SIEMENS
NIXDORF

SINIX

CES V5.40/V5.41

Referenzhandbuch für Programmierer

Beschreibung

Sie haben

uns zu diesem Handbuch etwas mitzuteilen?
Schicken Sie uns bitte Ihre Anregungen unter
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG
Manualredaktion STM QM 2
Otto-Hahn-Ring 6
W-8000 München 83

Fax: (089) 636-40443

email im EUnet:
man@sieqm2.uucp

Dieses Handbuch ist auf chlorfrei gebleichtem Papier gedruckt.

Sie haben

uns zu diesem Handbuch etwas mitzuteilen?
Schicken Sie uns bitte Ihre Anregungen unter
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG
Manualredaktion STM QM 2
Otto-Hahn-Ring 6
W-8000 München 83

Fax: (089) 636-40443

email im EUnet:
man@sieqm2.uucp

Dieses Handbuch ist auf chlorfrei gebleichtem Papier gedruckt.

CES (SINIX)

Referenzhandbuch für
Programmierer

Beschreibung

Ausgabe Juli 1992 (CES V5.40/V5.41)

Wollen Sie mehr wissen...

. . . über dieses Produkt
. . . oder ein anderes Thema der Informationstechnik?

Unsere Training Center stehen für Sie bereit.
Besuchen Sie uns in Berlin, Essen, Frankfurt/Main oder Hamburg,
in Hannover, Mainz, München, Stuttgart, Wien oder Zürich.

Auskunft und Informationsmaterial erhalten Sie über:

München (089) 636-2009

oder schreiben Sie an:

Siemens Nixdorf Training Center
Postfach 830951, W-8000 München 83

X/Open
PLUS XPG3

Inhalt

Einleitung

CES V5.4x ist ein **C**-Entwicklungs**S**ystem für das Betriebssystem SINIX V5.4x. Es enthält Funktionen und Kommandos, die die Erstellung von C-Programmen unterstützen.

Zielsetzung und Zielgruppe des Handbuchs

Die CES-Dokumentation wendet sich an alle, die mit dem C-Entwicklungssystem arbeiten und C-Programme erstellen.

Das vorliegende Handbuch ist ein Nachschlagewert, kein Lehrbuch für die Sprache C. Sie sollten deshalb mit der Programmiersprache C vertraut sein. Ein geeignetes Lehrbuch ist die Sprachbeschreibung von Kernighan & Ritchie "Programmieren in C", zweite Ausgabe.

Konzept des Handbuchs

In diesem "Referenzhandbuch für Programmierer" sind Kommandos für die C-Programmentwicklung, Systemaufrufe, Bibliotheken und Funktionen sowie C-spezifische Dateiformate beschrieben

Das Handbuch ist in fünf Kapitel unterteilt:

Kommandos (1)
Systemaufrufe (2)
Bibliotheksfunktionen
- Funktionen zur C-Programmierung aus der Standardbibliothek `libc` (3C)
- Funktionen für die Standard-Ein-/Ausgabe (3S)
- Funktionen für das ELF-Format aus der Bibliothek `libelf` (3E)
- allgemeine Funktionen aus der Bibliothek `libgen` (3G)
- mathematische Funktionen aus der Bibliothek `libm` (3M)
- Funktionen aus Spezialbibliotheken (3X)

Dateiformate (4)
Hilfsmittel und verschiedene Funktionen (5)

Im Anschluß an das letzte Kapitel finden Sie ein Literaturverzeichnis und ein Verzeichnis aller in diesem Handbuch beschriebenen Kommandos, Funktionen etc. alphabetisch sortiert und mit entsprechender Seitenangabe.

Jedes Kapitel besteht aus einer Reihe voneinander unabhängiger Beschreibungen. Innerhalb eines Kapitels sind die Beschreibungen alphabetisch sortiert, abgesehen von den jeweils einleitenden `intro`-Abschnitten. Die Kapitel (3C) und (3S) gehören zusammen, da die dort beschriebenen Funktionen die C-Standardbibliothek bilden.

In einigen Abschnitten werden mehrere Funktionen beschrieben. Sie sind jeweils unter einem 'Haupteintrag' aufgeführt, der als äußerer Kolumnentitel auf den entsprechenden Seiten erscheint.

Die Beschreibungen halten sich, soweit möglich und sinnvoll, an eine festgelegte Struktur:

- Der äußere Kolumnentitel enthält den 'Haupteintrag'.
- Der innere Kolumnentitel (optional) enthält Hinweise auf Maschinenspezifika.
- Die Hauptüberschrift enthält den/die Namen des Eintrags und eine kurze Erklärung.
- In den grau unterlegten Zeilen wird die einzubindende Bibliotheksdatei und die Syntax der Einträge angegeben.
- In dem folgenden Text sind Arbeitsweise, Umgebung, Fehlermeldungen etc. beschrieben.
- Soweit sinnvoll folgen BEISPIELE für die Verwendung des Kommandos, der Funktion etc.
- Unter dem Stichwort DATEIEN sind Verzeichnisse und Dateien angegeben, auf die zugegriffen wird bzw. die angelegt werden.
- Verweise auf weitere Informationen finden Sie unter SIEHE AUCH.
- Unter dem Stichwort ENDESTATUS wird der Wert angegeben, den ein Kommando nach seiner Ausführung an den aufrufenden Prozeß zurückliefert.
- Unter dem Stichwort ERGEBNIS wird der Rückgabewert einer Funktion beschrieben.
- Am Ende des Abschnitts können unter dem Stichwort HINWEIS weitere für die Verwendung des Kommandos bzw. der Funktion wichtige Informationen aufgeführt sein.

Darstellungsmittel

Die Darstellung der Anweisungsformate und Benutzereingaben im vorliegenden Handbuch folgt bestimmten Regeln. Sie werden im folgenden kurz vorgestellt.

`Schreibmaschine`	Konstante Dateinamen, Kommandos, Funktionen und andere konstante Begriffe sind in Schreibmaschinenschrift gedruckt. Außerdem sind Beispiele, die Sie der Vorlage entsprechend eingeben können, mit Schreibmaschinenschrift ausgezeichnet.
kursiv	Beispielhafte Namen für Parameter werden in Benutzereingaben und im fortlaufenden Text kursiv gedruckt.
[ ]	Angaben in eckigen Klammern können Sie wahlweise verwenden oder nicht verwenden. Die Klammern selbst dürfen Sie nicht angeben.
...	Fortsetzungszeichen stehen dort, wo Sie die vorhergehende syntaktische Einheit wiederholen können.

Funktionsübersicht

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

Die eingerückten Namen bedeuten, daß Sie die Beschreibung dieser funktionen unter dem Oberbegriff finden, unter dem sie eingerückt sind.

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

Dateibearbeitung

Dateizugriff

access(2)	Zugriffsrechte auf eine Datei feststellen
catopen(3C)	Meldungsverzeichnis öffnen
catclose(3C)	Meldungsverzeichnis schließen
close(2)	Dateideskriptor schließen
dup(2)	Dateideskriptor duplizieren
dup2(3C)	offenen Dateideskriptor duplizieren
fclose(3S)	Stream schließen
fflush(3S)	Stream leeren
fopen(3S)	Stream öffnen
freopen(3S)	
fdopen(3S)	
fseek(3S)	Schreib-/Lesezeiger eines Streams neu positionieren
rewind(3S)	
ftell(3S)	
fsetpos(3C)	Dateizeiger im Datenstrom neu positionieren
fgetpos(3C)	
fsync(2)	Dateizustand synchronisieren
lockf(3C)	Sätze in Dateien sperren
lseek(2)	Schreib-/Lesezeiger positionieren
open(2)	Datei zum Schreiben oder Lesen öffnen
pathfind(3G)	Verzeichnisse nach einer Datei durchsuchen

Dateiverwaltung

access(2)	Zugriffsrechte auf eine Datei feststellen
chdir(2)	aktuelles Dateiverzeichnis ändern
fchdir(2)	
chmod(2)	Dateimodus ändern
fchmod(2)	
chown(2)	Eigentümer/Gruppe einer Datei ändern
lchown(2)	
fchown(2)	
chroot(2)	Root-Dateiverzeichnis ändern
copylist(3G)	Datei in den Speicher kopieren
creat(2)	neue Datei erstellen oder vorhandene Datei überschreiben
directory:	Dateiverzeichnis-Operationen
opendir(3C)	öffnen
readdir(3C)	Eintrag suchen
telldir(3C)	Adresse suchen
seekdir(3C)	positionieren
rewinddir(3C)	auf Anfang positionieren
closedir(3C)	schließen
fcntl(2)	geöffnete Datei steuern
ftw(3C)	Dateibaum durchwandern
nftw(3C)	
getcwd(3C)	Pfadnamen des aktuellen Dateiverzeichnisses abfragen
link(2)	Verweis auf eine Datei einrichten
mkdir(2)	Dateiverzeichnis erstellen
mkdirp(3G)	Verzeichnisse in einem Pfad erzeugen
rmdirp(3G)	Verzeichnisse in einem Pfad entfernen
mkfifo(3C)	neue FIFO-Gerätedatei erzeugen
mknod(2)	Dateiverzeichnis, Gerätedatei oder normale Datei erstellen
mktemp(3C)	eindeutigen Dateinamen erstellen
mount(2)	Dateisystem einhängen
remove(3C)	Datei löschen
rename(2)	Dateinamen ändern
rmdir(2)	Dateiverzeichnis entfernen
stat(2)	Dateistatus abrufen
lstat(2)	
fstat(2)	
statvfs(2)	Dateisysteminformationen lesen
fstatvfs(2)	
tmpfile(3S)	temporäre Datei erstellen
tmpnam(3S)	Name für temporäre Datei erstellen
tempnam(3S)	

truncate(3C)	Datei auf angegebene Länge setzen
ftruncate(3C)	
uadmin(2)	Verwaltungsfunktionen
umask(2)	Dateimodus-Erstellungsmaske setzen und abfragen
umount(2)	Dateisystem aushängen
unlink(2)	Dateiverzeichnis-Eintrag entfernen
ustat(2)	Dateisystem-Statistik abfragen
utime(2)	Dateizugriffs- und Änderungszeiten setzen

