er die Operation ausführen und es wird kein Signal gesendet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0 zurück. Andernfalls wird der Wert -1 zurückgeliefert und die Variable *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcsetattr()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[EINVAL] Das Argument *optional_actions* besitzt keinen unterstützten Wert.

[ENOTTY] Die Datei, die zu *fildes* gehört ist keine Datensichtstation.

HINWEIS

Wenn eine Anwendung versucht, die Baudraten zu verändern, dann sollte sie zuerst *tcsetattr()* aufrufen und danach *tcgetattr()*, um zu bestimmen, welche Baudraten tatsächlich ausgewählt wurden.

PORTABILITÄT

Die Funktion *tcsetattr()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

cfgetispeed(), *tcgetattr()*, *<termios.h>*, *Abschnitt 2.4, Allgemeine Terminalschnittstelle.*

NAME

tcsetpgrp - set foreground process group ID
Vordergrund-Prozeßgruppennummer setzen

DEFINITION

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

int tcsetpgrp (fildes, pgrp_id)
int fildes;
pid_t pgrp_id;
```

BESCHREIBUNG

Falls {_POSIX_JOB_CONTROL} definiert ist:

Wenn der Prozeß ein kontrollierendes Terminal hat, dann setzt die Funktion *tcsetpgrp()* die Vordergrund-Prozeßgruppennummer, die zu dieser Datensichtstation gehört, auf den Wert *pgrp_id*. Die Datei, die zu *fildes* gehört, muß das kontrollierende Terminal des aufrufenden Prozesses sein. Das kontrollierende Terminal muß derzeit mit der Sitzung des aufrufenden Prozesses verbunden sein. Der Wert von *pgrp_id* muß gleich einer Prozeßgruppennummer eines Prozesses in der selben Sitzung wie der aufrufende Prozeß sein.

Andernfalls:

Entweder, die Implementierung unterstützt die Funktion *tcsetpgrp()* so, wie oben beschrieben, oder der Aufruf von *tcsetpgrp()* schlägt fehl.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgegeben und die Variable *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tcsetpgrp()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[EINVAL] Diese Implementierung unterstützt den Wert im Argument *pgrp_id* nicht.

[ENOSYS] Diese Implementierung unterstützt diese Funktionalität nicht.

[ENOTTY] Der aufrufende Prozeß besitzt kein kontrollierendes Terminal oder das kontrollierende Terminal ist nicht länger mit der Sitzung des aufrufenden Prozesses verbunden.

[EPERM] Der Wert von *pgrp_id* entspricht nicht der Prozeßgruppennummer eines Prozesses in der selben Sitzung wie der aufrufende Prozeß.

PORTABILITÄT

Die Funktion *tcsetpgrp()* ist im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß nicht jede Implementierung diese Funktion unterstützen muß.

SIEHE AUCH

tcgetpgrp(), *<sys/types.h>*.

NAME

tdelete - delete node from binary search tree
Knoten aus binärem Suchbaum entfernen

DEFINITION

```
char *tdelete (key, rootp, compar)
char *key, **rootp;
int (*compar)();
```

BESCHREIBUNG

Diese Funktion löscht einen Knoten aus einem binären Suchbaum. Siehe unter *tsearch()*.

PORTABILITÄT

Die Funktion *tdelete()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

telldir - current location of a named directory stream
Position im Dateiverzeichnisstrom ermitteln

DEFINITION

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

long telldir (dirp)
DIR *dirp;
```

BESCHREIBUNG

Die Funktion *telldir()* liefert die aktuelle Position, die dem angegebenen Dateiverzeichnisstrom zugeordnet ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *telldir()* die aktuelle Position.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *telldir()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

readdir(), *seekdir()*, *<dirent.h>*, *<sys/types.h>*.

NAME

tempnam - create a name for a temporary file
Namen für temporäre Datei erzeugen

DEFINITION

```
#include <stdio.h>

char *tempnam (dir, pfx)
char *dir, *pfx;
```

BESCHREIBUNG

Die Funktion *tempnam()* erzeugt einen Pfadnamen, der für eine temporäre Datei genutzt werden kann.

Die Funktion *tempnam()* erlaubt es dem Benutzer, die Wahl des Dateiverzeichnisses zu kontrollieren. Das Argument *dir* zeigt auf den Namen des Dateiverzeichnisses in dem die Datei erzeugt werden soll. Wenn *dir* der Nullzeiger ist oder auf eine Zeichenkette zeigt, die kein Name eines passenden Dateiverzeichnisses ist, dann wird der Pfandnamenanfang verwendet, der als {P_tmpdir} in der Include-Datei *<stdio.h>* definiert ist. Wenn auf dieses Dateiverzeichnis nicht zugegriffen werden kann, dann kann unter anderen Implermentierungen ein implementierungs-abhängiges Dateiverzeichnis verwendet werden.

Viele Anwendungen wollen, daß die Namen ihrer temporären Dateien mit bestimmten Buchstabenfolgen beginnen. Das Argument *pfx* sollte dafür verwendet werden. Dieses Argument kann der Nullzeiger sein, oder auf eine Zeichenkette mit bis zu fünf Zeichen zeigen, die als die erten Zeichen des Dateinamens verwendet werden sollen.

ERGEBNIS

Bei erfolgreicher Beendigung reserviert die Funktion *tempnam()* Speicherplatz für eine Zeichenkette, legt dort den erzeugten Pfadnamen ab und liefert einen Zeiger darauf zurück. Dieser Zeiger kann in einem nachfolgenden Aufruf von *free()* verwendet werden. Andernfalls liefert die Funktion den Nullzeiger und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tempnam()* schlägt fehl, wenn gilt:

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

HINWEIS

Diese Funktion erzeugt nur einen Pfadnamen. es liegt in der Verantwortung der Anwendung, die Dateien zu erzeugen oder zu löschen. Es ist möglich, daß zwischen dem Zeitpunkt, zu dem der Pfadname erzeugt wird, und dem Zeitpunkt des Öffnens der Datei ein anderer Prozeß eine Datei mit demselbem Namen erzeugt. Für Anwendungen ist daher die Funktion *tmpfile()* oft nützlicher.

Wenn die Funktion {TMP_MAX} Mal im selben Prozeß aufgerufen wird, dann ist ihr Verhalten undefiniert.

PORTABILITÄT

Die Funktion *tempnam()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fopen(), *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*, *<stdio.h>*.

NAME

tfind - search binary search tree
Binären Suchbaum durchsuchen

DEFINITION

```
char *tfind (key, rootp, compar)
char *key, **rootp;
int (*compar)();
```

BESCHREIBUNG

Diese Funktion sucht einen knoten in einem binären Suchbaum. Siehe unter *tsearch()*.

PORTABILITÄT

Die Funktion *tfind()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

tgetent - get termcap entry
termcap-Eintrag lesen

DEFINITION

```
int tgetent(bp, name)
char *bp, *name;
```

BESCHREIBUNG

Die Funktion *tgetent()* liest einen Eintrag aus der Datei */etc/termcap*. Der gelesene Eintrag wird an der Adresse, auf die das Argument *bp* zeigt, abgelegt. Dazu muß dort ein Speicherbereich von mindestens 1024 Byte bereitstehen.

Die Adresse *bp* wird von *tgetent()* intern abgespeichert, so daß sie an die Funktionen *tgetflag()*, *tgetnum()* und *tgetstr()*, die der Bearbeitung eines solchen gelesenen Eintrags dienen, nicht übergeben werden muß.

Das Argument *name* gibt an, welcher Eintrag aus der Datei */etc/termcap* gelesen werden soll. Üblicherweise ist *name* das Ergebnis eines Aufrufs der Funktion *getenv("TERM")* oder eine direkte Angabe eines Datensichstations-Typs.

ERGEBNIS

Bei Erfolg liefert die Funktion *tgetent()* als Ergebnis den Wert 1. Bei Mißerfolg liefert sie den Wert −1, wenn die *termcap*-Datei nicht geöffnet werden konnte bzw. den Wert 0, wenn in der *termcap*-Datei kein Eintrag für *name* existiert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Symbolisch definierte, nicht abdruckbare Zeichen werden erst durch die Routinen zur Ermittlung von *termcap*-Daten in ihre tatsächlichen Werte umgesetzt. So ist z.B. das Zeichen ESC nach dem Einlesen eines *termcap*-Eintrags immer noch als "\E" codiert. Erst die Funktion *tgetstr()* wandelt diese symbolische Codierung in '\033' um.

Wenn Sie die *termcap*-Funktionen in Ihrem Programmen nutzen wollen, dann müssen Sie diese beim Übersetzen durch die Option *-ltermcap* oder *-lcurses* dazubinden.

PORTABILITÄT

Die Funktion *tgetent()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Programm liest den *termcap*-Eintrag für die aktuelle Datensichtstation und gibt die einzelnen Felder des Eintrags am Bildschirm aus:

```
#include <stdio.h>

main()
{  char buffer[1024], erg;

   if ( (ret=tgetent(buffer,getenv("TERM"))) == 1)
   {  char * ptr;
      for (ptr=buffer; *ptr != '\0'; ptr++)
         if (*ptr == ':')
            putchar('\n');
         else
            putchar(*ptr);
   }
   else
   {  fprintf(stderr, "Fehler beim termcap-Einlesen: (%d)\n", erg);
      exit(1);
   }
}
```

SIEHE AUCH

tgetflag(), *tgetnum()*, *tgetstr()*, *tgoto()*, *tputs()*, *Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo.*

NAME

tgetflag - get boolean termcap entry
termcap: Boolesches Feld ermitteln

DEFINITION

```
int tgetflag(id)
char *id;
```

BESCHREIBUNG

Die Funktion *tgetflag()* sucht ein boolesches Feld im zuvor mit *tgetent()* eingelesenen *termcap*-Eintrag. Boolesche Felder haben die Form *id*, wobei *id* den Namen des Feldes bezeichnet. Ist ein boolesches Feld in einem *termcap*-Eintrag vorhanden, so besitzt die diesem Eintrag zugeordnete Datensichtstation die entsprechende Eigenschaft.

Das Argument *id* der Funktion *tgetflag()* ist ein Zeiger auf eine mit dem Nullbyte abgeschlossene Zeichenkette, die dem Namen des gesuchten Feldes entspricht.

ERGEBNIS

Das angegebene Feld wird im Puffer gesucht. Ist es vorhanden, so liefert die Funktion *tgetflag()* den Wert 1 als Ergebnis. Ist das gesuchte Feld nicht vorhanden, so wird der Wert 0 zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Funktionen der *termcap*-Bibliothek in einem Anwendungsprogramm nutzen zu können, müssen Sie diese Bibliothek zu Ihrem Programm dazubinden. Dazu geben Sie beim Übersetzen entweder den Schalter *-ltermcap* oder den Schalter *-lcurses* an.

PORTABILITÄT

Die Funktion *tgetflag()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Programm ermittelt, ob das aktuelle Terminal den Backspace kennt:

```
#include <stdio.h>

main()
{ int  tgetflag();
  char entry[1024], *terminal, *tgetent();

  if ((terminal = getenv("TERM")) == NULL)
  {  fprintf(stderr, "Umgebungsvariable TERM existiert nicht!\n");
     exit(1);
  }

  if (tgetent(entry, terminal) != 1)
  {  fprintf(stderr, "Eintrag fuer '%s' nicht gefunden!\n", terminal);
     exit(1);
  }

  if (tgetflag("bs"))
     printf("Terminal '%s' kennt den Backspace!\n", terminal);
  else
     printf("Terminal '%s' kennt den Backspace nicht!\n", terminal);
}
```

SIEHE AUCH

tgetent(),
Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo.

NAME

tgetnum - get numeric termcap entry
termcap: numerisches Feld ermitteln

DEFINITION

```
int tgetnum(id)
char *id;
```

BESCHREIBUNG

Die Funktion *tgetnum()* sucht ein Feld mit einem numerischen Wert im zuvor mit *tgetent()* eingelesenen *termcap*-Eintrag. Felder mit einem numerischen Wert haben die Form *id*#*val*, wobei *id* den Namen des Feldes und *val* den Wert des Feldes bezeichnet.

Das Argument *id* der Funktion *tgetnum()* ist ein Zeiger auf eine mit dem Nullbyte abgeschlossene Zeichenkette, die dem Namen des gesuchten Feldes entspricht.

ERGEBNIS

Das angegebene Feld wird im Puffer gesucht und sein Wert als *int* zurückgeliefert. Falls das Feld nicht existiert, wird der Wert −1 zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Funktionen der *termcap*-Bibliothek in einem Anwendungsprogramm nutzen zu können, müssen Sie diese Bibliothek zu Ihrem Programm dazubinden. Dazu geben Sie beim Übersetzen entweder den Schalter *-ltermcap* oder den Schalter *-lcurses* an.

PORTABILITÄT

Die Funktion *tgetnum()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Programm ermittelt die Anzahl der Zeilen der Datensichtstation:

```
#include <stdio.h>

main()
{ int lines, tgetnum();
  char entry[1024], *terminal, *tgetent();

  if ((terminal = getenv("TERM")) == NULL)
  {  fprintf(stderr, "Umgebungsvariable TERM existiert nicht!\n");
     exit(1);
  }

  if (tgetent(entry, terminal) != 1)
  {  fprintf(stderr, "Eintrag fuer '%s' nicht gefunden!\n", terminal);
     exit(1);
  }

  if ((lines = tgetnum("li")) == -1 )
  {  fprintf("termcap: Zeilenanzahl nicht angegeben!\n");
     exit(1);
  }
  else
     printf("Das Terminal '%s' hat %d Zeilen!\n", terminal, lines);
  exit(0);
}
```

SIEHE AUCH

tgetent(),
Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo.

NAME

tgetstr - get termcap string entry
termcap: Zeichenketten-Feld ermitteln

DEFINITION

```
char *tgetstr(id,area)
char *id, **area;
```

BESCHREIBUNG

Die Funktion *tgetstr()* sucht ein Feld, das eine Zeichenkette definiert, im vorher mit *tgetent()* eingelesenen *termcap*-Eintrag. Felder, die eine Zeichenkette definieren, haben die Form *id=str*, wobei *id* den Namen des Feldes und *str* die definierte Zeichenkette bezeichnet.

Das Argument *id* der Funktion *tgetstr()* ist ein Zeiger auf eine mit dem Nullbyte abgeschlossene Zeichenkette, die dem Namen des gesuchten Feldes entspricht. Das angegebene Feld wird im Puffer gesucht, die zugehörige Zeichenkette wird in ihre endgültige Form ungewandelt und in dem Speicherbereich abgelegt, auf den der Zeiger zeigt, dessen Adresse durch *area* angegeben wird.

Das Argument *area* ist die Adresse eines Zeigers auf *char*, den die Funktion *tgetstr()* benötigt, um sich die aktuelle Position in dem Speicherbereich zu merken, den der Benutzer zur Ablage der umgewandelten Zeichenketten bereitstellt. Der Zeiger, auf den *area* zeigt und der vom Benutzer bereitgestellt werden muß, wird von *tgetstr()* nach dem Eintragen der umgewandelten Zeichenkette weitergeschaltet, so daß er hinter das Ende der soeben eingetragenen Zeichenkette zeigt.

ERGEBNIS

Die Funktion *tgetstr()* liefert die Adresse der umgewandelten Zeichenkette zurück, die zu dem Feld *id* gehört. Falls das gesuchte Feld nicht existiert, wird der Nullzeiger zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Funktionen der *termcap*-Bibliothek in einem Anwendungsprogramm nutzen zu können, müssen Sie diese Bibliothek zu Ihrem Programm dazubinden. Dazu geben Sie beim Übersetzen entweder den Schalter *-ltermcap* oder den Schalter *-lcurses* an.

PORTABILITÄT

Die Funktion *tgetstr()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Das folgende Beispiel ermittelt die Steuerseqenzen für die Cursor-Positionierung ("cm") und für das Bildschirmlöschen ("cl"):

```
#include <stdio.h>

/* Hilfsfunktion fuer tputs() zur Ausgabe eines Zeichens */
int outc(c)
int c;
{  putchar(c);
}

main()
{ char entry[1024], *terminal, *tgetent();
  char *cm, *cl,        /* Adressen der Steuerseqenzen   */
       buffer[1024],    /* Speicher fuer Steuersequenzen */
       *bufptr=buffer, /* Zeiger auf buffer             */
       *tgetstr();

  if ((terminal = getenv("TERM")) == NULL)
  {  fprintf(stderr, "Umgebungsvariable TERM existiert nicht!\n")
     exit(1);
  }

  if (tgetent(entry, terminal) != 1)
  {  fprintf(stderr, "Eintrag fuer '%s' nicht gefunden!\n", termi
     exit(1);
  }

  if ((cm=tgetstr("cm",&bufptr)) == NULL )
  {  fprintf("termcap: Cursorpositionierung unmoeglich!\n");
     exit(1);
  }

  if ((cl=tgetstr("cl",&bufptr)) == NULL )
  {  fprintf("termcap: Bildschirmloeschen unmoeglich!\n");
     exit(1);
  }

  /* Bildschirmloeschen: */
  tputs("cl",0,outc);

  /* Cursor auf Zeile 5, Spalte 35: */
  tputs(tgoto("cl",34,4),0,outc);
}
```

SIEHE AUCH

tgetent(), *tgoto()*, *tputs()*, *Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo.*

NAME

tgoto - prepare cursor movement
termcap: Cursorpositionierung vorbereiten

DEFINITION

```
char * tgoto(cm, destcol, destline)
char * cm;
int destcol, destline;
```

BESCHREIBUNG

Die Funktion *tgoto()* wertet die Adresse *cm* als die Adresse einer *termcap*-Steuerzeichenfolge vom Typ "cm" aus und baut daraus eine Steuerzeichenfolge auf, die direkt auf den Bildschirm ausgegeben werden kann, um die gewünschte Positionierung zu erzielen.

Die Funktion *tgoto()* wandelt dazu alle Formatangaben in der Steuerzeichenfolge *cm* um, die mit einem '%' beginnen. Die Funktion *tgoto()* versucht intern alle Ausgaben von '\n','^D' und '\000' zu verhindern, da diese u.U. von Terminaltreibern besonders behandelt werden. Die Anwendung sollte in der Regel auch den Modus *Tabulatorzeichen expandieren* ausschalten, da u.U. auch Tabulatorzeichen zur Positionierung verwendet werden.

Das Ergebnis von *tgoto()* kann dann verwendet werden, um mit Hilfe der Funktion *tputs()* die Positionierung durchzuführen.

ERGEBNIS

Die Funktion *tgoto()* liefert eine Steuerzeichenfolge zurück, die unmittelbar zur Positionierung des Cursors an die gewünschte Position verwendet werden kann. Falls die Steuerzeichenfolge von *tgoto()* nicht verstanden wird, gibt die Funktion die Adresse der Zeichenkette "OOPS" zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die in früheren Versionen der *tercap*-Funktionen vorhandenen externen Variablen *BC* und *UP*, die von *tgoto()* verwendet wurden, stehen in dieser Version nicht mehr zur Verfügung.

Um die Funktionen der *termcap*-Bibliothek in einem Anwendungsprogramm nutzen zu können, müssen Sie diese Bibliothek zu Ihrem Programm dazubinden. Dazu geben Sie beim Übersetzen entweder den Schalter *-ltermcap* oder den Schalter *-lcurses* an.

PORTABILITÄT

Die Funktion *tgoto()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Siehe das Beispiel unter *tgetstr()*.

SIEHE AUCH

tgetent(), *tgetstr()*, *tputs()*, *Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo*.

NAME

time - get time
Zeit ermitteln

DEFINITION

```
#include <time.h>

time_t time (tloc)
time_t *tloc;
```

BESCHREIBUNG

Die Funktion *time()* liefert den Wert der Zeit in Sekunden seit dem Epochenwert.

Das Argument *tloc* zeigt auf einen Speicherbereich, in dem das Ergebnis abgelegt wird. Wenn *tloc* der Nullzeiger ist, dann wird kein Wert abgelegt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *time()* den Wert der Zeit.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *time()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

<time.h>.

NAME

times - get process und waited-for child process times
Laufzeit eines Prozesses und seiner Sohnprozesse ermitteln

DEFINITION

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

clock_t times (buffer)
struct tms *buffer;
```

BESCHREIBUNG

Die Funktion *times()* füllt die Struktur *tms*, auf die *buffer* zeigt, mit Informationen über Laufzeiten. Die Struktur *tms* ist definiert in *<sys/times.h>*.

Alle Zeitangaben werden im Raster von {CLK_TCK} Einheiten je Sekunde definiert.

Die Laufzeiten von beendeten Sohnprozessen werden in die Komponenten *tms_cutime* und *tms_cstime* des Vaterprozesses aufgenommen, sobald die Funktion *wait()* die Prozeßnummer dieses beendeten Sohnprozesses liefert. Wenn ein Sohnprozeß nicht auf seine Sohnprozesse wartet, dann werden deren Zeiten nicht mit aufgenommen.

- Die Komponente *tms_utime* ist die Rechenzeit, die für die Ausführung von Benutzeranweisungen des aufrufenden Prozesses verbraucht wurde.
- Die Komponente *tms_stime* ist die Rechenzeit, die für die Ausführung von Systemanweisungen des aufrufenden Prozesses verbraucht wurde.
- Die Komponente *tms_cutime* ist die Summe der Zeiten *tms_utime* und *tms_cutime* des Sohnprozesses.
- Die Komponente *tms_cstime* ist die Summe der Zeiten *tms_stime* und *tms_cstime* des Sohnprozesses.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *times()* die abgelaufene Echtzeit in {CLK_TCK} Einheiten je Sekunde seit einem bestimmten Zeitpunkt in der Vergangenheit (z.B. seit dem Einschalten des Systems). Dieser Zeitpunkt ändert sich nicht von einem Aufruf der Funktion *times()* innerhalb eines Prozesses zu einem anderen. Das Ergebnis kann den möglichen Wertebereich des Typs *clock_t* überschreiten (Überlauf).

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Portable Anwendungen sollten die Funktion *sysconf()* verwenden, um den Wert von {CLK_TCK} zu bestimmen, da dieser sich von System zu System unterscheiden kann.

PORTABILITÄT

Die Funktion *times()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Laufzeiten des aktuellen Prozesses auflisten:

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

struct tms buffer;

main()
{
   int status;

   system("sleep 8");

   times(&buffer);
   printf("Benutzerzeit : %ld\n", buffer.tms_utime);
   printf("Systemzeit : %ld\n", buffer.tms_stime);
   printf("Kind-Benutzerzeit : %ld\n", buffer.tms_cutime);
   printf("Kind-Systemzeit : %ld\n", buffer.tms_cstime);
}
```

SIEHE AUCH

exec, *fork()*, *sysconf()*, *time()*, *wait()*, *<sys/times.h>*.

NAME

timezone - difference from UTC und local standard time
Zeitzonen-Information

DEFINITION

```
extern long timezone;
```

BESCHREIBUNG

Diese Variable gibt die Differenz zwischen der Coordinated Universal Time (UTC) und der lokalen Zeit in Sekunden an. Siehe unter *tzset()*.

PORTABILITÄT

Die Variable *timezone* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

tmpfile - create a temporary file
Temporäre Datei erzeugen

DEFINITION

```
#include <stdio.h>

FILE *tmpfile()
```

BESCHREIBUNG

Die Funktion *tmpfile()* erzeugt eine temporäre Datei und öffnet einen dazugehörenden Datenstrom. Die Datei wird automatisch wieder gelöscht, sobald alle Verweise auf diese Datei geschlossen worden sind. Die Datei wird wie durch *fopen()* zum Ändern geöffnet (*"w+"*).

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *tmpfile()* einen Zeiger auf den Datenstrom für die erzeugte Datei. Andernfalls liefert sie den Nullzeiger und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Die Funktion *tmpfile()* schlägt fehl, wenn gilt:

[EMFILE] Für den Prozeß sind derzeit {FOPEN_MAX} Datenströme offen.

[ENFILE] Die Dateitabelle des Systems ist voll.

[ENOSPC] Das Dateiverzeichnis oder Dateisystem, das die neue Datei enthalten würde, kann nicht vergrößert werden.

Die Funktion *tmpfile()* schlägt fehl, wenn gilt:

[EACCES] Die Durchsucherlaubnis für eine Komponente des Pfadnamenanfangs der zu erzeugenden Datei oder die Schreiberlaubnis für das übergeordnete Dateiverzeichnis der zu erzeugenden Datei wird verweigert.

Die Funktion *tmpfile()* kann fehlschlagen, wenn gilt:

[EINTR] Ein Signal wurde während des Ablaufs der Funktion *tmpfile()* abgefangen.

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs der zu erzeugenden Datei ist kein Dateiverzeichnis.

[EROFS] Die zu erzeugende Datei würde sich in einem, nur zum Lesen eingehängten Dateisystem befinden.

HINWEIS

Der Datenstrom verweist auf eine Datei, für welche die Funktion *unlink()* aufgerufen wurde. Wenn der Prozeß zwischen dem Zeitpunkt der Erzeugung und dem Zeitpunkt des Aufrufs von *unlink()* abgebrochen wird, dann kann eine permanente Datei zurückbleiben.

Unter einigen Implementierungen kann eine Fehlermeldung ausgegeben werden, wenn der Datenstrom nicht geöffnet werden kann.

PORTABILITÄT

Die Funktion *tmpfile()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fopen(), *tmpnam()*, *unlink()*, *<stdio.h>*.

NAME

tmpnam - create a name for a temporary file
Namen für temporäre Datei erzeugen

DEFINITION

```
#include <stdio.h>

char *tmpnam (s)
char *s;
```

BESCHREIBUNG

Die Funktion *tmpnam()* erzeugt eine Zeichenkette, die ein gültiger, eindeutiger Dateiname ist.

Die Funktion *tmpnam()* erzeugt jedesmal, wenn sie vom selben Prozeß aufgerufen wird, einen anderen Dateinamen (bis zu {TMP_MAX} mal). Wenn die Funktion öfter als {TMP_MAX} mal aufgerufen wird, dann erzeugt sie erneut bereits erzeugte Namen.

Die Implementierung verhält sich so, als ob keine Bibliotheksfunktion die Funktion *tmpnam()* aufrufen würde.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *tmpnam()* einen Zeiger auf eine Zeichenkette.

Wenn das Argument *s* der Nullzeiger ist, dann legt die Funktion *tmpnam()* ihr Ergebnis in einem internen, statischen Objekt ab und liefert einen Zeiger auf das Objekt. Nachfolgende Aufrufe der Funktion *tmpnam()* können dasselbe Objekt verändern. Wenn das Argument *s* nicht der Nullzeiger ist, dann wird angenommen, daß es auf einen Vektor vom Typ *char* der Mindestlänge {L_tmpnam} zeigt; die Funktion *tmpnam()* schreibt ihr Ergebnis in diesen Vektor und liefert das Argument als Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Diese Funktion erzeugt nur Dateinamen. Es liegt in der Verantwortung der Anwendung, die Dateien zu erzeugen und zu löschen.

Es ist möglich, daß zwischen dem Zeitpunkt, zu dem der Pfadname erzeugt wird, und dem Zeitpunkt des Öffnens der Datei ein anderer Prozeß eine Datei mit demselbem Namen erzeugt. Für Anwendungen ist daher die Funktion *tmpfile()* oft nützlicher.

PORTABILITÄT

Die Funktion *tmpnam()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fopen(), *open()*, *tempnam()*, *tmpfile()*, *unlink()*, *<stdio.h>*.

NAME

toascii - translate integer to a 7-bit ASCII character
Zahl in 7-Bit ASCII-Zeichen umwandeln

DEFINITION

```
#include <ctype.h>

int toascii (c)
int c;
```

BESCHREIBUNG

Die Funktion *toascii()* wandelt ihr Argument in ein 7-Bit ASCII-Zeichen um.

ERGEBNIS

Die Funktion *toascii()* liefert den Wert (*c* & 0x7f).

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *toascii()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

isascii(), *<ctype.h>*.

NAME

_tolower - transliterate upper-case characters to lower-case
Groß- in Kleinbuchstaben umsetzen

DEFINITION

```
#include <ctype.h>

int _tolower (c)
int c;
```

BESCHREIBUNG

Das Makro *_tolower()* ist äquivalent zur Funktion *tolower(c)*, außer daß das Argument *c* ein Großbuchstabe sein muß.

ERGEBNIS

Siehe im Abschnitt BESCHREIBUNG.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *_tolower()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Programm liest eine Zeichenkette ein und wandelt die Zeichen zunächst in Kleinbuchstaben und dann in Großbuchstaben um:

```
#include <ctype.h>

 main()
 {
    int i;
    char s[81];

    printf("Bitte s eingeben \n");
    scanf("%s",s);

    printf("in klein: \n");
    i=0;

    while(s[i] != '\0')
       if (isupper(s[i]))
          printf("%c",_tolower(s[i++]));
       else
          printf("%c",s[i++]);

    printf("\nIN GROSS: \n");
    i=0;

    while(s[i] != '\0')
      if (islower(s[i]))
          printf("%c",_toupper(s[i++]));
      else
          printf("%c",s[i++]);

    printf("\n");
}
```

SIEHE AUCH

tolower(), *isupper()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

tolower - transliterate upper-case characters to lower-case
Groß- in Kleinbuchstaben umwandeln

DEFINITION

```
#include <ctype.h>

int tolower (c)
int c;
```

BESCHREIBUNG

Die Funktion *tolower()* besitzt als Argument ein *int*, dessen Wert als *unsigned char* oder als der Wert von EOF darstellbar sein muß. Wenn das Argument einen anderen Wert hat, dann ist das Verhalten unbestimmt. Wenn das Argument der Funktion *tolower()* einen Großbuchstaben darstellt, wie dies in der Kategorie *LC_CTYPE* der internationalen Umgebung des Programms definiert ist, dann ist das Ergebnis der entsprechende Kleinbuchstabe. Alle anderen Argumente im zulässigen Wertebereich werden unverändert zurückgeliefert.

In der internationalen Umgebung für die Programmiersprache C oder in einer Umgebung, in der keine Großbuchstaben-Information definiert ist, entscheidet die Funktion gemäß den Regeln des US-ASCII-Zeichensatzes, ob ein Zeichen ein Großbuchstabe ist. Zeichen außerhalb des Bereichs der ASCII-Zeichen werden unverändert zurückgegeben.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *tolower()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

setlocale(), *<ctype.h>*, *Abschnitt 2.5*.

NAME

_toupper - transliterate lower-case characters to upper-case
Klein- in Großbuchstaben umwandeln

DEFINITION

```
#include <ctype.h>

int _toupper (c)
int c;
```

BESCHREIBUNG

Das Makro *_toupper()* ist äquivalent zur Funktion *toupper()*, außer daß das Argument *c* ein Kleinbuchstabe sein muß.

ERGEBNIS

Siehe unter *toupper()*.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *_toupper()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Ein Beispiel zu *_toupper()* finden sie unter *_tolower()*.

SIEHE AUCH

islower(), *toupper()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

toupper - transliterate lower-case characters to upper-case
Klein- in Großbuchstaben umwandeln

DEFINITION

```
#include <ctype.h>

int toupper (c)
int c;
```

BESCHREIBUNG

Die Funktion *toupper()* besitzt als Argument ein *int*, dessen Wert als *unsigned char* oder als der Wert von EOF darstellbar sein muß. Wenn das Argument einen anderen Wert hat, dann ist das Verhalten unbestimmt. Wenn das Argument der Funktion *toupper()* einen Kleinbuchstaben darstellt, wie dies in der Kategorie *LC_CTYPE* der internationalen Umgebung des Programms definiert ist, dann ist das Ergebnis der entsprechende Großbuchstabe. Alle anderen Argumente im zulässigen Wertebereich werden unverändert zurückgeliefert.

In der internationalen Umgebung für die Programmiersprache C oder in einer Umgebung, in der keine Kleinbuchstaben-Information definiert ist, entscheidet die Funktion gemäß den Regeln des US-ASCII-Zeichensatzes, ob ein Zeichen ein Kleinbuchstabe ist. Zeichen außerhalb des Bereichs der ASCII-Zeichen werden unverändert zurückgegeben.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *toupper()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

setlocale(), *<ctype.h>*, *Abschnitt 2.5*.

NAME

tputs - put termcap string
termcap: Steuerzeichenfolgen ausgeben

DEFINITION

```
void tputs(cp, affcnt, outc)
char *cp;
int affcnt;
int (*outc)();
```

BESCHREIBUNG

Die Funktion *tputs()* wertet die Wartezeitangaben am Beginn von Steuerzeichenfolgen aus und übergibt die nötige Anzahl von Füllzeichen im Anschluß an die Steuerzeichenfolge. Die Ausgabe von *tputs()* wird zeichenweise an die Funktion übergeben, deren Adresse im Argument *outc* angegeben wird. Das Argument *cp* ist ein Zeiger auf die auszugebende Steuerzeichenfolge, *affcnt* enthält die Anzahl der von der Operation betroffenen Zeilen oder den Wert 1, wenn die Operation nicht mehrere Zeilen betrifft.

ERGEBNIS

Die Funktion *tputs()* liefert keinen Wert zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Für Datensichtstationen mit Übertragungsgeschwindigkeiten größer als 9600 Baud wird keine Wartezeitbearbeitung durchgeführt.

Die in früheren Versionen der *tercap*-Funktionen vorhandenen externen Variablen *ospeed* und *PC*, die von *tputs()* verwendet wurden, stehen in dieser Version nicht mehr zur Verfügung.

Um die Funktionen der *termcap*-Bibliothek in einem Anwendungsprogramm nutzen zu können, müssen Sie diese Bibliothek zu Ihrem Programm dazubinden. Dazu geben Sie beim Übersetzen entweder den Schalter *-ltermcap* oder den Schalter *-lcurses* an.

PORTABILITÄT

Die Funktion *tputs()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

BEISPIEL

Siehe das Beispiel unter *tgetstr()*.

SIEHE AUCH

tgetent(), *tgetstr()*, *tgoto()*, *Abschnitt 2.8: Bildschirmsteuerung mit termcap/terminfo.*

NAME

tdelete, tfind, tsearch, twalk - manage binary search tree
Binären Suchbaum verwalten

DEFINITION

```
#include <search.h>

void *tsearch (key, rootp, compar)
void *key, **rootp;
int (*compar)();

void *tfind (key, rootp, compar)
void *key, **rootp;
int (*compar)();

void *tdelete (key, rootp, compar)
void *key, **rootp;
int (*compar)();

void twalk (root, action)
void *root;
void (*action)();
```

BESCHREIBUNG

Die Funktionen *tsearch()*, *tfind()*, *tdelete()* und *twalk()* manipulieren binäre Suchbäume. Vergleiche werden durch Funktionen vorgenommen, die der Benutzer schreibt und deren Adresse im Argument *compar* übergeben wird. Diese Routinen werden mit zwei Argumenten aufgerufen, den Zeigern auf die zu vergleichenden Elemente. Die vom Benutzer zu schreibenden Routinen müssen eine ganze Zahl liefern, die kleiner, gleich oder größer 0 ist, je nachdem, ob das erste Element als kleiner, gleich oder größer dem zweiten Argument angesehen werden soll. Die Vergleichsfunktion muß nicht jedes Byte vergleichen, so daß zusätzlich zu den Vergleichswerten noch weitere Daten in den Elementen enthalten sein können.

Die Funktion *tsearch()* wird dazu verwendet, Bäume aufzubauen und darauf zuzugreifen. Das Argument *key* ist ein Zeiger auf den Eintrag, der gespeichert, oder auf den zugegriffen werden soll. Wenn es im Baum einen Eintrag gibt, der gleich **key* ist (dem Wert des Objekts auf das *key* zeigt), dann wird ein Zeiger auf diesen gefundenen Eintrag zurückgeliefert. Andernfalls wird **key* eingefügt, und ein Zeiger darauf wird zurückgeliefert. Es werden nur Zeiger kopiert, daher muß die aufrufende Routine die Daten speichern. Das Argument *rootp* zeigt auf eine Zeigervariable, die auf die Wurzel des Baumes zeigt. Der Wert NULL für die Variable, auf die *rootp* zeigt, bezeichnet einen leeren Baum; in diesem Fall wird die Variable so gesetzt, daß sie auf den Eintrag

zeigt, der an der Wurzel des neuen Baums steht.

Wie auch *tsearch()*, so sucht auch die Funktion *tfind()* nach einem Eintrag im Baum und liefert einen Zeiger auf den gefundenen Eintrag. Wenn der Eintrag nicht gefunden wird, dann liefert die Funktion *tfind()* den Nullzeiger. Die Argumente für die Funktion *tfind()* sind dieselben wie für *tsearch()*.

Die Funktion *tdelete()* löscht einen Knoten aus einem binären Suchbaum. Die Argumente sind dieselben wie für die Funktion *tsearch()*. Die Variable, auf die *rootp* zeigt wird verändert, wenn der gelöschte Knoten die Wurzel des Baums war. Die Funktion *tdelete()* liefert einen Zeiger auf den Vaterknoten des gelöschten Knotens bzw. den Nullzeiger, wenn der Knoten nicht gefunden wird.

Die Funktion *twalk()* arbeitet einen binären Suchbaum ab. Das Argument *root* ist die Wurzel des abzuarbeitenden Baums. Jeder Knoten eines Baums kann als Wurzel für eine Abarbeitung der Knoten unterhalb dieses Knotens dienen. *action* ist der Name der Routine, die für jeden Knoten aufgerufen wird. Diese Routine wird mit drei Argumenten aufgerufen. Das erste Argument ist die Adresse des aufgesuchten Knotens, das zweite Argument ist ein Wert aus dem Aufzählungstyp

typedef enum { preorder, postorder, endorder, leaf } VISIT

(definiert in der Include-Datei <*search.h*>), je nachdem, ob dies das erste, zweite oder dritte Mal ist, daß dieser Knoten aufgesucht wird (während einer von-links-nach-rechts-Abarbeitung bei der zuerst in der Tiefe gesucht wird), oder ob der Knoten ein Blatt ist. Das dritte Argument ist die Ebene des Knotens, wobei die Wurzel der Ebene 0 entspricht.

ERGEBNIS

Wenn der Eintrag gefunden wird liefert sowohl die Funktion *tsearch()* als auch die Funktion *tfind()* einen Zeiger auf diesen Eintrag. Wenn nicht, dann liefert *tfind()* den Nullzeiger und die Funktion *tsearch()* einen Zeiger auf den eingefügten Knoten.

Die Funktion *tsearch()* liefert den Nullzeiger, wenn nicht genügend Speicherplatz verfügbar ist, um den neuen Knoten zu erzeugen.

Die Funktionen *tsearch()*, *tfind()* und *tdelete()* liefern den Nullzeiger, wenn *rootp* gleich NULL ist.

Die Funktion *tdelete()* liefert einen Zeiger auf den Vaterknoten des gelöschten Knotens, bzw. den Nullzeiger, wenn der Knoten

nicht gefunden wird.