Datei-Information

access(2)	Zugriffsrechte auf eine Datei feststellen
basename(3G)	letztes Element eines Pfadnamens zurückgeben
dirname(3G)	Name des übergeordneten Dateiverzeichnisses ausgeben
ferror(3S)	Stream-Status abfragen
feof(3S)	Ende?
clearerr(3S)	
fileno(3S)	Kennzahl?
readlink(2)	Wert eines symbolischen Verweises lesen
realpath(3C)	echten Dateinamen ausgeben
stat(2)	Dateistatus abrufen
lstat(2)	
fstat(2)	
statvfs(2)	Dateisysteminformationen lesen
fstatvfs(2)	
sysfs(2)	Information über Dateisystemtyp abfragen
ustat(2)	Dateisystem-Statistik abfragen

Merfach benutzbares Objekt

dlclose(3X)	mehrfach benutzbares Objekt schließen
dlerror(3X)	Informationen zur Fehlerursache erfragen
dlopen(3X)	mehrfach benutzbares Objekt öffnen
dlsym(3X)	Adresse in mehrfach benutzbarem Objekt berechnen

Ein-/Ausgabe

bgets(3G)	Stream bis zum nächsten Begrenzer lesen
bufsplit(3G)	Puffer in Felder aufteilen
fread(3S)	binäre Eingabe
fwrite(3S)	binäre Ausgabe
getc(3S)	Zeichen/Wort aus Stream lesen
getchar(3S)	
fgetc(3S)	
getw(3S)	
getopt(3C)	Optionsbuchstaben aus Argumentvektor abfragen
gets(3S)	Zeichenkette aus einem Stream lesen
fgets(3S)	
getsubopt(3C)	Unteroptionen aus einer Zeichenkette heraustrennen
poll(2)	STREAMS-Ein-/Ausgabe multiplexen
printf(3S)	formatierte Ausgabe
fprintf(3S)	
sprintf(3S)	
putc(3S)	Zeichen/Wort auf einen Stream ausgeben
putchar(3S)	
fputc(3S)	
putw(3S)	
putmsg(2)	Nachricht auf einen Stream senden
putpmsg(2)	
puts(3S)	Zeichenkette auf einen Stream schreiben
fputs(3S)	
read(2)	aus einer Datei lesen
readv(2)	
scanf(3S)	formatierte Eingabe
fscanf(3S)	
sscanf(3S)	
setbuf(3S)	Pufferung für Stream festlegen
setvbuf(3S)	
stdio(3S)	Standardfunktionen für gepufferte Ein-/Ausgabe
ungetc(3S)	Zeichen in Eingabestrom zurückstellen
vprintf(3S)	variable Argumentenliste formatiert ausgeben
vfprint(3S)	
vsprintf(3S)	
write(2)	in eine Datei schreiben
writev(2)	

Prozesse

Prozeßverwaltung

acct(2)	Prozeßabrechnung ein- oder ausschalten
alarm(2)	Prozeß-Alarmuhr setzen
clock(3C)	verbrauchte CPU-Zeit melden
cuserid(3S)	Zeichenkette mit Benutzernamen erzeugen
getcontext(2)	Benutzerkontext ändern
setcontext(2)	
getenv(3C)	Wert für Umgebungsvariable zurückgeben
getpid(2)	Prozeßnummern abfragen
getpgrp(2)	Prozeßgruppennummer abfragen
getppid(2)	Vaterprozeßnummer abfragen
getpgid(2)	
getsid(2)	Sitzungsnummer lesen
getuid(2)	reale Benutzernummer abfragen
geteuid(2)	effektive Benutzernummer abfragen
getgid(2)	reale Gruppennummer abfragen
getegid(2)	effektive Gruppennummer abfragen
kill(2)	Signal an Prozeß oder Prozeßgruppe senden
makecontext(3C)	Benutzerkontext ändern
swapcontext(3C)	
pause(2)	Prozeß bis Signal anhalten
plock(2)	Prozeß, Text oder Daten sperren oder entsperren
priocntl(2)	Prozesse verwalten
priocntlset(2)	Prozesse kontrollieren
processflags(2)	Prozeßoptionen einstellen/abfragen
putenv(3C)	Umgebungsvariablen ändern/hinzufügen
setsid(2)	Sitzungsnummer einstellen
setpgid(2)	Prozeßgruppennummer einstellen
setpgrp(2)	Prozeßgruppennummer einstellen
setuid(2)	Benutzernummer einstellen
setgid(2)	Gruppennummer einstellen
sigaction(2)	detaillierte Signalverwaltung
signal(2)	vereinfachte Signalverwaltung
sigset(2)	
sighold(2)	
sigrelse(2)	
sigignore(2)	
sigpause(2)	

sigpending(2)	blockierte und wartende Signale prüfen
sigprocmask(2)	Signalmaske ändern oder testen
sigsetops(3C)	Signalmengen bearbeiten
sigemptyset(3C)	
sigfillset(3C)	
sigaddset(3C)	
sigdelset(3C)	
sigismember(3C)	
sleep(3C)	Ausführung unterbrechen
ssignal(3C)	Softwaresignale
gsignal(3C)	
times(2)	Zeiten für Prozeß und Sohnprozeß abfragen
ulimit(2)	Prozeßgrenzen abfragen und setzen
vfork(2)	neuen Prozeß im virtuellen Speicher erzeugen

Interprozeßkommunikation

getmsg(2)	Nachricht aus einem Stream abrufen
getpmsg(2)	
msgctl(2)	Steuerfunktionen für Nachrichten liefern
msgget(2)	Kennung für Nachrichten-Warteschlangen bestimmen
msgop:	
msgrcv(2)	Nachrichten lesen
msgsnd(2)	Nachrichten senden
raise(3C)	Signal an Programm senden
semctl(2)	Semaphor-Steuerfunktionen
semget(2)	Semaphor-Kennung bestimmen
semop(2)	Semaphor-Operationen
shmctl(2)	Steuerfunktionen für Shared Memory
shmget(2)	Kennung für Shared Memory bestimmen
shmop:	Operationen auf Shared Memory
shmat(2)	
shmdt(2)	
sigaltstack(2)	alternativen Stapelkontext eines Signals setzen/lesen
sigsend(2)	Signal an Prozeß/Prozeßgruppe senden
sigsendset(2)	
sigsuspend(2)	Signalmaske installieren und Prozeß deaktivieren
stdipc:	Standardpaket für Interprozeßkommunikation
ftok(3C)	

Pipes

p2open(3G)	Pipe zu Kommando öffnen
p2close(3G)	Pipe zu Kommando schließen
pipe(2)	Interprozeß-Kommunikationskanal einrichten
popen(3S)	Pipe zu Kommando öffnen
pclose(3S)	Pipe zu Kommando schließen

Zusammenwirken von Prozessen

abort(3C)	Erzeugung eines Signals für unnormale Beendigung
atexit(3C)	Beendigungsroutine einem Programm hinzufügen
exec:	Datei ausführen
execl(2)	
execle(2)	
execlp(2)	
execv(2)	
execve(2)	
execvp(2)	
exit(2)	Prozeß beenden
_exit(2)	
fork(2)	neuen Prozeß erzeugen
system(3S)	Shell-Kommando absetzen
wait(2)	auf Anhalten oder Beendigung eines Sohnprozesses warten
waitid(2)	auf Zustandsänderung von Sohnprozessen warten
waitpid(2)	auf Zustandsänderung von Sohnprozessen warten

Steuerung von Programmabläufen

monitor(3C)	Ausführungsprofil vorbereiten
nice(2)	Priorität eines Prozesses ändern
profil(2)	Ausführungsprofil erstellen
setjmp(3C)	nichtlokaler Sprung
longjmp(3C)	
sigsetjmp(3C)	nichtlokaler Sprung mit Signalstatus
siglongjmp(3C)	

Programmtest

assert(3X)	Zusicherung im Programm überprüfen
nlist(3E)	Einträge einer Namensliste lesen
ptrace(2)	Prozeß-Ablauf verfolgen