Die Funktion *twalk()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Das folgende Programmstück liest Zeichenketten ein und speichert Strukturen ab, die einen Zeiger auf jede Zeichenkette und deren Länge enthalten. Danach arbeitet es den Baum ab und gibt die gespeicherten Zeichenketten zusammen mit deren Länge in alphabetischer Reihenfolge wieder aus.

```
#include <search.h>
#include <stdio.h>

#define STRSZ               10000
#define NODSZ               500

struct node {       /* Zeiger auf diese Strukturen werden im
                       Baum gespeichert */
   char     *string;
   int      length;
};

char string_space[STRSZ];  /* Platz fuer Zeichenketten */
struct node nodes[NODSZ];  /* abzuspeichernde Knoten */
struct node *root = NULL;  /* Zeiger auf die Wurzel */

main( )
{
   char             *strptr = string_space;
   struct node      *nodeptr = nodes;
   void             print_node( ), twalk( );
   int              i = 0, node_compare( );

   while (gets(strptr) != NULL && i++ < NODSZ)
   {       /* Knoten besetzen */
           nodeptr->string = strptr;
           nodeptr->length = strlen(strptr);
           /* put node into the tree */
           (void) tsearch((void *)nodeptr, (void **)&root,
                     node_compare);
           /* Zeiger anpassen, so dass Baum nicht
              ueberschrieben wird */
           strptr += nodeptr->length + 1;
           nodeptr++;
   }
   /* Baum abarbeiten */
   twalk((void *)root, print_node);
}

/*
 * Diese Funktion vergleicht zwei Knoten gemaess ihrer
 * alphabetischen Ordnung
 */
```

```
int
node_compare(node1, node2)
char *node1, *node2;
{  return strcmp(((struct node *) node1)->string,
                 ((struct node *) node2)->string);
}

/*
 * Diese Funktion gibt einen Knoten dann aus, wenn twalk()
 * ihn zum zweiten Mal aufsucht oder wenn er ein Blatt ist
 */
void print_node(node, order, level)
struct node **node;
VISIT order;
int level;
{  if (order == postorder || order == leaf)
   {  (void)printf("string = %*s,  length = %d\n", STRSZ/NODSZ,
                   (*node)->string, (*node)->length);
   }
}
```

HINWEIS

Die Zeiger auf das Suchelement und die Wurzel des Baums sollten vom Typ *Zeiger auf Element* sein und in den Typ *(void *)* umgewandelt werden. Entsprechend sollte des Ergebnis, obwohl als Zeiger auf *char* deklariert, in den Typ *Zeiger auf Element* umgewandelt werden.

Das Argument *root* der Funktion *twalk()* hat eine Verweisstufe weniger, als die Argumente *rootp* der Funktionen *tsearch()* und *tdelete()*.

Es gibt zwei Nomenklaturen, die für dir Ordnung verwendet werden, in denen die Baumknoten durchlaufen werden. Die Funktion *tsearch()* verwendet *preorder*, um einen Knoten vor allen Knoten unterhalb diese Knotens, *postorder*, um einen Knoten nach dem linken aber vor dem rechten Unterknoten und *endorder*, um einen Knoten nach allen Unterknoten aufzusuchen. Die andere Nomenklatur vertauscht die Begriffe *inorder* und *postorder*, um dieselbe Reihenfolge zu beschreiben. Dies könnte zu Verwirrung über die Bedeutung der Begriffe führen.

Wenn die aufrufende Funktion den Zeiger auf die Wurzel ändert, dann ist das Ergebnis undefiniert.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

bsearch(), *hsearch()*, *lsearch()*, *<search.h>*.

NAME

ttyname - find pathname of a terminal
Dateiname einer Datensichtstation ermitteln

DEFINITION

```
char *ttyname (fildes)
int fildes;
```

BESCHREIBUNG

Die Funktion *ttyname()* liefert einen Zeiger auf eine Zeichenkette. Diese enthält den mit dem Nullbyte abschlossenen Pfadnamen der Datensichtstation, die der Dateikennzahl *fildes* zugeordnet ist. Das Ergebnis zeigt auf einen statischen Bereich, dessen Inhalt bei jedem Aufruf überschrieben wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ttyname()* einen Zeiger auf eine Zeichenkette. Andernfalls wird der Nullzeiger zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *ttyname()* schlägt fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige Dateikennzahl.

[ENOTTY] Das Argument *fildes* verweist nicht auf eine Datensichtstation.

PORTABILITÄT

Die Funktion *ttyname()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Folgendes Programm gibt den Namen der Datensichtstation aus, die dem Prozeß zugeordnet ist, überprüft, ob Standardeingabe und Fehlerausgabe auf den Bildschirm gelegt sind und gibt an, in welcher Zeile der Datei */etc/ttys* die Datensichtstation des Prozesses eingetragen ist:

```
#include <stdio.h>

char *ttyname();

main()
{
   printf("Dateiname fuer Standardausgabe : %s\n", ttyname(1));

   if(isatty(0) == 1)
      printf("stdin == Bildschirm\n");

   if(isatty(2) == 1)
      printf("stderr == Bildschirm\n");

   printf("Zeilen in /etc/ttys : %d\n", ttyslot());
}
```

SIEHE AUCH

isatty(), *ttyslot()*.

NAME

ttyslot - find the slot in the utmp file of the current user
Eintrag für aktuellen Benutzer in utmp-Datei suchen

DEFINITION

```
int ttyslot ( )
```

BESCHREIBUNG

Die Funktion *ttyslot* liefert den Indexeintrag für den aktuellen Benutzer in der Datei */etc/utmp*.

ERGEBNIS

Falls ein Fehler auftritt während nach dem Datensichtstationsnamen gesucht wird, oder falls keine der Dateikennzahlen (0,1 oder2) einer Gerätedatei für eine Datensichtstation zugeordnet ist, so wird der Wert 0 zurückgeliefert.

PORTABILITÄT

Die Funktion *ttyslot()* ist im X/Open-Standard nicht mehr enthalten.

BEISPIEL

Folgendes Programm gibt den Namen der Datensichtstation aus, die dem Prozeß zugeordnet ist, überprüft, ob Standardeingabe und Fehlerausgabe auf den Bildschirm gelegt sind und gibt an, in welcher Zeile der Datei */etc/ttys* die Datensichtstation des Prozesses eingetragen ist:

```
#include <stdio.h>

main()
{  char *ttyname();

   printf("Datename fuer Standardausgabe : %s\n", ttyname(1));

   if(isatty(0) == 1)
      printf("Bildschirm = Std-Eingabe\n");

   if(isatty(2) == 1)
      printf("Bildschirm = Std-Fehlerausgabe\n");

   printf("Zeilen in /etc/ttys : %d\n", ttyslot());
}
```

Wenn Sie die Standardausgabe umlenken (z.B. exec 1>aus) bevor Sie obiges Programm aufrufen, liefert ttyname(1) den Nullzeiger und die Ausgabe wird in die Datei *aus* geschrieben.

SIEHE AUCH

getut(), *ttyname()*.

NAME

twalk - traverse binary search tree
Binären Suchbaum durchlaufen

DEFINITION

```
#include <search.h>

void twalk (root, action)
char *root;
void (*action)();
```

BESCHREIBUNG

Diese Funktion arbeitet einen binären Suchbaum ab. Siehe unter *tsearch()*.

PORTABILITÄT

Die Funktion *twalk()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

tzname - timezone strings
Zeitzonen-Namen

DEFINITION

```
extern char *tzname[];
```

BESCHREIBUNG

Diese Variable beschreibt die Namen der aktuellen Zeitzone. Siehe unter *tzset()*.

PORTABILITÄT

Die Variable *tzname* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

tzset - set time zone conversion information
Umwandlungsinformation für Zeitzone setzen

DEFINITION

```
#include <time.h>

void tzset()

extern char *tzname[];

extern long timezone;

extern int daylight;
```

BESCHREIBUNG

Die Funktion *tzset()* verwendet den Wert der Umgebungsvariablen *TZ* um die Umwandlungsinformation zu setzen, die von *localtime()*, *ctime()*, *strftime()* und *mktime()* verwendet wird. Wenn *TZ* nicht existiert, dann wird eine implementierungs-abhängige Zeitzoneninformation verwendet.

Die Funktion *tzset()* besetzt die externe Variable *tzname* wie folgt:

tzname[0] = "*std*"; tzname[1] = "*dst*";
wobei *std* und *dst* jeweils drei oder mehr Bytes sind, die die Zeitzonen-Bezeichnungen für die Standard- ("*std*") oder Sommerzeit ("*dst*") darstellen. Diese Werte erhält die Variable *tzmane[]* aus der Umgebungsvariablen TZ, in der jedoch nur *std* angegeben sein muß.

Die Funktion *tzset()* besetzt auch die externe Variable *daylight* mit dem Wert 0, wenn eine Sommerzeit-Umwandlung für die verwendete Zeitzone niemals vorgenommen werden soll; andernfalls ist dieser Wert ungleich 0. Die externe Variable *timezone* wird gleich der Differnz in Sekunden zwischen der Coordinated Universal Time (UTC) und der lokalen Standardzeit gesetzt. Beispiele:

TZ	timezone
EST	5*60*60
GMT	0*60*60
JST	-9*60*60
MET	-1*60*60
MST	7*60*60
PST	8*60*60

ERGEBNIS

Die Funktion *tzset()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *tzset()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

ctime(), *localtime()*, *mktime()*, *strfttime()*, *<time.h>*.

NAME

ulimit - get und set process limits
Prozeßgrenzen setzen und ermitteln

DEFINITION

```
#include <ulimit.h>

long ulimit (cmd, ...)
int cmd;
```

BESCHREIBUNG

Die Funktion *ulimit()* bietet die Kontrolle über Prozeßgrenzen. Die Werte für *cmd*, die in *<ulimit.h>* definiert sind, beinhalten:

UL_GETFSIZE
: Liefert die Grenze für Dateigrößen des Prozesses. Die Grenze wird in 512-Byte-Blöcken angegeben und an Sohnprozesse vererbt. Dateien jeder Größe können gelesen werden.

UL_SETFSIZE
: Setzt die Grenze für die Dateigröße bei Ausgabeoperationen des Prozesses auf den Wert des zweiten Arguments, das als *long* interpretiert wird. Jeder Prozeß kann seine eigene Grenze heruntersetzen, aber nur ein Prozeß mit besonderen Rechten darf diese Grenze erhöhen. Das Ergebnis ist die neue Grenze für die Dateigröße.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ulimit()* den Wert der geforderten Grenze. Andernfalls wird der Wert -1 zurückgeliefert und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *ulimit()* schlägt fehl und die Grenze wird nicht verändert, wenn gilt:

[EINVAL] Das Argument *cmd* ist ungültig.

[EPERM] Ein Prozeß ohne besonderen Rechte versucht, die Grenze für die Dateigröße heraufzusetzen.

HINWEIS

Da bei Erfolg alle Ergebnisse erlaubt sind, sollte eine Anwendung, die Fehlersituationen überprüfen will, *errno* vor dem Aufruf von *ulimit()* gleich 0 setzen. Wenn das Ergebnis nach der Rückkehr gleich -1 und *errno* gesetzt ist, dann ist ein Fehler aufgetreten.

Der Wert 3 für *cmd*, der in der Version V5.21 für die Ermittlung des aktuellen brk-Werts verwendet wurde, wird auch von der Version V5.22 noch unterstützt. portable Anwendungen sollten diesen jedoch nicht mehr nutzen.

PORTABILITÄT

Die Funktion *ulimit()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Im Beispiel wird zunächst die maximale Dateigröße des Prozesses abgefragt, danach auf den Wert 10000 verringert. Der Versuch sie dann wieder auf den Wert 4000000 zu erhöhen scheitert, sofern die effektive Benutzernummer nicht die des Systemverwalters ist.

```
#include <ulimit.h>;
extern int errno;

main()
{
    long erg;

    if ((erg = ulimit (UL_GETFSIZE, 0L)) == -1
        perror ("ulimit 1");
    else
        printf ("Ergebnis 1 = %d\n", erg);

    if ((erg = ulimit (UL_SETFSIZE, 10000L)) == -1)
        perror ("ulimit 2");
    else
        printf ("Ergebnis 2 = %d\n", erg);

    if ((erg = ulimit (UL_GETFSIZE, 0L)) == -1
        perror ("ulimit 1");
    else
        printf ("Ergebnis 1 = %d\n", erg);

    if ((erg = ulimit (UL_SETFSIZE, 4000000L)) == -1)
        perror ("ulimit 2");
    else
        printf ("Ergebnis 2 = %d\n", erg);
}
```

SIEHE AUCH

write(), *<ulimit.h>*.

NAME

umask - set und get file mode creation mask
Schutzbitmaske setzen und ermitteln

DEFINITION

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

mode_t umask (cmask)
mode_t cmask;
```

BESCHREIBUNG

Die Funktion *umask()* setzt die Schutzbitmaske des Prozesses gleich *cmask* und liefert den bisherigen Wert der Maske. Nur die Schutzbits von *cmask* (siehe auch *<sys/stat.h>*) werden verwendet; die Bedeutung anderer Bits ist nicht festgelegt.

Die Schutzbitmaske des Prozesses wird von den Funktionen *open()*, *creat()*, *mkdir()* und *mkfifo()* verwendet, um Zugriffrechte im Argument *mode* zu entfernen. Bitpositionen, die in *cmask* gesetzt sind, werden bei den Zugriffsrechten der erzeugten Datei entfernt.

ERGEBNIS

Der bisherige Wert der Schutzbitmaske wird zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *umask()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das Beispielprogramm löscht eine eventuell vorhandene Datei *neu* ohne Rückfrage (!). Danach wird die Schutzbitmaske geändert, ihr alter Wert ausgegeben und eine Datei *neu* erstellt. Die neue Datei erhält von den geforderten Rechten (*rwx* für Eigentümer, Gruppe und Andere) nur die, die von der Schutzbitmaske nicht verboten sind. Der Wert 223 in der Schutzbitmaske verbietet für den Eigentümer und die Gruppe das Schreiben (s.o.), für alle Anderen Schreiben und ausführen. Als nächster Schritt werden die Zugriffsrechte ausgegeben und die Datei gelöscht. Dann wird die Schutzbitmaske erneut verändert, ihr letzter Wert ausgegeben und die Datei *neu* noch einmal angelegt.

```
main()
{
   int dk, i;

   system("rm -f neu");
   i = umask(0223);
   printf("i = %o\n", i);

   dk = creat("neu",0777);
   system("ls -l neu");
   system("rm -f neu");

   i = umask(0444);
   printf("i = %o\n", i);
   dk = creat("neu",0666);
   system("ls -l neu");
}
```

SIEHE AUCH

creat(), *mkdir()*, *mkfifo()*, *open()*, *<sys/stat.h>*, *<sys/types.h>*.

NAME

umount - unmount a file system
Dateisystem aushängen

DEFINITION

```
int umount (ger)
char *ger;
```

BESCHREIBUNG

Mit *umount()* wird ein zuvor mit *mount()* eingehängtes Dateisystem, das sich auf dem mit *ger* angegebenen blockorientierten Gerät befindet, wieder ausgehängt. *ger* zeigt auf einen Pfadnamen. Nach dem Aushängen des Dateisystems ist der Normalzustand des Dateiverzeichnisses, in das das Dateisystem eingehängt worden war, wieder hergestellt.

umount() darf nur unter der effektiven Benutzernummer eines Prozesses mit besonderen Rechten aufgerufen werden.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird −1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

umount() kann nicht ausgeführt werden, wenn einer oder mehrere der folgenden Fehler auftreten:

[EBUSY]	Eine Datei auf *ger* ist noch nicht frei.
[EFAULT]	*ger* verweist auf eine ungültige Adresse.
[EINVAL]	*ger* wurde nicht eingehängt (s. *mount()*).
[ENOENT]	Die angegebene Datei ist unbekannt.
[ENOTBLK]	*ger* ist kein blockorientiertes Gerät.
[ENOTDIR]	Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.
[ENXIO]	Das mit *ger* angegebene Gerät ist unbekannt.
[EPERM]	Die effektive Benutzernummer des Prozesses ist nicht die eines Prozesses mit besonderen Rechten.

HINWEIS

Statt mit der Funktion *mount()* ein Dateisystem einzuhängen, sollten Sie das Kommando */etc/mount* benutzen, wie in *SINIX Systemverwaltung [4]* beschrieben. Wenn ein Dateisystem eingehängt wird, müssen Systemtabellen entsprechend korrigiert werden; das Kommando *mount()* berücksichtigt dies automatisch.

Die Kommandos */etc/mount* und */etc/umount* tragen alle aktuell angehängten Dateisysteme in der Datei */etc/mtab* ein. Wenn Sie das Kommando */etc/mount* ohne Argument aufrufen, wird der Inhalt dieser Datei ausgegeben. Diese Datei hat rein informativen Charakter und wird von *mount()* und *umount()* nicht verwendet. D.h. ein Kommando */etc/mount* trägt ein eingehängtes Dateisystem in der *mtab* Datei ein, wird das eingehängte Dateisystem durch die Funktion *umount()* wieder abgehängt, bleibt der Eintrag in der Datei *mtab* bestehen, obwohl das Dateisystem nicht (mehr) eingehängt ist. Umgekehrt erscheint ein mittels der Funktion *mount()* eingehängtes Dateisystem nicht in der Datei *mtab*, das Kommando */etc/mount* scheitert.

PORTABILITÄT

Die Funktion *umount()* ist im X/Open-Standard nicht mehr enthalten.

SIEHE AUCH

mount().

NAME

uname - get name of current system
Systemnamen ermitteln

DEFINITION

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

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

BESCHREIBUNG

Die Funktion *uname()* speichert Informationen, die das aktuelle System identifizieren, in der Struktur ab, auf die *name* zeigt.

Die Funktion *uname()* verwendet die Struktur *utsname*, die in *<sys/utsname.h>* definiert ist.

Die Funktion *uname()* trägt eine, mit dem Nullbyte abgeschlossene, Zeichenkette, die den Namen des aktuellen Systems enthält, in den *char*-Vektor *sysname* ein. Analog dazu enthält *nodename* den Namen, unter dem das System in einem Kommunikationsnetz bekannt ist. Die Vektoren *release* und *version* identifizieren das Betriebssystem, der Vektor *machine* enthält einen Namen, der die Hardware kennzeichnet, auf der das System abläuft.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion einen nichtnegativen Wert.

FEHLER

Es sind im XPG3 [6] keine Fehler definiert.

Unter SINIX schlägt diese Funktion fehl, wenn gilt:

[EFAULT] Das Argument *name* ist eine ungültige Adresse.

HINWEIS

Die Aufnahme der Komponente *nodename* in diese Struktur besagt nicht, daß dies genügend Information ist, um Kommunikationsnetze anzusprechen.

In dieser Version hat sich die Bedeutung der Komponenten *version* und *release* geändert. Früher gab *version* die Release-Nr. an (z.B. V5.21), heute liefert sie die aktuelle Version (z.B. B70). Die Komponente *release* lieferte früher das Datum der Version, heute liefert sie die Release-Nr. (z.B. V5.22).

PORTABILITÄT

Die Funktion *uname()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Informationen über Ihr SINIX-System ausgeben:

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

main()
{
   int h;
   struct utsname name;

   if (( h = uname(&name) >= 0)
   {
      printf("Systemname : %s\n", name.sysname);
      printf("Nodename   : %s\n", name.nodename);
      printf("Release    : %s\n", name.release);
      printf("Version    : %s\n", name.version);
      printf("Machine    : %s\n", name.machine);
   }
   else
      printf("Fehler\n");
}
```

SIEHE AUCH

<sys/utsname.h>.

NAME

undial - release outgoing terminal line connection
Kommunikationsleitung freigeben

DEFINITION

```
#include <dial.h>

void undial(fd)
int fd;
```

BESCHREIBUNG

Die Funktion *undial()* gibt eine Kommunikationsleitung zu einer externen Datensichtstation wieder frei, die mit der Funktion *dial()* eingerichtet wurde.

Das Argument *fd* ist eine Dateikennzahl, die von einem vorhergehenden Aufruf von *dial()* geliefert worden sein muß.

ERGEBNIS

Die Funktion *undial()* hat kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Die Funktion *undial()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

SIEHE AUCH

dial(), *<dial.h>*.

NAME

ungetc - push character back into input stream
Zeichen in Datenstrom zurückstellen

DEFINITION

```
#include <stdio.h>

int ungetc (c, stream)
int c;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *ungetc()* stellt das durch *c* angegebene Zeichen (umgewandelt in den Typ *unsigned char*), in den Datenstrom zurück, auf den *stream* zeigt. Die zurückgestellten Zeichen werden von nachfolgenden Leseoperationen auf diesem Datenstrom zurückgeliefert, und zwar in der umgekehrten Reihenfolge ihrer Zurückstellung. Ein zwischendurch erfolgender, erfolgreicher Aufruf einer Funktion zur Positionierung (*fseek()* oder *rewind()*) für denselben Datenstrom löscht die zurückgestellten Zeichen aus dem Datenstrom. Der externe Speicher, der dem Datenstrom zugeordnet ist, bleibt unverändert.

Das Zurückstellen eines Zeichens ist garantiert. Wenn die Funktion *ungetc()* zu oft für denselben Datenstrom aufgerufen wird, ohne daß zwischendurch eine Leseoperation oder ein Positionieren stattfindet, dann kann die Operation fehlschlagen.

Wenn der Wert von *c* gleich dem Makro EOF ist, dann schlägt die Operation fehl und der Eingabestrom bleibt unverändert.

Ein erfolgreicher Aufruf der Funktion *ungetc()* löscht das Dateiende-Kennzeichen für diesen Datenstrom. Der Wert des Zeigers auf die Dateiposition für den Datenstrom ist nach einem Lesen oder Verwerfen aller zurückgestellter Zeichen derselbe wie vor dem Zurückstellen der Zeichen. Das zurückstellen eines Zeichens ist garantiert, vorausgesetzt, daß bereits etwas von dem Datenstrom gelesen wurde, und daß der Datenstrom wirklich gepuffert ist. Wenn *stream* gleich *stdin* und gepuffert ist, dann kann ein Zeichen in den Puffer zurückgestellt werden, ohne daß vorher gelesen werden muß. Für einen Datenstrom ist der Wert seines Zeigers auf die Dateiposition nach einem Aufruf von *ungetc()* solange unbestimmt, bis alle zurückgestellten Zeichen gelesen oder verworfen wurden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ungetc()* das zurückgestellte Zeichen nach der Umwandlung. Andernfalls liefert sie EOF.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Zurückstellen ohne vorherige Leseoperation wird auch im Fall von *stdin* nicht empfohlen.

PORTABILITÄT

Die Funktion *ungetc()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

```
#include <stdio.h>

main()
{  int c, count = 0;

   /* Es wird ein Zeichen von stdin eingelesen und dann wiederholt
      zurueckgestellt. Wieviele Zeichen insgesamt
      zurueckgestellt werden, wird jeweils ausgegeben. */
   c=getchar();
   while (ungetc(c,stdin) != EOF)
   {  count++;
      printf("%d. Zeichen zurueckgestellt!\n", count);
   }
}
```

SIEHE AUCH

fseek(), *getc()*, *rewind()*, *setbuf()*, *<stdio.h>*.

NAME

universe - get or change process universe
Prozeßuniversum wechseln

DEFINITION

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

int universe(uflag)
int uflag;
```

BESCHREIBUNG

Mit *universe()* können Sie feststellen, in welchem Universum Sie sich befinden oder Sie können in ein anderes Universum wechseln.

Für den Parameter *uflag* sind vier Werte möglich:

U_GET Mit U_GET können Sie das Universum abfragen, in dem Sie sich befinden. Das aktuelle Universum bleibt unverändert.

U_ATT Dieser Wert wechselt in das att- (xopen-)Universum.

U_SIE Dieser Wert wechselt in das sie-Universum.

U_UCB Dieser Wert wechselt in das ucb-Universum.

ERGEBNIS

Bei Erfolg wird der Wert des gegenwärtigen Universums zurückgeliefert; dieser Wert kann U_ATT, U_SIE bzw. U_UCB sein. Andernfalls wird −1 zurückgegeben und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *universe()* schlägt fehl, wenn gilt:

[EINVAL] *uflag* ist keiner der erlaubten Werte.

PORTABILITÄT

Die Funktion *universe()* ist im X/OPEN-Standard nicht enthalten.

SIEHE AUCH

<sys/universe.h>.

NAME

unlink - remove directory entry
Verweis entfernen

DEFINITION

```
int unlink (path)
char *path;
```

BESCHREIBUNG

Die Funktion *unlink()* entfernt den Verweis, der über den Pfadnamen *path* angegeben wird und dekrementiert den Verweiszähler für diese Datei.

Wird der Verweiszähler der Datei gleich 0 und hat kein Prozeß die Datei offen, dann wird der Platz freigegeben, den die Datei belegt. Auf die Datei kann dann nicht länger zugegriffen werden.

Wenn ein Prozeß die Datei geöffnet hat, dann wird die Datei erst dann entfernt, wenn kein Prozeß diese Datei mehr offen hat. Der Verweis auf die Datei dagegen wird sofort entfernt.

Das Argument *path* sollte kein Dateiverzeichnis bezeichnen, sofern der Prozeß keine besonderen Rechte hat und das System die Verwendung von *unlink()* für Dateiverzeichnisse unterstützt. Anwendungen sollten *rmdir()* verwenden, um Dateiverzeichnisse zu entfernen

Bei erfolgreicher Beendigung markiert die Funktion *unlink()* die Felder *st_ctime* und *st_mtime* des übergeordneten Dateiverzeichnisses.

Ebenso wird, wenn der Verweiszähler der Datei nicht gleich 0 ist, das Feld *st_ctime* der Datei zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert −1 zurückgeliefert, die angegebene Datei wird nicht verändert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *unlink()* schlägt fehl, wenn gilt:

[EACCES] Für eine Komponente des Pfadnamen-Anfangs ist kein Suchrecht vorhanden oder das Schreibrecht für das übergeordnete Dateiverzeichnis des zu löschenden Verweises ist nicht vorhanden.

[EBUSY] Der zu entfernende Verweis wird derzeit vom System oder einem anderen Prozeß benutzt und die Implementierung nimmt dies als einen Fehler an.

[ENAMETOOLONG] Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Das Argument *path* bezeichnet eine nichtexistierende Datei oder zeigt auf eine leere Zeichenkette.

[ENOTDIR] Eine Komponente des Pfads ist kein Dateiverzeichnis.

[EPERM] Die durch *path* angegebene Datei ist ein Dateiverzeichnis und der aufrufende Prozeß hat entweder keine besonderen Rechte oder die Implementierung erlaubt die Verwendung von *unlink()* für Dateiverzeichnisse nicht.

[EROFS] Der zu löschende Dateiverzeichniseintrag befindet sich in einem nur zum Lesen eingehängten Dateisystem.

Die Funktion *unlink()* kann fehlschlagen, wenn gilt:

[ETXTBSY] Der zu entfernende Eintrag ist der letzte Verweis auf eine reine Prozedurdatei (gemeinsamer Text), die ausgeführt wird.

Unter SINIX V5.22 kann zusätzlich zu den im XPG3 [6] definierten Fehlern auch noch der folgende Fehler auftreten:

[EFAULT] Es wurde eine ungültige Adresse als Argument übergeben.

PORTABILITÄT

Die Funktion *unlink()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

close(), *link()*, *remove()*, *rmdir()*.

NAME

ustat - get file system information
Dateisysteminformationen ermitteln

DEFINITION

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

int ustat (ger, puf)
int ger;
struct ustat *puf;
```

BESCHREIBUNG

Die Funktion *ustat()* liefert Informationen über ein mit der Funktion *mount()* eingehängtes Dateisystem. *ger* ist die Gerätenummer eines Gerätes, auf dem sich das eingehängte Dateisystem befindet. *puf* ist ein Zeiger auf eine *ustat*-Struktur mit folgenden Komponenten:

```
daddr_t  f_tfree;    /* freie Bloecke insgesamt */
ino_t    f_tinode;   /* Anzahl freier Indexeintraege */
char     f_fname[6]; /* Name des Dateisystems
                        oder NULL */
char     f_fpack[6]; /* Name des Dateisystemstapels
                        oder NULL */
```

Die letzten beiden Komponenten, *f_fname* und *f_fpack*, können das Nullzeichen enthalten.

ERGEBNIS

Bei erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. In allen anderen Fällen wird −1 geliefert und der Fehler in *errno* angezeigt.

FEHLER

Die Funktion *ustat()* schlägt fehl, wenn gilt:

[EFAULT]
puf zeigt auf eine Struktur außerhalb des diesem Prozeß zugewiesenen Adreßraums.

[EINVAL]
ger ist keine Gerätenummer eines Geräts, auf dem sich ein mit *mount()* eingehängtes Dateisystem befindet.

HINWEIS

Die Funktion *ustat()* wird in dieser Version von der Funktion *statfs()* abgelöst. Neuere Anwendungen sollten daher *statfs()* anstelle von *ustat()* verwenden.

PORTABILITÄT

Die Funktion *ustat()* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

stat(), *statfs()*, *<sys/types.h>*, *<ustat.h>*.

NAME

utime - set file access und modification times
Dateizeiten setzen

DEFINITION

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

int utime (path, times)
char *path;
struct utimbuf *times;
```

BESCHREIBUNG

Die Funktion *utime()* setzt die Zugriffs- und Modifikationszeit der Datei, die durch das Argument *path* angegeben wird.

Wenn *times* der Nullzeiger ist, dann werden die Zugriffsund die Modifikationszeit der Datei auf die aktuelle Zeit gesetzt. Die effektive Benutzernummer des Prozesses muß mit der des Eigentümers der Datei übereinstimmen, oder der Prozeß muß für die Datei Schreiberlaubnis oder besondere Rechte haben, damit die Funktion *utime()* auf diese Weise genutzt werden kann.

Wenn *times* nicht der Nullzeiger ist, dann wird *times* als Zeiger auf eine Struktur *utimbuf* interpretiert und die Zugriffs- und die Modifikationszeit wird gemäß den Werten in dieser Struktur gesetzt. Nur ein Prozeß, dessen effektive Benutzernummer mit der des Eigentümers der Datei übereinstimmt oder ein Prozeß mit besonderen Rechten kann *utime()* auf diese Art nutzen.

Die Struktur *utimbuf* wird in der Include-Datei *<utime.h>* definiert. Die Zeiten in der Struktur *utimbuf* werden alle in Sekunden seit dem Epochenwert angegeben.

Bei erfolgreicher Beendigung markiert die Funktion *utime()* die Zeit der letzten Zustandsänderung der Datei, *st_ctime*, zum Ändern, siehe *<sys/stat.h>*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert, *errno* wird gesetzt, um den Fehler anzuzeigen und die Dateizeiten werden nicht geändert.

FEHLER

Die Funktion *utime()* schlägt fehl, wenn gilt:

[EACCES] Die Durchsucherlaubnis für eine Komponente des Pfadnamen-Anfangs wird verweigert; oder das Argument *times* ist der Nullzeiger und die effektive Benutzernummer des Prozesses entspricht nicht der des Eigentümers der Datei und die Schreiberlaubnis wird verwehrt.

[ENAMETOOLONG] Die Länge des Arguments *path* überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX}, während {_POSIX_NO_TRUNC} aktiv ist.

[ENOENT] Die genannte Datei existiert nicht oder das Argument *path* zeigt auf eine leere Zeichenkette.

[ENOTDIR] Eine Komponente des Pfadnamen-Anfangs ist kein Dateiverzeichnis.

[EPERM] Das Argument *times* ist nicht der Nullzeiger und die effektive Benutzernummer des aufrufenden Prozesses darf auf die Datei schreiben, entspricht aber nicht der des Eigentümers der Datei und der aufrufende Prozeß besitzt keine besonderen Rechte.

[EROFS] Die angegebene Datei befindet sich auf einem, nur zum Lesen eingehängten Dateisystem.

Unter SINIX V5.22 kann zusätzlich zu den im XPG3 [6] definierten Fehlern auch noch der folgende Fehler auftreten:

[EFAULT] Es wurde eine ungültige Adresse als Argument übergeben.

PORTABILITÄT

Die Funktion *utime()* ist im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Beispiel setzt die Zugriffs- und Modifikationszeit der Datei *datei* auf die (frei gewählten) Werte 100 bzw. 200 Sekunden seit dem Epochenwert. Danach werden die Dateizeiten mit der Funktion *stat()* ermittelt und als Zahlwerte und als Zeichenkette (*ctime()*) ausgegeben:

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

main()
{  struct utimebuf timebuf;
   struct stat dinf;
   char *ctime();

   timebuf.asctime = (time_t) 100;
   timebuf.modtime = (time_t) 200;

   utime("datei", &timebuf);
   stat("datei", &dinf);

   printf("%ld : %ld : %ld\n", dinf.st_atime,
          dinf.st_mtime, dinf.st_ctime);
   printf("%s",ctime(&dinf.st_ctime));
}
```

SIEHE AUCH

<sys/stat.h>, *<sys/types.h>*, *<utime.h>*.

NAME

utmpname - change utmp file name
Neue utmp-Datei bestimmen

DEFINITION

```
#include <utmp.h>

void utmpname(datei)
char *datei;
```

BESCHREIBUNG

Die Funktion *utmpname()* gehört zu einer Gruppe von Funktionen, die Einträge in der Datei */etc/utmp* bearbeiten. Sie initialisiert diese Funktionsgruppe für eine neue *utmp*-Datei. Eine ausführliche Beschreibung finden Sie unter *getutent()*.

PORTABILITÄT

Die Funktion *utmpname()* ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

NAME

varargs - handle variable argument list
Variable Argumentliste behandeln

DEFINITION

```
#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;
```

BESCHREIBUNG

Die Makros der Include-Datei *<varargs.h>* erlauben es, portable Prozeduren mit variablen Argumentlisten zu schreiben.

Eine ausführliche Beschreibung dieser Makros finden Sie unter *<varargs.h>*.

PORTABILITÄT

Die Include-Datei *<varargs.h>* und die dort definierten Makros sind im X/Open-Standard (Ausgabe 3) definiert.

NAME

vfprintf, vprintf, vsprintf
- format output of a varargs argument list
Formatierte Ausgabe einer Argumentliste

DEFINITION

```
#include <stdio.h>
#include <varargs.h>

int vprintf (format, ap)
char *format;
va_list ap;

int vfprintf (stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf (s, format, ap)
char *s, *format;
va_list ap;
```

BESCHREIBUNG

Die Funktionen *vprintf()*, *vfprintf()* und *vsprintf()* entsprechen jeweils den Funktionen *printf()*, *fprintf()* und *sprintf()*, außer daß sie mit eine Argumentliste, wie sie in *<varargs.h>* definiert ist, aufgerufen werden, anstelle mit einer variablen Anzahl von Argumenten.

ERGEBNIS und FEHLER

Siehe unter *printf()*.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

BEISPIEL

Das folgende Beispiel zeigt, wie man unter Verwendung von *vfprintf* eine Fehlerroutine schreiben kann.

```
#include <stdio.h>
#include <varargs.h>
        .
        .
        .
/*
 *   error aufrufen mit
 *           error(funktionsname, format, arg1, arg2...);
 */
/*VARARGS0*/
void
error(va_alist)
/* Merke: die Argumente funktionsname und format koennen wg. der
 *        varargs-Definition nicht gesondert vereinbart werden.
 */
va_dcl
{
        va_list args;
        char *fmt;
        va_start(args);
        /* Name der fehlerverursachenden Funktion ausgeben */
        (void) fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
        fmt = va_arg(args, char *);
        /* Rest der Meldung ausgeben */
        (void) vfprintf(stderr,fmt, args);
        va_end(args);
        (void) abort( );
}
```

SIEHE AUCH

printf(), *<stdio.h>*, *<varargs.h>*.

NAME

wait, waitpid - wait for child process to stop oder terminate
Ende eines Sohnprozesses abwarten

DEFINITION

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

pid_t wait (stat_loc)
int *stat_loc;

pid_t waitpid (pid, stat_loc, options)
pid_t pid;
int *stat_loc, options;
```

BESCHREIBUNG

Die Funktionen *wait()* und *waitpid()* erlauben es dem aufrufenden Prozeß Zustandsinformationen zu einem seiner Sohnprozesse zu erlangen. Verschiedene Optionen erlauben die Zustandsinformation für Sohnprozesse die beendet oder angehalten sind. Wenn die Zustandsinformation für zwei oder mehr Sohnprozesse verfügbar ist, dann ist die Reihenfolge, in der diese Informationen geliefert werden, unbestimmt.

Die Funktion *wait()* unterbricht die Ausführung des aufrufenden Prozesses, bis die Zustandsinformation für einen seiner beendeten Sohnprozesse verfügbar ist, oder bis zur Zustellung eines Signals, dessen Aktion entweder die Ausführung einer Signalbehandlungsfunktion oder das Beenden des Prozesses ist. Wenn die Zustandsinformation bereits vor dem Aufruf von *wait()* verfügbar ist, dann erfolgt eine sofortige Rückkehr.

Die Funktion *waitpid()* verhält sich identisch zur Funktion *wait()*, wenn das Argument *pid* den Wert -1 und das Argument *options* den Wert 0 hat. Andernfalls wird ihr Verhalten durch die Werte der Argumente *pid* und *options* verändert.

Das Argument *pid* gibt eine Menge von Sohnprozessen an, für die der Zustand ermittelt werden soll. Die Funktion *waitpid()* liefert nur den Zustand eines Sohnprozesses aus dieser Menge zurück:

— Wenn *pid* gleich -1 ist, dann wird der Zustand irgendeines Sohnprozesses abgefragt. In dieser Hinsicht ist *waitpid()* dann äquivalent zu *wait()*.

- Wenn *pid* größer als 0 ist, dann gibt dieses Argument die Prozeßnummer eines einzelnen Sohnprozesses an, für den der Zustand abgefragt wird.
- Wenn *pid* gleich 0 ist, dann wird der Zustand für irgendeinen Sohnprozeß abgefragt, dessen Prozeßgruppennummer gleich der des aufrufenden Prozesses ist.
- Wenn *pid* kleiner als -1 ist, dann wird der Zustand irgendeines Sohnprozesses abgefragt, desses Prozeßgruppennnumer gleich der des Absolutbetrags von *pid* ist.

Das Argument *options* wird durch das bitweise Inklusiv-ODER mit beliebigen der folgenden beiden Kennzeichen gebildet, die in der Include-Datei *<sys/wait.h>* definiert sind.

WNOHANG
: Die Funktion *waitpid()* hält die Ausführung des aufrufenden Prozesses nicht an, wenn der Zustand für einen der durch *pid* angegebenen Sohnprozesse nicht unmittelbar verfügbar ist.

WUNTRACED
: Dieses Kennzeichen ist unter SINIX nicht verfügbar, kann aber für andere X/Open-kompatiblem Implementierungen die folgende Wirkung haben:
Wenn die Implementierung Auftragskontrolle unterstützt, dann wird der Zustand aller der durch *pid* angegebenen Prozesse, die angehalten sind und deren Zustand seit dem Anhalten noch nicht geliefert wurde, ebenfalls an den aufrufenden Prozeß gemeldet.