Speicherverwaltung und -operationen

brk(2)	Größe des Datensegments verändern
sbrk(2)	
end(3C)	End-Adressen im Programm
etext(3C)	
edata(3C)	
malloc(3C)/(3X)	Hauptspeicher verwalten
free(3C)/(3X)	
realloc(3C)/(3X)	
calloc(3C)(3X)	
memalign(3C)	
valloc(3C)	
mallopt(3X)	
mallinfo(3X)	
memcntl(2)	Speicherverwaltung kontrollieren
memory(3C)	Speicherfunktionen
memccpy(3C)	Zeichen kopieren
memchr(3C)	Zeichen suchen
memcmp(3C)	Zeichen vergleichen
memcpy(3C)	Zeichen kopieren
memmove(3C)	
memset(3C)	Zeichen setzen
mincore(2)	Residenz von Speicherseiten bestimmen
mlock(3C)	Speicherseiten sperren
munlock(3C)	Speicherseiten freigeben
mlockall(3C)	Adreßbereich sperren
munlockall(3C)	Adreßbereich freigeben
mmap(2)	Speicherseiten abbilden
mprotect(2)	Zugriffsschutz für Speicherabbildung ändern
msync(3C)	Speicher synchronisieren
munmap(2)	Abbildung von Speicherseiten aufheben

Systemorganisation

crypt(3C)/(3X)	Zeichenketten verschlüsseln
setkey(3C)/(3X)	
encrypt(3C)/(3X)	
crypt_close(3X)	
des_crypt(3X)	
des_setkey(3X)	
des_encrypt(3X)	
run_setkey(3X)	
run_crypt(3X)	
fpathconf(2)	konfigurierbare Pfadnamenvariablen lesen
pathconf(2)	
getdate(3C)	Datums- und Zeitangaben umwandeln
getdents(2)	Dateiverzeichnis-Einträge umwandeln
getgrent(3C)	Gruppendatei-Eintrag bestimmen
getgrgid(3C)	
getgrnam(3C)	
setgrent(3C)	
endgrent(3C)	
fgetgrent(3C)	
getgroups(2)	Gruppennummern lesen
setgroups(2)	Gruppennummern schreiben
getitimer(3C)	Intervall-Timer lesen und setzen
setitimer(3C)	
getlogin(3C)	Login-Namen abfragen
getmntent(3C)	Dateieintrag mnttab holen
getmntany(3C)	
getpass(3C)	Paßwort lesen
getpw(3C)	Benutzernummer feststellen
getpwent(3C)	Paßwortdatei-Eintrag ändern
getpwuid(3C)	
getpwnam(3C)	
setpwent(3C)	
endpwent(3C)	
fgetpwent(3C)	
getrlimit(2)	Betriebsmittelverbrauch kontrollieren
setrlimit(2)	

getspent(3C)	Eintrag der Shadow-Paßwortdatei ändern
getspnam(3C)	
lckpwdf(3C)	
ulckpwdf(3C)	
setspent(3C)	
endspent(3C)	
fgetspent(3C)	
getut:	auf utmp-Dateieintrag zugreifen
getutent(3C)	
getutid(3C)	
getutline(3C)	
pututline(3C)	
setutent(3C)	
endutent(3C)	
utmpname(3C)	
getutx:	auf utmpx-Eintrag zugreifen
getutxent(3C)	
getutxid(3C)	
getutxline(3C)	
pututxline(3C)	
setutxent(3C)	
endutxent(3C)	
utmpxname(3C)	
getutmp(3C)	
getutmpx(3C)	
updwtmp(3C)	
updwtmpx(3C)	
getvfsent(3C)	vfstab-Dateieintrag lesen
getvfsfile(3C)	
getvfsspec(3C)	
getvfsany(3C)	
initgroups(3C)	zusätzliche Gruppenzugriffslisten initialisieren
maillock(3X)	Sperrdatei für Mailbox eines Benutzers verwalten
mailunlock(3X)	
makedev(3C)	Gerätenummer verwalten
major(3C)	
minor(3C)	
mpcntl(3X)	Standardschnittstelle für Multiprozessor-Verarbeitung
putpwent(3C)	Eintrag in die Paßwortdatei schreiben
putspent(3C)	Eintrag in Shadow-Paßwortdatei schreiben

spinlock(3X)	Benutzersynchronisation
initspin(3X)	
cspinlock(3X)	
spinunlock(3X)	
yield(3X)	
swapctl(2)	Swap-Bereich verwalten
symlink(2)	symbolischen Verweis auf eine Datei erstellen
sync(2)	Superblock aktualisieren
sysconf(3C)	konfigurierbare Systemvariablen lesen
sysi86(2)	rechnerspezifische Funktionen
sysinfo(2)	Systeminformationen lesen und schreiben
uname(2)	Namen des aktuellen SINIX-Systems abfragen

Zeichen und Zeichenketten

Einzelne Zeichen bearbeiten

conv:	Zeichen umwandeln
toupper(3C)	Umwandlung in Großbuchstaben
tolower(3C)	Umwandlung in Kleinbuchstaben
_toupper(3C)	Umwandlung in Großbuchstaben
_tolower(3C)	Umwandlung in Kleinbuchstaben
toascii(3C)	Umwandlung in ASCII-Zeichen
ctype:	Zeichen bearbeiten
isalpha(3C)	Buchstabe?
isupper(3C)	Großbuchstabe?
islower(3C)	Kleinbuchstabe?
isdigit(3C)	Ziffer?
isxdigit(3C)	hexadezimales Zeichen?
isalnum(3C)	alphanumerisches Zeichen?
isspace(3C)	Leerzeichen?
ispunct(3C)	Sonderzeichen?
isprint(3C)	druckbares Zeichen?
isgraph(3C)	druckbares Zeichen außer Leerzeichen?
iscntrl(3C)	Lösch- oder Steuerzeichen?
isascii(3C)	ASCII-Zeichen?
ffs(3C)	erstes gesetztes Bit suchen
isencrypt(3G)	Zeichen-Puffer-Verschlüsselung feststellen

Umwandlung von Größen

a64l(3C)	ASCII-Zeichenkette in long integer umwandeln
l64a(3C)	long integer in ASCII-Zeichenkette umwandeln
ecvt(3C)	Gleitkommazahl in Zeichenkette umwandeln
fcvt(3C)	
gcvt(3C)	
l3tol(3C)	3-Byte integer in long integer
ltol3(3C)	long integer in 3-Byte integer
mbchar:	Mehrbyte-Zeichen umwandeln
mbtowc(3C)	
mblen(3C)	
wctomb(3C)	
mbstring:	Funktionen für Mehrbyte-Zeichen
mbstowcs(3C)	
wcstombs(3C)	
offsetof(3C)	Offset einer Strukturkomponente
sputl(3X)	maschinenunabhängig auf lange Ganzzahlen zugreifen
sgetl(3X)	
strtod(3C)	Zeichenkette in Gleikommazahl Typ double
atof(3C)	Zeichenkette in Gleitkommazahl
strtol(3C)	Zeichenkette in ganze Zahl umwandeln
strtoul(3C)	
atol(3C)	
atoi(3C)	
swab(3C)	Bytes austauschen

Zeichenketten bearbeiten

gmatch(3G)	globaler Mustervergleich in der Shell
str(3G)	Zeichenketten ändern
strfind(3G)	
strrspn(3G)	
strtrns(3G)	
strccpy(3G)	Zeichenketten kopieren
streadd(3G)	
strcadd(3G)	
strecpy(3G)	
strcoll(3C)	Zeichenketten sortieren
string:	Zeichenketten bearbeiten
strcat(3C)	
strdup(3C)	
strncat(3C)	
strlen(3C)	
strncmp(3C)	
strcmp(3C)	
strcpy(3C)	
strncpy(3C)	
strchr(3C)	
strpbrk(3C)	
strrchr(3C)	
strspn(3C)	
strcspn(3C)	
strtok(3C)	
strstr(3C)	
strxfrm(3C)	Transformation von Zeichenketten durchführen
swab(3C)	Bytes austauschen

Reguläre Audrücke

regcmp(3G)	regulären Ausdruck übersetzen und ausführen
regex(3G)	
regexpr:	regulären Ausdruck übersetzen und anwenden
compile(3G)/(5)	
step(3G)/(5)	
advance(3G)/(5)	

Meldungen

addseverity(3C)	Liste mit Warnstufen erstellen
catgets(3C)	Programmeldung lesen
catopen(3C)	Meldungsverzeichnis öffnen
catclose(3C)	Meldungsverzeichnis schließen
fmtmsg(3C)	Meldung über stderr oder die Systemkonsole anzeigen
gettxt(3C)	Zeichenkette aus Meldungsdatei holen
localeconv(3C)	numerische Formatierinformationen lesen
nl_langinfo(3C)	Sprachinformationen liefern
perror(3C)	Systemfehlermeldungen ausgeben
psignal(3C)	Signalmeldung des Systems ausgeben
psiginfo(3C)	
setlocale(3C)	Umgebung eines Programms abfragen/einstellen
strerror(3C)	Fehlermeldung feststellen

Zeitfunktionen

adjtime(2)	Synchronisation mit Systemuhr korrigieren
ctime(3C)	Datum und Zeit in Zeichenkette umwandeln
localtime(3C)	
gmtime(3C)	
asctime(3C)	
tzset(3C)	
difftime(3C)	Differenz zwischen zwei Kalenderdaten berechnen
gettimeofday(3C)	Datum und Zeit lesen und setzen
settimeofday(3C)	
mktime(3C)	tm-Struktur in Kalenderzeit umwandeln
stime(2)	Uhrzeit einstellen
strftime(3C)	Datum und Zeit in Zeichenkette umwandeln
cftime(3C)	
ascftime(3C)	
time(2)	Uhrzeit abfragen