Wenn *wait()* oder *waitpid()* zurückkehren, weil der Zustand eines Sohnprozesses verfügbar ist, dann ist das Ergebnis dieser Funktionen gleich der Prozeßnummer des Sohnprozesses. In diesem Fall, wenn der Wert des Arguments *stat_loc* nicht der Nullzeiger ist, wird die Information an der Stelle abgelegt, auf die *stat_loc* zeigt. Genau dann, wenn der Zustand von einem beendeten Sohnprozeß geliefert wird, der den Wert 0 aus *main()* zurückgegeben oder der den Wert 0 als Argument an eine der Funktionen *_exit()* oder *exit()* übergeben hat, ist auch der Wert, der an der Adresse *stat_loc* abgelegt wird, gleich 0. Gleichgültig, welchen Wert diese Information hat, sie kann mit Hilfe der folgenden Makros interpretiert werden, die in *<sys/wait.h>* definiert sind, und die ganzzahlige Ausdrücke berechnen; das Argument *stat_val* ist ein ganzzahliger Wert, auf den *stat_loc* zeigt.

WIFEXITED(stat_val)
Berechnet einen Wert ungleich 0 (*wahr* in C), wenn der Zustand für einen Sohnprozeß geliefert wurde, der sich normal beendete.

WEXITSTATUS(stat_val)
Wenn der Wert von WIFEXITED(*stat_val*) ungleich 0 ist, dann berechnet dieses Makro die niederwertigen 8 Bit des Arguments *status*, das der Sohnprozeß an *_exit()* oder *exit()* übergeben hat, bzw. des Werts, den der Sohnprozeß aus *main()* zurückgegeben hat.

WIFSIGNALED(stat_val)
Berechnet einen Wert ungleich 0, wenn der Zustand für einen Sohnprozeß geliefert wurde, der sich durch den Empfang eines Signals beendete, das nicht abgefangen wurde (siehe auch *<signal.h>*).

WTERMSIG(stat_val)
Wenn der Wert von WIFSIGNALED(*stat_val*) ungleich 0 ist, dann berechnet dieses Makro die Signalnummer, die den Abbruch des Sohnprozesses verursacht hat.

WIFSTOPPED(stat_val)
Berechnet einen Wert ungleich 0, wenn der Zustand für einen Sohnprozeß geliefert wurde, der zur Zeit angehalten ist.

WSTOPSIG(stat_val)
Wenn der Wert von WIFSTOPPED(*stat_val*) ungleich 0 ist, dann berechnet dieses Makro die Nummer des Signals, das den Sohnprozeß angehalten hat.

Wenn die Information, die an der Stelle *stat_loc* abgelegt ist, dort von einem Aufruf von *waitpid()* abgelegt wurde, die das Kennzeichen WUNTRACED angegeben hatte, dann liefert genau eines der Makros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*) und WIFSTOPPED(**stat_loc*) einen Wert ungleich 0. Wenn die Information, die unter der Adresse *stat_loc* abgelegt ist, dort von einem Aufruf von *waitpid()* abgelegt wurde, bei dem das Kennzeichen WUNTRACED nicht angegeben wurde, oder von einem Aufruf der Funktion *wait()*, dann liefert genau eines der Makros WIFEXITED(**stat_loc*) und WIFSIGNALED(**stat_loc*) einen Wert ungleich 0.

Wenn sich ein Vaterprozeß beendet, ohne auf alle seine Sohnprozesse zu warten, dann wird den verbleibenden Sohnprozessen eine neue Vaterprozeßnummers zugeordnet, nämlich die des Systemprozesses *init*. Andere X/Open-

kompatiblen Implementierungen können dafür einen anderen Systemprozeß vorsehen.

ERGEBNIS

Wenn die Funktion *wait()* oder *waitpid()* zurückkehrt, weil der Zustand eines Sohnprozesses verfügbar ist, dann liefern diese Funktionen ein Ergebnis, das gleich der Prozeßnummer des Sohnprozesses ist, dessen Zustand geliefert wird. Wenn die Funktion *wait()* oder *waitpid()* aufgrund der Zustellung eines Signals zurückkehrt, dann wird der Wert −1 zurückgeliefert und *errno* ist gleich [EINTR] gesetzt. Wenn die Funktion *waitpid()* mit dem Kennzeichen WNOHANG im Argument *options* aufgerufen wurde und die Funktion mindestens einen Sohnprozeß durch *pid* angegeben hat, dann wird der Wert 0 zurückgeliefert. Anderfalls wird der Wert −1 geliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *wait()* schlägt fehl, wenn gilt:

[ECHILD] Der aufrufende Prozeß besitzt keine Sohnprozessse, auf die nicht gewartet wird.

[EINTR] Die Funktion wurde von einem Signal unterbrochen. Der Wert des Objekts, auf das *stat_loc* zeigt, ist dann undefiniert.

Die Funktion *waitpid()* schlägt fehl, wenn gilt:

[ECHILD] Der mit *pid* angegebene Prozeß oder die Prozeßgruppe existiert nicht, oder ist kein Sohnprozeß des aufrufenden Prozesses.

[EINTR] Die Funktion wurde von einem Signal unterbrochen. Der Wert des Objekts, auf das *stat_loc* zeigt, ist dann undefiniert.

[EINVAL] Das Argument *options* ist nicht gültig.

HINWEIS

Anstelle der in Version V5.21 definierten Bitpositionen für die Werte von *stat_val* sollten portable Anwendungen unbedingt die oben beschriebenen Makros verwenden.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

exec, *exit()*, *fork()*, *<sys/types.h>*, *<sys/wait.h>*.

NAME

write - write on a file
In Datei schreiben

DEFINITION

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

BESCHREIBUNG

Die Funktion *write()* versucht, *nbyte* Bytes aus dem Puffer, auf den *buf* zeigt, in die der Dateikennzahl *fildes* zugeordnete Datei zu schreiben.

Wenn *nbyte* gleich 0 ist, dann liefert die Funktion *write()* den Wert 0 und hat keine sonstigen Ergebnisse, wenn die Datei eine normale Datei ist; andernfalls sind die Ergebnisse implementierungs-abhängig.

In einer normalen Datei, oder in einer Datei, in der positioniert werden kann, findet das jeweilige Schreiben von Daten an der Stelle in der Datei statt, die durch die Dateiposition bestimmt ist. Wenn die dann erhöhte Dateiposition größer als die Länge der Datei ist, dann wird die Länge der Datei gleich dieser Position gesetzt.

Wenn das Bit O_SYNC im System-Dateistatus-Byte gesetzt ist, und wenn *fildes* auf eine normale Datei verweist, dann kehrt ein erfolgreicher Aufruf von *write()* nicht eher zurück, bis die Daten an die zugrundeliegende Hardware ausgeliefert sind.

Bei der erfolgreichen Rückkehr von *write()* wird die Dateiposition um die Anzahl der tatsächlich geschriebenen Bytes erhöht.

In einer Datei, in der nicht positioniert werden kann, findet das Schreiben immer an der aktuellen Position statt. Der Wert der Dateiposition, der einem solchen Gerät zugeordnet ist, ist undefiniert.

Wenn das Bit O_APPEND im System-Dateistatus-Byte gesetzt ist, dann wird die Dateiposition vor dem Schreiben auf das Dateiende gesetzt.

Wenn ein *write()* mehr Bytes schreiben will, als Platz zur Verfügung steht (z.B. wegen *ulimit* oder dem physikalischen Ende eines Mediums), dann werden nur soviele Bytes geschrieben, wie noch Platz haben. Angenommen, es ist noch Platz für 20 Bytes, bevor eine Grenze erreicht wird. Ein Schreiben von 512 Bytes

liefert dann den Wert 20. Die nächste Schreiboperation mit einer Zahl von Bytes ungleich 0 würde dann ein Fehlerergebnis liefern (außer in den unten beschriebenen Fällen).

Bei erfolgreicher Beendigung liefert die Funktion *write()* die Anzahl der tatsächlich in die *fildes* zugeordnete Datei geschriebenen bytes. Diese Anzahl ist niemals größer als *nbyte*.

Wenn ein *write()* von einem Signal unterbrochen wird, bevor es Daten schreibt, dann liefert die Funktion den Wert -1, wobei *errno* gleich [EINTR] gesetzt ist.

Wenn die Funktion *write()* nach dem erfolgreichen Schreiben von Daten durch ein Signal unterbrochen wird, dann liefert sie entweder den Wert -1 und *errno* ist gleich [EINTR] gesetzt, oder sie liefert die Anzahl der geschriebenen Bytes. Ein *write()* auf eine Pipe oder FIFO kehrt niemals mit *errno* gleich [EINTR] zurück, wenn es irgendwelche Daten übertragen hatte und *nbyte* kleiner oder gleich {PIPE_BUF} ist.

Wenn der Wert von *nbyte* größer als {INT_MAX} ist, dann ist das Ergebnis implementierungs-abhängig.

Schreibanforderungen für eine Pipe oder FIFO werden auf dieselbe Weise gehandhabt, wie solche für eine normale Datei, mit den folgenden Ausnahmen:

- Es gibt keine Dateiposition, die einer Pipe zugeordnet ist, da jede Schreibanforderung an das Ende der Datei angefügt wird.
- Schreibanforderungen von {PIPE_BUF} Bytes oder weniger werden nicht mit Daten anderer Prozesse, die auf dieselbe Datei schreiben, gemischt. Schreiboperationen von mehr als {PIPE_BUF} Bytes können , in bestimmten Grenzen, mit Schreiboperationen anderer Prozesse gemischt werden, gleichgültig, ob das Bit O_NONBLOCK im System-Dateistatus-Byte gesetzt ist oder nicht.
- Wenn das Bit O_NONBLOCK nicht gesetzt ist, dann kann eine Schreibanforderung den Prozeß blockieren, aber eine normale Beendigung liefert das Ergebniss *nbyte*.
- Wenn das Bit O_NONBLOCK gesetzt ist, dann werden Anforderungen der Funktion *write()* auf die folgenden Arten verschieden behandelt: die Funktion *write()* blockiert den Prozeß nicht; Schreibanforderungen für {PIPE_BUF} oder weniger Bytes werden entweder erfolgreich beendet und liefern dann *nbyte*, oder sie liefern den Wert -1 und setzen *errno* gleich [EAGAIN].

Eine Schreibanforderung mit mehr als {PIPE_BUF} Bytes überträgt entweder soviele Daten wie möglich und gibt die Anzahl der geschriebenen Bytes zurück, oder überträgt keine Daten liefert den Wert -1 und setzt *errno* gleich [EAGAIN]. Außerdem gilt: wenn eine Anforderung größer als {PIPE_BUF} Bytes ist und alle Daten die vorher in diese Datei geschrieben wurden, sind bereits gelesen, dann überträgt *write()* mindestens {PIPE_BUF} Bytes.

Wenn ein Versuch erfolgt, auf eine Dateikennzahl zu schreiben, die keine Pipe oder FIFO ist und nichtblockierendes Schreiben unterstützt:

- Wenn das Bit O_NONBLOCK nicht gesetzt ist, dann blockiert *write()* solange, bis die Daten akzeptiert werden können.
- Wenn das Bit O_NONBLOCK gesetzt ist dann blockiert *write()* den Prozeß nicht. Wenn einige Daten ohne ein Blockieren des Prozesses geschrieben werden können,dann schreibt *write()* soviel wie möglich und liefert die Anzahl der übertragenen Bytes. Andernfalls liefert die Funktion den Wert -1 und *errno* ist gleich [EAGAIN] gesetzt.

Bei erfolgreicher Beendigung markiert die Funktion *write()* die Felder *st_ctime* und *st_mtime* der Datei; die Bits S_ISUID und S_ISGID des Dateistatus werden gelöscht, wenn der Prozeß keine besonderen Rechte besitzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion die Anzahl der tatsächlich geschriebenen Bytes. Andernfalls wird der Wert -1 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktion *write()* schlägt fehl, wenn gilt:

[EAGAIN] Das Bit O_NONBLOCK ist für die Dateikennzahl gesetzt und der Prozeß würde durch die Schreiboperation angehalten werden.

[EBADF] *fildes* ist keine gültige, zum Schreiben geöffnete Dateikennzahl.

[EFBIG] Es wurde der Versuch unternommen, auf eine Datei zu schreiben, deren Größe die Grenze für die Dateigröße des Prozesses, bzw. die maximal mögliche Dateigröße überschreitet. Siehe auch *ulimit()*.

[EINTR] Die Schreiboperation wurde durch den Empfang eines Signals unterbrochen, und es wurden entweder keine Daten übertragen oder das System meldet eine teilweise Übertragung von Daten nicht.

[EIO] Die Implementierung unterstützt Auftragskontrolle, der Prozeß ist Mitglied einer Hintergrundprozeßgruppe und versucht, auf sein kontrollierendes Terminal zu schreiben. TOSTOP ist gesetzt, der Prozeß blockiert oder ignoriert das Signal SIGTTOU nicht und die Prozeßgruppe des Prozesses ist verwaist.

[ENOSPC] Auf dem Gerät, das die Datei enthält, war kein freier Platz mehr übrig.

[EPIPE] Es wurde der Versuch unternommen auf eine Pipe oder FIFO zu schreiben, die kein Prozeß zum Lesen geöffnet hat. Auch das Signal SIGPIPE wird an den Prozeß gesendet.

Die Funktion *write()* kann fehlschlagen, wenn gilt:

[ENXIO] Es wurde der Versuch unternommen, auf ein nicht existierendes Gerät zuzugreifen, oder die Anforderung lag außerhalb der Fähigkeiten des Geräts.

Unter SINIX tritt zusätzlich der folgende Fehler auf:

[EFAULT] Das Argument *buf* ist eine ungültige Adresse.

HINWEIS

Das Bit O_NDELAY wurde durch O_NONBLOCK ersetzt. Es wird empfohlen, in neuen Anwendungen nur noch O_NONBLOCK zu verwenden.

PORTABILITÄT

Die Funktion *write()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

creat(), *dup()*, *fcntl()*, *lseek()*, *open()*, *pipe()*, *ulimit()*.

NAME

y0, y1, yn - Bessel functions of the second kind
Besselfunktionen der zweiten Art

DEFINITION

```
#include <math.h>

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

BESCHREIBUNG

Die Funktionen *y0()*, *y1()* und *yn()* liefern Besselfunktionen der zweiten Art für *x* jeweils für die Ordnungen 0, 1, und *n*. Der Wert von *x* muß positiv sein.

ERGEBNIS

Bei Erfolg liefern *y0()*, *y1()* und *yn()* den relevanten Besselwert für *x* der zweiten Art.

Wenn das Argument *x* der Funktionen *y0()*, *y1()* oder *yn()* nicht positiv ist, dann wird -HUGE_VAL oder NaN zurückgeliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

Andernfalls wird entweder 0.0 zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen, oder ein NaN wird zurückgeliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktionen *y0()*, *y1()* und *yn()* schlagen fehl, wenn gilt:

[EDOM] Der Wert von *x* ist nicht positiv.

[ERANGE] Der Wert von *x* ist zu groß.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* vor dem Aufruf einer der Funktionen gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt oder das Ergebnis -HUGE_VAL oder ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben.

Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen die Mathematik-Bibliothek durch *-lm* dazubinden:

cc programm.c -lm

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

isnan(), *j0()*, *matherr()*, *<math.h>*.

4 Include-Dateien

Dieses Kapitel beschreibt die Inhalte der Include-Dateien, die von den Funktionen aus Kapitel 3 verwendet werden.

Include-Dateien enthalten die Definition symbolischer Konstanten, allgemeiner Strukturen, Präprozessor-Makros und definierter Datentypen. Jede Funktion aus Kapitel 3 gibt die Include-Dateien an, die eine Anwendung mit einbinden muß, um diese Funktion zu benutzen. Diese Include-Dateien werden nur für die Entwicklung von Anwendungen benötigt; bei der Ausführung müssen sie nicht vorhanden sein.

NAME

a.out - compiler and linker output
Ausgaben von Übersetzer und Binder

DEFINITION

```
#include <a.out.h>
```

BESCHREIBUNG

a.out ist die Ausgabedatei des Übersetzers *as*(vgl. Kommando *cc*) und des Binders *ld*. Wenn die Datei fehlerfrei ist und keine unbefriedigten Externbezüge mehr enthält, wird sie von den beiden Programmen für die Ausführung freigegeben.

```
/*
* Kopf am Anfang jeder a.out-Datei
*/
struct exec {
        long            a_magic;        /* "magische Zahl" */
        unsigned long a_text;           /* Textsegment-Groesse im Speicher */
        unsigned long a_data;           /* Segmentgroesse f. initial. Daten */
        unsigned long a_bss;            /* Segm.gr. f. nicht-init. Daten */
        unsigned long a_syms;           /* Groesse d. Symboltabelle */
        unsigned long a_entry;          /* Einsprungadresse */
        unsigned long a_trsize;         /* Groesse d. Textverschiebung */
        unsigned long a_drsize;         /* Groesse d. Datenverschiebung */
        struct modtbl {                 /* leerer Mod.-Tabelleneintrag */
                unsigned long m_staticbase;
                unsigned long m_linkbase;
                unsigned long m_programbase;
                unsigned long m_reserved;
        } a_modtbl;
        unsigned long a_brtoentry[2];   /* Code f. Sprung zu Einsprungadr. */
        unsigned long a_shdata;         /* Groesse initial. gemeins. Datenseg. */
        unsigned long a_shbss;          /* Gr. nicht-initial. gem. Datenseg. */
        unsigned long a_shdrsize;       /* Gr. d. Verschieb. gem. Datens. */
        unsigned long a_reserved[15];   /* Reserviert */
};
#define OMAGIC          0x00ea          /* nicht-reines Format - f. .o's */
#define ZMAGIC          0x10ea          /* Anf.-Ladeformat - Null auf Null */
#define XMAGIC          0x20ea          /* Anf.-Ladeformat - unguelt. Null */
#define SMAGIC          0x30ea          /* Anf.-Ladeformat - autonom */
/*
* Makros zur Feststellung gueltiger Objektdatei u. Datei-Distanzen
*
* In einer ausfuehrbaren a.out-Datei legt a_magic den Bezug von Datei-
* position zu Speicheradresse fest: ZMAGIC und XMAGIC setzen Datei-
* position 0 auf Speicheradresse 0x800; bei SMAGIC sind Datei- und
* Speicheradresse gleich. a_text kodiert die logische Groesse d. Text-
* segments (Gr. im Speicher) u. nicht notwendigerweise die Segmentgroesse
* in der Datei. Der Dateikopf steht im Textsegment; dies wird mit
* N_ADDRADJ() kodiert.
*
* In einer nicht ausfuehrbaren a.out-Datei (OMAGIC) beginnt das Text-
* segment nach dem Dateikopf.
*
* N_MINSIZ(x) gibt die Mindestgroesse einer gueltigen ausfuehrbaren
* Datei an.
*/

#define N_BADMAG(x) \
   (((x).a_magic)!=OMAGIC && ((x).a_magic)!=ZMAGIC && \
    ((x).a_magic)!=XMAGIC && ((x).a_magic)!=SMAGIC)
#define N_ADDRADJ(x) \
       (((x).a_magic == ZMAGIC||(x).a_magic == XMAGIC) ? 2048 : 0)
#define N_TXTOFF(x)     ((x).a_magic == OMAGIC ? sizeof (struct exec) : 0)
#define N_DATAOFF(x)    (N_TXTOFF(x) + (x).a_text - N_ADDRADJ(x))
```

```
#define N_SHDATAOFF(x)  (N_DATAOFF(x) + (x).a_data)
#define N_TROFF(x)      (N_SHDATAOFF(x) + (x).a_shdata)
#define N_DROFF(x)      (N_TROFF(x) + (x).a_trsize)
#define N_SHDROFF(x)    (N_DROFF(x) + (x).a_drsize)
#define N_SYMOFF(x)     (N_SHDROFF(x) + (x).a_shdrsize)
#define N_STROFF(x)     (N_SYMOFF(x) + (x).a_syms)
#define N_MINSIZ(x)     N_TROFF(x)
```

Die Datei besteht aus vier Abschnitten: (1) Kopf, Programmtext und Daten, (2) Verschiebungs-Informationen, (3) Symboltabelle und (4) Stringtabelle (in dieser Reihenfolge). Die Abschnitte (2), (3) und (4) sind nicht erforderlich, wenn
das Programm mit *ld* mit gesetztem Schalter *-s* geladen wird oder wenn Symbole und Verschiebungs-Informationen mit *strip* entfernt wurden.

Im Kopf ist die Länge der einzelnen Abschnitte in Bytes angegeben. Die Länge des Kopfes ist in der Länge des Textabschnitts (a_text) enthalten.

Die Länge der einzelnen Abschnitte im Speicher ergibt sich aus der Distanz zu dem vom Makro N_ADDRADJ übergebenen Wert. D.h.: um die tatsächlichen Adressen zu erhalten, muß man N_ADDRADJ von den Adressen in der Symboltabelle subtrahieren. So errechnet sich z.B. die tatsächliche Länge des Textabschnitts der *a.out*-Datei (einschließlich Kopf) aus a_text-N_ADDRADJ.

Bei der Ausführung der *a.out*-Datei werden drei logische Segmente eingerichtet: ein Textsegment, ein Datensegment (das initialisierte Daten und daran anschließend nicht initialisierte, auf 0 gesetzte Daten enthält) und ein Keller. Das Textsegment beginnt bei Adresse 0 in der *a.out*-Datei und Speicheradresse 2048.

Ist die "magische Zahl" im Dateikopf OMAGIC, so ist die Datei eine ".o"-Datei, d.h. nicht ausführbar.

Ist die "magische Zahl" ZMAGIC, dann sind die ersten 2048 Bytes im Speicher auf 0 gesetzt und das Textsegment beginnt bei Adresse 2048. Das Datensegment beginnt an der ersten mit 0 mod 2048 errechneten Bytegrenze nach dem Textsegment. Sowohl die Länge des Text- als auch die des Datensegments ist ein Vielfaches von 2048 Bytes. Das Textsegment ist schreibgeschützt. Alle Prozesse, die dieses Programm ausführen, benutzen dasselbe Textsegment. Die Datei wird seitenweise je nach Bedarf dynamisch in den Speicher geladen ("demand paging"). ZMAGIC ist das von *ld*(1) erzeugte Standardformat, das für sehr umfangreiche Programme besonders geeignet ist.

Ist die "magische Zahl" XMAGIC, so ist das Format wie bei

ZMAGIC, die ersten 2048 Bytes im Speicher sind jedoch ungültig. XMAGIC kann dazu verwendet werden, Fehler ausfindig zu machen, die zu falschen Verweisen von Nullzeigern führen.

Ist die "magische Zahl" SMAGIC, so handelt es sich um eine autonome, nicht unter SINIX ablaufende Datei. Bei diesem Format beginnt das Textsegment an der Speicheradresse 0 (N_ADDRADJ ist 0). Die Länge der Text- und Datensegmente wird standardgemäß an 2048-Bytegrenzen ausgerichtet. Dies kann durch Setzen des **-p**-Schalters des Laders, der nur für SMAGIC-Dateien gilt, verhindert werden.

Der Keller belegt die höchstmöglichen Speicheradressen des geladenen Programms und wird nach unten erweitert, und zwar automatisch, je nach Bedarf, während das Datensegment nur bei Anforderung mit *brk*(2) erweitert wird.

Auf den Dateikopf folgen - in der hier angegebenen Reihenfolge -: Textsegment, Datensegment, gemeinsames Datensegment, Verschiebungs-Informationen zum Textsegment, zum Datensegment und zum gemeinsamen Datensegment, Symboltabelle und Stringtabelle. Bei Angabe des Namens einer exec-Struktur als Argument liefern die Makros N_TXTOFF, N_DATAOFF, N_SHDATAOFF, N_SYMOFF bzw.N_STROFF die absolute Anfangsadresse in der Datei von Text-, Daten-, gemeinsamem Datensegment, Symboltabelle bzw. Stringtabelle. Die Makros N_TROFF, N_DROFF bzw. N_SHDROFF liefern die absolute Anfangsadresse in der Datei für die Verschiebungs-Informationen zum Text-, Daten-, bzw. gemeinsamen Datensegment.

In den ersten 4 Bytes der Stringtabelle wird ihre Länge abgelegt. Diese Bytes werden bei der Längenangabe mitgerechnet, aber nicht für die Speicherung von Zeichenketten ("Strings") verwendet. Die Stringtabelle ist daher immer mindestens 4 Bytes lang.

Der Aufbau eines Eintrags in der Symboltabelle und die wichtigsten Werte für den Symboltyp sind in der Include-Datei wie folgt angegeben:

```
/*
 * Format eines Eintrags in der Symboltabelle
 */
 struct nlist {
        union {
                char *n_name;           /* nach dem Laden zu verwenden */
                long n_strx;            /* Index in Datei-Stringtabelle */
        } n_un;
        unsigned char n_type;           /* Typangabe, N_TEXT usw.; s.u. */
        char    n_other;                /* unbenutzt */
        short   n_desc;                 /* s. <stab.h> */
        unsigned long   n_value;        /* Symbolwert (o. sdb-Distanz) */
};
#define n_hash          n_desc          /* ld-intern verwendet */
/*
* einfache Werte v. n_type.
*/
#define N_UNDF          0x00            /* undefiniert */
#define N_ABS           0x02            /* absolut */
#define N_TEXT          0x04            /* Text (implizit gemeins. Text) */
#define N_DATA          0x06            /* private Daten */
#define N_BSS           0x08            /* private nicht-initial. Daten */
#define N_COMM          0x0a            /* gemeinsam (ld-intern) */
#define N_FN            0x0c            /* Dateiname */
#define N_SHARED        0x10            /* gemeinsame N_UNDF, N_DATA, N_BSS */
#define N_SHUNDF        (N_SHARED|N_UNDF)
#define N_SHDATA        (N_SHARED|N_DATA)
#define N_SHBSS         (N_SHARED|N_BSS)
#define N_SHCOMM        (N_SHARED|N_COMM)
#define N_EXT           0x01            /* externes Bit (ODER-Verkn.) */
#define N_TYPE          0x1e            /* Maske f. alle Typ-Bits */
/*
* Bei einigen anderen permanenten Eintraegen in der Symboltabelle
* sind einige der N_STAB-Bits gesetzt.
* Diese sind in <stab.h> angegeben:
*/
#define N_STAB  0xe0    /* bei gesetzten Bits: nicht wegwerfen */
/*
* Format fuer Werte der Namensliste
*/
#define N_FORMAT        "%08x"
```

Das n_un.n_strx-Feld eines Eintrags in der Symboltabelle der *a.out*-Datei enthält einen Index der Stringtabelle. Hat ein solches Feld den Wert 0, so heißt das, daß diesem Eintrag in der Symboltabelle kein Name zugeordnet ist. Das n_un.n_name-Feld kann nur dann als Verweis auf einen Symbolnamen verwendet werden, wenn dieser vom Programm unter Verwendung von n_strx und entsprechenden Daten aus der Stringtabelle erstellt wird.

Ist als Symboltyp (n_type) 'undefiniert extern' oder 'gemeinsam undefiniert extern' angegeben und das Wert-Feld (n_value) ist ungleich 0, so wird das Symbol vom Binder *ld* als Name eines gemeinsamen Bereichs interpretiert, dessen Länge dem Symbol-Wert entspricht.

Der Wert eines Bytes im Text- oder Datensegment, das nicht zu einem Verweis auf ein undefiniertes externes Symbol gehört, ist genau der Wert, der bei der Dateiausführung im Speicher steht. Gehört ein Byte im Text- oder Datensegment zu einem Verweis auf ein undefiniertes externes Symbol (Angabe in der Verschiebungs-Information), dann gibt der Wert in der Datei die Distanz zum entsprechenden externen Symbol an. Beim Bindelauf erhält das externe Symbol seinen definierten Wert, der zum Wert in der Datei hinzuaddiert wird.

Die Verschiebungs-Informationen bestehen, sofern vorhanden, pro verschiebbares Datenelement aus einem 8 Byte langen Eintrag, wie in der folgenden Struktur dargestellt:

```
/*
 * Format einer Verschiebungs-Angabe
 */
struct relocation_info {
      int             r_address;      /* zu verschiebende Adresse */
      unsigned int    r_symbolnum:24, /* lokal. Symbol-Ordinalzahl */
                      r_pcrel:1,      /* bereits relativ zu bz verschoben */
                      r_length:2,     /* 0=Byte, 1=Wort, 2=long */
                      r_extern:1,     /* Wert d.betr. Symb. nicht vorh. */
                      r_bsr:1,        /* Eintrag für bsr-Ang. */
                      r_disp:1,       /* Distanz */
                              :2;     /* noch nicht belegt */
};
```

Ist a_trsize+a_drsize+a_shdrsize= =0, dann existiert keine Verschiebungsinformation. Ist r_extern gleich 0, dann bezieht sich r_symbolnum auf den n_type des zu verschiebenden Datenelements (d.h. N_TEXT ist relativ zum ursprünglichen Textsegment).

HINWEIS

Die Laufzeitunterstützung für gemeinsame initialisierte und nicht initialisierte Daten ist noch nicht implementiert.

Nach einem *exec*-Aufruf werden nur die Text- und Datenbereiche (initialisierte und nicht initialisierte Daten) des prozeßspezifischen Adreßraums eingerichtet; gemeinsame initialisierte und nicht initialisierte Daten werden z.Zt. noch ignoriert.

PORTABILITÄT

Die Include-Datei *<a.out.h>* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

Kommandos as, ld, nm, dbx, strip, ddt, sdb.

NAME

assert.h - verify program assertion
Programmbedingung überprüfen

DEFINITION

```
#include <assert.h>
```

BESCHREIBUNG

Die Include-Datei *<assert.h>* definiert das Makro *assert()* und bezieht sich auf das Makro *NDEBUG*, das nicht in dieser Include-Datei definiert ist. Wenn *NDEBUG* als Makroname vor der Einbindung dieser Include-Datei definiert ist, dann wird das Makro *assert()* einfach wie folgt definiert:

```
#define assert(ignore) ((void) 0)
```

Andernfalls verhält sich das Makro so, wie unter *assert()* beschrieben.

Das Makro *assert()* ist als Makro, nicht als Funktion implementiert. Wenn die Makrodefinition unterdrückt wird, um auf eine konkrete Funktion zuzugreifen, dann ist das Verhalten undefiniert.

PORTABILITÄT

Die Include-Datei *<assert.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

assert().

NAME

ctype.h - character types
Zeichenklassifikation

DEFINITION

```
#include <ctype.h>
```

BESCHREIBUNG

Die Include-Datei *<ctype.h>* deklariert die folgenden Namen als Funktionen oder Makros:

isalnum()	*isgraph*()	*isupper*()
isalpha()	*islower*()	*isxdigit*()
isascii()	*isprint*()	*tolower*()
iscntrl()	*ispunct*()	*toupper*()
isdigit()	*isspace*()	
toascii()		

Die folgenden Namen werden als Makros vereinbart:

_toupper()	*_tolower*()

PORTABILITÄT

Die Include-Datei *<ctype.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

dirent.h - format of directory entries
Format von Dateiverzeichnis-Einträgen

DEFINITION

```
#include <dirent.h>
```

BESCHREIBUNG

Die Include-Datei *<dirent.h>* definiert den folgenden Datentyp mittels *typedef*:

```
DIR     Ein Typ, der einen Dateiverzeichnis-Strom
        definiert.
```

Definiert die Struktur vom Typ *dirent*, die die folgenden Komponenten besitzt:

```
ino_t       d_ino         fortlaufende Dateinummer
char        d_name[]      Name des Eintrags
```

Der Typ *ino_t* ist definiert in *<sys/types.h>*.

Der char-Vektor *d_name* hat keine bestimmte Größe, aber die Anzahl der Zeichen vor dem abschließenden Zeichen '\0' soll {NAME_MAX} nicht überschreiten.

Die nachfolgenden Funktionen werden deklariert:

closedir()	*rewinddir()*
opendir()	*seekdir()*
readdir()	*telldir()*

PORTABILITÄT

Die Include-Datei *<dirent.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

closedir(), *opendir()*, *readdir()*, *rewinddir()*, *seekdir()*, *telldir()*, *<sys/types.h>*.

NAME

errno.h - system error numbers
Fehlernummern des Systems

DEFINITION

```
#include <errno.h>
```

BESCHREIBUNG

Die Include-Datei *<errno.h>* stellt eine Deklaration der Variablen *errno* zur Verfügung und gibt den folgenden symbolischen Konstanten eindeutige, von 0 verschiedene Werte:

Anmerkung:
Die eckigen Klammern um die Fehlernamen sind nur eine Konvention für die Darstellung in diesem Handbuch, nicht jedoch Teil des Namens. Programme, die diese Namen verwenden wollen, dürfen die eckigen Klammern nicht mit angeben.

[E2BIG] Argumentliste zu lang
[EACCES] Erlaubnis verweigert
[EAGAIN] Betriebsmittel nicht verfügbar, nochmal versuchen
[EBADF] Ungültige Dateinummer
[EBUSY] Gerät oder Betriebsmittel aktiv
[ECHILD] Kein Sohnprozeß
[EDEADLK] Betriebsmittel-Verklemmung würde auftreten
[EDOM] Mathematisches Argument außerhalb des Wertebereichs für die Funktion
[EEXIST] Datei existiert
[EFAULT] Ungültige Adresse
[EFBIG] Datei zu groß
[EIDRM] Bezeichner entfernt
[EINTR] Funktion unterbrochen
[EINVAL] Ungültiges Argument
[EIO] Ein- oder Ausgabefehler
[EISDIR] Ist ein Dateiverzeichnis
[EMFILE] Zu viele offene Dateien
[EMLINK] Zu viele Verweise
[ENAMETOOLONG] Dateiname zu lang
[ENFILE] Überlauf der Dateitabelle
[ENODEV] Kein solches Gerät
[ENOENT] Keine solche Datei oder solch ein Dateiverzeichnis
[ENOEXEC] Formatfehler bei exec
[ENOLCK] Keine Sperren verfügbar

[ENOMEM] Nicht genug Platz
[ENOMSG] Keine Nachricht des verlangten Typs
[ENOSPC] Kein Platz mehr auf Gerät
[ENOSYS] Funktion nicht implementiert
[ENOTBLK] Blockorientiertes Gerät benötigt
[ENOTDIR] Kein Dateiverzeichnis
[ENOTEMPTY]
Dateiverzeichnis ist nicht leer
[ENOTTY] Ungültige Ein-/Ausgabe-Kontrolloperation
[ENXIO] Kein solches Gerät oder solch eine Adresse
[EPERM] Operation nicht erlaubt
[EPIPE] Pipe abgebrochen
[ERANGE] Ergebnis zu groß
[EROFS] Nur zum Lesen eingehängtes Dateisystem
[ESPIPE] Ungültige Suche
[ESRCH] Kein solcher Prozeß
[ETXTBSY] Reine Prozedurdatei aktiv
[EXDEV] Verweis über Gerätegrenzen

HINWEIS

Auf einigen X/Open-kompatiblen Systemen können zusätzliche Fehlernummern definiert sein.

PORTABILITÄT

Die Include-Datei *<errno.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

Abschnitt 2.2, Fehlernummern.

NAME

fcntl.h - file control options
Operationen für Dateikontrolle

DEFINITION

```
#include <fcntl.h>
```

BESCHREIBUNG

Die Include-Datei *<fcntl.h>* definiert die folgenden Kommandos und Argument für die Verwendung durch die Funktionen *fcntl()* und *open()*:

Werte für *cmd* in der Funktion *fcntl()* (die folgenden Werte sind eindeutig):

```
F_DUPFD   Dateikennzahl duplizieren
F_GETFD   Prozeß-Dateistatusbyte lesen
F_SETFD   Prozeß-Dateistatusbyte setzen
F_GETFL   System-Dateistatusbyte lesen
F_SETFL   System-Dateistatusbyte setzen
F_GETLK   Information über Sperre eines Dateibereich lesen
F_SETLK   Information über Sperre eines Dateibereichs setzen
F_SETLKW  Information über Sperre eines Dateibereichs setzen,
          warten wenn blockiert
```

Bits für das Prozeß-Dateistatusbyte in *fcntl()*:

FD_CLOEXEC (sbe-Bit) Schließe Dateikennzahl bei der Ausführung einer *exec*-Funktion

Werte für *l_type* zum Sperren von Dateibereichen mit *fcntl()* (die folgenden Werte sind eindeutig):

```
F_RDLCK   Lesesperre
F_UNLCK   Entsperren
F_WRLCK   Schreibsperre
```

Die folgenden drei Mengen von Werten sind bitweise verschieden. Werte für *oflag* zur Verwendung in *open()*:

```
O_CREAT   Erzeuge nicht existierende Datei
O_EXCL    Kennzeichen für exklusive Verwendung
O_NOCTTY  Kein kontrollierendes Terminal zuweisen
O_TRUNC   Kennzeichen für Abschneiden
```

Bits für das System Dateistatusbyte in *open()* und *fcntl()*:

```
O_APPEND      Anfügemodus setzen
O_NDELAY      (Zurückgezogen)

O_NONBLOCK    Nichtblockierender Modus
O_SYNC        Synchrones Schreiben
```

Maske für die Verwendung bei den Zugriffsarten einer Datei:

O_ACCMODE Maske für Zugriffsarten

Zugriffsarten für *open()* und *fcntl()*:

```
O_RDONLY        Nur zum Lesen offen
O_RDWR          Zum Lesen und Schreiben offen
O_WRONLY        Nur zum Schreiben offen
```

Die Struktur *flock* beschreibt eine Dateisperre. Sie schließt die folgenden Komponeten ein:

```
short       l_type      Sperrtyp; F_RDLCK, F_WRLCK, F_UNLCK
short       l_whence    Kennzeichen für Ausgangspunkt
off_t       l_start     Relativer Anfang in Bytes
off_t       l_len       Größe; wenn 0 dann bis EOF
pid_t       l_pid       Prozeßnummer des Prozeses, der die
                        Sperre besitzt; wird mit F_GETLK
                        zurückgeliefert
```

Die folgenden Namen werden entweder als Funktionen oder Makros deklariert:

creat() *fcntl()* *open()*

HINWEIS

O_NDELAY wurde zurückgezogen, da seine Funktionalität von O_NONBLOCK ersetzt wurde. Siehe auch die vollständige Beschreibung unter *open()*.

PORTABILITÄT

Die Include-Datei <*fcntl.h*> ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

creat(), *exec*, *fcntl()*, *open()*.

NAME

ftw.h - file tree traversal
Durchsuchen eines Dateibaums

DEFINITION

```
#include <ftw.h>
```

BESCHREIBUNG

Die Include-Datei *<ftw.h>* definiert Codes für das dritte Argument der benutzerdefinierten Funktion, die als zweites Argument an *ftw()* übergeben wird:

FTW_F Datei
FTW_D Dateiverzeichnis
FTW_DNR Dateiverzeichnis ohne Leseerlaubnis
FTW_NS Unbekannter Typ, *stat()* fehlgeschlagen

Deklariert den folgenden Namen als Funktion oder Makro:

ftw()

PORTABILITÄT

Die Include-Datei *<ftw.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

ftw().

NAME

grp.h - group structure
Gruppenstruktur

DEFINITION

```
#include <grp.h>
```

BESCHREIBUNG

Die Include-Datei *<grp.h>* deklariert die Struktur *struct group*, welche die folgenden Komponeten enthält:

char	*gr_name	Gruppenname
gid_t	gr_gid	Gruppennummer
char	**gr_mem	Zeiger auf einen mit dem Nullzeiger abgeschlossenen Vektor von Zeiger auf die Mitgliedsnamen

Die folgenden Namen werden entweder als Funktion oder als Makros deklariert:

getgrgid() *getgrnam()*

PORTABILITÄT

Die Include-Datei *<grp.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

getgrgid(), *getgrnam()*.

NAME

langinfo.h - language information constants
Konstanten für Sprach-Information

DEFINITION

```
#include <langinfo.h>
```

BESCHREIBUNG

Die Include-Datei *<langinfo.h>* enthält die Konstanten, die verwendet werden, um Daten zu *langinfo* zu identifizieren (siehe auch *nl_langinfo()*). Die Art dieser Konstanten ist in *<nl_types.h>* gegeben. Folgende Konstanten sind auf allen X/Open-kompatiblen Systemen definiert.

Die Einträge unter *Kategorie* geben an, zu welcher Kategorie von *setlocale()* jeder Eintrag definiert ist.

Konstante	Kategorie	Bedeutung
D_T_FMT	LC_TIME	Zeichenkette zur Formatierung von Datum und Zeit
D_FMT	LC_TIME	Datumsformat-Zeichenkette
T_FMT	LC_TIME	Zeitformat-Zeichenkette
AM_STR	LC_TIME	Kennzeichen für Vormittag
PM_STR	LC_TIME	Kennzeichen für Nachmittag
DAY_1	LC_TIME	Name des ersten Wochentags (e.g., Sonntag)
DAY_2	LC_TIME	Name des zweiten Wochentags (e.g., Montag)
DAY_3	LC_TIME	Name des dritten Wochentags (e.g., Dienstag)
DAY_4	LC_TIME	Name des vierten Wochentags (e.g., Mittwoch)
DAY_5	LC_TIME	Name des fünften Wochentags (e.g., Donnerstag)
DAY_6	LC_TIME	Name des sechsten Wochentags (e.g., Freitag)
DAY_7	LC_TIME	Name des siebten Wochentags (e.g., Samstag)
ABDAY_1	LC_TIME	abgekürzter Name des ersten Wochentags
ABDAY_2	LC_TIME	abgekürzter Name des zweiten Wochentags
ABDAY_3	LC_TIME	abgekürzter Name des dritten Wochentags
ABDAY_4	LC_TIME	abgekürzter Name des vierten

		Wochentags
ABDAY_5	LC_TIME	abgekürzter Name des fünften Wochentags
ABDAY_6	LC_TIME	abgekürzter Name des sechsten Wochentags
ABDAY_7	LC_TIME	abgekürzter Name des siebten Wochentags
MON_1	LC_TIME	Name des ersten Monats im gregorianischen Kalender
MON_2	LC_TIME	Name des zweiten Monats
MON_3	LC_TIME	Name des dritten Monats
MON_4	LC_TIME	Name des vierten Monats
MON_5	LC_TIME	Name des fünften Monats
MON_6	LC_TIME	Name des sechsten Monats
MON_7	LC_TIME	Name des siebten Monats
MON_8	LC_TIME	Name des achten Monats
MON_9	LC_TIME	Name des neunten Monats
MON_10	LC_TIME	Name des zehnten Monats
MON_11	LC_TIME	Name des elften Monats
MON_12	LC_TIME	Name des zwölften Monats
ABMON_1	LC_TIME	abgekürzter Name des ersten Monats
ABMON_2	LC_TIME	abgekürzter Name des zweiten Monats
ABMON_3	LC_TIME	abgekürzter Name des dritten Monats
ABMON_4	LC_TIME	abgekürzter Name des vierten Monats
ABMON_5	LC_TIME	abgekürzter Name des fünften Monats
ABMON_6	LC_TIME	abgekürzter Name des sechsten Monats
ABMON_7	LC_TIME	abgekürzter Name des siebten Monats
ABMON_8	LC_TIME	abgekürzter Name des achten Monats
ABMON_9	LC_TIME	abgekürzter Name des neunten Monats
ABMON_10	LC_TIME	abgekürzter Name des zehnten Monats
ABMON_11	LC_TIME	abgekürzter Name des elften Monats
ABMON_12	LC_TIME	abgekürzter Name des zwölften Monats
RADIXCHAR	LC_NUMERIC	Dezimalpunkt
THOUSEP	LC_NUMERIC	Tausender-Trennzeichen
YESSTR	LC_ALL	Bestätigende Antwort für Ja/Nein-Abfragen

NOSTR	LC_ALL	Negative Antwort für Ja/Nein-Abfragen
CRNCYSTR	LC_MONETARY	Währungssymbol; wenn das Symbol vor dem Wert erscheinen soll, dann muß das Zeichen '-' vorangestellt werden; wenn es nach dem Wert erscheinen soll, dann muß das Zeichen '+' voranstehen; wenn es den Dezimalpunkt ersetzen soll, dann muß das Zeichen '.' vorangestellt werden.

Dklariert den folgenden Namen als Funktion:

nl_langinfo()

HINWEIS

Wann immer dies möglich ist, sollten Benutzer die Funktionen verwenden, die kompatibel zu denen in *Draft ANSI X3.159 Programming Language C* sind, um auf Daten aus *langinfo* zuzugreifen. Insbesondere sollte die Funkmtion *strftime()* verwendet werden, um auf die Datums- und Zeitinformationen der Kategorie *LC_TIME* zuzugreifen.

PORTABILITÄT

Die Include-Datei *<langinfo.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

nl_langinfo(), *Abschnitt 2.5*.

NAME

limits.h - implementation specific constants
implementierungsabhängige Konstanten

DEFINITION

```
#include <limits.h>
```

BESCHREIBUNG

Die Include-Datei *<limits.h>* definiert verschiedene Namen, die in diesem Handbuch verwendet werden. Verschiedene Kategorien von Namen werden in den unten angeführten Tabellen beschrieben.

Die Namen repräsentieren verschiedene Grenzen von Betriebsmitteln, die das System Anwendungen zur Verfügung stellt.

Implementierungen können jeden beliebigen Wert für jede Grenze wählen, vorausgesetzt, dieser ist nicht eingeschränkter als der betreffende Wert in der Spalte mit der Überschrift *Kleinster akzeptabler Wert* in den unten angeführten Tabellen.

Anwendungen sollten für ein Grenze keinen bestimmten Wert annehmen. Um ein Maximum an Portabilität zu erreichen, sollte eine Anwendung nicht mehr von einem Betriebsmittel anfordern, als in der Spalte *Kleinster akzeptabler Wert* angegeben. Dennoch kann eine Anwendung, die die gesamte Menge eines Betriebsmittels, die unter einer Implementierung zur Verfügung steht, anfordern will, den Wert aus der Datei *<limits.h>* auf diesem speziellen System verwenden, indem sie die symbolischen Namen benutzt, die jeweils in der ersten Spalte der Tabellen aufgeführt sind. An dieser Stelle muß angemerkt werden, daß viele der aufgeführten Grenzen nicht fest sind, und daß zur Laufzeit der Wert einer Grenze von dem Wert abweichen kann, der in dieser Include-Datei vorgegeben ist. Dies hat folgende Gründe:

- Die Grenze hängt von einem Pfadnamen ab.
- Die Grenze unterscheidet sich auf Übersetzungszeit- und Laufzeit-Rechnern

Aus diesen Gründen kann eine Anwendung die Funktionen *fpathconf()*, *pathconf()* und *sysconf()* verwenden, um den aktuellen Wert einer Grenze zur Laufzeit zu ermitteln.

Die Einträge in der Liste, die auf *_MIN* enden, stellen die negativsten Werte der mathematischen Typen dar, die garantiert dargestellt werden können. Negativere Werte können unter einigen Systemen unterstützt werden, wie in der jeweiligen Include-Datei *<limits.h>* des Systems angegeben, aber Anwendungen, die solche Zahlen benötigen, sind nicht unbedingt unter allen Systemen portabel.

Das Symbol '*' in der Spalte *Kleinster akzeptabler Wert* gibt an, daß es keinen garantierten Wert für alle X/Open-kompatiblen Systeme gibt.

Die Definition für jeden der folgenden Namen kann aus *<limits.h>* weggelassen werden, wenn der Wert der Grenze unbestimmt, aber größer als das geforderte Minimum ist. Anwendungen sollten diese Symbole daher nur in bedingt übersetzten Programmteilen nutzen, die nur bei der Existenz dieses Symbols übersetzt werden oder in Aufrufen von *fpathconf()*, *pathconf()* oder *sysconf()*.

Name	Beschreibung / Kleinster akzeptabler Wert
ARG_MAX	Maximale Länge des Arguments der *exec*-Funktionen, einschließlich der Umgebungsdaten / _POSIX_ARG_MAX
CHILD_MAX	Maximale Anzahl der Prozesse je Benutzernummer / _POSIX_CHILD_MAX
LINK_MAX	Maximale Anzahl von Verweisen auf eine einzelne Datei / _POSIX_LINK_MAX
MAX_CANON	Maximale Anzahl von Bytes in einer Eingabezeile der Datensichtstation bei standardmäßiger Eingabeverarbeitung / _POSIX_MAX_CANON
MAX_INPUT	Maximale Anzahl von Byte, die in der Eingabeschlange einer Datensichtstation erlaubt sind / _POSIX_MAX_INPUT
NAME_MAX	Maximalzahl der Zeichen in einem Dateinamen, ohne das abschließende Nullbyte / _POSIX_NAME_MAX
OPEN_MAX	Maximalzahl der gleichzeitig offenen Dateien je Prozeß / _POSIX_OPEN_MAX
PASS_MAX	Maximalzahl signifikanter Zeichen in einem Kennwort, ohne abschließendes Nullbyte / 8
PATH_MAX	Maximalzahl der Zeichen in einem Pfadnamen, ohne abschließendes Nullbyte / _POSIX_PATH_MAX
PIPE_BUF	Maximalzahl der Bytes die beim Schreiben auf eine Pipe garantiert atomar sind /_POSIX_PIPE_BUF

Die folgende Konstante ist in *<limits.h>* immer definiert und deshalb auch über *sysconf()* verfügbar:

Name	Beschreibung / Kleinster akzeptabler Wert
NGROUPS_MAX	Maximalzahl gleichzeitig vorhandener weiterer Gruppennummern je Prozeß / 0

Die folgenden Konstanten sind immer in *<limits.h>* definiert:

Name	Beschreibung / Kleinster akzeptabler Wert
NL_ARGMAX	Maximaler Wert einer Ziffer in Aufrufen der Funktionen *printf* und *scanf*() / 9
NL_LANGMAX	Maximalzahl der Bytes in einem *LANG*-Namen / 14
NL_MSGMAX	Maximale Meldungsnummer / 32767
NL_NMAX	Maximale Anzahl von Bytes in N-zu-1 abgebildeten Zeichen / *
NL_SETMAX	Maximale Mengennummer / 255
NL_TEXTMAX	Maximale Anzahl von Zeichen in einem Meldungstext / 255
NZERO	Standard-Prozeßpriorität / 20
TMP_MAX	Maximalzahl eindeutiger Namen, die von *tmpnam()* generiert werden können / 10000

Die folgenden Konstanten sind durch POSIX festgelegt und werden immer in *<limits.h>* definiert. Sie sind fest.

Name	Beschreibung / Wert
_POSIX_ARG_MAX	Die Länge der Argumentzeichenkette für die *exec*-Funktionen in Bytes, einschließlich der Umgebungsdaten / 4096
_POSIX_CHILD_MAX	Die Anzahl gleichzeitiger Prozesse je realer Benutzernummer / 6
_POSIX_LINK_MAX	Der Maximalwert des Verweiszählers einer Datei / 8
_POSIX_MAX_CANON	Die Anzahl der Bytes in der Eingabeschlange einer Datensichtstation bei standardmäßiger Eingabeverarbeitung / 255
_POSIX_MAX_INPUT	Die Anzahl von Bytes, für die in der Eingabeschlange einer Datensichtstation Platz verfügbar ist / 255
_POSIX_NAME_MAX	Die Anzahl der Bytes in einem Dateinamen / 14
_POSIX_NGROUPS_MAX	Die Anzahl gleichzeitiger weiterer Gruppennummern je Prozeß / 0
_POSIX_OPEN_MAX	Die Anzahl von Dateien, die ein Prozeß gleichzeitig offen haben kann / 16
_POSIX_PATH_MAX	Die Anzahl der Bytes in einem Pfadnamen / 255
_POSIX_PIPE_BUF	Die Anzahl der Bytes, die beim Schreiben auf eine Pipe automatisch geschrieben werden kann / 512

Die folgenden Konstanten werden in *<limits.h>* immer definiert. Sie sind fest:

Name	Beschreibung / Kleinster akzeptabler Wert
CHAR_BIT	Anzahl der Bits in einem *char* / 8
CHAR_MAX	Maximaler ganzzahliger Wert eines *char* / 127
DBL_DIG	Genaue Ziffern eines *double* / *
DBL_MAX	Maximaler dezimale Wert eines *double* / *
FLT_DIG	Genaue Ziffern eines *float* / *
FLT_MAX	Maximaler dezimale Wert eines *float* / *
INT_MAX	Maximaler dezimale Wert eines *int* / 32767
LONG_BIT	Anzahl der Bits in einem *long* / 32
LONG_MAX	Maximaler dezimale Wert eines *long* / 2147483647
MB_LEN_MAX	Nützliche POSIX-Konstante / 1
SCHAR_MAX	Maximalwert eines *signed char* / 127
SHRT_MAX	Maximaler dezimale Wert eines *short* / 32767
UCHAR_MAX	Maximalwert eines *unsigned char* / 255L
UINT_MAX	Maximalwert eines *unsigned int* / 65535L
ULONG_MAX	Maximalwert eines *unsigned long* / 4294967295L
USHRT_MAX	Maximalwert für ein *unsigned short* / 65535L
WORD_BIT	Bitanzahl eines *Worts* bzw. *int* / 16

Name	Beschreibung / Kleinster akzeptabler Wert
CHAR_MIN	Minimaler ganzzahlige Wert eines *char* / 0
DBL_MIN	Minimaler dezimale Wert eines *double* / *
FLT_MIN	Minimaler dezimale Wert eines *float* / *
INT_MIN	Minimaler dezimale Wert eines *int* / -32768
LONG_MIN	Minimaler dezimale Wert eines *long* / -2147483648
SCHAR_MIN	Minimalwert eines *signed char* / -127
SHRT_MIN	Minimaler dezimale Wert eines *short* / -32768

Die folgenden Konstanten wurden zurückgezogen und werden nicht mehr länger unterstützt. Siehe auch HINWEIS für weitere Einzelheiten.

Name	Beschreibung
CLK_TCK	ZURÜCKGEZOGEN
FCHR_MAX	ZURÜCKGEZOGEN
LOCK_MAX	ZURÜCKGEZOGEN
MAX_CHAR	ZURÜCKGEZOGEN
PROC_MAX	ZURÜCKGEZOGEN
STD_BLK	ZURÜCKGEZOGEN
SYS_OPEN	ZURÜCKGEZOGEN
SYSPID_MAX	ZURÜCKGEZOGEN
USI_MAX	ZURÜCKGEZOGEN

HINWEIS

CLK_TCK wurde in die Datei *<time.h>* verschoben. FCHR_MAX, LOCK_MAX, STD_BLK, SYS_OPEN und SYSPID_MAX wurden zurückgezogen, da es sich um systemweite Grenzen handelte, die für Anwendungen bedeutungslos waren. MAX_CHAR wurde zurückgezogen und durch MAX_INPUT ersetzt. USI_MAX wurde zurückgezogen und durch UINT_MAX ersetzt.

Wenn der Wert eines Objekts vom Typ *char* bei der Verwendung in einem Ausdruck der Vorzeichenerweiterung unterworfen wird, dann ist der Wert von CHAR_MIN derselbe wie der von SCHAR_MIN und der Wert von CHAR_MAX ist identisch zu SCHAR_MAX. Wenn der Wert von CHAR_MIN gleich 0 ist, dann ist der Wert von CHAR_MAX identisch mit dem von UCHAR_MAX.

PORTABILITÄT

Die Include-Datei *<limits.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

fpathconf(), *pathconf()*, *sysconf()*.

NAME

locale.h - category macros
Kategorie-Makros

DEFINITION

```
#include <locale.h>
```

Beschreibung

Die Include-Datei <*locale.h*> definiert wenigstens die folgenden Makros:

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MONETARY
LC_NUMERIC
LC_TIME

die zu vollständig verschiedenen, konstanten Ausdrücken expandiert werden, um als erstes Argument der Funktion *setlocale()* verwendet zu werden.

Sie deklariert die Funktion *setlocale()*.

Zusätzliche Makro-Definitionen, die mit den Zeichen LC_ und einem Großbuchstaben beginnen, können hier ebenfalls definiert werden.

PORTABILITÄT

Die Include-Datei <*locale.h*> ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

setlocale(), *Abschnitt 2.5*.

NAME

malloc.h - main memory allocator
Speicherplatzreservierung

DEFINITION

```
#include <malloc.h>
```

BESCHREIBUNG

Diese Datei enthält die Definitionen der für *mallopt()* eingesetzten Konstanten

```
M_MXFAST  /* Blockgröße für "fast" setzen */
M_NLBLKS  /* Blockanzahl in großen Blöcken setzen */
M_GRAIN   /* Anzahl der bei kleinen Blöcken aufgerundeten
             Größen setzen */
M_KEEP    /* Blockinhalt nach Freigabe bis zur nächsten
             Reservierung aufheben */
```

sowie die Deklaration der Struktur *mallinfo*, die aus folgenden Elementen besteht:

```
int arena;     /* Speicherplatz insgesamt im Bereich */
int ordblks;   /* Anzahl normaler Blöcke */
int smblks;    /* Anzahl kleiner Blöcke */
int hblks;     /* Anzahl großer Blöcke */
int hblkhd;    /* Platz in Köpfen der großen Blöcke */
int usmblks;   /* Platz in benutzten kleinen Blöcken */
int fsmblks;   /* Platz in freien kleinen Blöcken */
int uordblks;  /* Platz in benutzten normalen Blöcken */
int fordblks;  /* Platz in freien normalen Blöcken */
int keepcost;  /* Aufwand für Verwendung von keep */
```

PORTABILITÄT

Die Include-Datei *<malloc.h>* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

malloc().

NAME

math.h - mathematical declarations
Mathematische Deklarationen

DEFINITION

```
#include <math.h>
```

BESCHREIBUNG

Die Include-Datei <*math.h*> definiert die folgenden Konstanten. Die Werte sind vom Typ *double* und liegen genau innerhalb der Präzision eines *double*.

M_E	Wert von e
M_LOG2E	Wert von $\log^2 e$
M_LOG10E	Wert von $\log^{10} e$
M_LN2	Wert von $\log^e 2$
M_LN10	Wert von $\log^e 10$
M_PI	Wert von π
M_PI_2	Wert von $\pi/2$
M_PI_4	Wert von $\pi/4$
M_1_PI	Wert von $1/\pi$
M_2_PI	Wert von $2/\pi$
M_2_SQRTPI	Wert von 2/Wurzel aus π
M_SQRT2	Wert von Wurzel aus 2

Die Include-Datei enthält eine #*define*-Anweisung für das Symbol MAXFLOAT, das systemabhängig ist, und für den Wert HUGE_VAL der im Fehlerfall von den Funktionen aus der Mathematik-Bibliothek zurückgeliefert wird.

MAXFLOAT Wert der größten nichtunendlichen Gleitpunktzahl einfacher Genauigkeit.

HUGE_VAL Von den Funktionen der Mathematik-Bibliothek zurückgelieferter Fehlerwert.

Das Makro HUGE_VAL ist definiert, um Fehlerwerte zu repräsentieren, die von den Mathematikfunktionen geliefert werden. HUGE_VAL ist entweder gleich *+inf* auf einem System, das *IEEE Std 754-1985* unterstützt, oder gleich +{DBL_MAX} auf einem System, das diesen Standard nicht unterstützt.

Die folgenden Namen werden als Funktionen oder Makros deklariert:

acos()	*fabs()*	*jn()*	*sqrt()*
asin()	*floor()*	*ldexp()*	*tan()*
atan2()	*fmod()*	*lgamma()*	*tanh()*
atan()	*frexp()*	*log10()*	*y0()*
ceil()	*isnan()*	*log()*	*y1()*
cos()	*gamma()*	*modf()*	*yn()*
cosh()	*hypot()*	*pow()*	
erf()	*j0()*	*sin()*	
exp()	*j1()*	*sinh()*	

signgam wird als externe Variable des Typs *int* deklariert.

PORTABILITÄT

Die Include-Datei *<math.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

acos(), *asin()*, *atan()*, *atan2()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *gamma()*, *hypot()*, *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *modf()*, *pow()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*.

NAME

memory - memory operations
Speicheroperationen

DEFINITION

```
#include <memory.h>
```

BESCHREIBUNG

Diese Datei enthält die Typ-Deklarationen für die Speicheroperationen ausführenden Funktionen.

PORTABILITÄT

Die Include-Datei <*memory.h*> ist im X/Open-Standard nicht enthalten. Sie wird nur aus Gründen der Rückwärtskompatibilität zu exisitierenden Anwendungen noch unterstützt. Die Speicheroperationen verwenden in dieser Version die Include-Datei <*string.h*>.

SIEHE AUCH

mem.....

NAME

mon.h - prepare execution profile
Ausführungsprofil

DEFINITION

```
#include <mon.h>
```

BESCHREIBUNG

Diese Datei enthält folgende Struktur-Deklarationen (jeweils einschließlich Strukturkomponenten):

struct hdr

```
char    *lpc;   /* niedrigster BZ (Befehlszaehler) d. m. profil */
                /* bearbeiteten Bereichs */
char    *hpc;   /* hoechster BZ d. m. profil bearbeiteten Bereichs */
int     nfns;   /* Anzahl cnt-Strukturen */
```

struct cnt

```
char    *fnpc;  /* Funktions-BZ */
long    mcnt;   /* Aufrufzaehler */
```

Die Datei enthält ferner eine Typdefinition für Typ *WORD*.

Folgende Namen werden definiert:

MON_OUT name des Profils, z.B. ''mon.out''
MPROGS0
MSCALE0
NULL

PORTABILITÄT

Die Include-Datei <*mon.h*> ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

monitor().

NAME

nl_types.h - data types
Datentypen für NLS

DEFINITION

```
#include <nl_types.h>
```

BESCHREIBUNG

Die Include-Datei *<nl_types.h>* enthält die Definitionen zumindest der folgenden Datentypen:

nl_catd — von den Meldungskatalog-Funktionen *catopen()*, *catgets()* und *catclose()* verwendet, um einen Katalog-Deskriptor zu identifizieren.

nl_item — von *nl_langinfo()* verwendet, um Daten von *langinfo* zu identifizieren. Werte für Objekte des Typs *nl_item* werden in *<langinfo.h>* definiert.

und zumindest die folgende Konstante:

NL_SETD — verwendet von *gencat*, wenn keine *$set*-Direktive in einer Meldungstext-Quelldatei angegeben wird. Diese Konstante kann als Wert von *set_id* an nachfolgende Aufrufe von *catgets()* verwendet werden um Meldungen aus der Standard-Meldungsmenge zu ermitteln. Der Wert von NL_SETD ist implementierungsabhängig.

Die folgenden Funktionen werden deklariert:

catclose() *catgets*() *catopen*()

PORTABILITÄT

Die Include-Datei *<nl_types.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

catclose(), *catgets*(), *catopen*(), *nl_langinfo*(), *<langinfo.h>*, Kommando *gencat*.

NAME

pwd.h - password structure
Kennwortstruktur

DEFINITION

```
#include <pwd.h>
```

BESCHREIBUNG

Die Include-Datei *<pwd.h>* stellt eine Definition für die Struktur *struct passwd* zur Verfügung, welche die folgenden Komponenten enthält:

```
char  *pw_name  Benutzerkennung
uid_t  pw_uid   Benutzernummer
gid_t  pw_gid   Gruppennummer
char  *pw_dir   HOME-Dateiverzeichnis
char  *pw_shell Startprogramm
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

getpwnam() *getpwuid*()

HINWEIS

Die Struktur *passwd* kann zusätzliche Komponenten besitzen.

PORTABILITÄT

Die Include-Datei *<pwd.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

getpwnam(), *getpwuid()*.

NAME

regexp.h - regular-expression declarations
Deklarationen für reguläre Ausdrücke

DEFINITION

```
#include <regexp.h>
```

BESCHREIBUNG

Die Include-Datei *<regexp.h>* deklariert die folgenden Funktionen als Makros:

advance() *compile*() *step*()

und deklariert die folgenden externen Variablen:

loc1 *loc2* *locs*

PORTABILITÄT

Die Include-Datei *<regexp.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

regexp().

NAME

search.h - search tables
Tabellen durchsuchen

DEFINITION

```
#include <search.h>
```

BESCHREIBUNG

Die Include-Datei *<search.h>* führt ein *typedef ENTRY* für den Datentyp *struct entry* durch, der die folgenden Komponenten enthält:

```
char    *key;
char    *data;
```

und definiert *ACTION* und *VISIT* als Aufzählungstypen mit *typedef* wie folgt:

```
enum { FIND, ENTER } ACTION; enum { preorder, postorder,
endorder, leaf } VISIT;
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

hcreate()	*lfind()*	*tdelete()*
hdestroy()	*lsearch()*	*tfind()*
hsearch()	*tsearch()*	*twalk()*

PORTABILITÄT

Die Include-Datei *<search.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

hsearch(), *lsearch()*, *tsearch()*.

NAME

setjmp.h - stack environment declarations
Deklarationen für Stackumgebung

DEFINITION

```
#include <setjmp.h>
```

BESCHREIBUNG

Die Include-Datei *<setjmp.h>* enthält Typdefinitionen mit *typedef* für die Datentypen *jmp_buf* und *sigjmp_buf*.

Die Funktionen *longjmp()* und *siglongjmp()* werden deklariert.

setjmp() und *sigsetjmp()* werden entweder als Funktionen oder als Makros deklariert.

PORTABILITÄT

Die Include-Datei *<setjmp.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

longjmp(), *setjmp()*, *siglongjmp()*, *sigsetjmp()*.

NAME

signal.h - signals
Signale

DEFINITION

```
#include <signal.h>
```

BESCHREIBUNG

Die Include-Datei *<signal.h>* definiert die folgenden symbolischen Konstanten, von denen jede zu einem eindeutigen Ausdruck des Typs *void (*)()* expandiert, dessen Wert keiner deklarierbaren Funktion entspricht.

SIG_DFL standardmäßige Signalbehandlung
SIG_ERR Ergebnis von *signal*() im Fehlerfall
SIG_IGN Ignorieren des Signals

Der Datentyp *sigset_t* wird verwendet, um Signalmengen zu repräsentieren. Er ist immer ein ganzzahliger oder Strukturtyp. Verschiedene Funktionen werden verwendet, Objekte des Typs *sigset_t* zu manipulieren; siehe auch *sigfillset()*.

Diese Include-Datei deklariert die Konstanten, die dazu verwendet werden, im System auftretende Signale darzustellen. Jedes dieser Signale besitzt einen eindeutigen, positiven ganzzahligen Wert. Der Wert 0 ist für das Nullsignal reserviert (siehe auch *kill()*). Eine Implementierung kann zusätzliche Signale, die im System auftreten können, definieren.

Die folgenden Signale werden von allen Implementierungen unterstützt (Standardaktionen werden unten erklärt):

Signal	Standard-Aktion	Beschreibung
SIGABRT	ii	Signal für Prozeßabbruch
SIGALRM	i	Alarmuhr
SIGFPE	ii	Gleitkommaausnahme
SIGHUP	i	Hangup
SIGILL	ii	Illegale Instruktion
SIGINT	i	Unterbrechung an Datensichtstation
SIGKILL	i	Kill (kann nicht abgefangen oder ignoriert werden)
SIGPIPE	i	Schreiben auf Pipe, die nicht gelesen wird
SIGQUIT	ii	Beenden von Datensichtstation
SIGSEGV	ii	Ungültiger Verweis im Speicher
SIGTERM	i	Beendigungs-Signal
SIGUSR1	i	benutzer-definiertes Signal 1
SIGUSR2	i	benutzer-definiertes Signal 2

Die folgenden Signale sind auf allen Systemen definiert, aber ein System, das Auftragskontrolle nicht unterstützt, braucht auch die Funktionalität dieser Signale nicht zu unterstützen:

Signal	Standard-Aktion	Beschreibung
SIGCHLD	iii	Sohnprozeß beendet oder angehalten
SIGCONT	v	Fortsetzen, wenn angehalten
SIGSTOP	iv	Ausführung anhalten (kann nicht abgefangen oder ignoriert werden)
SIGTSTP	iv	Stopsignal für Datensichtstation
SIGTTIN	iv	Hintergrundprozeß versucht zu lesen
SIGTTOU	iv	Hintergrundprozeß versucht zu schreiben

Die Standardaktionen sind die folgenden:

- i Beenden des Prozesses.
- ii Anormales Beenden des Prozesses. Der Prozeß wird beendet, aber zusätzlich können implementierungs-abhängige Routinen für die anormale Beendigung, wie zum Beispiel Speicherabzüge, ausgeführt werden.
- iii Signal ignorieren.
- iv Prozeß anhalten.
- v Prozeß fortsetzen, wenn er angehalten ist, andernfalls Signal ignorieren.

Die Include-Datei stellt eine Deklaration von *struct sigaction* zur Verfügung, die zumindest die folgenden Komponenten einschließt:

void	sa_handler)()	Signalbehandlungs-Funktion
sigset_t	sa_mask	Signalmenge, die während der Ausführung der Signalbehandlungsfunktion blockiert ist
int	sa_flags	Spezielle Kennzeichen

Die folgenden Konstanten werden deklariert:

SA_NOCLDSTOP
: Beim Anhalten eines Sohnprozesses nicht SIGCHLD generieren

SIG_BLOCK
: Die Ergebnismenge ist die Vereinigung der aktuellen Menge und der Signalmenge, auf die *set* zeigt

SIG_UNBLOCK
: Die Ergebnismenge ist die Schnittmenge der aktuellen Menge und der Signalmenge, auf die *set* zeigt

SIG_SETMASK
: Die Ergebnismenge ist die Signalmenge, auf die *set* zeigt

Die folgenden Funktionen oder Makros werden deklariert:

kill()	*sigemptyset()*	*sigpending()*
sigaction()	*sigfillset()*	*sigprocmask()*
sigaddset()	*sigismember()*	*sigsuspend()*
sigdelset()	*signal()*	

PORTABILITÄT

Die Include-Datei <*signal.h*> ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

alarm(), *kill()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *wait()*.

NAME

stdio.h - standard buffered input/output
Gepufferte Standar-Ein- und Ausgabe

DEFINITION

```
#include <stdio.h>
```

BESCHREIBUNG

Die Include-Datei *<stdio.h>* definiert die folgenden symbolischen Namen:

BUFSIZ	Größe der stdio-Puffer
EOF	Ergebniswert für Dateiende
FOPEN_MAX	Maximalzahl offener Datenströme
_IOFBF	voll gepufferte Ein-/Ausgabe
_IOLBF	zeilenweise gepufferte Ein-/Ausgabe
_IONBF	ungepufferte Ein-/Ausgabe
L_ctermid	Maximalgröße des Zeichenvektors für *ctermid()*-Ausgaben
L_cuserid	Maximalgröße des Zeichenvektors für *cuserid()*-Ausgaben
L_tmpnam	Maximalgröße des Zeichenvektors für *tmpnam()*-Ausgaben
NULL	Nullzeiger
SEEK_CUR	Suche relativ zur aktuellen Position
SEEK_END	Suche relativ zum Dateiende
SEEK_SET	Suche relativ zum Dateianfang
size_t	Ergebnistyp des *sizeof()*-Operators von C
stderr	Standardfehlerausgabe
stdin	Standardeingabe
stdout	Standardausgabe
TMP_MAX	Minimalzahl eindeutiger Dateinamen, die von *tmpnam()* generiert werden

Der folgende Datentyp wird mit *typedef* definiert:

FILE	Eine Struktur, die Informationen über eine Datei enthält

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

clearerr()	*fseek()*	*rename()*
fclose()	*ftell()*	*rewind()*
fdopen()	*fwrite()*	*scanf()*
feof()	*getc()*	*setbuf()*
ferror()	*getchar()*	*setvbuf()*
fflush()	*gets()*	*sprintf()*
fgetc()	*getw()*	*sscanf()*
fgets()	*pclose()*	*tempnam()*
fileno()	*perror()*	*tmpfile()*
fopen()	*popen()*	*tmpnam()*
fprintf()	*printf()*	*ungetc()*
fputc()	*putc()*	*vfprintf()*
fputs()	*putchar()*	*vprintf()*
fread()	*puts()*	*vsprintf()*
freopen()	*putw()*	
fscanf()	*remove()*	

HINWEIS

Die folgenden Namen können ebenfalls in dieser Include-Datei definiert sein, und sollten von Anwendungsentwicklern nur in Übereinstimmung mit den Definitionen anderer Schnittstellenbeschreibungen verwendet werden, sofern vorhanden.

FILENAME_MAX	*fgetpos()*	*fpos_t*
P_tmpdir	*fsetpos()*	

PORTABILITÄT

Die Include-Datei *<stdio.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

ctermid(), *cuserid()*, *fclose()*, *ferror()*, *fflush()*, *fgets()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *getopt()*, *gets()*, *getw()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*, *puts()*, *putw()*, *rename()*, *scanf()*, *setbuf()*, *stdin*, *system()*, *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vprintf()*.

NAME

stdlib.h - standard library definitions
Definitionen für die Standardbibliothek

DEFINITION

```
#include <stdlib.h>
```

BESCHREIBUNG

Die Include-Datei *<stdlib.h>* definiert die folgenden symbolischen Namen:

EXIT_FAILURE	Fehlerhafte Beendigung
EXIT_SUCCESS	Erfolgreiche Beendigung
NULL	Nullzeiger
RAND_MAX	Maximalwert für das Ergebnis von *rand()*

Der folgende Datentyp wird durch *typedef* definiert:

size_t	Ergebnistyp des C-Operators *sizeof()*

Die folgenden Namen sind entweder als Funktionen oder als Makros definiert:

abort()	*calloc()*	*rand()*
abs()	*exit()*	*realloc()*
atof()	*free()*	*srand()*
atoi()	*getenv()*	*strtod()*
atol()	*malloc()*	*strtol()*
bsearch()	*qsort()*	*system()*

HINWEIS

Die folgenden Namen können ebenfalls in dieser Include-Datei definiert sein und sollten von Anwendungs-Entwicklern nur in Übereinstimmung mit den Definitionen in anderen Schnittstellen-Beschreibungen verwendet werden, wo dies gegeben ist.

atexit()	*ldiv_t*	*wchar_t*
div()	*mblen*()	*wcstombs*()
div_t	*mbstowcs*()	*wctomb*()
labs()	*mbtowc*()	
ldiv()	*strtoul*()	

PORTABILITÄT

Die Include-Datei *<stdlib.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

atof(), *bsearch()*, *calloc()*, *free()*, *malloc()*, *qsort()*, *rand()*, *realloc()*, *srand()*, *strtod()*.

NAME

string.h - string operations
Zeichenketten-Operationen

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<string.h>* definiert den folgenden symbolischen Namen:

NULL Nullzeiger

und den folgenden Datentyp mit *typedef*:

size_t Vorzeichenloses ganzzahliges Ergebnis des *sizeof*-Operators in C.

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

memccpy()	*strcoll()*	*strncpy()*
memchr()	*strcpy()*	*strpbrk()*
memcmp()	*strcspn()*	*strrchr()*
memcpy()	*strerror()*	*strspn()*
memset()	*strlen()*	*strstr()*
strcat()	*strncat()*	*strtok()*
strchr()	*strncmp()*	*strxfrm()*
strcmp()		

SIEHE AUCH

memccpy(),	*memchr()*,	*memcmp()*,
memcpy(),	*memset()*,	*strcat()*,
strchr(),	*strcmp()*,	*strcoll()*,
strcpy(),	*strerror()*,	*strlen()*,
strpbrk(),	*strspn()*,	*strtok()*,
strxfrm().		

HINWEIS

Der Name *memmove()* kann in dieser Include-Datei definiert sein und sollte von Anwendungsentwicklern nur in Übereinstimmung mit den Definitionen in anderen Schnittstellenbeschreibungen verwendet werden, sofern vorhanden.

PORTABILITÄT

Die Include-Datei *<string.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

sys/ipc.h - inter-process communication access structure
Strukturen für Interprozeßkommunikation

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/ipc.h>* verwendet drei Mechanismen für die Interprozeßkommunikation (IPC): Nachrichten (messages), Semaphore und gemeinsam genutzer Hauptspeicher (shared memory). Alle verwenden einen gemeinsamen Strukturtyp *ipc_perm*, um die Informationen für die Ermittlung der Berechtigung für IPC-Operationen zu übergeben.

Die Struktur *ipc_perm* enthält die folgenden Komponenten:

```
uid_t     uid     Benutzernummer des Eigentümers
gid_t     gid     Gruppennummer des Eigentümers
uid_t     cuid    Benutzernummer des Erzeugers
gid_t     cgid    Gruppennummer des Erzeugers
mode_t    mode    Lese- und Schreibrechte
```

Die folgenden Konstanten werden definiert:

Zustandsbits:

```
IPC_CREAT  erzeuge Eintrag, wenn Schlüssel nicht existiert
IPC_EXCL   fehlschlagen, wenn Schlüssel existiert
IPC_NOWAIT Fehler, wenn die Anforderung warten muß
```

Schlüssel:

IPC_PRIVATE privater Schlüssel

Kontroll-Kommandos:

IPC_RMID Identifikator entfernen
IPC_SET Optionen setzen
IPC_STAT Optionen ermitteln

PORTABILITÄT

Die Include-Datei *<sys/ipc.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

sys/lpr.h - structures for liblpr
Strukturen und Definitionen für liblpr

DEFINITION

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

BESCHREIBUNG

Diese Include-Datei enthält Strukturen und Definitionen für aktuelle Angaben zum Spoolbetrieb. Mit den Funktionen zur C-Schnittstelle des Druckspools können die Dateien CONFIG.bin und POOLDAT gelesen werden, in denen diese Angaben enthalten sind. Die wichtigsten Strukturen sind unter den Funktionen selbst beschrieben (siehe *getcfadent*()-*getcftynam*() und *getpdjbent*()-*getpdprent*()). Im folgenden werden nur Definitionen zu Elementen aus den POOLDAT-Strukturen *prstat* und *jobpar* beschrieben.

Die Definitionen zu aktuellen Drucker-Zuständen in *st_cstate* aus der Struktur *prstat* lauten:

```
#define S_RDY      0    /* BEREIT */
#define S_POLL     1    /* POLL */
#define S_START    2    /* START AUSGABE */
#define S_RUN      3    /* LAEUFT */
#define S_CANC1    4    /* ABBRUCH 1 */
#define S_CANC2    5    /* ABBRUCH 2 */
#define S_CANC3    6    /* ABBRUCH 3 */
#define S_WAIT     7    /* WARTET */
#define S_INOP     8    /* GESTOERT */
#define S_LOCK1    9    /* GESPERRT 1 */
#define S_LOCK2   10    /* GESPERRT 2 */
#define S_EXCL    11    /* EXKL. BELEGT durch fremde Druckerverwal
#define S_TEST    12    /* PROBEDRUCK 1 (entspricht S_START) */
#define S_RTEST   13    /* PROBEDRUCK 2 (entspricht S_RUN) */
#define S_VOID1   14    /* UNBEKANNT */
#define S_VOID2   15    /* UNBEKANNT nach Zustand GESPERRT */

#define S_DMAX    16    /* max. Anzahl verschiedener Drucker-Zusta
```

Die Definitionen zu aktuell eingetretenen Ereignissen in *st_evcod* aus der Struktur *prstat* lauten:

```
#define E_AFN    0  /* Auftrag nicht ausgeführt */
#define E_AFP    1  /* Auftrag ausgeführt */
#define E_ANN    2  /* Auftrag abgelehnt */
#define E_ANP    3  /* Auftrag angenommen */
#define E_BDIE   4  /* das Backend hat sich beendet */
#define E_CANC   5  /* Abbruch des laufenden Auftrags durch das Backer
#define E_OF     6  /* Flag -of war gesetzt */
#define E_DD     7  /* Flag -dd war gesetzt */
#define E_DG     8  /* das Backend muß sich beenden */
#define E_DK     9  /* Flag -dk war gesetzt */
#define E_DU    10  /* Flag -du war gesetzt */
#define E_FILE  11  /* Druckauftrag für diesen Drucker vorhanden */
#define E_LD    12  /* das Backend wird geladen */
#define E_EX    13  /* das Backend wird beendet */
#define E_VEX   14  /* Flag -vex war gesetzt */
#define E_NOP   15  /* Dummy-Ereignis */
#define E_TEST  16  /* Beginn des Probedruck-Betriebs */
#define E_TIME  17  /* Alarmuhr ist abgelaufen */
#define E_TXT1  18  /* Nachricht vom Backend eingetroffen */

#define E_DMAX  19  /* max. Anzahl verschiedener Ereignisse */
```

Die Definitionen der Auftrags-Flags in *jb_flags* aus der Struktur *jobpar* lauten:

```
#define DP_COP   1  /* Kopie der auzudruckenden Datei wird gedruckt */
#define DP_DEL   2  /* Löschen der auszudruckenden Datei nach der */
                    /*  Druckausführung */
#define DP_MSG   4  /* Nachricht an den Auftraggeber nach der */
                    /*  Druckausführung */
#define DP_PIP   8  /* Auftragserteilung erfolgt über eine Pipe */
#define DP_TO   16  /* Auftrag wird an anderen Benutzer gesendet */
#define DP_WRK  32  /* Auftrag wird gerade ausgeführt */
```

PORTABILITÄT

Die Include-Datei *<sys/lpr.h>* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

getcfadent()-getcftynam(), getpdjbent()-getpdprent(), Kommando lpr.

NAME

sys/msg.h - message queue structures
Strukturen für Nachrichtenwarteschlangen

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/msg.h>* definiert die folgenden Konstanten und Komponenten der Struktur *msqid_ds*.

Kennzeichen für Nachrichtenbearbeitung:

MSG_NOERROR kein Fehler, wenn große Nachricht

Die Struktur *msqid_ds* enthält die folgenden Komponenten:

```
struct ipc_perm  msg_perm        Struktur für Erlaubnis der
                                 Operation
unsigned short   msg_qnum        aktuelle Anzahl der Nachrichte
                                 in der Warteschlange
unsigned short   msg_qbytes      maximalzahl der erlaubten
                                 Bytes in der Warteschlange
pid_t    msg_lspid       Prozeßnummer des letzten msgsnd()
pid_t    msg_lrpid       Prozeßnummer des letzten msgrcv()
time_t   msg_stime       Zeit des letzten msgsnd()
time_t   msg_rtime       Zeit des letzten msgrcv()
time_t   msg_ctime       Zeit der letzten Änderung
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

```
msgctl()        msgrcv()
msgget()        msgsnd()
```

PORTABILITÄT

Die Include-Datei *<sys/msg.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

msgctl(), *msgget()*, *msgrcv()*, *msgsnd()*.