Mathematische Funktionen

abs(3C)	ganzzahligen Absolutwert liefern
labs(3C)	
bessel:	Bessel-Funktionen
j0(3M)	
j1(3M)	
jn(3M)	
y0(3M)	
y1(3M)	
yn(3M)	
div(3C)	Quotienten und Rest berechnen
ldiv(3C)	
erf(3M)	Fehlerfunktion und komplementäre Fehlerfunktion
erfc(3M)	
exp(3M)	Exponentialfunktion
expf(3M)	
cbrt(3M)	
log(3M)	natürlicher Logarithmus x
logf(3M)	
log10(3M)	Logarithmus x zur Basis 10
log10f(3M)	
pow(3M)	allgemeine Exponentialfunktion
powf(3M)	
sqrt(3M)	Quadratwurzel
sqrtf(3M)	
floor(3M)	ganzzahlig abrunden
floorf(3M)	
ceil(3M)	ganzzahlig aufrunden
ceilf(3M)	
copysign(3M)	
fmod(3M)	Gleitkommarest von x/y
fmodf(3M)	
fabs(3M)	Absolutbetrag einer Gleitkommazahl
fabsf(3M)	
rint(3M)	
remainder(3M)	
frexp(3C)	Teile von Gleitkommazahlen bearbeiten
ldexp(3C)	
logb(3C)	
nextafter(3C)	
scalb(3C)	
modf(3C)	
modff(3C)	

gamma(3M)	Logarithmus der Gammafunktion
lgamma(3M)	
hypot(3M)	euklidischer Abstand
isnan(3C)	Gleitkommazahl-Typ bestimmen
isnand(3C)	
isnanf(3C)	
finite(3C)	
fpclass(3C)	
unordered(3C)	
matherr(3M)	Fehlerbehandlungsfunktion
sinh(3M)	Hyperbel-Funktionen
sinhf(3M)	
cosh(3M)	
coshf(3M)	
tanh(3M)	
tanhf(3M)	
asinh(3M)	
acosh(3M)	
atanh(3M)	
trig:	Trigonometriefunktionen
sin(3M)	
sinf(3M)	
cos(3M)	
cosf(3M)	
tan(3M)	
tanf(3M)	
asin(3M)	
asinf(3M)	
acos(3M)	
acosf(3M)	
atan(3M)	
atanf(3M)	
atan2(3M)	
atan2f(3M)	

Funktionen der ELF-Bibliothek

elf_begin(3E)	Dateideskriptor erzeugen
elf_cntl(3E)	Dateideskriptor kontrollieren
elf_end(3E)	Benutzung einer Objektdatei beenden
elf_error:	Fehlerbehandlung
elf_errmsg(3E)	
elf_errno(3E)	
elf_fill(3E)	Füll-Byte einstellen
elf_flag:	Schalter manipulieren
elf_flagdata(3E)	
elf_flagehdr(3E)	
elf_flagelf(3E)	
elf_flagphdr(3E)	
elf_flagscn(3E)	
elf_flagshdr(3E)	
elf_fsize:	Länge einer Objektdatei zurückgeben
elf32_fsize(3E)	
elf_getarhdr(3E)	Mitglied einer Archivdatei auslesen
elf_getarsym(3E)	Symboltabelle eines Archivs lesen
elf_getbase(3E)	Basis-Offset einer Objektdatei lesen
elf_getdata(3E)	Abschnittsdaten lesen
elf_newdata(3E)	
elf_rawdata(3E)	
elf_getehdr:	klassenabhängigen Objektdateikopf lesen
elf32_getehdr(3E)	
elf32_newehdr(3E)	
elf_getident(3E)	Identifkationsdaten einer Datei lesen
elf_getphdr:	klassenabhängige Programmkopftabelle lesen
elf32_getphdr(3E)	
elf32_newphdr(3E)	
elf_getscn(3E)	Abschnittsinformationen lesen
elf_ndxscn(3E)	
elf_newscn(3E)	
elf_nextscn(3E)	
elf_getshdr:	klassenabhängigen Abschnittskopf lesen
elf32_getshdr(3E)	
elf_hash(3E)	Hash-Wert berechnen
elf_kind(3E)	Dateityp bestimmen
elf_next(3E)	auf Archivkomponenten sequentiell zugreifen
elf_rand(3E)	auf Archivkomponenten wahlfrei zugreifen
elf_rawfile(3E)	nicht interpretierten Dateiinhalt lesen
elf_strptr(3E)	Zeiger auf Zeichenkette erzeugen
elf_update(3E)	ELF-Deskriptor aktualisieren

elf_version(3E)	Versionen abgleichen
elf_xlate:	klassenabhängige Datenumsetzung
elf32_xlatetof(3E)	
elf32_xlatetom(3E)	

Zufallszahlen

drand48(3C)	nicht negative Zufallszahl Typ double
erand48(3C)	nicht negative Zufallszahl Typ double
lrand48(3C)	nicht negative Zufallszahl Typ long
nrand48(3C)	nicht negative Zufallszahl Typ long
mrand48(3C)	Zufallszahl Typ long
jrand48(3C)	Zufallszahl Typ long
srand48(3C)	initialisieren
seed48(3C)	initialisieren
lcong48(3C)	initialisieren
rand(3C)	einfacher Zufallszahlengenerator
srand(3C)	

Such- und Sortierverfahren

bsearch(3C)	sortierte Tabelle binär absuchen
hsearch(3C)	Hash-Suchtabellen verwalten
hcreate(3C)	
hdestroy(3C)	
insque(3C)	Element in Queue einfügen
remque(3C)	Element aus Queue entfernen
lsearch(3C)	linear suchen und ändern
lfind(3C)	
qsort(3C)	mit Quicksort sortieren
tsearch(3C)	binäre Suchbäume verwalten
tfind(3C)	
tdelete(3C)	
twalk(3C)	

Manipulation einer seriellen Schnittstelle

ctermid(3S)	Dateinamen für Terminal erstellen
ioctl(2)	Geräte und STREAMS steuern
tcsetpgrp(3C)	Vordergrund-Prozeßgruppennummer setzen
termios:	allgemeine Terminalschnittstelle
tcgetattr(2)	
tcsetattr(2)	
tcsendbreak(2)	
tcdrain(2)	
tcflush(2)	
tcflow(2)	
cfgetospeed(2)	
cfsetospeed(2)	
cfgetispeed(2)	
cfsetispeed(2)	
tcgetpgrp(2)	
tcsetpgrp(2)	
tcgetsid(2)	
ttyname(3C)	Name eines Terminals abfragen
isatty(3C)	
ttyslot(3C)	Eintrag des aktuellen Benutzers in der utmp-Datei finden

Hardware-Verhalten steuern

fpgetround(3C)	IEEE-Gleitkomma-Umgebung steuern
fpsetround(3C)	
fpgetmask(3C)	
fpsetmask(3C)	
fpgetsticky(3C)	
fpsetsticky(3C)	

Grafikfähiges Terminal

jagent:	Host-Steuerung für grafikfähiges Terminal
ioctl(5)	
libwindows:	Funktionsbibliothek für grafikfähiges Terminal
New(3X)	
Newlayer(3X)	
openchan(3X)	
Runlayer(3X)	
Current(3X)	
Delete(3X)	
Top(3X)	
Bottom(3X)	
Move(3X)	
Reshape(3X)	
Exit(3X)	

Kommandos

intro - Einführung zu den Programmierhilfen

Der folgende Abschnitt beschreibt Kommandos zur Unterstützung bei der Programmentwicklung in alphabetischer Reihenfolge. Wenn nicht anders angegeben, akzeptieren die Kommandos Optionen und andere Argumente nach der folgenden Syntax:

```
Name [Option...] [Argument...]
```

Name Name des Kommandos

Option *-noargletter(s)* oder *-argletter < > optarg*, wobei:

noargletter ein einzelner Buchstabe ist, der eine Option ohne Optionsargument spezifiziert

argletter ein einzelner Buchstabe ist, der eine Option angibt, welche ein Optionsargument erfordert

< > ein optional anzugebendes Leerzeichen (White Space) ist

optarg ein Optionsargument (Zeichenkette) ist, welches von der vorhergehenden Option *argletter* akzeptiert wird

Argument entweder ein Bindestrich - selbst, der die Standardeingabe bezeichnet, oder ein *nicht* mit - beginnender Pfadname (oder ein anderes Kommandoargument).

Im Handbuch werden die Namen *TMPDIR, BINDIR, INCDIR* und *LIBDIR* verwendet. Diese Namen stehen für Verzeichnisnamen. Der Name der Verzeichnisse wird jeweils an den entsprechenden Stellen definiert. Beispielsweise kann sich *TMPDIR* auf `/var/tmp` beziehen. Es handelt sich bei den Namen nicht um Umgebungsvariablen, die gesetzt werden können. (Es gibt jedoch auch eine Umgebungsvariable `TMPDIR`. Siehe `tmpnam`(3S).) Es gibt außerdem Referenzen auf *LIBPATH*, den voreingestellten Suchpfad des Binders und anderer Werkzeuge.