NAME

sys/sem.h - semaphore facility
Semaphor-Strukturen

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/sem.h>* definiert die folgenden Konstanten und Strukturen.

Kennzeichen für Semaphor-Operationen:

SEM_UNDO	automatische Freigabe von Semaphoren bei Prozeßende

Kommandodefinitionen für die Funktion *semctl()*:

GETNCNT	semcnt ermitteln
GETPID	sempid ermitteln
GETVAL	semval ermitteln
GETALL	alle semvals ermitteln
GETZCNT	semzcnt ermitteln
SETVAL	semval setzen
SETALL	alle semvals setzen

Die Struktur *semid_ds* enthält die folgenden Komponenten:

struct ipc_perm	sem_perm	Berechtigungsstruktur
unsigned short	sem_nsems	Anzahl der Semaphore in der Menge
time_t	sem_otime	letzte Ausführung von *semop()*
time_t	sem_ctime	letzte Änderung durch *semctl*()

Die Anzahl der Semaphore einer Menge ist *sem_nsems*, innerhalb der Semaphormenge wird von 0 bis *sem_nsems-1* durchnumeriert. Die Nummer eines Semaphors heißt *sem_num*.

Ein Semaphor wird durch eine unbenannte Struktur repräsentiert, die die folgenden Komponenten enthält:

unsigned short	semval	Semaphorwert
pid_t	sempid	Prozeßnummer der letzen Operation
unsigned short	semncnt	Anzahl der Prozesse, die darauf warten, daß semval größer wird als es zur Zeit ist
unsigned short	semzcnt	Anzahl der Prozesse, die darauf warten, daß semval gleich 0 wird

Die Struktur *sembuf* enthält die folgenden Komponenten:

unsigned short	sem_num Semaphornummer
short	sem_op Semaphoroperation
short	sem_flg Operationskennzeichen

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

semctl() *semget()* *semop()*

PORTABILITÄT

Die Include-Datei *<sys/sem.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

semctl(), *semget()*, *semop()*.

NAME

sys/shm.h - shared memory facility

DEFINITION

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

BESCHREIBUNG

Die inlcude-Datei *<sys/shm.h>* definiert die folgenden Konstanten und eine Struktur.

Schalter für Operationen:

```
SHM_RDONLY  nur zum Lesen anhängen (sonst: Lesen und Schreiben)
SHMLBA      Vielfaches der Adresse der Segmentuntergrenze
SHM_RND     Anhängadresse auf SHMLBA aufrunden
```

Die Struktur *shmid_ds* enthält die folgenden Komponenten:

```
struct ipc_perm shm_perm    Berechtigungsstruktur
int             shm_segsz   Größe des Segments in Bytes
pid_t           shm_lpid    Prozeßnummer der letzen Operation
pid_t           shm_cpid    Prozeßnummer des Erzeugers
unsigned short  shm_nattch  Anzahl der aktuelle angehängten
                            Bereiche
time_t          shm_atime   Zeit des letzten shmat()
time_t          shm_dtime   Zeit des letzten shmdt()
time_t          shm_ctime   Zeit der letzten Änderung
                            durch shmctl()
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

shmat() *shmctl()* *shmdt()* *shmget()*

PORTABILITÄT

Die Include-Datei *<sys/shm.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

shmat(), *shmctl()*, *shmdt()*, *shmget()*.

NAME

sys/stat.h - data returned by stat function
Ergebnisse für stat()

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/stat.h>* definiert die Struktur, die von den Funktionen *stat()* und *fstat()* zurückgeliefert wird.

Die Struktur *stat* enthält mindestens die folgenden Komponenten:

dev_t	st_dev	Nummer des Geräts, das die Datei enthält
ino_t	st_ino	Dateinummer
mode_t	st_mode	Dateityp (siehe auch unten)
nlink_t	st_nlink	Anzahl der Verweise
uid_t	st_uid	Benutzernummer des Eigentümers
gid_t	st_gid	Gruppennummer des Eigentümers
dev_t	st_rdev	Gerätenummer (falls Gerätedatei)
off_t	st_size	Dateigröße in Bytes (nur normale Datei)
time_t	st_atime	letzte Zugriffszeit
time_t	st_mtime	letzte Modifikationszeit
time_t	st_ctime	Zeit der letzten Zustandsänderung

Auch die folgenden symbolischen Namen für Werte von *st_mode* werden definiert.

Dateityp:

S_IFMT	Dateityp
S_IFBLK	Datei für blockorientiertes Gerät
S_IFCHR	Datei für zeichenorientiertes Gerät
S_IFDIR	Dateiverzeichnis
S_IFIFO	FIFO-Gerätedatei
S_IFREG	normale Datei

Zugriffsrechte:

S_IRWXU	Lesen, Schreiben, Ausführen/Durchsuchen für Eigentümer
S_IRUSR	Lesen für Eigentümer
S_IWUSR	Schreiben für Eigentümer
S_IXUSR	Ausführen/Durchsuchen für Eigentümer
S_IRWXG	Lesen, Schreiben, Ausführen/Durchsuchen für Gruppe
S_IRGRP	Lesen für Gruppe
S_IWGRP	Schreiben für Gruppe
S_IXGRP	Ausführen/Durchsuchen für Gruppe
S_IRWXO	Lesen, Schreiben, Ausführen/Durchsuchen für übrige Benutzer
S_IROTH	Lesen für übrige Benutzer
S_IWOTH	Schreiben für übrige Benutzer
S_IXOTH	Ausführen/Durchsuchen für übrige Benutzer
S_ISUID	Benutzernummer bei Ausführung setzen (s-Bit für Eigentümer)

```
S_ISGID       Gruppennummer bei Ausführung setzen
              (s-Bit für Gruppe)
S_ISVTX       siehe unter HINWEIS

S_IREAD       ZURÜCKGEZOGEN
S_IWRITE      ZURÜCKGEZOGEN
S_IEXEC       ZURÜCKGEZOGEN
```

Dateityp-Testmakros:

```
S_ISBLK()       Test auf Gerätedatei für blockorientierte
                Geräte
S_ISCHR()       Test auf Gerätedatei für zeichenorientierte
                Geräte
S_ISDIR()       Test auf Dateiverzeichnis
S_ISFIFO()      Test auf FIFO-Gerätedatei
S_ISREG()       Test auf normale Datei
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

chmod() *mkfifo()*
fstat() *stat()*
mkdir() *umask()*

HINWEIS

Um den Typ einer Datei zu ermitteln, wird die Verwendung der Makros empfohlen.

Es sollte angemerkt werden, daß S_IREAD, S_IWRITE und S_IEXEC zurückgezogen wurden. Anwendungen sollten statt dessen S_IRUSR, S_IWUSR und S_IXUSR verwenden.

S_ISTVX war für das Sticky-Bit (save swapped text after use) reserviert. Diese Funktionalität ist jetzt veraltet.

PORTABILITÄT

Die Include-Datei *<sys/stat.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

chmod(), *fstat()*, *mkdir()*, *mkfifo()*, *stat()*, *umask()*, *<sys/types.h>*.

NAME

sys/termio.h - define values for termio and ioctl
Definition der Werte für termio und ioctl

DEFINITION

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

BESCHREIBUNG

Diese Include-Datei enthält die Definitionen für die von *ioctl()* verwendeten Strukturen und Namen (siehe dort).

PORTABILITÄT

Die Include-Datei *<sys/termio.h>* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

ioctl().

NAME

sys/times.h - file access und modification times structure
Struktur für Dateizeiten

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/times.h>* definiert die Struktur *struct tms*, die von *times()* zurückgeliefert wird. Diese schließt die folgenden Komponenten ein:

```
clock_t     tms_utime     Benutzerzeit
clock_t     tms_stime     Systemzeit
clock_t     tms_cutime    Benutzerzeit beendeter Sohnprozesse
clock_t     tms_cstime    Systemzeit beendeter Sohnprozesse
```

Der Datentyp *clock_t* wird durch ein *typedef* definiert.

Deklariert die Funktion *times()*.

PORTABILITÄT

Die Include-Datei *<sys/times.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

times().

NAME

sys/types.h - data types
Datentypen

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/types.h>* definiert Datentypen und schließt zumindest die folgenden Typen ein:

```
  clock_t   für die Systemzeit in der Maßeinheit CLK_TCK
  dev_t     für Gerätenummern
  gid_t     für Gruppennummern
  ino_t     für Dateinummern
* key_t     für die Interprozeßkommunikation
  mode_t    für einige Dateiattribute
  nlink_t   für den Verweiszähler
  off_t     für Dateigrößen
  pid_t     für Prozeßnummern
  size_t    für die Größe von Objekten
  time_t    für die Zeit in Sekunden
  uid_t     für Benutzernummern
```

Alle diese Datentypen, außer dem mit i*' markierten sind als arithmetische Typen einer angemessenen Größe definiert.

Zusätzlich gilt, daß *size_t* vorzeichenlos ist und *pid_t* ein Vorzeichen hat.

HINWEIS

Die folgenden Namen werden häufig als Ergänzung zu den oben angeführten Typen verwendet. Sie sind daher reserviert und portable Anwendungen sollten sie nicht verwenden:

addr_t caddr_t

PORTABILITÄT

Die Include-Datei *<sys/types.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

bsearch(), *chmod()*, *chown()*, *closedir()*, *creat()*, *fcntl()*, *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *getpgrp()*, *getpid()*, *getppid()*, *getuid()*, *kill()*, *lseek()*, *mkdir()*, *mkfifo()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *open()*, *opendir()*, *readdir()*, *rewinddir()*, *semctl()*, *semget()*, *semop()*, *setgid()*, *setpgid()*, *setsid()*, *setuid()*, *shmat()*, *shmctl()*, *shmdt()*, *shmget()*, *stat()*, *tcgetpgrp()*, *tcsetpgrp()*, *umask()*, *utime()*.

NAME

universe - options for change of universe
Schalter für den Wechsel eines Universums

DEFINITION

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

BESCHREIBUNG

Folgende symbolische Konstanten werden definiert:

```
U_ATT /* Wechsel in das att-Universum */
U_SIE  /* Wechsel in das sie-Universum */
U_UCB/* Wechsel in das ucb-Universum */
U_GET/* Abfragen des gegenwärtigen Universums */
```

PORTABILITÄT

Die Include-Datei *<sys/universe.h>* ist im X/Open-Standard nicht enthalten.

NAME

sys/utsname.h - system name structure
Struktur für Systemnamen

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/utsname.h>* definiert die Struktur *struct utsname*, welche die die folgenden Komponenten enthält:

```
char  sysname[]  Name der Implementierung des Betriebssystems
char  nodename[] Name dieses Knotens für ein
                 implementierungs-spezifisches Kommunikationsnet
char  release[]  aktuelle Releade-Nummer dieser Implementierung
char  version[]  aktuelle Version dieses Release
char  machine[]  Name des Hardwaretyps, auf dem das System läuft
```

Die Größe der Zeichen-Vektoren ist nicht festgelegt, aber die dort gespeicherten Daten werden durch das Nullbyte abgeschlossen.

Deklariert die Funktion *uname()*.

PORTABILITÄT

Die Include-Datei *<sys/utsname.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

uname().

NAME

sys/wait.h - declarations for waiting
Deklarationen für das Warten von Prozesses

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<sys/wait.h>* definiert die folgenden symbolischen Konstanten für die Verwendung in der Funktion *waitpid()*:

```
WNOHANG      nicht Warten, wenn kein Status verfügbar,
             sofort zurückkehren
WUNTRACED    Status eines angehaltenen Sohnprozesses liefern
```

und die folgenden Makros für die Analyse der Prozeß-Statuswerte:

```
WEXITSTATUS()    liefert Endestatus
WIFEXITED()      wahr, wenn Sohnprozeß normal beendet wurde
WIFSIGNALED()    wahr, wenn Sohnprozeß durch nicht
                 abgefangenes Signal beendet wurde
WIFSTOPPED()     Wahr, wenn Sohnprozeß derzeit angehalten ist
WSTOPSIG()       liefert die Signalnummer, die den Prozeß
                 angehalten hat
WTERMSIG()       liefert die Signalnummer, die den Prozeß
                 beendet hat
```

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

wait() *waitpid()*

PORTABILITÄT

Die Include-Datei *<sys/wait.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

wait().

NAME

termios.h - define values for termios
Werte für termios definieren

DEFINITION

```
#include <termios.h>
```

BESCHREIBUNG

Die Include-Datei *<termios.h>* enthält die Definitionen die von der *termios*-Schnittstelle verwendet werden (siehe *Abschnit 2.4, Allgemeine Terminalschnittstelle* zu den Strukturen und definierten Namen).

Es existieren verschiedene *unsigned*-Typdefinitionen für:

cc_t *speed_t* *tcflag_t*

Die Struktur *termios* enthält die folgenden Komponenten:

tcflag_t	c_iflag	Eingabeverarbeitung
tcflag_t	c_oflag	Ausgabeverarbeitung
tcflag_t	c_cflag	Hardware-Eigenschaften
tcflag_t	c_lflag	lokale Verarbeitung
cc_t	c_cc[NCCS]	Kontrollzeichen

Es wird eine Definition gegeben für:

NCCS Größe des Vektors c_cc für Kontroll-Zeichen

Die besonderen Kontroll-Zeichen sind durch den Vektor *c_cc* definiert:

Ersatzzeichen:

Standard-Eingabeverarbeitung	rohe	Beschreibung
VEOF		EOF-Zeichen
VEOL		EOL-Zeichen
VERASE		ERASE-Zeichen
VINTR	INTR	INTR-Zeichen
VKILL		KILL-Zeichen
	VMIN	MIN-Wert
VQUIT	VQUIT	QUIT-Zeichen
VSTART	VSTART	START-Zeichen
VSTOP	VSTOP	STOP-Zeichen
VSUSP	VSUSP	SUSP-Zeichen
	VTIME	TIME-Zeichen

Die Ersatzwerte sind eindeutig, außer daß VMIN und VTIME dieselben Werte wie VEOF und VEOL haben können.

Implementierungen, die die Option Auftragskontrolle nicht unterstützen, können den Wert für das Zeichen SUSP ignorieren, das im Vektor *c_cc* durch das Ersatzzeichen VSUSP indiziert wird.

Eingabeverarbeitung

Die Komponente *c_iflag* beschreibt die grundlegende Eingabekontrolle für Datensichtstationen:

BRKINT	Signal-Unterbrechung bei "break"
ICRNL	Bei der Eingabe CR auf NL abbilden
IGNBRK	"break"-Bedingung ignorieren
IGNCR	CR ignorieren
IGNPAR	Zeichen mit Paritätsfehlern ignorieren
INLCR	Bei der Eingabe NL auf CR abbilden
INPCK	Eingabe-Pariatätsprüfung einschalten
ISTRIPa	Löschzeichen
IUCLC	Bei der Eingabe Groß- auf Kleinbuchstaben abbilden
IXANY	Irgendein Zeichen als Startzeichen für Ausgabewiederholung freigeben
IXOFF	Start/Stop-Eingabekontrolle ermöglichen
IXON	Start/Stop-Ausgabekontrlolle ermöglichen
PARMRK	Paritätsfehler markieren

Ausgabeverarbeitung

Die Komponente *c_oflag* bestimmt, wie das System Ausgaben behandelt:

OPOST	Ausgabe nachbearbeiten
OLCUC	Bei Ausgabe Klein- in Großbuchstaben umwandeln
ONLCR	Bei Ausgabe NL in CR-NL umwandeln
OCRNL	Bei Ausgabe CR in NL umwandeln
ONOCR	Keine Ausgabe von CR bei Spalte 0
ONLRET	NL führt CR-Funktion aus
OFILL	Benutze Füllzeichen für Wartezeiten
OFDEL	Füllzeichen ist DEL, sonst NUL
NLDLY	Wartezeiten für NL wählen:
NL0	Neue-Zeile-Zeichen Typ 0
NL1	Neue-Zeile-Zeichen Typ 1
CRDLY	Wartezeiten für CR wählen:
CR0	CR-Wartezeit Typ 0
CR1	CR-Wartezeit Typ 1

CR2	CR-Wartezeit Typ 2
CR3	CR-Wartezeit Typ 3
TABDLY	Wartezeiten für Horizontal-Tabulatoren auswählen:
TAB0	Horizontal-Tabulator Wartezeit Typ 0
TAB1	Horizontal-Tabulator Wartezeit Typ 1
TAB2	Horizontal-Tabulator Wartezeit Typ 2
TAB3	Tabulatoren in Leerzeichen umwandeln
BSDLY	Wartezeiten für Rückschritt auswählen:
BS0	Rückschritt-Wartezeit Typ 0
BS1	Rückschritt-Wartezeit Typ 1
VTDLY	Wartezeiten für Vertikal-Tabulatoren auswählen:
VT0	Vertikal-Tabulator Wartezeit Typ 0
VT1	Vertikal-Tabulator Wartezeit Typ 1
FFDLY	Wartezeit für Seitenvorschub auswählen:
FF0	Seitenvorschub Wartezeit Typ 0
FF1	Seitenvorschub Wartezeit Typ 1

Auswahl der Baudrate

Die Eingabe- und Ausgabe-Baudraten sind in der Struktur *termios* abgespeichert. Diese sind gültige Werte für Objekte des Typs *speed_t*. Die folgenden Werte sind definiert, aber nicht alle Baudraten müssen von der zugrundeliegenden Hardware unterstützt werden:

B0	Verbindungsabbruch (Hang Up)
B50	50 Baud
B75	75 Baud
B110	110 Baud
B134	134.5 Baud
B150	150 Baud
B200	200 Baud
B300	300 Baud
B600	600 Baud
B1200	1200 Baud
B1800	1800 Baud
B2400	2400 Baud
B4800	4800 Baud
B9600	9600 Baud
B19200	19200 Baud
B38400	38400 Baud

Hardware-Eigenschaften

Die Komponente *c_cflag* beschreibt die Hardware-Kontrolle der Datensichtstation; nicht alle der beschriebenen Werte müssen von der zugrundeliegenden Hardware unterstützt werden:

CSIZE	Zeichengröße
CS5	5 Bit
CS6	6 Bit
CS7	7 Bit
CS8	8 Bit
CSTOPB	Zwei Stopbits senden, sonst eins
CREAD	Empfanger einschalten
PARENB	Paritätsprüfung einschalten
PARODD	Ungerade Parität, sonst gerade
HUPCL	Verbindungsabbruch bei letztem Schließen
CLOCAL	lokale Verbindung, sonst Wählverbindung

Lokale Verarbeitung

Die Komponente *c_lflag* der Argument-Struktur wird verwendet, um verschiedene Funktionen der Datensichtstation zu kontrollieren:

ECHO	Echo einschalten
ECHOE	Rückschritt durch Löschzeichen darstellen
ECHOK	KILL ausgeben
ECHONL	NL ausgeben
ICANON	Standard-Eingabeverarbeitung
IEXTEN	Besondere Eingabezeichen-Verarbeitung
ISIG	Signalverarbeitung einschalten
NOFLSH	Verwerfen nach Unterbrechung oder Ende ausschalten
TOSTOP	Signal SIGTTOU für Hintergrund-Ausgabe senden
XCASE	Kanonische Groß-/Kleinbuchstaben-Darstellung

Auswahl der Eigenschaften

Die folgenden symbolischen Konstanten für die Verwendung in *tcsetattr()* sind definiert:

TCSANOW	Eigenschaften sofort ändern
TCSADRAIN	Eigenschaften erst nach Abwarten der Ausgabe ändern
TCSAFLUSH	Eigenschaften erst nach Abwarten der Ausgabe ändern; zusätzlich anstehende Eingaben verwerfen

Leitungskontrolle

Die folgenden symbolischen Konstanten für die Verwendung in *tcflush()* sind definiert:

TCIFLUSH	anstehende Eingaben verwerfen
TCOFLUSH	nicht übertragene Ausgaben verwerfen
TCIOFLUSH	anstehende Eingaben und nicht übertragene Ausgaben verwerfen

Die folgenden symbolischen Konstanten für die Verwendung in *tcflow()* sind definiert:

TCIOFF	übertrage ein Stop-zeichen, um die Eingabe zu unterbinden
TCION	übertrage ein Start-Zeichen, um die Eingabe erneut zu starten
TCOOFF	unterbinde Ausgabe
TCOON	starte Ausgabe erneut

Die folgenden Namen sind entweder als Funktionen oder als Makros definiert:

cfgetispeed()	*tcdrain()*	*tcgetattr()*
cfgetospeed()	*tcflow()*	*tcsendbreak()*
cfsetispeed()	*tcflush()*	*tcsetattr()*
cfsetospeed()		

HINWEIS

Die folgenden Namen werden häufig als Erweiterung der oben angesprochenen benutzt. Sie sind daher reserviert und portable Anwendungen sollten sie nicht verwenden.

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

PORTABILITÄT

Die Include-Datei *<termios.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

cfgetispeed(), *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcgetattr()*, *tcsendbreak()*, *tcsetattr()*, *Abschnitt 2.4, Allgemeine Terminalschnittstelle.*

NAME

time.h - time types
Datentypen für Zeiten

DEFINITION

```
#include <time.h>
```

BESCHREIBUNG

Die Include-Datei *<time.h>* deklariert die Struktur *stuct tm*, welche zumindest die folgenden Komponenten enthält:

```
int   tm_sec        Sekunden [0, 61]
int   tm_min        Minuten [0, 61]
int   tm_hour       Stunde [0, 23]
int   tm_mday       Tag des Monats [1, 31]
int   tm_mon        Monat des Jahres [0, 11]
int   tm_year       Jahr seit 1900
int   tm_wday       Wochentag [0, 6] (Sonntag = 0)
int   tm_yday       Tag im Jahr [0, 365]
int   tm_isdst      Tageszeitkennzeichen
```

Diese Include-Datei definiert die folgenden symbolischen Namen:

```
NULL          Nullzeiger
CLK_TCK       Anzahl von clock ticks je Sekunde
```

und die folgenden Datentypen mittels *typedef*:

```
size_t      vorzeichenlose Ganzzahl für das Ergebnis des
            sizeof-Operators
time_t      arithmetischer Typ, der die Zeit in Sekunden
            darstellen kann
```

Der Wert von CLK_TCK kann variabel sein und man darf nicht annehmen, daß CLK_TCK eine Konstante zur Übersetzungszeit ist. Der Wert von CLK_TCK ist identisch mit dem Ergebnis von *sysconf(_SC_CLK_TCK)*.

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

asctime()	*localtime()*	*time()*
ctime()	*mktime()*	*tzset()*
gmtime()	*strftime()*	

Außerdem werden die folgenden Variablen deklariert:

daylight	*timezone*	*tzname*[]

HINWEIS

Die folgenden Namen können in dieser Include-Datei definiert sein und sollten von Anwendungs-Entwicklern nur in Übereinstimmung mit den Definitionen in anderen Schnittstellenbeschreibungen verwendet werden, sofern vorhanden:

clock()	*difftime()*

PORTABILITÄT

Die Include-Datei <*time.h*> ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

asctime(), *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *sysconf()*, *time()*, *tzset()*.

NAME

ulimit.h - ulimit commands
Kommandos für ulimit()

DEFINITION

```
#include <ulimit.h>
```

BESCHREIBUNG

Die Include-Datei <*ulimit.h*> definiert die symbolischen Konstanten, die für die Funktion *ulimit*() verwendet werden können.

Symbolische Konstanten:

UL_GETFSIZE	ermittle maximale Dateigröße
UL_SETFSIZE	setze maximale Dateigröße

Definiert den folgenden Namen entweder als Funktion oder Makro:

ulimit()

PORTABILITÄT

Die Include-Datei <*ulimit.h*> ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

ulimit().

NAME

unistd.h - standard symbolic constants und structures
Symbolische Standardkonstanten und Standardstrukturen

DEFINITION

```
#include <unistd.h>
```

BESCHREIBUNG

Die Include-Datei *<unistd.h>* definiert die symbolischen Konstanten und die Strukuren, die irgendwo in der Beschreibung verwendet, aber in keiner anderen Include-Datei definiert oder deklariert werden. Der Inhalt dieser Include-Datei wird unten gezeigt.

Die folgenden symbolischen Konstanten werden für die Funktion *access()* definiert:

```
R_OK      Test auf Leseerlaubnis
W_OK      Test auf Schreiberlaubnis
X_OK      Test auf Ausführungs-/Durchsucherlaubnis
F_OK      Test auf Existenz der Datei
```

Die Konstanten F_OK, R_OK, W_OK und X_OK und die Ausdrücke R_OK|W_OK, R_OK|X_OK und R_OK|W_OK|X_OK besitzen alle verschiedene Werte.

Deklariert die Konstante:

```
NULL      Nullzeiger
```

Die folgenden symbolischen Konstanten werden für die Funktion *lockf()* definiert (sie haben alle verschiedene Werte):

```
F_ULOCK   ZURÜCKGEZOGEN
F_LOCK    ZURÜCKGEZOGEN
F_TLOCK   ZURÜCKGEZOGEN
F_TEST    ZURÜCKGEZOGEN
```

Die folgenden symbolischen Konstanten werden für Pfadnamen definiert:

```
GF_PATH   ZURÜCKGEZOGEN
IF_PATH   ZURÜCKGEZOGEN
PF_PATH   ZURÜCKGEZOGEN
```

Die folgenden symbolischen Konstanten werden für die Funktionen *lseek()* und *fcntl()* definiert (sie haben eindeutige Werte):

```
SEEK_SET      Dateiposition gleich offset setzen
SEEK_CUR      Dateiposition gleich aktuelle Position
              plus offset setzen
SEEK_END      Dateiposition gleich EOF plus offset
              setzen
```

Die folgenden symbolischen Konstanten sind mit festen Werten definiert:

_POSIX_VERSION	Ganze Zahl, die die Version des Standards *IEEE Standard 1003.1-1988* angibt
_XOPEN_VERSION	Ganze Zahl, die die Version des XPG angibt, zu der das System kompatibel ist

Die folgenden symbolischen Konstanten werden definiert, wenn die entsprechende Option vorhanden ist:

_POSIX_CHOWN_RESTRICTED	Die Verwendung von *chown()* ist auf Prozesse mit besonderen Rechten beschränkt
_POSIX_JOB_CONTROL	Die Implementierung unterstützt Auftragskontrolle
_POSIX_NO_TRUNC	Pfadnamen-Komponenten die länger als {NAME_MAX} sind, erzeugen einen Fehler
_POSIX_SAVED_IDS	veranlaßt *exec*, die effektive Benutzer- und Gruppennummer zu sichern (ist auf allen XPG-kompatiblen Systemen definiert)
_POSIX_VDISABLE	die speziellen Zeichen für Datensichtstationen, die in *<termios.h>* definiert sind, können mit diesem Zeichen abgeschaltet werden.

Die folgenden symbolischen Konstanten werden für *sysconf()* definiert:

_SC_ARG_MAX	_SC_CHILD_MAX	_SC_CLK_TCK
_SC_JOB_CONTROL	_SC_NGROUPS_MAX	_SC_OPEN_MAX
_SC_PASS_MAX	_SC_SAVED_IDS	_SC_VERSION
_SC_XOPEN_VERSION		

Die folgenden symbolischen Konstanten werden für *pathconf()* definiert:

_PC_CHOWN_RESTRICTED	_PC_LINK_MAX
_PC_MAX_CANON	_PC_MAX_INPUT
_PC_NAME_MAX	_PC_NO_TRUNC
_PC_PATH_MAX	_PC_PIPE_BUF
_PC_VDISABLE	

Die folgenden symbolischen Konstanten werden für Datenströme definiert:

STDIN_FILENO	Dateinummer von *stdin*. Sie ist gleich 0.
STDOUT_FILENO	Dateinummer von *stout*. Sie ist gleich 1.
STDERR_FILENO	Dateinummer von *stderr*. Sie ist gleich 2.

Die folgenden Namen werden entweder als Funktionen oder als Makros deklariert:

access()	*execv()*	*getpgrp()*	*rmdir()*
alarm()	*execve()*	*getpid()*	*setgid()*
chdir()	*execvp()*	*getppid()*	*setpgid()*
chown()	*_exit()*	*getuid()*	*setsid()*
close()	*fork()*	*isatty()*	*setuid()*
ctermid()	*fpathconf()*	*link()*	*sleep()*
cuserid()	*getcwd()*	*lseek()*	*sysconf()*
dup2()	*getegid()*	*pathconf()*	*tcgetpgrp()*
dup()	*geteuid()*	*pause()*	*tcsetpgrp()*
execl()	*getgid()*	*pipe()*	*ttyname()*
execle()	*getgroups()*	*read()*	*unlink()*
execlp()	*getlogin()*	*rename()*	*write()*

HINWEIS

GF_PATH, PF_PATH und IF_PATH wurden aus der Beschreibung zurückgezogen, da auf einigen Systemen die Kennwort-, Gruppen- und Kopfinformation nicht als Dateien unterstützt wird. Anwendungen sollten diese Namen nicht benutzen, sondern *getpwnam()* und *getgrnam()* verwenden, um auf Benutzer- und Gruppendateien zuzugreifen.

Die folgenden Werte für Konstaanten sind für Systeme definiert, die zu dieser Ausgabe des Handbuchs kompatibel sind:

```
_POSIX_VERSION      198808L
_XOPEN_VERSION      3
```

PORTABILITÄT

Die Include-Datei *<unistd.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

access(), *alarm()*, *chdir()*, *chown()*, *close()*, *ctermid()*, *cuserid()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fpathconf()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *getlogin()*, *getpgrp()*, *getpid()*, *getppid()*, *getuid()*, *isatty()*, *kill()*, *link()*, *lseek()*, *open()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *rmdir()*, *setgid()*, *setpgid()*, *setsid()*, *setuid()*, *sleep()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*, *ttyname()*, *unlink()*, *utime()*, *write()*, *<limits.h>* #, *<termios.h>*.

NAME

ustat.h - file system statistis
Statistische Angaben zum Dateisystem

DEFINITION

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

BESCHREIBUNG

Die Include-Datei *<ustat.h>* definiert den Datentyp *struct ustat*, der mindestens die folgenden Komponenten besitzt:

```
daddr_t  f_tfree;      /* freie Bloecke insgesamt */
ino_t    f_tinode;     /* freie Indexeintraege insgesamt */
char     f_fname[6];   /* Name d. Dateisystems */
char     f_fpack[6];   /* Name d. Dateisystemstapels */
```

PORTABILITÄT

Die Include-Datei *<ustat.h>* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

ustat().

NAME

utime.h - access und modification times structure
Zeitstrukturen manipulieren

DEFINITION

```
#include <utime.h>
```

BESCHREIBUNG

Die Include-Datei *<utime.h>* deklariert die Struktur *utimbuf*, welche die folgenden Komponenten besitzt:

time_t	actime	Zugriffszeit
time_t	modtime	Modifikationszeit

Die Zeiten werden in Sekunden seit dem Epochenwert gemessen. Der Typ *time_t* ist in *<sys/types.h>* deklariert.

Sie deklariert außerdem die Funktion *utime()*.

PORTABILITÄT

Die Include-Datei *<utime.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

utime(), *<sys/types.h>*.

NAME

values.h - machine dependent values
Hardwareabhängige Werte

DEFINITION

```
#include <values.h>
```

BESCHREIBUNG

Diese Include-Datei enthält eine Reihe von vorgegebenen, hardwareabhängigen Konstanten.

Bei ganzen Zahlen (int) wird Binärdarstellung (Einer- oder Zweierkomplement-Darstellung) angenommen, wobei der Wert des höchstwertigen Bits das Vorzeichen darstellt.

BITSPERBYTE
: Anzahl Bits pro Byte

BITS(*type*)
: Anzahl Bits im angegebenen Datentyp (z.B. int)

HIBITS
: Kleiner ganzzahliger Wert (Datentyp short int), bei dem nur das höherwertige Bit gesetzt ist (in den meisten Implementierungen 0x8000)

HIBITL
: Großer ganzzahliger Wert (Datentyp long int), bei dem nur das höherwertige Bit gesetzt ist (in den meisten Implementierungen 0x80000000)

HIBITI
: Regulärer ganzzahliger Wert (Datentyp int), bei dem nur das höherwertige Bit gesetzt ist (normalerweise identisch mit HIBITS oder HIBITL)

MAXSHORT
: Größtmöglicher Wert vom Typ short int mit Vorzeichen (in den meisten Implementierungen 0x7FFF (= 32767)

MAXLONG
: Größtmöglicher Wert vom Typ long int mit Vorzeichen (in den meisten Implementierungen 0x7FFFFFFF (= 2147483647)

MAXINT
: Größtmöglicher Wert vom Typ int mit Vorzeichen (in den meisten Implementierungen identisch mit MAXSHORT oder MAXLONG)

DMAXEXP
: Größtmöglicher Exponent eines Gleitkommawerts mit doppelter Genauigkeit (Datentyp double)

FMAXEXP
Größtmöglicher Exponent eines Gleitkommawerts mit einfacher Genauigkeit (Datentyp float)
DMINEXP
Kleinster Exponent einer double
FMINEXP
Kleinster Exponent einer float
DMAXPOWTWO
Größte genau (ohne Runden) als double darstellbare Zweierpotenz
FMAXPOWTWO
Größte genau (ohne Runden) als float darstellbare Zweierpotenz
_FEXPLEN
Anzahl Bits für den Exponenten einer float
_EXPBASE
Exponentenbasis
_DEXPLEN
Anzahl Bits für den Exponenten einer double
_IEEE
Bei Verwendung der IEEE-Standard-Darstellung: 1
_LENBASE
Anzahl Bits in der Exponentenbasis
_HIDDENBIT
Bei implizitem höchstwertigem Bit in der Mantisse: 1 Größte genau (ohne Runden) als double darstellbare Zweierpotenz
FSIGNIF
Anzahl signifikante Bits in der Mantisse einer einfach genauen Gleitkommazahl
DSIGNIF
Anzahl signifikante Bits in der Mantisse einer doppelt genauen Gleitkommazahl
MAXFLOAT, LN_MAXFLOAT
Größtmöglicher Wert einer einfach genauen Gleitkommazahl und ihr natürlicher Logarithmus
MAXDOUBLE, LN_MAXDOUBLE
Größtmöglicher Wert einer doppelt genauen Gleitkommazahl und ihr natürlicher Logarithmus
MINFLOAT, LN_MINFLOAT
Kleinster positiver Wert einer einfach genauen Gleitkommazahl und ihr natürlicher Logarithmus
MINDOUBLE, LN_MINDOUBLE
Kleinster positiver Wert einer doppelt genauen Gleitkommazahl und ihr natürlicher Logarithmus

PORTABILITÄT

Die Include-Datei <*values.h*> ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

<*limits.h*>, <*math.h*>.

NAME

varargs.h - handle variable argument list
Variable Argumentliste behandeln

DEFINITION

```
#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;
```

BESCHREIBUNG

Die Include-Datei *<varargs.h>* enthält einige Makros, die es erlauben portable Prozeduren mit variablen Argumentlisten zu schreiben. Routinen, die variable Argumentlisten haben (wie z.B. *printf()*) , die aber nicht *varargs* benutzen, sind an sich nicht portabel, da verschiedene Rechner verschiedene Konventionen für die Argumentübergabe verwenden.

va_alist wird als Parameterliste in einem Funktionskopf verwendet

va_dcl ist eine Deklaration für *va_alist*. Auf *va_dcl* sollte kein Semikolon folgen.

va_list ist ein Datentyp, der für die Variable definiert ist, welche die Liste abarbeitet.

va_start() wird aufgerufen, um *pvar* auf den Anfang der Liste zu initialisieren.

va_arg() liefert das nächste Argument aus der Liste, auf die *pvar* zeigt. *type* ist der Typ, der für das Argument angenommen wird. Verschiedene Typen können gemischt werden, aber die Routine muß wissen, welchen Typ das nächste Argument hat, da dies zur Laufzeit nicht festgestellt werden kann.

va_end() wird zum "Aufräumen" verwendet.