SIEHE AUCH

`exit`(2), `wait`(2), `getopt`(3C).
`getopts`(1) in "SINIX V5.41 Kommandos"

ENDESTATUS

Nach Beendigung eines Kommandos werden zwei Statusbytes zurückgegeben. Das erste Byte wird vom Betriebssystem versorgt und gibt den Grund für die Beendigung an. Das zweite Byte gibt, bei normaler Beendigung, den Ende-Status des Programms an (siehe `wait`(2) und `exit`(2)). Bei normaler Beendigung des Kommandos hat das erste Statusbyte den Wert 0; das zweite Statusbyte enthält üblicherweise 0 für erfolgreiche Ausführung und ungleich 0, um Fehler anzuzeigen, wie zum Beispiel falsche Parameter oder fehlerhafte Daten. Der Ende-Status wird nur dort beschrieben, wo er speziellen Konventionen folgt.

admin - SCCS-Dateien erstellen und verwalten

admin [Option...] Datei...

`admin` (administer) dient zur Erstellung neuer SCCS-Dateien und zur Änderung der Parameter bestehender SCCS-Dateien. Die Argumente für `admin`, die in beliebiger Reihenfolge erscheinen können, bestehen aus Optionen, die mit - beginnen, und Dateinamen. Beachten Sie, daß SCCS-Dateinamen mit den Zeichen `s.` anfangen müssen. Wenn die Datei nicht existiert, wird sie erstellt und ihre Parameter werden entsprechend den angegebenen Optionsargumenten initialisiert. Nicht mit einem Optionsargument initialisierten Parametern wird ein Standardwert zugewiesen. Ist die Datei vorhanden, werden die Parameter entsprechend den angegebenen Optionsargumenten geändert, während die übrigen Parameter unverändert bleiben.

Ist ein Dateiverzeichnis angegeben, verhält sich `admin` so, als ob jede Datei im Dateiverzeichnis angegeben wurde. Unlesbare Dateien und Dateien, deren Name nicht mit `s.` beginnt, werden kommentarlos ignoriert. Wenn als Dateiname - angegeben wird, wird von Standardeingabe gelesen; jede Zeile der Standardeingabe wird als Name einer zu verarbeitenden SCCS-Datei angesehen. Auch hier werden Dateien, die keine SCCS-Dateien sind, und unlesbare Dateien kommentarlos ignoriert.

Die Optionen und ihre Argumente sind nachstehend aufgelistet. Dabei wird jedes Argument so erklärt, als ob nur jeweils eine Datei verarbeitet werden soll, weil jedes Argument jede Datei unabhängig betrifft.

`-n` zeigt an, daß eine neue SCCS-Datei erstellt werden soll.

`-i[`*Name*`]` gibt den Namen einer Datei an, von der der Text für eine neue SCCS-Datei entnommen werden soll.
Der Text stellt das erste Delta der Datei dar (siehe `-r` Option für Delta-Numerierungssystem). Bei Verwendung der Option `-i` ohne Angabe eines Dateinamens wird der Text von der Standardeingabe erwartet. Wird diese Option weggelassen, ist die erstellte SCCS-Datei leer. Wird `-i` angegeben, kann nur eine SCCS-Datei erstellt werden.
Beachten Sie, daß die Option `-i` die Option `-n` impliziert.

`-r`*Version* gibt die *Version* (Releasenummer) an, in die das erste Delta eingefügt wird.
Diese Option darf nur bei gleichzeitiger Benutzung der Option `-i` verwendet werden. Wird die Option `-r` nicht eingesetzt, wird das erste Delta in Version 1 eingefügt. Der Level des ersten Deltas ist immer 1 (erste Deltas werden standardmäßig mit 1.1 bezeichnet).

`-t[`*Name*`]` gibt den Namen einer Datei an, aus der der beschreibende Text für die SCCS-Datei entnommen werden soll.
Wenn die Option `-t` zusammen mit `-n` und/oder `-i` verwendet wird und

admin eine neue SCCS-Datei erstellt, muß auch der Dateiname für den beschreibenden Text angegeben werden. Bei bereits vorhandenen SCCS-Dateien gilt folgendes: Eine Option -t ohne Dateinamen bewirkt, daß beschreibender Text entfernt wird, der zum gegebenem Zeitpunkt in der SCCS-Datei steht. Eine Option -t mit einem Dateinamen bewirkt, daß in der Datei vorhandener Text den in der aktuellen SCCS-Datei stehenden beschreibenden Text ersetzt.

-f*Flag* gibt den Wert für ein Kennzeichen an, das die SCCS-Datei erhalten soll. In einer admin-Kommandozeile können mehrere Optionen -f eingesetzt werden. Folgende *Flags* und ihre Werte sind zugelassen:

b läßt die Option -b bei dem Kommando get zur Erstellung von Verzweigungsdeltas zu.

c*ceil*
gibt die höchste Version an, die die Datei erhalten kann. *ceil* muß zwischen 1 und 9999 (einschließlich) liegen. Der Standardwert bei fehlendem c ist 9999.

f*floor*
gibt die Nummer der ersten Version an. *floor* muß zwischen 0 und 10000 (ausschließlich) liegen. Der Standardwert bei fehlendem f ist 1.

d*SID* ist die Standard-Deltanummer (SID), die von einem Kommando get zu verwenden ist.

i[*str*]
bewirkt, daß die von get oder delta ausgegebene Meldung No ID keywords (cm7) (Keine Schlüsselwörter) als schwerwiegender Fehler behandelt wird.
Standardmäßig ist diese Meldung lediglich eine Warnung. Die Meldung wird ausgegeben, wenn keine SCCS-Schlüsselwörter (siehe get(1)) in dem Text gefunden werden, der in der SCCS-Datei bereitgestellt bzw. gespeichert ist. Wird ein Wert angegeben, müssen die gelesenen Schlüsselwörter exakt mit dieser Zeichenkette übereinstimmen, wobei die Zeichenkette genau ein Schlüsselwort darstellen muß und keine eingebetteten Neue-Zeile-Zeichen aufweisen darf.

j läßt gleichzeitig mehrere Kommandos get für die Bearbeitung derselben SID einer SCCS-Datei zu. Dadurch können mehrere Aktualisierungen gleichzeitig an derselben Version der SCCS-Datei vorgenommen werden.

`l`*Liste*

Liste von Versionen (Releasenummern), an denen keine Deltas mehr vorgenommen werden können. `get -e` ist bei diesen 'gesperrten' Versionen erfolglos.
Liste hat die Syntax:

<Liste> ::= *<Bereich>* | *<Liste>* , *<Bereich>*
<Bereich> ::= *<Releasenummer>* | `a`

Das Zeichen `a` in *Liste* bedeutet, daß alle Versionen für die angegebene SCCS-Datei zu sperren sind.

`n` veranlaßt `delta` zur Erstellung eines Nulldeltas in jeder der Versionen, die eventuell übersprungen werden, wenn mit einer neuen Version gearbeitet wird, z. B. werden bei der Ausführung von Delta 5.1 im Anschluß an Delta 2.7 die Versionen 3 und 4 übersprungen. Diese Nulldeltas dienen als Ankerpunkte, an denen später Verzweigungen angelegt werden können. Das Fehlen dieser Option bewirkt, daß die übersprungenen Versionen in der SCCS-Datei nicht existieren und es daher nicht möglich ist, später Verzweigungsdeltas anzulegen.

`q`*Text*

vom Benutzer wählbarer Text, der bei jedem Auftreten des Schlüsselwortes `%Q%` in der mit `get` bereitgestellten SCCS-Datei ersetzt wird.

`m`*Mod* der Modulname der SCCS-Datei, der bei jedem Auftreten des Schlüsselwortes `%M%` in dem mit `get` bereitgestellten Text der SCCS-Datei ersetzt wird.
Ist die Option `m` nicht angegeben, handelt es sich bei dem zugewiesenen Wert um den Namen der SCCS-Datei ohne das führende `s.`

`t`*Typ* gibt den *Typ* des Moduls in der SCCS-Datei an, der bei jedem Auftreten des Schlüsselwortes `%Y%` in dem mit `get` bereitgestellten Text der SCCS-Datei ersetzt wird.

`v[`*pgm*`]`

`delta` fordert Änderungsanforderungs-Nummern (MR-Nr.) als Grund für die Erstellung eines Deltas an.
Das optionale Argument gibt den Namen eines Gültigkeitsprüfprogramms für MR-Nummern an (siehe `delta`(1)). Dieses Programm erhält als Argumente den Modulnamen, den Wert des *Typ*-Flags (siehe `t`*Typ* oben) und die *MR-Liste*. Wenn diese Option während der Erstellung einer SCCS-Datei gesetzt wird, muß auch die Option `m` verwendet werden, selbst wenn die MR-Liste leer ist.

`-d`*Flag* *Flag* wird aus der SCCS-Datei gelöscht.
In einem Kommando `admin` können mehrere `-d` Optionen verwendet werden. Für zulässige Werte für *Flag* siehe Option `-f`.

`-a`*Login* zur Liste der Benutzer, die Deltas (Änderungen) in der SCCS-Datei vornehmen dürfen, kann ein Login-Name oder eine Gruppennummer hinzugefügt werden.
Eine Gruppennummer steht stellvertretend für alle Login-Namen, die zu dieser Gruppe gehören. In einer `admin`-Kommandozeile können mehrere Optionen `a` verwendet werden. In der Liste kann eine beliebige Anzahl von *Logins* oder Gruppennummern gleichzeitig aufgeführt sein. Wenn die Liste der Benutzer leer ist, darf jeder Benutzer Deltas hinzufügen. Wenn vor dem Login oder der Gruppennummer ein `!` steht, wird diesen Benutzern ausdrücklich keine Erlaubnis zum Erzeugen von Deltas gegeben.