Mehrere Abarbeitungen sind möglich; jede wird durch *va_start*

... *va_end* eingeschlossen.

BEISPIEL

Dieses Beispiel ist eine mögliche Implementierung von *execl()*.

```
#include <varargs.h>

#define MAXARGS    100

/*   execl wird so aufgerufen:
 *      execl(file, arg1, arg2, ..., (char *)0);
 */
execl(va_alist)
va_dcl
{       va_list ap;
        char *file;
        char *args[MAXARGS];
        int argno = 0;

        va_start(ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
           ;
        va_end(ap);
        return execv(file, args);
}
```

HINWEIS

Die aufrufende Routine muß angeben, wieviele Argumente es gibt, da es nicht immer möglich ist, dies aus dem Inhalt des Stacks zu bestimmen. Für *execl()* zum Beispiel wird als Kennzeichen für das Listenende ein Nullzeiger übergeben. Die Funktion *printf()* ermitelt die Zahl der Argumente aus dem ersten Argument (Format).

Es ist nicht portabel, ein zweites Argument des Typs *char*, *short* oder *float* für *va_arg* anzugeben, da es für die gerufene Funktion keine Argumente des Typs *char*, *short* oder *float* gibt. Die Sprache C konvertiert Argumente vom Typ *char* und *short* zu *int* und Argumente des Typs *float* zu *double*, bevor es sie an eine Funktion übergibt.

PORTABILITÄT

Die Include-Datei *<varargs.h>* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

exec, *printf()*.

A Anhang

dezi-mal	oktal	hexa-dez.		Bedeutung	Control
0	00	00	NUL	Null, keine Operation	@
1	01	01	SOH	Start of Heading Vorspannanfang	A
2	02	02	STX	Start of Text Textanfang	B
3	03	03	ETX	End of Text Textende	C
4	04	04	EOT	End of Transmission Taste END Übertragungsende	D
5	05	05	ENQ	Enquiry Stationsanruf	E
6	06	06	ACK	Acknowledge Bestätigung	F
7	07	07	BEL	Bell Klingel	G
8	10	08	BS	Backspace Korrekturtaste	H
9	11	09	HT	Horizontal Tabulation Tabulatorzeichen	I
10	12	0A	LF	Line Feed Zeilenvorschub, neue Zeile	J
11	13	0B	VT	Vertical Tabulation	K
12	14	0C	FF	Form Feed Formularvorschub	L
13	15	0D	CR	Carriage Return Wagenrücklauf	M
14	16	0E	SO	Shift Out Umschalten Zeichensatz	N
15	17	0F	SI	Shift In Zurückschalten Zeichensatz	O
16	20	10	DLE	Data Link Escape Austritt aus der Datenverbindung	P
17	21	11	DC1	Device Control 1 Gerätesteuerung 1, Ausgabe fortsetzen	Q
18	22	12	DC2	Device Control 2	R
19	23	13	DC3	Device Control 3 Ausgabe anhalten	S
20	24	14	DC4	Device Control 4	T
21	25	15	NAK	Negative Acknowledge Fehlermeldung	U
22	26	16	SYN	Synchronous Idle Synchronisierung	V
23	27	17	ETB	End of Transm. Block Datenblockende	W
24	30	18	CAN	Cancel ungültig, Zeilenlöscher	X
25	31	19	EM	End of Medium Datenträgerende, quit (Signal3)	Y
26	32	1A	SUB	Substitute Character Zeichen ersetzen	Z

dezi-mal	oktal	hexa-dez.		Bedeutung		Control
27	33	1B	ESC	Escape	Rücksprung	
28	34	1C	FS	File Separator	Dateitrennung	\
29	35	1D	GS	Group Separator	Gruppentrennung	]
30	36	1E	RS	Record Separator	Satztrennung	
31	37	1F	US	Unit Separator	Einheitentrennung	ø oder DEL
32	40	20	SP	SPACE	Leerzeichen	
33	41	21	!			
34	42	22	"			
35	43	23	#	Nummernzeichen		
36	44	24	$	oder nationales Währungssymbol		
37	45	25	%			
38	46	26	&			
39	47	27	'			
40	50	28	(			
41	51	29	)			
42	52	2A	*	(Stern gilt oft als Multiplikations-zeichen)		
43	53	2B	+			
44	54	2C	,			
45	55	2D	-			
46	56	2E	.			
47	57	2F	/	(dient als Divisionszeichen)		
48	60	30	0			
49	61	31	1			
50	62	32	2			
51	63	33	3			
52	64	34	4			
53	65	35	5			
54	66	36	6			
55	67	37	7			
56	70	38	8			
57	71	39	9			
58	72	3A	:			
59	73	3B	;			
60	74	3C	<			
61	75	3D	=			
62	76	3E	>			
63	77	3F	?			
64	100	40	@	(kaufmännisches "at" oder §)		
65	101	41	A			
66	102	42	B			
67	103	43	C			
68	104	44	D			
69	105	45	E			
70	106	46	F			
71	107	47	G			
72	110	48	H			
73	111	49	I			
74	112	4A	J			
75	113	4B	K			
76	114	4C	L			
77	115	4D	M			

dezi-mal	oktal	hexa-dez.		Bedeutung	Control
78	116	4E	N		
79	117	4F	O		
80	120	50	P		
81	121	51	Q		
82	122	52	R		
83	123	53	S		
84	124	54	T		
85	125	55	U		
86	126	56	V		
87	127	57	W		
88	130	58	X		
89	131	59	Y		
90	132	5A	Z		
91	133	5B	[	oder Ä	
92	134	5C	\	Gegenschrägstrich oder Ö	
93	135	5D	]	oder Ü	
94	136	5E	^	oder ↑	
95	137	5F	-	Unterstrich oder >	
96	140	60	`		
97	141	61	a		
98	142	62	b		
99	143	63	c		
100	144	64	d		
101	145	65	e		
102	146	66	f		
103	147	67	g		
104	150	68	h		
105	151	69	i		
106	152	6A	j		
107	153	6B	k		
108	154	6C	l		
109	155	6D	m		
110	156	6E	n		
111	157	6F	o		
112	160	70	p		
113	161	71	q		
114	162	72	r		
115	163	73	s		
116	164	74	t		
117	165	75	u		
118	166	76	v		
119	167	77	w		
120	170	78	x		
121	171	79	y		
122	172	7A	z		
123	173	7B	{	oder ä	
124	174	7C	|	oder ö	
125	175	7D	}	oder ü	
126	176	7E	~	oder ß	
127	177	7F	DEL	Delete Löschzeichen, Interrupt (Signal2)	

Abkürzungen

Die englischsprachige Bezeichnung ist in Klammern gesetzt.

<datei>	
<sys/datei>	Abkürzende Schreibweise für Include-Dateien, deren vollständiger Pfadname lautet: */usr/include/datei* bzw. */usr/include/sys/datei*
bzw.	beziehungsweise
CPU	Zentraleinheit (central process unit)
d.h.	das heißt
DVZ	Dateiverzeichnis
E/A	Ein-/Ausgabe
egid	effektive Gruppennummer (effective group ID)
EOF	Datei-Ende, in <stdio.h> definiert (End Of File)
euid	effektive Benutzernummer (effective user ID)
FIFO	Abkürzung für first-in-first-out
GID, gid	Gruppennummer, gemeint ist die reale Gruppennummer ((real) group ID)
GMT	Abkürzung für Greenwich Mean Time
ID	Identifikations-Nummer (identification)
I/O	Ein-/Ausgabe (input/output)
i. allg.	im allgemeinen
i.d.R.	in der Regel
MET	Abkürzung für Mean European Time
MEZ	Mitteleuropäische Zeit
o.g.	oben genannt
PID, pid	Prozeßnummer (Process ID)
PPID, ppid	Vaterprozeßnummer (Parent Process ID)
r-Bit	Schutzbit für Leseberechtigung (read)
s.	siehe
s-Bit	Schutzbit, um einem Programmanwender die effektive Benutzernummer des Programmeigentümers während des Programmablaufs zu verleihen (set-user-ID Bit)
sbe-Bit	"schließe-bei-exec-Bit"; früher verwandte Bezeichnung für das close-on-exec-Bit
s.o.	siehe oben
sog.	sogenannte, -r, -s
s.u.	siehe unten
t-Bit	Sticky-Bit
usw.	und so weiter

UID, uid	Benutzernummer, gemeint ist die reale Benutzernummer ((real) user ID)
u.U.	unter Umständen
vgl.	vergleiche
w-Bit	Schutzbit für Schreibberechtigung (write)
wg.	wegen
x-Bit	Schutzbit für Ausführberechtigung (execute)
z.B.	zum Beispiel

Fachwörter englisch - deutsch

absolute pathname	absoluter Pfadname
access	Zugriff
access mode	Zugriffsart
access permission	Zugriffsberechtigung
action	Aktion
address	Adresse
address space	Adreßraum
alarm clock	Alarmuhr
alignment bytes	Füllbytes
APPLICATION USAGE	HINWEIS
appropriate privileges	besondere Rechte
archive	Bibliothek
argument	Argument
assignment	Zuweisung
back space key	Korrekturtaste
backgoound process group	Hintergrund-Prozeßgruppe
backslash	Gegenschrägstrich
base adress	Basisadresse
baud rate	Baudrate
binary file	Binärdatei
binary operator	zweistelliger Operator
block special device	blockorientiertes Gerät
block special file	Gerätedatei für b.o. Gerät
break	break-Anweisung
broken-down time	aufgeschlüsselte Zeit
buffer area	Pufferbereich
buffer	Puffer
buffered I/O	gepufferte Ein- und Ausgabe
byte	Byte = 8 Bit
call	aufrufen, Aufruf
canonical mode input processing	Standard-Eingabeverarbeitung
cast	Typumwandlung mit dem Cast-Operator
cast	cast-Operator
catch	abfangen (Signale)
category	Kategorie
change	ändern
character set	Zeichensatz,Zeichenvorrat
character special device	zeichenorientiertes Gerät
character special file	Gerätedatei für z.o. Gerät
character	Zeichen

child process	Sohnprozeß
cleanup	Bereinigung,Aufräumarbeiten
close-on-exec-bit	sbe-Bit (schließe bei exec)
code in use	aktuell gültiger Zeichensatz
code set	Zeichensatz
collating element	Zeicheneinheit
collating sequence	Sortierreihenfolge
collating symbol	Zeicheneinheits-Symbol
command interpreter	Kommandointerpreter
command prompt	Eingabeaufforderung
compile time	Übersetzungzeit
completion	Beendigung
concatenate	verketten
concatenation	Verkettung
console	Konsole
control character	Kontrollzeichen
control mode (Terminal Interface)	Hardware-Eigenschaften
controlling process	kontrollierender Prozeß
controlling terminal	kontrollierendes Terminal
core (dump|file|image)	Speicherabzug
currency symbol	Währungszeichen
current	aktuell
current working directory	aktuelles Verzeichnis
data transmission	Datenübertragung
decimal	Dezimal-, zur Basis 10
default	Standard-
delete	löschen,entfernen
description	Beschreibung
device	Gerät
device driver	Gerätetreiber
device ID	Gerätenummer
diagnostics	Diagnoseteile
digit groupng symbol	Tausendertrennzeichen
digit	Ziffer
directory	Dateiverzeichnis
directory entry	Dateiverzeichniseintrag
directory stream	Dateiverzeichnis-Strom
disconnect the line	die Verbindung abbrechen
disk	Festplatte
double floating point	doppelt genaue Gleitpunktzahl
driver	Treiber
duration	Dauer
effective group ID	effektive Gruppennummer
effective user ID	effektive Benutzernummer

empty directory	leeres Dateiverzeichnis
empty string	leere Zeichenkette
end-of-file	Dateiende
environment	Umgebung
environment variables	Umgebungsvariablen
Epoch	Epochenwert (1.1.1070, 0:00 Uhr)
equivalence class	Äquivalenzklasse
error	Fehler
error condition	Fehlerbedingung
error handling	Fehlerbehandlung
error number	Fehlernummer
escape character	Fluchtsymbol
escape	entwerten
escape sequence	Escape-Sequenz
execute	ausführen
execute permission	Ausführberechtigung
exit status	Endestatus
expression	Ausdruck
extended security controls	erweiterte Sicherheitskontrollen
FIFO	FIFO
FIFO special file	FIFO-Gerätedatei
file	Datei
file access permissions	Zugriffsrechte für Dateien
file description	Datei-Beschreibung
file descriptor	Dateikennzahl
file descriptor flag	Prozeß-Dateistatus-Byte
file flag	System-Dateistatus-Byte
file group class	Benutzerklasse Gruppe
file hierarchy	Dateibaum
file mode	Dateistatus
file name	Dateiname
file offset	Dateiposition
file other class	Benutzerklasse übrige Benutzer
file owner class	Benutzerklasse Eigentümer
file permission bits	Schutzbits der Datei
file pointer	Dateizeiger
file serial number	Dateinummer
FILE structure	FILE-Struktur
file system	Dateisystem
file times update	Änderung der Dateizeiten
flag	Anzeiger,Schalter,Bit
floating point (number)	Gleitkommazahl,-punktzahl
floppy disk	Diskette

foreground process group	Vordergrund-Prozeßgruppe
foreground process group ID	Vordergrund-Prozeßgruppennr.
formatted I/O	formatierte Ein- und Ausgabe
free	freigeben
functionatity	Funktionalität
group	Gruppe
group ID	Gruppennummer
group name	Gruppenname
header (file)	Include-Datei
hexadecimal	hexadezimal
home directory	HOME-Dateiverzeichnis
implementation	Implemenierung, implementierungs-
include	einbinden, -schließen
include-file	Include-Datei
initialize	initialisieren
input	Eingabe
input mode	Eingabeverarbeitung
input stream	Eingabestrom
integer	ganze Zahl, Ganzzahl
inter-process communication	Interprozeßkommunikation
interface	Schnittstelle
interrupt	Unterbrechung, unterbrechen
issue	Ausgabe
job control	Auftragskontrolle
key	Schlüssel,Taste
link	Verweis
link count	Verweiszähler
local mode (Terminal Interface)	lokale Verarbeitung
locale	internationale Umgebung
macro	Makro
mandatory	obligatorisch
match	passen,gefunden
memory	Speicher(platz)
message (NLS)	Meldung
message catalogue	Meldungskatalog
message catalogue descriptor	Meldungskatalog-Deskriptor
mode (of a file)	Dateistatus
mount	einhängen
NaN (Not a Number)	NaN
native language	Landessprache

new process image file	neue Abbilddatei des Prozesses
nice value	Priorität
node	Knoten,Struktur
non-canonical mode input processing	rohe Eingabeverarbeitung
non-spacing character	Leerzeichen ohne Schreibmarkenbewegung
null byte (\0)	Nullbyte
null character	Nullbyte
null pointer	Nullzeiger
null string	leere Zeichenkette
null terminated string	mit Nullbyte abgeschlossene Zeichenkette
octal	oktal
offset	Offset,Abstand
open file	offene Datei
open file description	geöffnete Dateibeschreibung
open file decriptor	offene Dateikennzahl
ordinary file	Textdatei
orphaned	verwaist
output	Ausgabe
output mode	Ausgabeverarbeitung
output stream	Ausgabestrom
overflow	Überlauf
owner	Eigentümer
parent directory	übergeordnetes Dateiverzeichnis
parent process	Vaterprozeß
parent process ID	Prozeßnummer des Vaterprozesses
path	Pfad
path prefix	Pfadnamen-Anfang
pathname	Pfadname
pathname component	Pfadnamen-Komponente
pathname resolution	Pfadnamen-Auflösung
pattern	Muster
pipe	Pipe
pointer	Zeiger
portable	portabel
portable filename character set	Zeichensatz für portable Dateinamen
portable pathname	portabler Pfadname
preprocessor	Präprozessor
privilege	Recht
process	Prozeß

process group	Prozeßgruppe
process group ID	Prozeßgruppennummer
process group leader	Prozeßgruppenchef
process group lifetime	Lebensdauer einer Prozeßgruppe
process ID	Prozeßnummer
process image	Prozeßabbild
process lifetime	Lebensdauer eines Prozesses
process termination	Prozeßende
program message	Meldungstext
protection bit	Schutzbit
radix character	Dezimalpunkt
read	lesen
read-only file system	nur zum Lesen eingehängtes Dateisystem
read permission	Leseerlaubnis
real group ID	reale Gruppennummer
real user ID	reale Benutzernummer
regular file	normale Datei
regular expression	regulärer Ausdruck
relative pathname	relativer Pfadname
released	freigegeben
request	Anfrage
result	Ergebnis
resume	fortsetzen
return	zurückliefern,-kehren,Rückkehr
root-directory	Root-Dateiverzeichnis
run time	Laufzeit
saved set-group-ID	gesicherte Gruppennummer
saved set-user-ID	gesicherte Benutzernummer
schedule	steuern,festlegen
screen	Bildschirm
search	durchsuchen
search permission	Durchsuchberechtigung
security mechanism	Sicherheitsmechanismus
security policy	Sicherheitsverfahren
session	Sitzung
session leader	Sitzungsführer
session lifetime	Lebensdauer einer Sitzung
set-group-ID mode bit	s-Bit für die Gruppe
set-user-ID mode bit	s-Bit für den Eigentümer
shared text	gemeinsamer Text
sign	Vorzeichen
signal	Signal

signal mask	Signalmaske
size	Größe
slash	Schrägstrich
space character	Zwischenraumzeichen
space	Zwischenraum
special character	Sonderzeichen
special file	Gerätedatei
static data area	statischer Datenbereich
stream	Strom
string	Zeichenkette
structure	Struktur
superuser (root)	Superuser (root)
supplementary group ID	weitere Gruppennummer
suspend (a process)	anhalten (einen Prozeß
synopsis	Definition
system	System
system administrator	Systemverwalter
system call	Systemaufruf
system process	Systemprozeß
system time	Systemzeit
tab	Tabulator(zeichen)
temporary	temporär
terminal (device)	Datensichtstation
territory	Gebiet
timezone	Zeitzone
transmission	Übertragung
transmit	übertragen
underflow	Unterlauf
underscore	Unterstrich
user	Benutzer
user ID	Benutzernummer
user time	Benutzerzeit
value	Wert
variable	variabel, Variable
white space character	Zwischenraumzeichen
withdrawn	zurückgezogen
working directory	aktuelles Dateiverzeichnis
write permission	Schreiberlaubnis
zombie process	Zombieprozeß

Fachwörter deutsch - englisch

abfangen (Signale)	catch
absoluter Pfadname	absolute pathname
Adresse	address
Adreßraum	address space
ändern	change
Änderung der Dateizeiten	file times update
Äquivalenzklasse	equivalence class
Aktion	action
aktuell	current
aktuell gültiger Zeichensatz	code in use
aktuelles Dateiverzeichnis	working directory
aktuelles Verzeichnis	current working directory
Alarmuhr	alarm clock
Anfrage	request
anhalten (einen Prozeß	suspend (a process)
Anzeiger,Schalter,Bit	flag
Argument	argument
aufgeschlüsselte Zeit	broken-down time
aufrufen, Aufruf	call
Auftragskontrolle	job control
Ausdruck	expression
Ausführberechtigung	execute permission
ausführen	execute
Ausgabe	issue
Ausgabe	output
Ausgabestrom	output stream
Ausgabeverarbeitung	output mode
Basisadresse	base adress
Baudrate	baud rate
Beendigung	completion
Benutzer	user
Benutzerklasse Eigentümer	file owner class
Benutzerklasse Gruppe	file group class
Benutzerklasse übrige Benutzer	file other class
Benutzernummer	user ID
Benutzerzeit	user time
Bereinigung,Aufräumarbeiten	cleanup
Beschreibung	description
besondere Rechte	appropriate privileges
Bibliothek	archive

Bildschirm	screen
Binärdatei	binary file
blockorientiertes Gerät	block special device
break-Anweisung	break
Byte = 8 Bit	byte
cast-Operator	cast
Datei	file
Datei-Beschreibung	file description
Dateibaum	file hierarchy
Dateiende	end-of-file
Dateikennzahl	file descriptor
Dateiname	file name
Dateinummer	file serial number
Dateiposition	file offset
Dateistatus	file mode
Dateistatus	mode (of a file)
Dateisystem	file system
Dateiverzeichnis	directory
Dateiverzeichnis-Strom	directory stream
Dateiverzeichniseintrag	directory entry
Dateizeiger	file pointer
Datensichtstation	terminal (device)
Datenübertragung	data transmission
Dauer	duration
Definition	synopsis
Dezimal-, zur Basis 10	decimal
Dezimalpunkt	radix character
Diagnoseteile	diagnostics
die Verbindung abbrechen	disconnect the line
Diskette	floppy disk
doppelt genaue Gleitpunktzahl	double floating point
Durchsuchberechtigung	search permission
durchsuchen	search
effektive Benutzernummer	effective user ID
effektive Gruppennummer	effective group ID
Eigentümer	owner
einbinden, -schließen	include
Eingabe	input
Eingabeaufforderung	command prompt
Eingabestrom	input stream
Eingabeverarbeitung	input mode
einhängen	mount
Endestatus	exit status

entwerten	escape
Epochenwert (1.1.1070, 0:00 Uhr)	Epoch
Ergebnis	result
erweiterte Sicherheitskontrollen	extended security controls
Escape-Sequenz	escape sequence
Fehler	error
Fehlerbedingung	error condition
Fehlerbehandlung	error handling
Fehlernummer	error number
Festplatte	disk
FIFO	FIFO
FIFO-Gerätedatei	FIFO special file
FILE-Struktur	FILE structure
Fluchtsymbol	escape character
formatierte Ein- und Ausgabe	formatted I/O
fortsetzen	resume
freigeben	free
freigegeben	released
Funktionalität	functionatity
Füllbytes	alignment bytes
ganze Zahl, Ganzzahl	integer
Gebiet	territory
Gegenschrägstrich	backslash
gemeinsamer Text	shared text
gepufferte Ein- und Ausgabe	buffered I/O
Gerät	device
Gerätedatei	special file
Gerätedatei für b.o. Gerät	block special file
Gerätedatei für z.o. Gerät	character special file
Gerätenummer	device ID
Gerätetreiber	device driver
gesicherte Benutzernummer	saved set-user-ID
gesicherte Gruppennummer	saved set-group-ID
geöffnete Dateibeschreibung	open file description
Gleitkommazahl,-punktzahl	floating point (number)
Gruppe	group
Gruppenname	group name
Gruppennummer	group ID
Größe	size
Hardware-Eigenschaften	control mode (Terminal Interface)
hexadezimal	hexadecimal

Hintergrund-Prozeßgruppe	backgoound process group
HINWEIS	APPLICATION USAGE
HOME-Dateiverzeichnis	home directory
Implemenierung, implementierungs-	implementation
Include-Datei	header (file)
Include-Datei	include-file
initialisieren	initialize
internationale Umgebung	locale
Interprozeßkommunikation	inter-process communication
Kategorie	category
Knoten,Struktur	node
Kommandointerpreter	command interpreter
Konsole	console
kontrollierender Prozeß	controlling process
kontrollierendes Terminal	controlling terminal
Kontrollzeichen	control character
Korrekturtaste	back space key
Landessprache	native language
Laufzeit	run time
Lebensdauer einer Prozeßgruppe	process group lifetime
Lebensdauer einer Sitzung	session lifetime
Lebensdauer eines Prozesses	process lifetime
leere Zeichenkette	empty string
leere Zeichenkette	null string
leeres Dateiverzeichnis	empty directory
Leerzeichen ohne Schreibmarkenbewegung	non-spacing character
Leseerlaubnis	read permission
lesen	read
lokale Verarbeitung	local mode (Terminal Interface)
löschen,entfernen	delete
Makro	macro
Meldung	message (NLS)
Meldungskatalog	message catalogue
Meldungskatalog-Deskriptor	message catalogue descriptor
Meldungstext	program message
mit Nullbyte abgeschlossene Zeichenkette	null terminated string
Muster	pattern
NaN	NaN (Not a Number)

neue Abbilddatei des Prozesses	new process image file
normale Datei	regular file
Nullbyte	null byte (\0)
Nullbyte	null character
Nullzeiger	null pointer
nur zum Lesen eingehängtes Dateisystem	read-only file system
obligatorisch	mandatory
offene Datei	open file
offene Dateikennzahl	open file decriptor
Offset,Abstand	offset
oktal	octal
passen,gefunden	match
Pfad	path
Pfadname	pathname
Pfadnamen-Anfang	path prefix
Pfadnamen-Auflösung	pathname resolution
Pfadnamen-Komponente	pathname component
Pipe	pipe
portabel	portable
portabler Pfadname	portable pathname
Priorität	nice value
Prozeß	process
Prozeß-Dateistatus-Byte	file descriptor flag
Prozeßabbild	process image
Prozeßende	process termination
Prozeßgruppe	process group
Prozeßgruppenchef	process group leader
Prozeßgruppennummer	process group ID
Prozeßnummer	process ID
Prozeßnummer des Vaterprozesses	parent process ID
Präprozessor	preprocessor
Puffer	buffer
Pufferbereich	buffer area
reale Benutzernummer	real user ID
reale Gruppennummer	real group ID
Recht	privilege
regulärer Ausdruck	regular expression
relativer Pfadname	relative pathname
rohe Eingabeverarbeitung	non-canonical mode input processing
Root-Dateiverzeichnis	root-directory
s-Bit für den Eigentümer	set-user-ID mode bit
s-Bit für die Gruppe	set-group-ID mode bit

sbe-Bit (schließe bei exec)	close-on-exec-bit
Schlüssel,Taste	key
Schnittstelle	interface
Schreiberlaubnis	write permission
Schrägstrich	slash
Schutzbit	protection bit
Schutzbits der Datei	file permission bits
Sicherheitsmechanismus	security mechanism
Sicherheitsverfahren	security policy
Signal	signal
Signalmaske	signal mask
Sitzung	session
Sitzungsführer	session leader
Sohnprozeß	child process
Sonderzeichen	special character
Sortierreihenfolge	collating sequence
Speicher(platz)	memory
Speicherabzug	core,(dump|file|image)
Standard-	default
Standard-Eingabeverarbeitung	canonical mode input processing
statischer Datenbereich	static data area
steuern,festlegen	schedule
Strom	stream
Struktur	structure
Superuser (root)	superuser (root)
System	system
System-Dateistatus-Byte	file flag
Systemaufruf	system call
Systemprozeß	system process
Systemverwalter	system administrator
Systemzeit	system time
Tabulator(zeichen)	tab
Tausendertrennzeichen	digit groupng symbol
temporär	temporary
Textdatei	ordinary file
Treiber	driver
Typumwandlung mit dem Cast-Operator	cast
übergeordnetes Dateiverzeichnis	parent directory
Überlauf	overflow
Übersetzungzeit	compile time
übertragen	transmit
Übertragung	transmission
Umgebung	environment

Umgebungsvariablen	environment variables
Unterbrechung, unterbrechen	interrupt
Unterlauf	underflow
Unterstrich	underscore
variabel, Variable	variable
Vaterprozeß	parent process
verketten	concatenate
Verkettung	concatenation
verwaist	orphaned
Verweis	link
Verweiszähler	link count
Vordergrund-Prozeßgruppe	foreground process group
Vordergrund-Prozeßgruppennr.	foreground process group ID
Vorzeichen	sign
weitere Gruppennummer	supplementary group ID
Wert	value
Währungszeichen	currency symbol
Zeichen	character
Zeicheneinheit	collating element
Zeicheneinheits-Symbol	collating symbol
Zeichenkette	string
zeichenorientiertes Gerät	character special device
Zeichensatz	code set
Zeichensatz für portable Dateinamen	portable filename character set
Zeichensatz,Zeichenvorrat	character set
Zeiger	pointer
Zeitzone	timezone
Ziffer	digit
Zombieprozeß	zombie process
Zugriff	access
Zugriffsart	access mode
Zugriffsberechtigung	access permission
Zugriffsrechte für Dateien	file access permissions
zurückgezogen	withdrawn
zurückliefern,-kehren,Rückkehr	return
Zuweisung	assignment
zweistelliger Operator	binary operator
Zwischenraum	space
Zwischenraumzeichen	space character
Zwischenraumzeichen	white space character

Literatur

SINIX-Handbücher

Die mit * gekennzeichneten Titeln sind nicht von der Siemens AG herausgegeben.

[1] Betriebssystem SINIX V5.22
Kommandos

[2] Betriebssystem SINIX V5.22
SINIX-Schnittstellen
Benutzerhandbuch

[3] Betriebssystem SINIX V5.22
Einführung

[4] Betriebssystem SINIX V5.22
Systemverwaltung

[5] Internationalisation in SINIX
Benutzerhandbuch

X/OPEN Guide

* [6] X/OPEN Portability Guide (December 1988)
Issue 3, Volumes 1 - 7
Eaglewood Cliffs: Prentice Hall 1989

Literatur zu UNIX

* T. Baggenstos, R. Marty u.a.
UNIX als Basis für Softwareentwicklung
Berlin, Heidelberg, New York, Tokyo: Springer 1986

* S. R. Bourne
Das UNIX System
o.O.: Addison-Wesley Verlag (Deutschland) GmbH 1985

* Jürgen Gulbins
UNIX
Berlin: Springer-Verlag , 2. Auflage 1985

* M. J. Rochkind
Advanced UNIX Programming
Englewood Cliffs: Prentice Hall 1985

* R. Thomas, L. R. Rogers, J. L. Yates
Advanced Programmer's Guide To UNIX System V
Berkeley: Osborne McGraw-Hill 1986

* M. J. Bach
The Design Of The UNIX System
Englewood Cliffs: Prentice Hall 1986

Literatur zur Sprache C

* H. Herold, W. Unger
Das C-Buch
München: te-wi Verlag 1986

B. W. Kernighan, D. M. Ritchie
Programmieren in C
Zweite Ausgabe - ANSI C
Coedition Hanser/Prentice Hall 1990
Sonderausgabe SIEMENS AG 1990

C. L. Tondo, S. E. Gimpel
Das C-Lösungsbuch zu Kernighan/Ritchie, Programmieren in C
Zweite Ausgabe - ANSI C
Coedition Hanser/Prentice Hall 1990
Sonderausgabe SIEMENS AG 1990

* T. Plum
Das C-Lernbuch
München, Wien: Hanser, London: Prentice Hall 1985

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis Datentechnik*. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen Datentechnik*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Siemens-Zweigniederlassung; außerhalb der Bundesrepublik Deutschland hilft Ihnen die zuständige Siemens-Vertretung.

Stichwörter

3-Byte-Ganzzahl in long int umwandeln 3-425
7-Bit US-ASCII-Zeichen, Test auf 3-398
<assert.h> 3-39, 4-8
<ctype.h> 4-9
<dirent.h> 4-10
<errno.h> 2-5, 4-11
<fcntl.h> 4-12f
<ftw.h> 4-15
<grp.h> 4-16
<langinfo.h> 4-17
<limits.h> 4-20ff
<lokale.h> 4-25
<malloc.h> 3-55
<math.h> 3-42, 3-408, 4-27
<memory.h> 3-473ff
<nl_types.h> 3-56f, 4-31
<pwd.h> 4-32
<regexp.h> 4-33
<search.h> 3-54, 4-34
<setjmp.h> 4-35
<signal.h> 4-36
<stdio.h> 3-53, 4-39
<stdlib.h> 3-52ff, 4-41
<string.h> 4-43
<sys/ipc.h> 4-43f
<sys/msg.h> 4-47
<sys/sem.h> 4-48
<sys/shm.h> 4-50
<sys/stat.h> 1-7f, 1-16, 4-51
<sys/times.h> 4-54
<sys/types.h> 3-346f, 4-55
<sys/utsname.h> 4-57
<sys/wait.h> 4-58
<termios.h> 3-62, 4-59
<time.h> 4-64
<ulimit.h> 4-66
<unistd.h> 3-296, 4-67
<utime.h> 4-71
<values.h> 4-72
<varargs.h> 3-821, 4-75
[n, m] 1-21
_exit() 3-256f

_IOFBF 4-39
_IOLBF 4-39
_IONBF 4-39
_PC_CHOWN_RESTRICTED 3-527, 4-68
_PC_LINK_MAX 3-527, 4-68
_PC_MAX_CANON 3-527, 4-68
_PC_MAX_INPUT 3-527, 4-68
_PC_NAME_MAX 3-527, 4-68
_PC_NO_TRUNC 3-527, 4-68
_PC_PATH_MAX 3-527, 4-68
_PC_PIPE_BUF 3-527, 4-68
_PC_VDISABLE 3-527, 4-68
_POSIXHNAME_MAX 4-22
_POSIX_ARG_MAX 4-21f
_POSIX_CHILD_MAX 4-21f
_POSIX_CHOWN_RESTRICTED 3-527, 4-68
_POSIX_JOB_CONTROL 1-4, 4-68
_POSIX_LINK_MAX 4-21f
_POSIX_MAX_CANON 4-21f
_POSIX_MAX_INPUT 4-21f
_POSIX_NAME_MAX 4-21
_POSIX_NGROUPS_MAX 4-22
_POSIX_NO_TRUNC 1-13, 4-68
_POSIX_OPEN_MAX 4-21f
_POSIX_PATH_MAX 4-21f
_POSIX_PIPE_BUF 4-21f
_POSIX_SAVED_IDS 4-68
_POSIX_VDISABLE 3-527, 4-68
_POSIX_VERSION 4-68f
_SC_ARG_MAX 4-68
_SC_CHILD_MAX 4-68
_SC_CLK_TCK 4-68
_SC_JOB_CONTROL 4-68
_SC_NGROUPS_MAX 4-68
_SC_OPEN_MAX 4-68
_SC_PASS_MAX 4-68
_SC_SAVED_IDS 4-68
_SC_VERSION 4-68
_SC_XOPEN_VERSION 4-68
_tolower() 3-783
_toupper() 3-786
_XOPEN_VERSION 4-68f

a64l() 3-23
Abbilddatei 3-248

Abfangen eines Signals 3-654
Abfragen der Dateikennzahl 3-284
abort() 3-25, 3-39, 4-41
Abrundungsfunktion 3-285
abs() 3-26, 4-41
Absolutbetrag
  einer ganzen Zahl 3-26
  einer Zahl bestimmen 3-260
absoluter Pfadname 1-12
Abwarten des Endes eines Sohnprozesses 3-824
access 3-27
access() 3-27
acos() 3-30
addch() (*curses*) 3-92
addr_t, Definition 4-55
addstr() (*curses*) 3-94
Adresse
  initialisierter Datenbereich 3-235
  nicht initialisierter Datenbereich 3-237
  Programmcode 3-247
Adreßraum 1-4, 1-14f, 3-49
advance() 3-32, 3-581, 3-584
Ändern
  der Dateiposition 3-455
  der Dateizugriffsrechte 3-68
  der Prozeßpriorität 3-509
  der Signalmaske 3-673
  von Gruppe und Eigentümer einer Datei 3-71
  von Signal-Aktionen 3-650
  von Umgebungsvariablen 3-562
Änderung
  der Dateizeiten 1-8
  lineare 3-452
Aktivieren
  des Bildschirmlöschens (*curses*) 3-106
  des Funktionstastenblocks (*curses*) 3-138
Aktualisieren
  des Fensters (*curses*) 3-179
  eines Pad (*curses*) 3-174
Aktualisierung, effiziente (*curses*) 3-212
aktuellen Datensichtstationsmodus speichern (*curses*) 3-109
aktuelles Dateiverzeichnis 1-7, 1-13
  ermitteln 3-335
  wechseln 3-66
alarm() 3-33

Alarmsignal festsetzen 3-33
allgemeine Bildschirmbehandlung (*curses*) 2-60
Anfügen einer Zeichenkette 3-703
  teilweise 3-717
Angeben der verbrauchten Rechenzeit 3-77
Anhängen eines gemeinsamen Speicherbereichs 3-642
Anhalten
  eines Prozesses bis Signal eintrifft 3-530
  eines Prozesses für festgesetzte Zeitspanne 3-683
  von Datenübertragung oder -empfang 3-746
Anlegen
  einer neuen Datei 3-86
  einer Semaphormenge 3-612
  eines gemeinsamen Speichersegments 3-647
anormaler Prozeßabbruch 3-25
anstehendes Signal 3-652
  prüfen 3-672
Anzahl der Zeichen in einer Zeile 2-17
Archive, (catopen()) 3-58
Arcus
  Cosinus 3-30
  Sinus 3-37
  Tangens 3-40
Argumentliste
  formatierte Ausgabe 3-822
  variable behandeln 3-821, 4-75
Argumentvektor, Optionen ermitteln 3-351
argv[], Optionen ermitteln 3-351
ARG_MAX 3-250, 4-21
Arten von Eingabeverarbeitung 2-16
ASCII-Code A-1
asctime() 3-35
asin() 3-37
assert() 3-39
  Definition 4-8
  -Makro 4-8
asynchrone
  Datensichtstationen (*curses*) 2-62
  Leitung, lokal 2-13
atan() 3-40
atan2() 3-42
atexit() 4-41
atof() 3-44, 4-41
atoi() 3-45, 4-41
atol() 3-46, 4-41

att-Universum 3-812
Attribute für Fenster behandeln 3-96
attroff() (*curses*) 3-96
attron() (*curses*) 3-96
attrset() (*curses*) 3-96
audiovisuelle Signale (*curses*) 3-101
auf ein Signal warten 3-677
Aufrundungsfunktion 3-60
Aufspalten
  einer Gleitkommazahl 3-490
  einer Zeichenkette in Teile 3-728
Auftragskontrolle 1-4
  Prozeßgruppennummer setzen 3-632
aus Datei lesen 3-572
Ausführen
  einer Datei 3-248
  eines Kommandos 3-739
Ausführungsprofil 4-30
Ausführungsrecht 1-16
Ausgabe
  binäre 3-324
  (*curses*) 2-60, 2-63
  einer Argumentliste, formatierte 3-822
  formatierte 3-297, 3-543
  in Fenster, formatiert (*curses*) 3-176
  unterdrücken (*curses*) 3-116
  vom Übersetzer und Binder 4-2
  verzögert (*curses*) 3-110
Ausgabe-Baudrate
  ermitteln 3-63
  festlegen 3-65
Ausgabe-Übertragung abwarten 3-745
Ausgabefunktionen (*curses*) 2-60
Ausgeben
  einer Zeichenkette auf einen Datenstrom 3-300
  eines Maschinenworts auf Datenstrom 3-566
  eines Zeichens auf Datenstrom 3-298
  eines Zeichens auf einen Datenstrom 3-560
  Zeichen in Fenster (*curses*) 3-92
Ausschalten
  des Blätterns (*curses*) 3-186
  des Raw-Modus (*curses*) 3-178