`-e`*Login* ein *Login*-Name oder eine Gruppennummer aus der Liste der Benutzer mit Erlaubnis zur Durchführung von Deltas in der SCCS-Datei wird gelöscht. Die Angabe einer Gruppennummer ist gleichbedeutend mit der Angabe aller *Login*-Namen, die zu dieser Gruppe gehören. In einer `admin`-Kommandozeile können mehrere Optionen `-e` verwendet werden.

`-m[`*MR-Liste*`]`
die Liste der Änderungsanforderungs-Nummern (MR-Nummern) wird in die SCCS-Datei eingefügt als Grund für die Erstellung des ersten Deltas, und zwar in der gleichen Weise wie bei `delta`.
Das Flag `v` der Option `-f` muß gesetzt werden, und die MR-Nummern werden validiert, wenn die Option `v` ein Argument hat (den Namen eines MR-Nummer-Validierungsprogramms). Eine Meldung wird ausgegeben, wenn die Option `v` nicht gesetzt wurde oder die MR-Validierung erfolglos war.

`-y[`*Kommentar*`]`
der *Kommentar* wird in derselben Weise wie bei `delta` als Kommentar für das erste Delta in die SCCS-Datei eingefügt.
Das Weglassen der Option `-y` führt zum Einfügen einer Standard-Kommentarzeile.
Die Option `-y` ist nur gültig, wenn die Optionen `-i` und/oder `-n` angegeben werden, d.h. wenn eine neue SCCS-Datei erstellt wird.

`-h` `admin` prüft die Struktur der SCCS-Datei (siehe `sccsfile`(4))
Eine neuberechnete Prüfsumme (die Summe aller Zeichen in der SCCS-Datei mit Ausnahme der Zeichen in der ersten Zeile) wird mit der Prüfsumme verglichen, die in der ersten Zeile der SCCS-Datei gespeichert ist, und gibt ggf. eine Fehlermeldung aus. Die Option `-h` verhindert das Schreiben in die Datei, so daß der Effekt von allen anderen angegebenen Optionen aufgehoben wird.

`-z` die Prüfsumme der SCCS-Datei wird erneut berechnet und in der ersten Zeile der SCCS-Datei gespeichert (siehe `-h`).
Beachten Sie, daß die Verwendung dieser Option in einer stark beschädigten Datei eine spätere Feststellung des Fehlers verhindern kann.

Der Dateiname ohne Pfadangabe jeder SCCS-Datei muß die Form `s.`*Dateiname* haben. Neue SCCS-Dateien werden mit Modus 444 angelegt. Für die Erstellung einer Datei benötigen Sie die Schreiberlaubnis in dem entsprechenden Dateiverzeichnis. `admin` schreibt in eine temporäre Datei mit dem Namen `x.`*Dateiname*. Erst nach erfolgreicher Ausführung der gewünschten Aktion wird die ggf. vorhandene Originaldatei gelöscht und die x-Datei umbenannt. Damit wird sichergestellt, daß die Änderungen an der Original-SCCS-Datei nur dann vorgenommen werden, wenn keine Fehler aufgetreten sind.

Für Dateiverzeichnisse mit SCCS-Dateien wird der Modus 755 empfohlen und für die SCCS-Dateien selbst der Modus 444. Der Modus der SCCS-Dateien gewährleistet, daß Änderungen nur mit den SCCS-Kommandos vorgenommen werden können.

Um eine gleichzeitige Aktualisierung der SCCS-Datei durch verschiedene Benutzer zu verhindern, verwendet `admin` außerdem eine vorübergehende Sperrdatei, die `z.`*Dateiname* genannt wird. Weitere Angaben hierzu siehe `get`(1).

DATEIEN

x-Datei	siehe `delta`(1)
z-Datei	siehe `delta`(1)

SIEHE AUCH

`bdiff`(1), `ed`(1), `delta`(1), `get`(1), `help`(1), `prs`(1), `what`(1), `sccsfile`(4).

HINWEIS

Sollte aus irgendeinem Grund eine Direktkorrektur einer SCCS-Datei erforderlich sein, können Sie als Eigentümer den Modus auf 644 ändern, so daß mit einem Text-Editor gearbeitet werden kann. Anschließend müssen Sie `admin -z` zur Erzeugung einer richtigen Prüfsumme aufrufen. Sie sollten `admin -h` aufrufen, um sicherzustellen, daß die SCCS-Datei gültig ist.

ar - portierbare Archive oder Bibliotheken verwalten

ar [-V] -Schlüssel [Arg] [Position] Archiv [Name...]

Das Kommando `ar` (archive) verwaltet Gruppen von Dateien, die in einer einzigen Archivdatei zusammengefaßt sind. Der Hauptzweck dieses Kommandos besteht in der Erstellung und Aktualisierung von Bibliotheksdateien. Dieses Kommando läßt sich jedoch auch für alle ähnlichen Zwecke einsetzen.

Die im Archiv enthaltene interne Verwaltungsinformation hat ein Format, das auf alle Maschinen übertragbar ist. Format und Struktur dieses portierbaren Archivs sind im einzelnen in `ar`(4) beschrieben. Die Symboltabelle des Archivs wird vom Binder-Programm `ld` benutzt, um mehrmalige Durchläufe durch Objektdatei-Bibliotheken effektiv durchführen zu können. Eine Symboltabelle wird nur dann von `ar` erstellt und verwaltet, wenn sich mindestens eine Objektdatei im Archiv befindet. Wird `ar` aufgerufen, um den Inhalt eines solchen Archivs zu erstellen oder zu aktualisieren, dann wird die Symboltabelle neu aufgebaut. Mit der nachstehend beschriebenen Option `s` wird der erneute Aufbau der Symboltabelle erzwungen.

Die Option `-V` veranlaßt `ar`, die Versionsnummer des aktuell laufenden C-Entwicklungssystems (SCDE) auf Standard-Fehlerausgabe auszugeben.

Schlüssel ist erforderlicher Bestandteil der Kommandozeile von `ar`. *Schlüssel* wird aus einem der folgenden Buchstaben gebildet: `drqtpmx`; Argumente für *Schlüssel* werden alternativ aus einem oder mehreren der folgenden Zeichen gebildet: `vucls`. Mit *Position* kann man steuern, wohin eine Datei innerhalb des Archivs plaziert werden soll, und zwar in Bezug auf eine andere im Archiv enthaltene Datei *posdatei*. *Position* kann folgendes Format haben:

`a` *posdatei* Die Datei(en) *Name...* werden hinter *posdatei* plaziert.

`b` *posdatei* oder `i` *posdatei*
Die Dateien werden vor *posdatei* plaziert.

Wenn *Position* nicht angegeben ist, werden die angegebenen Dateien an das Ende des Archivs plaziert.

Archiv ist der Name der Archivdatei.

Name... sind die Namen der zu bearbeitenden Dateien.

Die *Schlüssel*-Zeichen haben folgende Bedeutung:

`d` angegebene Dateien aus der Archivdatei löschen

r angegebene Dateien in der Archivdatei ersetzen
Bei Verwendung des optionalen Zeichens u in Verbindung mit r werden nur die Dateien ersetzt, deren Änderungsdatum neuer als das Datum der entsprechenden archivierten Datei ist.

q angegebene Dateien schnell an das Ende der Archivdatei setzen
Das Kommando prüft nicht, ob sich die hinzugefügten Dateien bereits im Archiv befinden. Diese Option ist für die Vermeidung von quadratischem Laufzeitverhalten bei schrittweiser Erstellung eines großen Archivs nützlich.

t ein Inhaltsverzeichnis der Archivdatei ausgeben
Werden keine Namen angegeben, erfolgt die Auflistung aller Dateien im Archiv. Bei Angabe von Namen werden nur die genannten Dateien erfaßt.

p die Inhalte der benannten Dateien im Archiv auf Standardausgabe schreiben.

m die angegebenen Dateien an die durch *Position* bestimmte Stelle oder an das Archivende verlagern.

x die angegebenen Dateien aus dem Archiv holen
Werden keine Namen angegeben, werden alle Dateien aus dem Archiv geholt. Die Archivdatei bleibt unverändert.

Die anderen *Schlüssel*-Zeichen haben folgende Bedeutung:

v gibt die Namen der Dateien, die bearbeitet werden aus bzw. mit dem Schlüssel t ausführlichere Information über die gewünschten Dateien.

c unterdrückt die Meldung, die standardmäßig bei der Erstellung von *Archiv* ausgegeben wird

l Diese Option ist veraltet. Sie wird erkannt aber ignoriert und im nächsten Release nicht mehr aufgeführt.

s stellt die Archivsymboltabelle wieder her, falls sie entfernt worden ist (siehe strip(1)).

SIEHE AUCH

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4)

HINWEIS

Wenn dieselbe Datei zweimal in einer Argumentliste erwähnt wird, kann sie auch zweimal in das Archiv aufgenommen werden.

Archivnamen sollten mit .a enden.

cb - C-Formatier-Programm

cb [-s] [-j] [-l Länge] [-V] [Datei...]

Das Kommando `cb` (C program beautifier) liest C-Programme entweder aus den als Argumente angegebenen Dateien oder von der Standardeingabe und schreibt sie auf die Standardausgabe mit Abständen und Einrückungen, so daß die Struktur des Codes angezeigt wird. Standardmäßig bewahrt `cb` alle Neue-Zeile-Zeichen des Benutzers.