B0 2-29, 4-61
B110 2-29, 4-61

B1200 2-29, 4-61
B134 2-29, 4-61
B150 2-29, 4-61
B1800 2-29, 4-61
B19200 2-29, 4-61
B200 2-29, 4-61
B2400 2-29, 4-61
B300 2-29, 4-61
B38400 2-29, 4-61
B4800 2-29, 4-61
B50 2-29, 4-61
B600 2-29, 4-61
B75 2-29, 4-61
B9600 2-29, 4-61
Baudrate 3-62ff
  für Verbindungsabbruch 4-61
  Konstanten 4-61
baudrate() (*curses*) 3-100
Bedeutung der Schriftart 1-3
Beenden eines Prozesses 3-256
beep() (*curses*) 3-101
Behandeln einer variablen Argumentliste 3-821, 4-75
Benutzer- und Gruppennummer 2-52
Benutzerdatei
  Eintrag für Benutzernummer ermitteln 3-367
  Eintrag für Namen ermitteln 3-366
Benutzereintrag 3-796
Benutzerkennung ermitteln 3-447
Benutzerklasse
  Eigentümer 1-4, 1-20
  Gruppe 1-4,1-20
  übrige Benutzer 1-4, 1-20
Benutzernamen ermitteln 3-349
Benutzernummer 1-5, 1-9, 3-362f
  effektive 1-5
  effektive ermitteln 3-339
  gesicherte 1-5
  reale 1-5
  reale ermitteln 3-370
  setzen 3-637
Bereinigen eines Datenstrom-Puffers 3-276
besondere Rechte 1-5, 1-8
Bessel-Funktionen
  der ersten Art 3-419
  der zweiten Art 3-832

Bestimmen des Absolutbetrags einer Zahl 3-260
Betriebsart
  speichern, Datensichtstation (*curses*) 3-181
  zurücksetzen, Datensichtstation (*curses*) 3-180f
Bibliothek, mathematische 1-24, 3-468
Bildlauf durchführen (*curses*) 3-185
Bildlaufbereich einstellen (*curses*) 3-188
Bildschirmbehandlung, allgemeine (*curses*) 2-60
Bildschirmlöschen aktivieren (*curses*) 3-106
Bildschirmschnittstellen-Definition (*curses*) 2-54
Bildschirmsteuerung mit curses 2-54
Binäre Ausgabe 3-302, 3-324
Binären Suchbaum
  durchlaufen 3-797
  durchsuchen 3-763
  verwalten 3-790
  Knoten entfernen 3-759
Binder, Ausgabe 4-2
Blättern ein-/ausschalten (*curses*) 3-186
Blockieren eines Signals 3-652
blockiertes Signal 3-652
blockorientiertes Gerät 1-9
boolesches termcap-Feld ermitteln 3-766
box() (*curses*) 3-102
break 3-48
brk() 3-48
BRKINT 2-24, 4-60
BS0 4-61
BS1 4-61
BSDLY 4-61
bsearch() 3-52f, 4-41
Buchstabe
  oder Ziffer, Test auf 3-394
  Test auf 3-396
BUFSIZ 4-39
Byte(s)
  im Speicher finden 3-474
  im Speicher initialisieren 3-477
  im Speicher kopieren 3-473, 3-476
  im Speicher vergleichen 3-475
  vertauschen 3-735

C Standard 1-5
C-Bibliotheksfunktionen zur Bildschirmbehandlung (*curses*) 2-54
caddr_t, Definition 4-55

calloc() 3-55, 4-41
catclose() 3-56
catgets() 3-57
catopen() 3-58f
cbreak() (*curses*) 3-104
CBREAK-Modus (*curses*) 3-104
cc_t 4-59
ceil() 3-60
cfgetispeed() 3-62
cfgetospeed() 3-63
cfsetispeed() 3-64
cfsetospeed() 3-65
CHAR_BIT 4-23
CHAR_MAX 4-23f
CHAR_MIN 4-23f
chdir() 3-66
CHILD_MAX 4-21
chmod() 1-19f, 3-68f
chown 3-71f
chroot() 3-74
clear() (*curses*) 3-105
clearerr() 3-76
clearok() (*curses*) 3-106
CLK_TCK 1-5, 4-23f
  Definition 4-64
CLOCAL 2-29, 4-62
  Öffnen einer Gerätedatei für Datensichtstation 2-14
Clock Tick 1-5
clock() 3-77
clock_t 1-5
  Definition 4-55
close() 3-78f
close(), Löschen von Verweisen 2-2
closedir() 3-80, 4-10
clrtobot() (*curses*) 3-107
clrtoeol() (*curses*) 3-108
compile() 3-81, 3-581ff
cos() 3-82
cosh() 3-84
Cosinus 3-82
  Hyperbolicus 3-84
CR 2-22
CR0 4-60
CR1 4-60
CR2 4-60

CR3 4-60
CRDLY 4-60
CREAD 4-62
creat() 1-19, 3-86
CRNCYSTR 4-19
crypt() 3-87
CS5 2-29, 4-62
CS6 2-29, 4-62
CS7 2-29, 4-62
CS8 2-29, 4-62
CSIZE 2-29, 4-62
CSTOPB 2-29, 4-62
ctermid() 3-89
ctime() 3-90
curses, Bildschirmsteuerung 2-54
Cursorpositionierung (*termcap*) vorbereiten 3-772
cuserid() 3-219
c_cc[]
  Größe 4-59
  Indexnamen 2-34
  Vektorgröße 2-34

Datei 1-5
  ausführen 3-248
  einrichten 3-482
  kontrollieren 3-265
  lesen 3-572
  normale 1-5, 1-7
  öffnen 3-517
  offene 1-7
  Position ändern 3-455
  schreiben 3-828
  Schutzbits 1-16
  umbenennen 3-588
  Verweis einrichten 3-432
  Zugriffsrechte prüfen 3-27
Datei-Beschreibung, offene 1-6, 1-14
Dateianfang der Gruppendatei, positionieren auf 3-623
Dateibaum 1-6, 1-12f
  durchlaufen 3-320
  durchsuchen (<ftw.h>) 4-15
Dateibereich
  fremde Sperren 3-438
  (gesperrten), freigeben 3-438
  sperren 3-438

Dateibeschreibung
offene 2-2
Verweis auf 2-2
Dateieigenschafts-Struktur 1-8
Dateien löschen 3-587
Dateiende, Datenstrom prüfen 3-274
Dateihierarchie 1-6
Dateiinformation ermitteln 3-314
Dateikennzahl 1-6f, 3-56
abfragen 3-284
duplizieren 3-230
erzeugen 2-2
schließen 3-78
Dateimodus 1-7, 1-16
Dateiname 1-6f,1-12, 1-15
einer Datensichtstation ermitteln 3-794
erzeugen 3-486
für temporäre Datei erzeugen 3-761, 3-780
portabler Zeichensatz 1-19
Dateinummer 1-7
Dateiposition 1-6, 1-14
ändern 3-455
auf Anfang setzen 3-592
Dateistatus ermitteln 3-694
Dateisystem 1-7
aushängen 3-492, 3-805
Informationen ermitteln 3-697
statistische Angaben 4-70
zum Lesen 1-7
Dateisysteminformationen ermitteln 3-815
Dateiverzeichnis 1-5ff, 1-12ff
aktuelles 1-7
einrichten 3-482, 3-485
ermitteln, aktuelles 3-335
erzeugen 3-478
leeres 1-7
löschen 3-595
öffnen 3-524
offenes 1-8
Operationen 3-225
Root 1-15
übergeordnetes 1-17
wechseln 3-66
Dateiverzeichnis-Strom schließen 3-80

Dateiverzeichniseintrag 1-7, 1-17
  Format 4-10
  lesen 3-577
Dateiverzeichnisstrom 1-8
  Position ermitteln 3-760
  positionieren 3-608
Dateizeiten
  setzen 3-817
  Struktur 4-54
Dateizeiten-Änderung 1-8
Dateizugriffsrechte ändern 3-68
Dateizustand synchronisieren 3-316
Daten- und Textbereich, Platzbedarf 3-48
Datenbereich
  initialisierter 3-235
  nicht initialisierter 3-237
Datenempfang neu starten oder anhalten 3-746
Datensegment
  Größe ändern 3-48, 3-597
  Speicherplatz für dynamisch ändern 3-48
Datensichtstation 1-4, 1-8, 1-10f
  ähnliche 2-89
  Betriebsarten speichern (*curses*) 3-181
  Betriebsarten zurücksetzen (*curses*) 3-180f
  (*curses*) 2-62
  Dateiname ermitteln 3-794
  Gerätedatei 1-9
  Initialisierung 2-84
  Name ermitteln (*curses*) 3-142
  neu eröffnen (*curses*) 3-164
  Öffnen der Gerätedatei 2-13
  Parameter zurücksetzen (*curses*) 3-117
  Test auf 3-399
  Übertragungsgeschwindigkeit (*curses*) 3-100
  Umgebung initialisieren (*curses*) 3-133
  umschalten zwischen (*curses*) 3-187
  zugeordnete Parameter lesen 3-750
  zugeordnete Parameter setzen 3-755
Datensichtstations-Beschreibungen erstellen 2-73
Datensichtstations-Eigenschaften
  grundlegende 2-73
  Typen 2-71
Datensichtstationsmodus (aktuellen), speichern (*curses*) 3-109
Datenstrom
  auf Dateiende prüfen 3-274

Dateikennzahl abfragen 3-284
erzeugen 2-2
Fehlerkennzeichen prüfen 3-275
Maschinenwort ausgeben 3-566
öffnen 3-289, 3-305
Position liefern 3-317
Positionszeiger neu positionieren 3-311
Puffer zuweisen 3-618
Pufferung zuordnen 3-640
schließen 3-262
Zeichen ausgeben 3-298, 3-560
Zeichen lesen 3-278, 3-329
Zeichenkette ausgeben 3-300
zu gegebener Dateikennzahl zuordnen 3-272
Datenstrom-Puffer bereinigen 3-276
Datenstrukturen
(IPC) 2-51
sog. Fenster (*curses*) 2-54
Datentyp
cc_t 4-59
DIR 4-10
für Zeiten 4-64
<nl_types.h> 4-31
size_t 4-41
speed_t 4-59
struct dirent 4-10
struct termios 4-59
tcflag_t 4-59
Datenübertragung neu starten oder anhalten 3-746
Datum und Zeit in Zeichenkette umwandeln 3-35, 3-90, 3-713
daylight 3-221, 3-799
DBL_DIG 4-23
DBL_MAX 4-23, 4-27
DBL_MIN 4-23
Definieren einer Signalbehandlung 3-666
Definitionen für die Standardbibliothek 4-41
def_prog_mode() (*curses*) 3-109
def_shell_mode() (*curses*) 3-109
Deklarationen
für das Warten von Prozesses 4-58
für reguläre Ausdrücke 4-33
für Stackumgebung 4-35
mathematische 4-27
delay_output() (*curses*) 3-110
delch() (*curses*) 3-111

deleteln() (*curses*) 3-113
delwin() (*curses*) 3-114
dev_t, Definition 4-55
dial() 3-222
DIR 4-10
Distanzfunktion, euklidische 3-384
div() 4-41
div_t 4-41
doupdate() (*curses*) 3-212
Draft ANSI X3.159 Programming Language C 1-5
drand48() 3-226
Druckaufträge, C-Funktionen 3-356
Druckbares Zeichen, Test auf 3-409
Drucken einer formatierten Ausgabe 3-297
Druckerbeschreibung, C-Funktionen 3-331
Druckspool, C-Schnittstelle 3-331, 3-356
dup() 3-230f
  Erzeugen von Verweisen 2-2
dup2() 3-230f
Duplizieren
  einer offenen Dateikennzahl 3-230
  einer Zeichenkette 3-710
Durchführen des Bildlaufs (*curses*) 3-185
Durchlaufen
  eines binären Suchbaums 3-797
  eines Dateibaums 3-320
Durchsuchen
  einer Zeichenkette 3-704, 3-722
  eines binären Suchbaums 3-763
  eines Dateibaums 4-15
  von Tabellen, <search.h> 4-34
Durchsucherlaubnis 1-20
Durchsuchrecht 1-16
d_ino 4-10
d_name[] 4-10

E2BIG 2-5, 4-11
EACCES 2-5, 4-11
EAGAIN 2-5, 4-11
EBADF 2-6, 4-11
EBUSY 2-6, 4-11
ECHILD 2-6, 4-11
ECHNL 4-62
ECHO 2-31, 4-52
echo() (*curses*) 3-116

Echo, einlesen ohne 3-354
ECHOE 2-31, 4-62
ECHOK 2-31, 4-62
ECHONL 2-31
ecvt() 3-233
edata 3-235
EDEADLK 2-6, 4-11
EDOM 2-6, 4-11
EEXIST 2-6, 4-11
EFAULT 2-10, 4-11
EFBIG 2-6, 4-11
Effekte bei Funktionen, Signale 3-654
effektive Benutzernummer 1-5
  ermitteln 3-339
effektive Gruppennummer 1-10, 1-18
  ermitteln 3-336
effiziente Aktualisierung eines Fensters (*curses*) 3-212
EIDRM 2-6, 4-11
Eigenschaften, Test durch Makro 1-11
Eigentümer und Gruppe einer Datei ändern 3-71
Ein- oder Ausgabedaten verwerfen 3-748
Einfügen
  einer Zeile in Fenster (*curses*) 3-136
  eines Zeichens in Fenster (*curses*) 3-134
  (hardwaremäßig) von Zeichen (*curses*) 3-130
  von Zeichen (*curses*) 3-128
  von Zeilen (*curses*) 3-129
Eingabe
  binäre 3-302
  (*curses*) 2-60, 2-63
  formatierte 3-310
Eingabe-Baudrate
  ermitteln 3-62
  festlegen 3-64
Eingabefunktionen (*curses*) 2-60
Eingabepuffer, für Datensichtstation 2-16
Eingabeverarbeitung, Arten 2-16
Eingabezeile, max. Länge 2-17
Einlesen
  einer Zeichenkette (*curses*) 3-125
  eines Zeichens (*curses*) 3-122
  von Zeichen, Blockierung (*curses*) 3-169
Einrichten
  einer Kommunikationsleitung 3-222
  einer Nachrichtenwarteschlange 3-498

einer Pipe zu einem Prozeß 3-538
eines neuen Fensters (*curses*) 3-166
eines neuen *Pad* (*curses*) 3-163
eines Verweises auf Datei 3-432
Einschalten
des Blätterns (*curses*) 3-186
des Raw-Modus (*curses*) 3-178
Einstellen
des Bildlaufbereichs (*curses*) 3-188
eines Schlüssels 3-626
EINTR 2-6, 4-11
Eintrag
aus Gruppendatei 3-341
aus Kennwortdatei abfragen 3-363
aus Symboltabelle ermitteln 3-515
für aktuellen Benutzer 3-796
für Benutzernummer in Benutzerdatei ermitteln 3-367
für Gruppenname aus Gruppendatei ermitteln 3-344f
für Kennwortdatei schreiben 3-564
für Namen in der Benutzerdatei ermitteln 3-366
in linearer Suchtabelle finden 3-430
EINVAL 2-6, 3-346, 4-11
EIO 2-7, 4-11
EISDIR 2-7, 4-11
EMFILE 2-7, 4-11
EMLINK 2-7, 4-11
Empfang
von Daten anhalten 3-746
von Daten neu starten 3-746
Empfangen einer Nachricht 3-503
ENAMETOOLONG 2-7, 4-11
encrypt() 3-236
end 3-237
endcfent() 3-331
Ende eines Sohnprozesses abwarten 3-824
endgrent() 3-341
endpdent() 3-356
endpwent() 3-363
endutent() 3-371
endwin() (*curses*) 3-117
ENFILE 2-7, 4-11
ENODEV 2-7, 4-11
ENOENT 2-7, 4-11
ENOEXEC 2-7, 4-11
ENOLCK 2-8, 4-11

ENOMEM 2-8, 3-55, 4-11
ENOMSG 2-8, 4-11
ENOSPC 2-8, 4-12
ENOSYS 2-8, 4-12, 3-236
ENOTBLK 4-12
ENOTDIR 2-8, 4-12
ENOTEMPTY 2-8, 4-12
ENOTTY 2-8, 4-12
Entfernen eines Knotens aus binärem Suchbaum 3-759
environ 3-242, 3-248ff
ENXIO 2-8, 4-12
EOF 2-21, 4-39
EOL 2-21
EPERM 2-9, 4-12
EPIPE 2-9, 4-12
Epochenwert 1-8, 3-488
erand48() 3-226f, 3-243
ERANGE 2-9, 4-12
ERASE 2-20
  Wirkung 2-18
erase() (*curses*) 3-118
ERASE-Zeichen ermitteln (*curses*) 3-119
erasechar() (*curses*) 3-119
erf() 3-244f
erfc() 3-244
Ergebnisparameter Zeiger 1-23
Ergebnistyp Zeiger 1-23
Erkennen regulärer Ausdrücke 3-581
Ermitteln
  der Benutzerkennung 3-447
  der Dateiinformation 3-314
  der effektiven Benutzernummer 3-339
  der effektiven Gruppennummer 3-336
  der Länge der Teilzeichenkette 3-723
  der Länge einer Zeichenkette 3-716
  der Laufzeit eines Prozesses 3-775
  der Position in Dateiverzeichnisstrom 3-760
  der Prozeßgruppennummer 3-359
  der Prozeßnummer 3-360
  der realen Benutzernummer 3-370
  der realen Gruppennummer 3-340
  der Speicherplatzauslastung 3-459
  der Taktfrequenz 3-348
  der Vaterprozeßnummer 3-361
  der Zeit 3-774

des Benutzernamens 3-349
des Dateinamens einer Datensichtstation 3-794
des Dateistatus 3-694
des ERASE-Zeichens (*curses*) 3-119
des KILL-Zeichens (*curses*) 3-140
des Systemnamens 3-807
eines Eintrags aus Gruppendatei für Gruppenname 3-345
eines Eintrags aus Gruppendatei für Gruppennummer 3-344
eines Eintrags für Benutzernummer in Benutzerdatei 3-367
eines Eintrags für Namen in der Benutzerdatei 3-366
von Dateistatusinformationen 3-815
von Informationen zum Dateisystem 3-697
von Mantisse und Exponent einer doppelt genauen Gleitpunktzahl3-
von Optionen im Argumentvektor 3-351
Eröffnen
einer neuen Datensichtstation (*curses*) 3-164
eines Unter-Fensters (*curses*) 3-191
EROFS 2-9, 4-12
errno 2-5, 3-55, 3-246
Variable 4-11
errno() 3-461, 3-832
ERROR() 3-581
erweiterte Sicherheitskontrollen 1-8
Erzeugen
einer Dateikennzahl 2-2
einer Hash-Tabelle 3-379
einer Pipe 3-534
einer temporären Datei 3-778
einer Zeichenkette mit Benutzerkennung 3-219
eines Namens für temporäre Datei 3-761
eines Namens für temporäre Datei 3-780
eines neuen Prozesses 3-293
eines Pfadnamens für kontrollierendes Terminal 3-89
gleichmäßig verteilter Pseudo-Zufallszahlen 3-243
von gleichverteilten Pseudo-Zufallszahlen 3-226
ESPIPE 2-9, 4-12
ESRCH 2-9, 4-12
etext 3-247
ETXTBSY 2-9, 4-12
Euklidische Distanzfunktion 3-384
EXDEV 2-9, 4-12
exec 1-5, 1-10
exec-Funktionen, Löschen von Verweisen 2-2
execl() 3-248, 3-250
Beispiel für die Implementierung 4-76

execle() 3-248
execlp() 3-248ff
execv() 3-248, 3-250
execve() 3-248ff
exit() 3-256f, 3-353, 4-41
EXIT_FAILURE 4-41
EXIT_SUCCESS 4-41
exp() 3-258
Exponent
  einer Gleitpunktzahl laden 3-428
  und Mantisse einer doppelt genauen Gleitpunktzahl ermitteln 3-308
Exponentialfunktion 3-258
externe Variable errno 2-5

fabs() 3-260
FCHR_MAX 4-23f
fclose() 3-262f
  Löschen von Verweisen 2-2
fcntl() 3-265, 3-267ff
  Erzeugen von Verweisen 2-2
fcvt() 3-271
fdopen() 3-272f
  Erzeugen von Verweisen 2-2
FD_CLOEXEC 3-250, 4-13
Fehler- und komplementäre Fehlerfunktion 3-244
Fehlerabfrage 1-22
Fehleranzeigen für einen Strom löschen 3-76
Fehlerbehandlungsfunktion 3-468
Fehlerfunktion 3-244
  komplementäre 3-244
Fehlerkennzeichen eines Datenstroms testen 3-275
Fehlermeldungstexte 3-712
Fehlernummer 2-5
  globale 3-246
  des Systems 4-11
Fehlersuche 3-554
Fehlfunktion 2-88
Fenster
  aktualisieren (*curses*) 3-179
  Attribute behandeln (*curses*) 3-96
  (*curses*) 2-60
  Datenstrukturen (*curses*) 2-54
  effiziente Aktualisierung (*curses*) 3-212
  formatierte Ausgabe (*curses*) 3-176
  formatiertes Lesen (*curses*) 3-183

großes (Pad, *curses*) 3-163
löschen (*curses*) 3-105, 3-114, 3-118
mit Leerzeichen füllen (*curses*) 3-118
neu einrichten (*curses*) 3-166
Rahmen zeichnen (*curses*) 3-102
Schreibmarke verschieben (*curses*) 3-143
überlagern (*curses*) 3-173
verschieben (*curses*) 3-158
Zeichen ausgeben in (*curses*) 3-92
Zeichen einfügen (*curses*) 3-134
Zeichen liefern (*curses*) 3-131
Zeichen löschen (*curses*) 3-111
Zeichenkette schreiben auf (*curses*) 3-94
Zeile einfügen (*curses*) 3-136
Zeile löschen (*curses*) 3-113
Fensteränderungs-Information löschen (*curses*) 3-192
feof() 3-274
ferror() 3-275
FF0 4-61
FF1 4-61
FFDLY 4-61
fflush() 3-276f
fgetc() 3-278f, 3-302
fgetgrent() 3-341
fgetpwent() 3-363
fgets() 3-283
FIFO 1-5, 1-9, 1-13f
einrichten 3-482
FIFO-Gerätedatei 1-9, 1-13
erzeugen 3-480
FILENAME_MAX 4-40
fileno() 3-284
Erzeugen von Verweisen 2-2
Finden
eines Bytes im Speicher 3-474
eines Eintrags in linearer Suchtabelle 3-430
flash() (*curses*) 3-101
floor() 3-285
FLT_DIG 4-23
FLT_MAX 4-23
FLT_MIN 4-23
flushinp() (*curses*) 3-121
fmod() 3-287f
fopen() 3-272, 3-289f
Erzeugen eines Datenstroms 2-2

FOPEN_MAX 4-39
fork() 1-14f, 3-293f
  Erzeugen von Verweisen 2-2
Format von Dateiverzeichniseinträgen 4-10
formatiert ausgeben 3-543
formatierte Ausgabe
  drucken 3-297
  einer Argumentliste 3-822
  in Fenster (*curses*) 3-176
  in Zeichenkette 3-686
formatierte Eingabe
  aus Zeichenkette 3-691
  umwandeln 3-310, 3-598
formatiertes Lesen im Fenster (*curses*) 3-183
Fortran
  E-Format 3-328
  F-Format 3-271
fpathconf() 3-296, 3-526
fprintf 3-353
fprintf() 3-297, 3-543, 3-548f
fputc() 3-298f
fputs() 3-300
fread() 3-302
free() 3-304, 4-41
freigeben
  einer Kommunikationsleitung 3-809
  eines zugewiesenen Speicherbereiches 3-49
  reservierten Speichers 3-304
  von Speicher (Druckeraufträge) 3-239
  von Speicher (Druckerbeschreibung) 3-238
freopen() 3-305f
frexp() 3-308
fscanf() 3-310, 3-598, 3-603f
fseek() 3-311f
fstat() 1-8, 1-19f, 3-314
fstatfs() 3-697
fsync() 3-316
ftell() 3-317
ftok() 3-318
ftw() 3-320ff
FTW_D 3-320, 4-15
FTW_DNR 3-320, 4-15
FTW_F 3-320, 4-15
FTW_NS 3-320, 4-15
Füllzeichen für Pause senden 3-753

Funktion
  für die Signalbehandlung 3-654
  zur Fehlerbehandlung 3-468
Funktionstastenblock 2-83
  aktivieren (*curses*) 3-138
fwrite() 3-324
F_DUPFD 4-13, 3-265
F_GETFD 4-13, 3-265
F_GETFL 4-13, 3-265
F_GETLK 4-13, 3-266
F_LOCK 4-67
F_OK 4-67
F_RDLCK 4-13
F_SETFD 4-13, 3-265
F_SETFL 4-13, 3-266
F_SETLK 4-13, 3-266
F_SETLKW 4-13, 3-267
F_TEST 4-67
F_TLOCK 4-67
F_ULOCK 4-67
F_UNLCK 4-13
F_WRLCK 4-13

gamma() 3-326
  Vorzeichen 3-671
Gamma-Funktion 3-326, 3-431
gcvt() 3-328
Gemeinsamer Speicherbereich (shared memory) 2-49
  Kontrolloperationen 3-644
  abhängen 3-646
  anhängen 3-642
Gemeinsames Speichersegment anlegen 3-647
Generierung eines Signals 3-652
Gepufferte Standard-Ein- und Ausgabe 4-39
Gerät 1-9
Gerätedatei 1-5
  einrichten 3-482
  FIFO 1-9
  für blockorientierte Geräte 1-9
  für Datensichtstation 1-9
  für zeichenorientierte Geräte 1-8f, 1-14
Gerätenummer 1-9
Gerätesteuerung 3-387
gesicherte
  Benutzernummer 1-9

Gruppennummer 1-9
gesperrten Datcibereich freigeben 3-438
GETALL 4-48
getc() 3-329
GETC() 3-581
getcfadent() 3-331
getcfadnam() 3-331
getcfgrent() 3-331
getcfgrnam() 3-331
getcfprent() 3-331
getcfprnam() 3-331
getcftyent() 3-331
getcftynam() 3-331
getch() (*curses*) 3-122
getchar() 3-334
getcwd() 3-335
getegid() 3-336, 3-347
getenv() 3-337
getenv() 4-41
geteuid() 3-339
getgid() 3-340
getgrent() 3-341, 3-344
getgrnam() 3-341, 3-345
getgroups() 3-346
gethz() 3-348
getlogin() 3-349
GETNCNT 4-48
getopt() 3-351ff
getpass() 3-354
getpdjbent() 3-356
getpdjbnam() 3-356
getpdjbnum() 3-356
getpdprent() 3-356
getpdprnam() 3-356
getpgrp() 3-359
GETPID 4-48
getpid() 3-359f
getpooldat() 3-356
getppid() 3-361
getpw() 3-362
getpwent() 3-363
getpwnam() 3-366
getpwuid() 3-367
gets() 3-369
getstr() (*curses*) 3-125

getuid() 3-370
getutent() 3-371
getutline() 3-371
GETVAL 4-48
getw() 3-375
getyx() (*curses*) 3-127
GETZCNT 4-48
GF_PATH 4-67
gid_t 1-10
  Definition 4-55
Gleichmäßig verteilte Pseudo-Zufallszahlen erzeugen 3-243
Gleichverteilte Pseudo-Zufallszahlen erzeugen 3-226
Gleitkommazahl
  aufspalten 3-490
  in Zeichenkette umwandeln 3-233, 3-271, 3-328
Gleitpunktzahl
  Exponent laden 3-428
  Restfunktion 3-287
globale Fehlernummer 3-246
Globale Variable
  loc1 3-435
  loc2 3-435
  locs 3-442
gmtime() 3-377
Größe des Vektors c_cc[] 4-59
Groß- in Kleinbuchstaben
  umsetzen 3-783
  umwandeln 3-785
Großbuchstabe, Test auf 3-415
Gruppe und Eigentümer einer Datei ändern 3-71
Gruppendatei
  Eintrag abfragen 3-341
  Eintrag für Gruppenname ermitteln 3-345
  Eintrag für Gruppennummer ermitteln 3-344
  positionieren auf Dateianfang 3-623
Gruppenkennung 3-341
Gruppennummer 1-9f, 1-18, 3-341, 3-346, 3-363
  effektive 1-10
  effektive ermitteln 3-336
  gesicherte 1-10
  reale 1-10
  reale ermitteln 3-340
  setzen 3-621
Gruppenstruktur 4-16
  Zeiger 3-281

gsignal() 3-692

Handle 2-2
Hardware-Kontrolle einer Datensichtstation 4-62
Hardwareabhängige Werte 4-72
Hash-Tabelle
  erzeugen 3-379
  verwalten 3-381
  zerstören 3-380
has_ic() (*curses*) 3-128
has_il() (*curses*) 3-129
hcreate() 3-379, 3-381ff
hdestroy() 3-380f
Header-Files 0-3
Hervorhebung 2-81
Hexadezimalziffer, Test auf 3-417
Hintergrund-Prozeßgruppe 1-9
Hinzufügen von Umgebungsvariablen 3-562
hsearch() 3-381ff
HUGE_VAL 4-27
HUPCL 2-29, 4-62
hypot() 3-384

ICANON 2-31, 4-62
ICRNL 2-24, 4-60
idlok() (*curses*) 3-130
IEXTEN 2-31, 4-62
IF_PATH 4-67
IGNBRK 2-24, 4-60
IGNCR 2-24, 4-60
Ignorieren eines Signals 3-652
IGNPAR 2-24, 4-60
implementierungsabhängige Konstanten 4-20
In Datei schreiben 3-828
inch() (*curses*) 3-131
Include-Datei 1-24
  <dirent.h> 4-10
  <locale.h> 4-25
  <nl_types.h> 4-31
  <regexp.h> 4-33
  <search.h> 4-34
  <setjmp.h> 4-35
  <signal.h> 4-36
  <stdio.h> 4-39
  <stdlib.h> 4-41

<string.h> 4-43
<sys/ipc.h> 4-44
<sys/msg.h> 4-47
<sys/sem.h> 4-48
<sys/times.h> 4-54
<sys/utsname.h> 4-57
<sys/wait.h> 4-58
<termios.h> 4-59
<time.h> 4-64
<ulimit.h> 4-66
<unistd.h> 4-67
<utime.h> 4-71
<values.h> 4-72
<varargs.h> 3-821
<varargs.h> 4-75
Indexeintrag 3-736
Indexnamen für Sonderzeichen 2-34
Information zu Landessprachen 3-511
Initialisieren
der Umgebung einer Datensichtstation (*curses*) 3-133
einer Signalmenge 3-661f
von Bytes im Speicher 3-477
Initialisierung
Datensichtstation 2-84
des Pseudo-Zufallszahlengenerators 3-689
initscr() (*curses*) 3-133
INLCR 2-24, 4-60
ino_t, Definition 4-55
INPCK 2-24, 4-60
insch() (*curses*) 3-134
insertln() (*curses*) 3-136
internationale Umgebung 2-36
für Programm setzen 3-628
internationalisiertes Programm 2-36
Interprozeßkommunikation 2-49
Datenstrukturen 2-51
Schlüssel 3-318
Standardpaket 3-318
Statusinformation 2-52
Strukturen 4-44
Systemkennzahl 2-51
Interpunktionszeichen, Test auf 3-411
Interrupt-Taste 3-666
INTLINFO 2-40
INTR 2-20

intrflush() (*curses*) 3-137
INT_MAX 4-23
INT_MIN 4-23
ioctl() 3-387
IPC 2-49
IPC_CREAT 4-44
IPC_EXCL 4-44
IPC_NOWAIT 4-44
IPC_PRIVATE 4-44
IPC_RMID 4-44
IPC_SET 4-44
IPC_STAT 4-44
isalnum() 3-394
isalpha() 3-396
isascii() 3-398
isatty() 3-399
iscntrl() 3-400
isdigit() 3-402
isfirst() 3-403
isgraph() 3-404
ISIG 2-31, 4-62
islower() 3-406
isnan() 3-408
isprint() 3-409
ispunct() 3-411
isspace() 3-413
ISTRIP 2-24, 4-60
isupper() 3-415
isxdigit() 3-417
IUCLC 2-24, 4-60
IXANY 2-24, 4-60
IXOFF 2-24, 4-60
IXON 2-24, 4-60

j0() 3-419f
j1() 3-419f
jn() 3-419f
jrand48() 3-226f, 3-421

Kategorie-Makros 4-25
Keller (stack) 3-50
Kennwort 3-363
  Struktur, Zeiger 3-282
Kennwortdatei
  schließen 3-240

auf Dateianfang positionieren 3-635
Benutzernummer 3-362
Eintrag abfragen 3-363
Eintrag schreiben 3-564
Kennwortstruktur 4-32
Kennzeichen für Sommerzeit 3-221
keypad() (*curses*) 3-138
key_t, Definition 4-55
KILL 2-21
Wirkung 2-18
Zeichen ermitteln (*curses*) 3-140
kill() 3-422f, 3-654
killchar() (*curses*) 3-140
Klassifikation von Zeichen 4-9
Klein- in Großbuchstaben umwandeln 3-786f
Kleinbuchstabe, Test auf 3-406
Knoten aus binärem Suchbaum entfernen 3-759
Kommando
ausführen 3-739
für ulimit() 4-66
Interpreter 1-4, 1-10
Kommunikationleitung
einrichten 3-222
freigeben 3-809
Kommunikationselement 2-50
Komplementäre Fehlerfunktion 3-244
konfigurierbare Pfadnamen-Variablen ermitteln 3-296, 3-526
konfigurierbare Systemvariablen prüfen 3-737
Konstante 1-23
symbolische 1-23
Konstanten für
Baudraten 4-61
Hardware-Kontrolle 4-62
tcflow() 4-63
tcflush() 4-63
tcsetattr() 4-62
Kontrollieren einer Datei 3-265
kontrollierender Prozeß 1-10, 2-15
kontrollierendes Terminal 1-9ff, 1-18
des Sitzungsführers 2-14
Pfadname erzeugen 3-89
Kontrolloperationen für
gemeinsame Speicherbereiche 3-644
Nachrichtenwarteschlangen 3-496
Semaphore 3-609

Kontrollzeichen, Test auf 3-400
Kopieren einer Zeichenkette 3-708
  teilweise 3-720
Kopieren von Bytes im Speicher 3-473, 3-476

l3tol() 3-425
l64a() 3-23
labs() 4-41
Laden des Exponenten einer Gleitpunktzahl 3-428
Länge
  der Teilzeichenkette ermitteln 3-723
  einer Eingabezeile 2-17
  einer Zeichenkette ermitteln 3-716
Landeskonventionen 2-35
Landessprachen-Information 3-511
LANG 2-37
Laufzeit 1-4
  eines Prozesses und seiner Sohnprozesse ermitteln 3-775
lcong48() 3-226f
LC_ALL 2-37, 3-628, 4-25
LC_COLLATE 2-38f, 3-628, 4-25
LC_CTYPE 2-37f, 3-628, 4-25
LC_MONETARY 2-37f, 3-628, 4-25
LC_NUMERIC 2-37f, 3-628, 4-25
LC_TIME 2-37f, 3-628, 4-25
ldexp() 3-428
ldiv() 4-41
leaveok() (*curses*) 3-141
Lebensdauer einer Prozeßgruppe 1-5, 1-10, 1-14f
leere
  Signalmenge 3-661
  Zeichenkette 1-18
leerer Pfadname 1-13
leeres Dateiverzeichnis 1-7
Leerzeichen, Test auf 3-413
Leitung, asynchrone lokal 2-13
Leitungskontrolle 4-63
Leseerlaubnis 1-20
Lesen aus Datei 3-572
Lesen der Vordergrund-Prozeßgruppennummer 3-751
Lesen einer Zeichenkette
  ohne Echo 3-354
  von stdin 3-369
Lesen eines termcap-Eintrags 3-764

Lesen eines Zeichens
  aus einem Datenstrom 3-278
  von einem Datenstrom 3-329
  von Standardeingabe 3-334
Lesen im Fenster, formatiert (*curses*) 3-183
Leserecht 1-16
lfind() 3-430, 3-452f
lgamma() 3-431
Liefern der Position im Datenstrom liefern 3-317
Lineare Suche und Änderung 3-452
Lineare Suchtabelle, Eintrag finden 3-430
link() 3-432
LINK_MAX 3-527, 4-21
locl 3-581, 3-583f, 4-33
  globale Variable 3-435
loc2 3-581, 3-583f, 4-33
loc2, globale Variable 3-435
localtime() 3-436
lockf() 3-438
LOCK_MAX 4-23f
locs 3-581, 4-33
locs, globale Variable 3-442
Löschen
  bis zum Ende des Bildschirms (*curses*) 3-107
  bis zum Zeilenende (*curses*) 3-108
  der Fensteränderungs-Information (*curses*) 3-192
  des Zeichenpuffers (*curses*) 3-121
  (hardwaremäßig) von Zeichen (*curses*) 3-130
  eines Dateiverzeichnisses 3-595
  eines Fensters (*curses*) 3-118
  von Dateien 3-587
  von Fehleranzeigen für einen Strom 3-76
  von Zeichen (*curses*) 3-128f
  von Zeichen im Fenster (*curses*) 3-111
  Zeilen im Fenster 3-113
log() 3-443
log10() 3-445
logarithmische Gamma-Funktion 3-326, 3-431
Logarithmus
  zur Basis 10 3-445
  natürlicher 3-443
Login-Name 3-363
logname() 3-447
lokal angeschlossene asynchrone Leitungen 2-13
Lokale Zeit ermitteln 3-436

long int in 3-Byte-Ganzzahl umwandeln 3-458
long() 3-226
longjmp() 3-448f
longname() (*curses*) 3-142
LONG_BIT 4-23
LONG_MAX 4-23
LONG_MIN 4-23
lrand48() 3-226f, 3-451ff
lseek() 1-9, 3-455f, 3-458

Major-Device 3-483
Makro
  assert() 4-8
  für den Test von Eigenschaften 1-11
mallinfo() 3-459
malloc() 3-383, 3-461, 4-41
mallopt() 3-466
Manipulieren
  der Signalmaske 3-652
  von Zeitstrukturen 4-71
Mantisse und Exponent einer doppelt genauen Gleitpunktzahl
  ermitteln 3-308
Maschinenwort auf Datenstrom ausgeben 3-566
mathematische
  Bibliothek 1-24, 3-468
  Deklarationen 4-27
matherr() 3-468
MAXFLOAT 4-27
Maximalwert für rand() 4-41
MAX_CANON 2-17, 3-527, 4-21, 4-23f
MAX_INPUT 3-527, 4-21, 4-24
mblen() 4-41
mbstowcs() 4-41
mbtowc() 3-471, 4-41
MB_LEN_MAX 4-23
Meldungskatalog 1-11, 3-56f
  öffnen 3-58
Meldungskatalog-Deskriptor 1-11, 3-57
  schließen 3-56
memccpy() 3-473
memchr() 3-474f
memcmp() 3-475
memcpy() 3-476
memset() 3-477
messages 2-49

Minor-Device 3-483
mkdir() 1-19, 3-478
mkfifo 1-9, 1-19, 3-480
mknod() 3-482
mktime() 3-488
mktmp() 3-486
mode_t, Definition 4-55
modf() 3-490
Modus 1-11
mount() 3-492
move() (*curses*) 3-143
mrand48() 3-226f, 3-495
msgctl() 3-496ff
msgrcv() 3-503ff
msgsnd() 3-505ff
MSG_NOERROR 4-47
Mustervergleich mit regulärem Ausdruck 3-701
mvaddch() (*curses*) 3-92
mvaddstr() (*curses*) 3-94
mvdelch() (*curses*) 3-111
mvgetch() (*curses*) 3-122
mvgetstr() (*curses*) 3-125
mvinch() (*curses*) 3-131
mvinsch() (*curses*) 3-134
mvprintw() (*curses*) 3-176
mvscanw() (*curses*) 3-183
mvwaddch() (*curses*) 3-92
mvwaddstr() (*curses*) 3-94
mvwdelch() (*curses*) 3-111
mvwgetch() (*curses*) 3-122
mvwgetstr() (*curses*) 3-125
mvwin() (*curses*) 3-158
mvwinch() (*curses*) 3-131
mvwinsch() (*curses*) 3-134
mvwprintw() (*curses*) 3-176
mvwscanw() (*curses*) 3-183
M_1_PI 4-27
M_2_PI 4-27
M_2_SQRTPI 4-27
M_E 4-27
M_LN10 4-27
M_LN2 4-27
M_LOG10E 4-27
M_LOG2E 4-27
M_PI 4-27

M_PI_2 4-27
M_PI_4 4-27
M_SQRT1_2 4-27
M_SQRT2 M_SQRT1_2
  Wert von 1/Wurzel aus 2 4-27

Nachricht
  empfangen 3-503
  (messages) 2-49
  senden 3-506
Nachrichten-Warteschlange 2-49
  einrichten 3-498
  Kontrolloperationen 3-496
  Strukturen 4-47
Name der Datensichtstation ermitteln (*curses*) 3-142
Namen für temporäre Datei erzeugen 3-761, 3-780
Namen für Zeitzonen 3-798
NAME_MAX 1-6, 1-13, 3-527, 4-21
NaN 1-11, 3-408
Nationalsprachen-System 2-35
Natürlicher Logarithmus 3-443
NCCS 2-34, 4-59
NDEBUG 4-8
Neue Abbilddatei des Prozesses 3-248
neue Datei anlegen oder vorhandene überschreiben 3-86
Neue-Zeile-Zeichen, Umsetzung (*curses*) 3-167
Neuen Prozeß erzeugen 3-293
Neustarten von Datenübertragung oder -empfang 3-746
newpad() (*curses*) 3-163
newterm() (*curses*) 3-164
newwin() (*curses*) 3-166
NGROUPS_MAX 1-18, 3-346, 4-22
nice() 3-509
Nichtlokaler Sprung 3-448
  Signalbehandlung 3-664
  Sprungmarke setzen 3-624, 3-675
NL 2-21
nl() (*curses*) 3-167
NL0 4-60
NL1 4-60
NLDLY 4-60
nlink_t, Definition 4-55
nlist() 3-515
NLS 2-35
NLSPATH 2-39

NL_ARGMAX 4-22
nl_codesize() 3-510
nl_istype() 3-513
nl_langinfo() 3-511, 3-628
NL_LANGMAX 4-22
NL_MSGMAX 4-22
NL_NMAX 4-22
NL_SETMAX 4-22
nl_settype() 3-513
NL_TEXTMAX 4-22
nocbreak() (*curses*) 3-104
nodelay() (*curses*) 3-169
noecho() (*curses*) 3-116
NOFLSH 2-31, 4-62
nonl() (*curses*) 3-167
noraw() (*curses*) 3-178
normale Datei 1-7
nrand48() 3-226f, 3-516
NULL 3-52, 4-41
  Definition 4-64
NULL-Zeiger 3-52, 3-55
Nullbyte ('\0') 1-18
Nullbyte 1-11ff
Nullsignal 3-422
Nullzeiger 1-12, 4-39, 4-41, 4-43, 4-67
numerisches termcap-Feld ermitteln 3-768
NZERO 4-22

OCRNL 4-60
Öffnen
  einer Datei 3-517
  eines Dateiverzeichnisses 3-524
  eines Datenstroms 3-289
  eines Datenstroms 3-305
  eines Meldungskatalogs 3-58
OFDEL 4-60
offene Datei 1-6f
offene Datei-Beschreibung 1-6, 1-14, 2-2
offene Dateikennzahl duplizieren 3-230
offenes Dateiverzeichnis 1-8
off_t, Definition 4-55
OFILL 4-60
OLCUC 4-60
ONLCR 4-60
ONLRET 4-60

ONOCR 4-60
open() 1-9, 1-19, 2-2, 3-517ff
opendir() 3-524f, 4-10
OPEN_MAX 1-6, 4-21
Operationen
  auf Dateiverzeichnissen 3-225
  auf Zeichen aus mehreren Bytes 3-471
  auf Zeichenketten 4-43
  für Semaphor 3-614
OPOST 2-27, 4-60
optarg 3-351ff
optarg() 3-351
opterr 3-351f
opterr() 3-351
optind 3-351ff
optind() 3-351
Optionen im Argumentvektor ermitteln 3-351
optisches Signal 2-81
overlay() (*curses*) 3-173
overwrite() (*curses*) 3-173
O_ACCMODE 4-13
O_APPEND 4-13
O_CREAT 4-13
O_EXCL 4-13
O_NDELAY 4-13
O_NOCTTY 4-13
O_NONBLOCK 4-13
  nicht gesetzt 2-16
  Öffnen einer Gerätedatei für Datensichtstation 2-14
  Pufferung von AusgabeN 2-20
O_RDONLY 4-14
O_RDWR 4-14
O_SYNC 4-13
O_TRUNC 4-13
O_WRONLY 4-14

Pad (*curses*) 2-55, 2-60
  aktualisieren (*curses*) 3-174
  Definition (*curses*) 3-163
  neu einrichten (*curses*) 3-163
Parameter
  der Datensichtstation zurücksetzen (*curses*) 3-117
  einer Datensichtstation setzen 3-755
  lesen, die der Datensichtstation zugeordnet sind 3-750
Parametrisierte Steuerzeichenfolgen 2-75

PARENB 2-29, 4-62
PARMRK 2-24, 4-60
PARODD 2-29, 4-62
Passenden regulären Ausdruck in Zeichenkette anhalten 3-442
PASS_MAX 4-21
pathconf() 1-13, 3-296, 3-526
PATH_MAX 1-12f, 3-527, 4-21
pause() 3-530
Pausenzeichen senden 3-753
pclose() 3-532
PEEK() 3-581
perror() 3-533
Pfadname 1-12, 1-15
  absoluter 1-12
  für kontrollierendes Terminal erzeugen 3-89
  portabler 1-13
  relativer 1-13
Pfadnamen-Anfang 1-12f
Pfadnamen-Auflösung 1-6f, 1-12f, 1-15
Pfadnamen-Variablen 3-296, 3-526
PF_PATH 4-67
pid_t 1-14f
  Definition 4-55
Pipe 1-13f
  erzeugen 3-534
  schließen 3-532
  zu einem Prozeß einrichten 3-538
pipe() 1-13, 2-2, 3-534
PIPE_BUF 3-527, 4-21
PM_STR 4-17
pnoutfresh() (*curses*) 3-174
popen() 1-10, 3-538
  Erzeugen eines Datenstroms 2-2
Portabilität 1-24
portable Pfadnamen 1-13
portabler Zeichensatz für Dateinamen 1-19
Portierbarkeitsgründe 3-667
Position
  der Schreibmarke unverändert lassen (*curses*) 3-141
  der Schreibmarke ermitteln (*curses*) 3-127
  im Datenstrom liefern 3-317
  in Dateiverzeichnisstrom auf Anfang setzen 3-594
  in Dateiverzeichnisstrom ermitteln 3-760
  in Dateiverzeichnisstrom 3-608
Positionszeiger eines Datenstroms neu positionieren 3-311

Potenzierfunktion 3-541
pow() 3-541
prefresh() (*curses*) 3-174
printf() 3-53, 3-543, 3-548f
printw() (*curses*) 3-176
PROC_MAX 4-23
profil() 3-552
  Prozeßausführung 4-30
Programm
  internationalisiertes 2-36
Programmbedingung überprüfen 4-8
Programmcode 3-247
Programmeldung lesen 3-57
Prozeß 1-4ff, 1-9ff,
  anhalten, bis Signal eintrifft 3-530
  beenden 3-256
  effektive Benutzernummer 1-4
  Einrichten einer Pipe 3-538
  erzeugen 3-293
  fortsetzen 1-4
  für festgesetzte Zeitspanne anhalten 3-683
  Gruppennummer 1-4
  kontrollierender 1-10
  Laufzeit ermitteln 3-775
  Lebensdauer 1-5
  reale Benutzernummer 1-5
  stoppen, aussetzen 1-4
  Signal senden 3-422
Prozeßabbruch 3-666
  anormaler 3-25
Prozeßausführung
  Profil 4-30
  Zeitprofil 3-552
Prozesse, Deklarationen für das Warten 4-58
Prozeßgrenzen setzen und ermitteln 3-801
Prozeßgruppe 1-9, 1-11, 1-14ff
  Lebensdauer 1-14
  Signal senden 3-422
Prozeßgruppenchef 1-14
Prozeßgruppennummer 1-14f, 1-18, 3-634
  ermitteln 3-359
  für Auftragskontrolle setzen 3-632
  setzen 3-634, 3-636

Prozeßnummer 1-14ff, 3-634
  ermitteln 3-360
Prozeßpriorität ändern 3-509
Prozeßüberwachung 3-554
Prozeßuniversum wechseln 3-812
Prüfen
  anstehender Signale 3-672
  der Signalmaske 3-673
  der Zugriffsrechte einer Datei 3-27
  des Fehlerkennzeichens eines Datenstroms 3-275
  des Puffers (*curses*) 3-193
  eines Datenstroms auf Dateiende 3-274
  von konfigurierbaren Systemvariablen 3-737
Pseudo-Zufallszahlen erzeugen 3-226, 3-243
Pseudo-Zufallszahlengenerator 3-570
  initialisieren 3-689
ptrace() 3-554
Puffer
  an Datenstrom zuweisen 3-618
  eines Datenstroms bereinigen 3-276
  prüfen (*curses*) 3-193
Pufferung für Datenstrom zuordnen 3-640
Punkt 1-7, 1-15
Punkt-Punkt 1-7, 1-15
putc() 3-560
putchar() 3-561
putenv() 3-562
putpwent() 3-564
puts() 3-565f
putw() 3-566

qsort() 3-568, 4-41
Quadratwurzel 3-687
QUIT 2-20
Quit-Taste 3-666

Rahmen zeichnen (*curses*) 3-102
raise() 3-654
rand() 3-570, 4-41
  Maximalwert 4-41
RAND_MAX 4-41
raw() (*curses*) 3-178
Raw-Modus ein/ausschalten (*curses*) 3-178
RCEAD 2-29
read() 1-9, 3-572ff
readdir() 3-577f, 4-10

reale Benutzernummer 1-5
  ermitteln 3-370
reale Gruppennummer 1-10
  ermitteln 3-340
realloc() 3-579, 4-41
Rechenzeit angeben, verbrauchte 3-77
Rechte, besondere 1-20, 1-5
refresh() (*curses*) 3-179
regexp 3-32
Reguläre Ausdrücke
  Deklarationen 4-33
  Erkennen und Übersetzen 3-581
regulärer Ausdruck 3-581
  Mustervergleich 3-701
  übersetzten 3-81
relativer Pfadname 1-13
remove() 3-587
rename() 3-588
Reservieren von Speicher 3-461
Reservierten Speicher freigeben 3-304
Reservierung von Speicherplatz 3-55
resetty() (*curses*) 3-181
reset_prog_mode() (*curses*) 3-180
reset_shell_mode() (*curses*) 3-180
Restfunktion für Gleitpunktzahlen 3-287
RETURN() 3-581
rewind() 3-592
rewinddir() 3-594, 4-10
rmdir() 3-595
Root-Dateiverzeichnis 1-12ff
  wechseln 3-74
R_OK 4-67

s-Bit 3-483
savetty() (*curses*) 3-181
SA_NOCLDSTOP 3-651, 4-38
sbe-Bit 3-250
sbrk() 3-48
scanf() 3-53, 3-598f, 3-603ff
scanw() (*curses*) 3-183
SCHAR_MAX 4-23f
SCHAR_MIN 4-23
SCHAR_MIN 4-24
Schließen
  der Kennwortdatei 3-240

einer Dateikennzahl 3-78
einer Pipe 3-532
eines Dateiverzeichnis-Stroms 3-80
eines Datenstroms 3-262
Schlüssel
einstellen 3-626
für die Interprozeß-Kommunikation 3-318
Schrägstrich 1-6f, 1-12ff
Schreiben
einer Zeichenkette auf Standardausgabe 3-565
eines Eintrags für Kennwortdatei 3-564
eines Zeichens auf Standardausgabe 3-561
in Datei 3-828
Zeichenkette auf Fenster 3-94
Schreiberlaubnis 1-20
Schreibmarke
im Fenster verschieben (*curses*) 3-143
Ermitteln der Position der (*curses*) 3-127
Position unverändert lassen (*curses*) 3-141
Positionierung 2-78
Schreibrecht 1-16
Schriftart, Bedeutung der 1-3
Schutzbitmaske setzen und ermitteln 3-803
Schutzbits 1-19f, 3-478
ändern 3-68
der Datei 1-7
einer Datei 1-16
scroll() (*curses*) 3-185
scrollok() (*curses*) 3-186
Sedezimalziffer, Test auf 3-417
seed48() 3-226, 3-607
seekdir() 3-608, 4-10
SEEK_CUR 4-39, 4-67
SEEK_END 4-39, 4-67
SEEK_SET 4-39, 4-67
Semaphore 2-49
Kontrolloperationen 3-609
Operationen 3-614
Strukturen 4-48
Semaphorkennzahl (semid) 2-51
Semaphormenge anlegen 3-612
semctl() 3-609, 3-611ff
semget() 3-612f
semop() 3-614ff
SEM_UNDO 4-48

Senden
einer Nachricht 3-506
eines Signals an Prozeß oder Prozeßgruppe 3-422
von Füllzeichen für Pause 3-753
SETALL 4-48
setbuf() 3-618
setcfadent() 3-331
setcfgrent() 3-331
setcftyent() 3-331
setgid() 1-9f, 3-347, 3-621
setgrent() 3-341
setjmp() 3-624
setkey() 3-626
setlocale() 3-628, 4-25
setpdjbent() 3-356
setpdprent() 3-356
setpgid() 1-14, 1-16, 3-632
setpgrp() 3-634
setpwent() 3-363
setscrreg() (*curses*) 3-188
setsid() 1-14, 1-16, 3-636
setterm() (*curses*) 3-187
setuid() 1-5, 1-9, 3-637
setutent() 3-371
SETVAL 4-48
setvbuf() 3-640f
Setzen der
Benutzernummer 3-637
Dateiposition auf Anfang 3-592
Gruppennummer 3-621
Prozeßgruppennummer 3-634
Vordergrund-Prozeßgruppennummer 3-757
Setzen von einer Datensichtstation zugeordneten Parametern 3-755
sh 1-10
shared memory 2-49, 3-644
Speicherkennzahl (shmid) 2-51
shmat() 3-642f
shmctl() 3-644f
shmdt() 3-646
shmget() 3-647f
SHMLBA 4-50
SHM_RDONLY 4-50
SHRT_MAX 4-23
SHRT_MIN 4-23
Sicherheitskontrollen, erweiterte 1-8

Sichtbares Zeichen, Test auf 3-404
sie-Universum 3-812
SIGABRT 4-36
sigaction() 3-650
sigaction-Struktur 4-37
sigaddset() 3-659
SIGALRM 4-36
  Signal 3-33
SIGCHLD 3-654, 4-37
SIGCONT 3-653, 4-37
sigdelset() 3-660
sigemptyset() 3-661
sigfillset() 3-662
SIGFPE 3-654, 4-36
SIGHUP 4-36
SIGILL 3-654, 4-36
SIGINT 4-36
sigismember() 3-663
SIGKILL 3-654, 4-36
siglongjmp() 3-664
Signal 1-11, 1-16
  abfangen 3-654
  an einen Prozeß schicken 3-652
  an Prozeß oder Prozeßgruppe senden 3-422
  aus Signalmenge löschen 3-660
  blockieren 3-652
  generieren 3-652
  ignorieren 3-652, 3-666
  in Signalmenge prüfen 3-663
  zu Signalmenge hinzufügen 3-659
  zustellen 3-652
signal() 3-651, 3-666
Signal
  anstehend 3-652
  Effekte bei Funktionen 3-654
  optisches 2-81
  Prozeß bis Eintreffen anhalten 3-530
  warten auf 3-677
Signal-Aktionen 3-653
  ändern 3-650
  untersuchen 3-650
Signalbehandlung
  definieren 3-666
  bei nichtlokalem Sprung 3-664
Signalbehandlungs-Funktion 3-654

Signale (audiovisuelle) (*curses*) 3-101
Signale 2-11, 4-36
  prüfen, anstehende 3-672
  <signal.h> 4-36
  Vordergrund-Prozeßgruppen 2-14
Signalmaske 3-652
  auf Signal prüfen 3-663
  initialisieren 3-661f
  manipulieren 3-652
  prüfen und ändern 3-673
  Signal hinzufügen 3-659
  Signal löschen 3-660
signgam 3-326, 3-431, 3-671
sigpending() 3-672
SIGPIPE 4-36
sigprocmask() 3-673
SIGQUIT 4-36
SIGSEGV 3-654, 4-36
sigsetjmp() 3-675
SIGSTOP 3-653f, 4-37
sigsuspend() 3-677
SIGTERM 4-36
SIGTSTP 3-653, 4-37
SIGTTIN 3-653, 4-37
SIGTTIN-Signal, Bedingungen 2-15
SIGTTOU 3-653, 4-37
SIGUSR1 4-36
SIGUSR2 4-36
SIG_BLOCK 3-673, 4-38
SIG_DFL 3-653, 4-36
SIG_ERR 4-36
SIG_IGN 3-653, 4-36
SIG_SETMASK 3-673, 4-38
SIG_UNBLOCK 3-673, 4-38
sin() 3-679
sinh() 3-681
Sinus 3-679
  Hyperbolicus 3-681
Sitzung 1-9ff, 1-16ff
  erzeugen 3-636
  Lebensdauer 1-16
Sitzungsführer 1-10, 1-16
  kontrollierendes Terminal 2-14
sizeof(), Ergebnistyp 4-41

size_t 3-52, 3-55, 4-41, 4-43, 4-64
  Definition 4-55
sleep() 3-683f
Software-Signal 3-692
Software-Signal auslösen 3-378
Sohnprozeß 1-14, 1-17
  Ende abwarten 3-824
Sommerzeit-Kennzeichen 3-221
Sortieren eines Vektors 3-568
Sortierreihenfolge, Zeichenketten vergleichen 3-706
speed_t 4-59
Speicher freigeben
  (Druckeraufträge) 3-239
  (Druckerbeschreibung) 3-238
Speicher reservieren 3-461
Speicher
  Bytes initialisieren 3-477
  Bytes kopieren 3-476
  Bytes vergleichen 3-475
  Kopieren von Bytes 3-473
Speicherbereich (zugewiesener) freigeben 3-49
Speicherblockgröße verändern 3-579
Speicheroperationen 4-29
Speicherplatz für
  das Vorzeichen von gamma() 3-671
  Datensegment dynamisch ändern 3-48
Speicherplatzauslastung ermitteln 3-459
Speicherplatzreservierung 3-55, 4-26
  Algorithmus 3-466
Speicherverwaltung 3-48
Sperren von Dateibereichen 3-438
sprintf() 3-543, 3-549, 3-686
Sprung
  mit Signalbehandlung, nichtlokaler 3-664
  nichtlokaler 3-448, 3-624
Sprungmarke für nichtlokalen Sprung setzen 3-624, 3-675
sqrt() 3-687
srand() 3-571, 3-689, 4-41
srand48() 3-226, 3-690
sscanf() 3-598, 3-604, 3-691f
ssignal() 3-692
Stackumgebung, Deklarationen 4-35
Standard Interprozess-Kommunikationspaket 3-318
Standard-Ein- und Ausgabe, gepufferte 4-39
Standard-Ein-/Ausgabeströme 3-699

Standard-Fenster (*curses*) 2-54
Standardausgabe 3-699
  Zeichen schreiben 3-561
  Zeichenkette schreiben 3-565
Standardbibliothek, Definitionen 4-41
Standardeingabe 3-699
  Zeichen lesen 3-334
Standardfehlerausgabe 3-699
Standardfehlerbehandlungsprozeduren 3-469
Standardkonstanten, symbolische 4-67
Standardstrukturen 4-67
standend() (*curses*) 3-96
standout() (*curses*) 3-96
START 2-22
stat() 1-8, 1-19f, 3-694
statfs() 3-697
Statusinformationen 2-52
stderr 3-39, 3-353, 3-699, 4-39
STDERR_FILENO 4-68
stdin 3-699, 4-39
  Zeichen lesen 3-334
  Zeichenkette lesen 3-369
STDIN_FILENO 4-68
stdout 3-699, 4-39
STDOUT_FILENO 4-68
STD_BLK 4-23f
step() 3-581, 3-583f, 3-701
Steuerzeichenfolgen
  (*termcap*) ausgeben 3-788
  parametrisiert 2-75
stime() 3-702
STOP 2-21
strcat() 3-703
strcfrm() 3-733
strchr() 3-704
strcmp() 3-53, 3-705, 3-718
strcoll() 3-706, 3-707
strcpy() 3-708
strdup() 3-710
strerror() 3-712
strftime() 3-713
strlen() 3-716
strncat() 3-717
strncmp() 3-718
strncpy() 3-720

strncpy() 3-720
Strom, Fehleranzeigen löschen 3-76
strpbrk() 3-721
strrchr() 3-722
strspn() 3-723
strstr() 3-724
strtod() 3-725f, 4-41
strtok() 3-728f
strtol() 3-730f, 4-41
strtoul() 4-41
struct dirent 4-10
struct sigaction 3-650, 4-37
struct termios 4-59
Struktur für
  Dateizeiten 4-54
  Interprozeßkommunikation 4-44
  Nachrichtenwarteschlangen 4-47
  Semaphore 4-48
  Systemnamen 4-57
subwin() (*curses*) 3-191
Suche, lineare 3-452
Suchen einer Teilzeichenkette 3-724
Superblock aktualisieren 3-736
SUSP 2-22
swab() 3-735
symbolische
  Konstante 1-23
  Standardkonstanten 4-67
Symboltabelle, Einträge ermitteln 3-515
sync() 3-736
synchrone Datensichtstationen (*curses*) 2-62
Synchronisieren des Dateizustands 3-316
sysconf() 3-737
SYSPID_MAX 4-23f
System 1-17
system() 1-10, 3-739, 4-41
System-Fehlernummern 4-11
Systemkennzahl für Interprozeßkommunikation 2-51
Systemnamen
  ermitteln 3-807
  Struktur 4-57
Systemprozeß 1-15, 1-17
Systempuffer 3-736
Systemuhr stellen 3-702
Systemvariablen, konfigurierbare 3-737

SYS_OPEN 4-23f
S_IEXEC 4-52
S_IFBLK 4-51
S_IFCHR 4-51
S_IFDIR 4-51
S_IFIFO 4-51
S_IFMT 4-51
S_IFREG 4-51
S_IREAD 4-52
S_IRGRP 4-51
S_IROTH 4-51
S_IRUSR 4-51
S_IRWXG 4-51
S_IRWXO 4-51
S_IRWXU 4-51
S_ISBLK() 4-52
S_ISCHR() 4-52
S_ISDIR() 4-52
S_ISFIFO() 4-52
S_ISGID 4-52
  ändern 3-68
S_ISREG() 4-52
S_ISUID 4-51
S_ISUID ändern 3-68
S_ISVTX 4-52
S_IWGRP 4-51
S_IWOTH 4-51
S_IWRITE 4-52
S_IWUSR 4-51
S_IXGRP 4-51
S_IXOTH 4-51
S_IXUSR 4-51

TAB0 4-61
TAB1 4-61
TAB2 4-61
TAB3 4-61
TABDLY 4-61
Tabellen durchsuchen, <search.h> 4-34
Tabulator 2-84
Taktfrequenz ermitteln 3-348
tan() 3-741
Tangens 3-741
  Hyperbolicus 3-743
tanh() 3-743

tcdrain() 3-745
tcflag_t 4-59
tcflow() 3-746
  Konstanten 4-63
tcflush() 3-748
  Konstanten 4-63
tcgetattr() 3-750
tcgetpgrp() 3-751
TCIFLUSH 3-748, 4-63
TCIOFF 4-63
TCIOFLUSH 3-748, 4-63
TCION 4-63
TCOFLUSH 3-748, 4-63
TCOOFF 3-746, 4-63
TCOON 3-746, 4-63
TCSADRAIN 4-62
TCSAFLUSH 3-755, 4-62
TCSANOW 3-755, 4-62
TCSDRAIN 3-755
tcsendbreak() 3-753
tcsetattr() 3-755
  Konstanten 4-62
tcsetpgrp() 3-757
tdelete() 3-759, 3-790ff
Teilzeichenkette suchen 3-724
telldir() 3-760, 4-10
tempnam() 3-761
Temporäre Datei erzeugen 3-778
termcap-Schnittstelle 2-90
termcap-Eintrag lesen 3-764
termcap:
  boolesches Feld ermitteln 3-766
  Cursorpositionierung vorbereiten 3-772
  numerisches Feld ermitteln 3-768
  Steuerzeichenfolgen ausgeben 3-788
  Zeichenketten-Feld ermitteln 3-770
Terminal, kontrollierendes 1-9, 1-18
Terminalschnittstelle 2-13
termios, Definition der Werte 4-59
Test auf
  7-Bit US-ASCII-Zeichen 3-398
  Buchstabe 3-396
  Buchstabe oder Ziffer 3-394
  Datensichtstation 3-399
  druckbares Zeichen 3-409

Großbuchstabe 3-415
Hexadezimalziffer 3-417
Interpunktionszeichen 3-411
Kleinbuchstabe 3-406
Kontrollzeichen 3-400
Leerzeichen 3-413
Multibyte-Zeichen 3-403
auf NaN 3-408
Sedezimalziffer 3-417
sichtbares Zeichen 3-404
Ziffer 3-402
Text- und Datenbereich, Platzbedarf 3-48
tfind() 3-763, 3-790f
tgetent() 3-764
tgetflag() 3-766
tgetnum() 3-768
tgetstr() 3-770
tgoto() 3-772
time() 3-774
times() 1-5, 3-775
timezone 3-777, 3-799
timezone() 3-777
time_t 4-64
Definition 4-55
tmpfile() 3-778
tmpnam() 3-780
TMP_MAX 4-22, 4-39
toascii() 3-782
tolower() 3-785
_tolower() 3-783
TOSTOP 2-31, 4-62
touchwin() (*curses*) 3-192
toupper() 3-787
_toupper() 3-786
tputs() 3-788
tsearch() 3-790ff
ttyname() 3-794
ttyslot() 3-796
twalk() 3-790ff, 3-797
typeahead() (*curses*) 3-193
tzname 3-798f
tzname[] 3-799
tzset() 3-799f

ucb-Universum 3-812

UCHAR_MAX 4-23f
Übergeordnetes Dateiverzeichnis 1-17
Überlagern von Fenstern (*curses*) 3-173
Überprüfen einer Programmbedingung 4-8
Überschreiben einer vorhandenen Datei 3-86
Übersetzen
  eines regulären Ausdrucks 3-81
  regulärer Ausdrücke 3-581
Übersetzer, Ausgabe 4-2
übersetzter Ausdruck 3-581
Übersetzungszeit 1-4
Übertragung
  einer Ausgabe abwarten 3-745
  von Daten anhalten 3-746
  von Daten neu starten 3-746
Übertragungsgeschwindigkeit der Datensichtstation (*curses*) 3-100
uid_t 1-5
  Definition 4-55
UINT_MAX 4-23f
ulimit() 3-801, 4-66
  Kommandos 4-66
ULONG_MAX 4-23
UL_GETFSIZE 4-66
UL_SETFSIZE 4-66
umask() 3-803
Umbenennen einer Datei 3-588
Umgebung
  der Datensichtstation initialisieren (*curses*) 3-133
  Informationen (*curses*) 2-60
  internationale 2-36, 3-628
Umgebungsvariable
  INTLINFO 2-40
  LANG 2-37
  LC_CTYPE 2-37
  LC_MONETARY 2-37
  LC_NUMERIC 2-37
  LC_TIME 2-37
  NLSPATH 2-39
  ändern 3-562
  hinzufügen 3-562
  Zeiger-Vektor 3-242
umount() 3-805
Umschalten zwischen Datensichtstationen (*curses*) 3-187
Umsetzen von Groß- in Kleinbuchstaben 3-783
Umsetzung des Neue-Zeile-Zeichens (*curses*) 3-167

Umwandeln einer Gleitkommazahl in Zeichenkette 3-271, 3-328
Umwandeln einer Zeichenkette
  in double 3-725
  in long 3-730
Umwandeln
  eines Zeitwerts in die aufgeschlüsselte lokale Zeit 3-436
  formatierter Eingabe 3-598
  von Datum und Zeit in Zeichenkette 3-90
  von Klein- in Großbuchstaben 3-786f
  Gleitkommazahl in Zeichenkette 3-233
  long int in 3-Byte-Ganzzahl 3-458
  Zahl in Zeichenkette 3-23
  formatierter Eingaben 3-310
  von Groß- in Kleinbuchstaben 3-785
umwandlung von
  3-Byte-Ganzzahl in long int 3-425
  Zeichenketten 3-733
Umwandlungsinformation für Zeitzone setzen 3-799
uname() 3-807
unctrl() (*curses*) 3-194
undial() 3-809
UNGETC() 3-581
ungetc() 3-810f
universe() 3-812
Universum wechseln 4-56
unlink() 3-813
Unter-Fenster eröffnen (*curses*) 3-191
Unterbaum 3-492
Unterschied zwischen
  longjmp() und siglongjmp() 3-665
  setjmp() und sigsetjmp() 3-675
Unterstreichung 2-81
Untersuchen
  einer Zeichenkette auf Zeichen 3-721
  von Signal-Aktionen 3-650
US-ASCII-Zeichen, Test auf 3-398
USHRT_MAX 4-23
USI_MAX 4-23f
ustat() 3-815
utime() 3-817f
  Deklaration 4-71
utmp-Datei, Einträge bearbeiten 3-371
utmpname() 3-371

Variable Argumentliste behandeln 3-821, 4-75
Variable errno 2-5, 4-11
Vaterprozeß 1-14, 1-17ff
Vaterprozeßnummer 1-17
  ermitteln 3-361
Vektor c_cc[]
  Indexnamen 2-34
  Vektorgröße 2-34
Vektor sortieren 3-568
VEOF 2-34, 4-59
Verändern der Speicherblockgröße 3-579
VERASE 2-34, 4-59
Verbindungsabbruch, Baudrate 4-61
verbrauchte Rechenzeit angeben 3-77
Vergleichen von
  Bytes im Speicher 3-475
  Zeichenketten 3-705
  Zeichenketten gemäß Sortierreihenfolge 3-706
  Zeichenketten, teilweise 3-718
Verklemmung (deadlock) 3-439
Verschieben
  der Schreibmarke im Fenster (*curses*) 3-143
  eines Fensters (*curses*) 3-158
Verschlüsseln einer Zeichenkette 3-87
Verschlüsselungsfunktion 3-236
Vertauschen von Bytes 3-735
verwaiste Prozeßgruppe 1-17
Verwalten
  eines binären Suchbaums 3-790
  von Hash-Tabellen 3-381
Verweis 1-17
  auf Datei einrichten 3-432
  auf Dateibeschreibung 2-2
  entfernen 3-813
  auf ein Dateiverzeichnis 3-485
Verweiszähler 1-17
Verwerfen
  nichtgelesener Eingabedaten 3-748
  nichtgesendeter Ausgabedaten 3-748
verzögerte Ausgabe (*curses*) 3-110
vfprintf() 3-822
VINTR 2-34, 4-59
VKILL 2-34, 4-59
VMIN 2-34, 4-59
Vollduplexbetrieb 2-16

Vordergrund-Prozeßgruppe 1-18
  Beschreibung 2-14
Vordergrund-Prozeßgruppennummer 1-18
  lesen 3-751
  setzen 3-757
vorhandene Datei überschreiben oder neue anlegen 3-86
Vorzeichen von gamma() 3-671
vprintf() 3-822
VQUIT 2-34, 4-59
vsprintf() 3-822
VSTART 2-34, 4-59
VSTOP 2-34, 4-59
VSUSP 2-34, 4-59
VT0 4-61
VT1 4-61
VTDLY 4-61
VTIME 2-34, 4-59

waddch() (*curses*) 3-92
waddstr() (*curses*) 3-94
wait() 1-15, 1-19, 3-824ff
waitpid() 1-15, 1-19, 3-824ff
Warten auf
  die Übertragung einer Ausgabe 3-745
  Signal 3-677
Warten von Prozesses, Deklarationen 4-58
Warteschlange bei Unterbrechungssignal löschen (*curses*) 3-137
Warteschlangenkennzahl (msqid) 2-51
wattroff() (*curses*) 3-96
wattron() (*curses*) 3-96
wattrset() (*curses*) 3-96
wchar_t 4-41
wclear() (*curses*) 3-105
wclrtobot() (*curses*) 3-107
wclrtoeol() (*curses*) 3-108
wcstombs() 4-41
wctomb() 4-41
wdelch() (*curses*) 3-111
wdeleteln() (*curses*) 3-113
Wechseln des
  aktuellen Dateiverzeichnisses 3-66
  Prozeßuniversums 3-812
  Root-Dateiverzeichnisses 3-74
  Universums 4-56

weitere Gruppennummer 1-18
  lesen 3-346
werase() (*curses*) 3-118
Werte für termios 4-59
WEXITSTATUS() 4-58
wgetch() (*curses*) 3-122
wgetstr() (*curses*) 3-125
WIFEXITED() 4-58
WIFSIGNALED() 4-58
WIFSTOPPED() 4-58
winch() (*curses*) 3-131
winsch() (*curses*) 3-134
winsertln() (*curses*) 3-136
wmove() (*curses*) 3-143
WNOHANG 4-58
wnoutrefresh() (*curses*) 3-212
WORD_BIT 4-23
wprintw() (*curses*) 3-176
wrefresh() (*curses*) 3-179
write() 1-9, 3-828
wscanw() (*curses*) 3-183
wsetscrreg() (*curses*) 3-188
wstandend() (*curses*) 3-96
wstandout() (*curses*) 3-96
WSTOPSIG() 4-58
WTERMSIG() 4-58
WUNTRACED 4-58
Wurzelfunktion 3-687
W_OK 4-67

X kontrollierendes Terminal 2-14
XCASE 2-31, 4-62
X_OK 4-67

y0() 3-832
y1() 3-832
yn() 3-832

Zahl in 7-Bit ASCII-Zeichen umwandeln 3-782
Zahl in Zeichenkette umwandeln 3-23
Zeichen 1-18
  auf Datenstrom ausgeben 3-298, 3-560
  auf Standardausgabe schreiben 3-561
  aus Datenstrom lesen 3-278
  aus Fenster liefern (*curses*) 3-131
  ausfügen 2-79

einfügen 2-79
einlesen (*curses*) 3-122
einlesen, Blockierung (*curses*) 3-169
im Fenster löschen (*curses*) 3-111
in darstellbares Format bringen (*curses*) 3-194
in Datenstrom zurückstellen 3-810
in Fenster ausgeben (*curses*) 3-92
in Fenster einfügen (*curses*) 3-134
von Datenstrom lesen 3-329
von Standardeingabe lesen 3-334
einfügen und löschen (*curses*) 3-128
hardwaremäßig einfügen und löschen (*curses*) 3-130
Zeichenkette
anfügen 3-703
auf Datenstrom ausgeben 3-300
auf Fenster schreiben (*curses*) 3-94
auf Standardausgabe schreiben 3-565
auf Zeichen untersuchen 3-721
duplizieren 3-710
durchsuchen 3-704, 3-722
einlesen (*curses*) 3-125
in double umwandeln 3-44, 3-725
in ganze Zahl umwandeln 3-45
in long umwandeln 3-46, 3-730
in Teile aufspalten 3-728
in termcap-Eintrag ermitteln 3-770
kopieren 3-708
mit Benutzerkennung erzeugen 3-219
ohne Echo lesen 3-354
teilweise anfügen 3-717
teilweise kopieren 3-720
und regulären Ausdruck vergleichen 3-581
verschlüsseln 3-87
von stdin lesen 3-369
formatierte Ausgabe 3-686
formatierte Eingabe 3-691
Länge ermitteln 3-716
leere 1-18
gemäß Sortierreihenfolge vergleichen 3-706
teilweise vergleichen 3-718
vergleichen 3-705
Zeichenketten-Operationen 4-43
Zeichenketten-Umwandlung 3-733

Zeichenklassifikation 4-9
(NLS) 3-513
zeichenorientiertes Gerät 1-9
Zeichenpuffer löschen (*curses*) 3-121
Zeichensatz 1-18
analysieren 3-510
für portable Dateinamen 1-19
Zeichnen eines Rahmens (*curses*) 3-102
Zeiger
auf die nächste Gruppenstruktur 3-281
auf nächste Kennwort-Struktur 3-282
Ergebnisparameter 1-23
Ergebnistyp 1-23
Zeiger-Vektor auf die Umgebungsvariablen 3-242
Zeile ausfügen 2-78
Zeile einfügen 2-78
Zeile im Fenster löschen
löschen (*curses*) 3-113
einfügen (*curses*) 3-136
Zeilen, einfügen und löschen (*curses*) 3-129
Zeit
ermitteln 3-774
umwandeln 3-488
und Datum in Zeichenkette umwandeln 3-35, 3-90, 3-713
lokale 3-436
Zeiten, Datentypen 4-64
Zeitprofil der Prozeßausführung 3-552
Zeitstrukturen manipulieren 4-71
Zeitwert in aufgeschlüsselte lokale Zeit umwandeln 3-436
Zeitzone, Umwandlungsinformation setzen 3-799
Zeitzonen-Information 3-777
Zeitzonen-Namen 3-798
Zerstören von Hashtabellen 3-380
Ziffer
oder Buchstabe, Test auf 3-394
Test auf 3-402
Zombieprozeß 1-19
Zugriff 1-5, 1-20
Zugriff auf Geräte 1-9
Zugriffsart 1-6
Zugriffsberechtigung 1-5, 1-11, 1-18f
Zugriffsberechtigungen für Datei 1-5,1-10, 1-16

Zugriffsrecht 1-21, 3-482
  einer Datei prüfen 3-27
Zuordnen
  einer Pufferung für Datenstrom 3-640
  eines Datenstroms zu einer gegebenen Dateikennzahl 3-272
Zustellung eines Signals 3-652
Zuweisen eines Puffers an Datenstrom 3-618

Bestell-Nr. des leeren Ordners U3764-J-Z18-1 (950)

SIEMENS

Betriebssystem SINIX V5.22
CES C-Entwicklungssystem
Teil 2

SIEMENS

SINIX Computer

Daten- und Informationstechnik
Data and Information Systems