`cb` akzeptiert folgende Optionen:

`-s`	paßt den Code an den Stil von Kernighan/Ritchie an
`-j`	getrennte Zeilen werden wieder zusammengesetzt
`-l` *Länge*	`cb` spaltet Zeilen, die länger als *Länge* sind
`-V`	druckt auf Standard-Fehlerausgabe die `cb` Version

HINWEIS

`cb` behandelt `asm` wie ein Schlüsselwort.

Das Format der Strukturinitialisierungen bleibt durch `cb` unverändert.

In Präprozessor-Anweisungen verborgene Interpunktion verursacht Fehler in der Einrükkung.

SIEHE AUCH

`cc`(1).
Kernighan/Ritchie: "Die Programmiersprache C".

cc - C-Compiler

cc [Option...] Datei... [-larchivkuerzel...]

Mit cc können Sie alle Schritte ausführen, die zur Übersetzung eines C-Quellprogramms erforderlich sind. Standardmäßig erzeugt cc aus einem C-Quellprogramm mit Präprozessor-Anweisungen ein ablauffähiges Programm in der Datei a.out.

Durch Angabe von Optionen und von bereits teilweise übersetzten Dateien können Sie das Standardverhalten ändern, so daß cc nur einen Teil der Übersetzungsphasen durchführt. Ist der Übersetzungsprozeß nicht vollständig, dann werden Optionen für nicht durchlaufene Phasen ignoriert.

cc erzeugt Dateien für Zwischen- und Endergebnisse. Daher müssen Sie für das Dateiverzeichnis Schreiberlaubnis haben, in dem cc die Dateien erzeugt. Standardmäßig erzeugt cc die Dateien in dem Dateiverzeichnis, in dem cc aufgerufen wird.

Durch Angabe von *Optionen* im cc-Aufruf können Sie den Ablauf steuern und beeinflussen, mit welchen Argumenten die Programme für die einzelnen Übersetzungsphasen versorgt werden. Wird keine *Option* angegeben erstellt cc, falls die Dateiinhalte der angegebenen Dateien syntaktisch korrekt sind und alle offenen Referenzen gelöst werden können, eine ausführbare Datei a.out, die das ablauffähige Programm enthält. Sind mehrere Dateien angegeben, dann speichert cc den Objektcode zu den einzelnen Dateien zusätzlich in gleichnamigen .o-Dateien ab.

Datei ist der Name der Eingabedatei. cc schließt aus der Endung des Dateinamens auf den Dateiinhalt und führt die jeweils noch erforderlichen Übersetzungsschritte aus. Der Dateiname muß daher das Suffix haben, das gemäß den SINIX-Konventionen zu dem Dateiinhalt paßt. Im einzelnen gibt es folgende Möglichkeiten:

Suffix	Dateiinhalt
.c	C-Quellcode vor Präprozessorlauf
.i	C-Quellcode nach Präprozessorlauf
.s	Assembler-Quellcode
.o	Objektcode
.a	Bibliothek mit Objektmodulen
.so	mehrfach benutzbare Bibliothek mit Objektmodulen

Sie müssen mindestens eine Nicht-Bibliotheksdatei angeben. Sie können mehrere Dateien angeben.

cc unterstützt die getrennte Übersetzung. Sie können ein C-Programm auf mehrere Dateien verteilen, die Dateien einzeln übersetzen und dann zusammenbinden. Die Dateien, die Sie im cc-Aufruf angeben, müssen nicht vom gleichen Typ sein. Sie können in demselben Aufruf C-Quelldateien, Assembler-Dateien und Objektmodule angeben. Geben Sie mehrere Bibliotheken an, ist die Reihenfolge und die Position auf der Kommandozeile für den Binder wichtig.

Folgende allgemeine Optionen werden von cc ausgewertet:

-V
Während der Ausführung von cc und den von cc aufgerufenen Übersetzungsphasen werden Meldungen über den Ablauf auf die Standardfehlerausgabe geschrieben.

-W[*n*]
Mit dieser Option können Sie festlegen, in welchem Umfang Warnungen ausgegeben werden. *n* kann sein: 0, 1 oder 2.

0 Alle Warnungen werden unterdrückt.
1 Nur bestimmte, wichtige Warnungen werden ausgegeben.
2 Alle Warnungen werden ausgegeben.

Die bei *n* = 2 zusätzlich ausgegebenen Warnungen betreffen interne Abläufe, die für ein Anwendungsprogramm ohne Bedeutung sind.

Wenn -W[*n*] nicht angegeben wird, dann werden nur bestimmte, wichtige Warnungen ausgegeben (wie -W1). Wenn -W ohne *n* angegeben wird, gilt *n* = 2.

Folgende Optionen steuern den Übersetzungsablauf:

-P
Nur der Präprozessor wird aufgerufen.
cc wird nach dem Präprozessor-Lauf angehalten. Das Ergebnis zu einer Datei *datei*.c wird in eine Datei *datei*.i geschrieben. Die Datei *datei*.i kann mit cc weiter übersetzt werden.

-E
Wie bei der Option -P wird nur der Präprozessor aufgerufen. Das Ergebnis wird aber auf die Standardausgabe statt in eine Datei geschrieben.

-S
Die angegebenen Dateien werden in Assembler-Quellcode übersetzt. Danach stoppt cc. Der Assembler-Quellcode zu einer Datei *datei*.c wird in die Datei *datei*.s geschrieben.

-c
Die angegebenen Dateien werden in Objektcode übersetzt. Danach wird cc gestoppt. Der Objektcode zu einer Datei *datei*.c wird in die Datei *datei*.o geschrieben.

Alle Optionen des Präprozessors können beim cc-Aufruf angegeben werden. Sie werden durch das cc-Kommando an den Präprozessor weitergereicht.

-C
C-Kommentare /* ... */ werden nicht entfernt.

-D*name*[=*wert*]
Die symbolische Konstante *name* wird definiert wie durch die Präprozessor-Anweisung #define. Wird auch *wert* angegeben, so entspricht dies #define *name wert*.

-I*dv*
dv wird in die Liste der Dateiverzeichnisse aufgenommen, in denen der Präprozessor nach Include-Dateien sucht.
Bei Include-Dateien, deren relativer Dateiname (beginnt nicht mit

Schrägstrich /) in Anführungszeichen "..." eingeschlossen ist, durchsucht der Präprozessor die Dateiverzeichnisse in folgender Reihenfolge:

1. das Dateiverzeichnis der Quelldatei, die die `#include`-Anweisung enthält
2. die Dateiverzeichnisse, die mit der Präprozessor-Option `-I` angegeben wurden
3. das Dateiverzeichnis `/usr/include`.

Bei Include-Dateien, deren relativer Dateiname in spitzen Klammern <...> eingeschlossen ist, durchsucht der Präprozessor die Dateiverzeichnisse in folgender Reihenfolge:

1. die Dateiverzeichnisse, die mit der Präprozessor-Option `-I` angegeben wurden
2. das Dateiverzeichnis `/usr/include`.

Enthält eine Include-Datei selbst auch eine Include-Anweisung ohne spitze Klammern <...>, durchsucht der Präprozessor die Dateiverzeichnisse in folgender Reihenfolge:

1. das Dateiverzeichnis der Include-Datei
2. die Dateiverzeichnisse, die mit der Präprozessor-Option `-I` angegeben wurden.

`-U`*name*

Die Definition für ein Makro bzw. eine symbolische Konstante *name* wird gelöscht wie durch die Präprozessor-Anweisung `#undef`.
Kommt *name* in der Eingabedatei nicht vor, wird die Option `-U` ignoriert. Enthält die Eingabedatei `#define`-Anweisungen für *name*, hat die Option `-U` keine Wirkung.

name ist ein vordefinierter Präprozessor-Name oder ein Name, der **vor** dem Präprozessor-Aufruf definiert wurde.

Wird `-U`*name* nicht angegeben, dann gelten nur die vom System vordefinierten Namen.

Folgende Übersetzer-Optionen werden von `cc` ausgewertet:

`-k[`*modus*`]`

Angabe des Sprachmodus.
Wird die Option `-k` nicht angegeben, verwendet der Übersetzer den K&R-Modus, so wie er in der ersten Ausgabe "The C Programming Language" von Kernighan und Ritchie beschrieben ist (`__STDC__=0`).

Für *modus* kann folgendes angegeben werden:

`ansi` alle ANSI-C Syntax- und Semantik-Konstrukte werden akzeptiert und `__STDC__` wird auf 1 gesetzt.

`cc` K&R-Modus:
Die C-Definition gemäß Kernighan&Ritchie wird verwendet.

`cle` Einige CLE-Erweiterungen werden akzeptiert. Diese Option kann nur in einigen Systemen angegeben werden. Sehen Sie bitte in der Kurzbeschreibung "cc-Kommando" für Ihr System nach.

Wird *modus* nicht angegeben, dann gilt der K&R-Modus (`__STDC__=0`).

`-p` Der Übersetzer erzeugt zusätzlich für jede Funktion Code, der mitzählt, wie oft die Funktion aufgerufen wird. Führen Sie das übersetzte Programm aus, wird bei erfolgreicher Beendigung des Programms eine Profildatei `mon.out` erzeugt, die die Zählerwerte enthält.
Mit dem Kommando `prof` können Sie aus der Profildatei ein Laufzeitprofil des Programms erstellen. `prof` liefert CPU-Zeit und Aufrufstatistik für alle Funktionen des Programms, die mit der Übersetzeroption `-p` übersetzt wurden.

`-qp` Diese Option hat die gleiche Wirkung wie `-p`.

`-g` Der Übersetzer speichert Informationen für symbolische Testhilfen. Geben Sie in einem Aufruf gleichzeitig die Optionen `-g` und `-O` an, dann setzt die Option `-g` die Option `-O` außer Kraft. Geben Sie gleichzeitig die Binder-Option `-s` an, ist die Option `-g` wirkungslos. Dadurch entstehen Einschränkungen beim Testen mit symbolischen Testhilfen. Da die Symboltabellen mit der Binder-Option `-s` entfernt werden, kann nur noch auf Maschinencode-Ebene getestet werden.

Die cc-Optionen zur Optimierung und Codegenerierung sind maschinenabhängig. Sehen Sie dafür bitte in der Kurzbeschreibung "cc-Kommando" für Ihr System nach!

`-O` `cc` ruft zusätzlich ein Programm auf, das die Optimierungen durchführt. Üblicherweise können Sie damit die Größe der Objektdatei reduzieren und die Laufzeit verkürzen. Allerdings wird die Übersetzungszeit erhöht. Ob die Option `-O` Auswirkungen auf eine `.s`-Datei als Eingabedatei hat, ist maschinenabhängig.

`-Kpic` oder `-KPIC`
Diese Option steuert die Generierung von positionsunabhängigem Code. Welcher Code auf Ihrem System standardmäßig eingestellt ist und wie ggf. umgeschaltet werden kann, entnehmen Sie bitte der für Ihr System relevanten Kurzbeschreibung "cc-Kommando".

Einige der für das ld-Kommando definierten Optionen können Sie auch im cc-Aufruf angeben (siehe auch ld(1)):

-Bdynamic
: Die Option -Bdynamic veranlaßt den Binder ld ab der Angabe dieser Option auf der Aufrufzeile nur nach dynamischen Bibliotheken (libx.so) zu suchen. Dieses wird bis zum Ende der Aufrufzeile oder bis zum Auftreten einer -Bstatic-Option fortgesetzt.
Sie dürfen nicht gleichzeitig die Optionen -Bdynamic und -dn angeben.

-Bstatic
: Die Option -Bstatic veranlaßt den Binder ld ab der Angabe dieser Option auf der Aufrufzeile nur nach statischen Bibliotheken (libx.a) zu suchen. Dieses wird bis zum Ende der Aufrufzeile oder bis zum Auftreten einer -Bdynamic-Option fortgesetzt.
Sie dürfen nicht gleichzeitig die Optionen -Bstatic und -G angeben.

-Bsymbolic
: Angabe, wie bei dynamischem Binden offene Referenzen aufgelöst werden.
Enthält ein Objektmodul Definitionen für globale Symbole, löst ld die offenen Referenzen auf diese Symbole schon beim Erzeugen des mehrfach benutzbaren Objektmoduls auf. Auf diese Weise wird, falls vorhanden, immer die objekteigene Definition für ein Symbol verwendet.
Normalerweise werden Referenzen zu globalen Symbolen innerhalb mehrfach benutzbarer Objektmodule erst zur Ausführungszeit aufgelöst, auch wenn die Definitionen vorhanden sind. Dadurch kann es vorkommen, daß statt der objekteigenen Definition für ein Symbol eine Definition verwendet wird, die in einem zuvor angegebenen ausführbaren Programm oder Objektmodul steht. Mit -Bsymbolic kann dies verhindert werden.
Sie dürfen nicht gleichzeitig die Optionen -Bsymbolic und -dn angeben. Die Option -Bsymbolic funktioniert nur im PIC-Code

-d*o*
: Globale Einstellung von ld für dynamisches oder statisches Binden. *o* ist entweder y oder n. dy spezifiziert Dynamisch Binden. Dabei werden die Objektmodule erst zur Ausführungszeit gebunden. dn spezifiziert Statisch Binden. Die Objektmodule werden schon zur Übersetzungszeit gebunden.
Sie dürfen nicht gleichzeitig die Optionen -Bdynamic und -dn angeben.

-G
: Mehrfach benutzbares Objekt erzeugen.
Sie dürfen nicht gleichzeitig die Optionen -G und -dn bzw. -Bstatic angeben.

`-h`*name*

Für das erzeugte mehrfach benutzbare Objektmodul wird der Name *name* verwendet. *name* wird in den Abschnitt für dynamisches Binden (`.dynamic section`) der Objektdatei und in alle Programme, die mit dem Objektmodul gebunden werden, eingetragen. Der dynamische Binder sucht nach *name*, wenn er externe Referenzen auflösen will.
Die Option `-h`*name* ist nur zusammen mit der Option `-G` wirksam.

`-LD["]`*ld_option*`["]`

Binder-Option angeben.
Für *ld_option* können Sie jede beliebige `ld`-Option angeben.
Sie müssen die `-LD`-Angabe in Anführungszeichen setzen, wenn *ld_option* Leerzeichen enthält.

`-L`*dv*

Weiteres Suchverzeichnis angeben.
Diese Option wird vor allem zusammen mit der `-l`-Option verwendet. Mit *dv* wird ein zusätzliches Dateiverzeichnis angegeben, in dem der Binder nach der mit der Option `-l` angegebenen Bibliothek suchen soll. Die so angegebenen Verzeichnisse werden noch vor den Standard-Bibliotheksverzeichnissen nach den angegebenen Bibliotheken durchsucht.

`-o` *ausgabedatei*

Wird ein ablauffähiges Programm erstellt oder ist die Option `-c` gesetzt und wird nur ein Objektmodul erstellt, dann heißt die Ausgabedatei *ausgabedatei*.

`-l`*archivkuerzel*

`ld` bindet die Bibliothek mit dem Kürzel *archivkuerzel* dazu.
Standardmäßig durchsucht `ld` folgende Dateiverzeichnisse in der angegebenen Reihenfolge nach der Bibliothek:

1. `/usr/ccs/lib`
2. `/usr/lib`

Maschinenabhängig können weitere Verzeichnisse durchsucht werden.

Wird dynamisch gebunden, sucht `ld` zuerst nach der dynamischen Version der Bibliothek mit dem Namen `lib`*archivkuerzel*`.so`.
Wird statisch gebunden, sucht `ld` nur die statische Version `lib`*archivkuerzel*`.a` und bindet diese dazu.

Statt des absoluten Dateinamens `lib`*archivkuerzel*`.a` bei statischem Binden bzw. `lib`*archivkuerzel*`.so` bei dynamischem Binden geben Sie mit *archivkuerzel* nur den für die Bibliothek spezifischen Teil an.

Haben Sie mit der Option -L*dv* ein Dateiverzeichnis angegeben, das zusätzlich nach Bibliotheken durchsucht werden soll, müssen Sie eine Bibliothek aus diesem Dateiverzeichnis mit der Option -l in der Form -l*archivkuerzel* angeben. Die Bibliothek muß unter dem Namen lib*archivkuerzel*.so oder lib*archivkuerzel*.a im Dateiverzeichnis vorhanden sein.

Wird -l nicht angegeben, dann ruft cc den Binder immer mit der Option -lc auf, so daß automatisch die Standard-C-Bibliothek /usr/ccs/lib/libc.so bzw. /usr/ccs/lib/libc.a dazugebunden wird.

SIEHE AUCH

dbx(1), ld(1), lint(1), prof(1), monitor(3C), tmpnam(3S).

Kapitel "Das C-Übersetzungssystem" in "Leitfaden und Werkzeuge für die Programmierung mit C".

"CES V5.41 cc-Kommando" Kurzbeschreibung für die verschiedenen Maschinen.

Kernighan/Ritchie: "Programmieren in C".

HINWEIS

Eine vollständige Beschreibung aller auf Ihrer Maschine möglichen Optionen für das cc-Kommando finden Sie in der entsprechenden Kurzbeschreibung "CES V5.41 cc-Kommando" für die verschiedenen Maschinen.

Zum Testen der mit CES erstellten Programme sollte die Symbolische Testhilfe dbx(1) verwendet werden. Die Zusammenarbeit von CES mit anderen Debuggern wird nicht garantiert.

Achtung: Die Option -KPIC kann erst ab CES V5.41 verwendet werden.

Der CES-C-Übersetzer ist der Übersetzer des integrierten C-Entwicklungssystems für die SINIX-Rechnerfamilie der Siemens Nixdorf Informationssysteme AG. Alternativ kann auch das C-Entwicklungssystem von AT&T mit dem C-Übersetzer PCC genutzt werden. Den PCC-Übersetzer rufen Sie wie folgt auf: /usr/ccs/bin/cc.

cdc - Deltakommentar eines SCCS-Deltas ändern

cdc -r SID [-m[MR-Liste]] [-y[Kommentar]] Datei...

`cdc` (change delta comment) ändert den Deltakommentar für die mit der Option `-r` angegebene SID (SCCS-Identifikationszeichenkette) jeder angegebenen SCCS-Datei.

Der Deltakommentar besteht aus den Änderungsanforderungen (modification request MR) und den Kommentardaten, die normalerweise bei dem Kommando `delta` (Optionen `-m` und `-y`) angegeben werden.

Wenn ein Dateiverzeichnis angegeben wird, verhält sich `cdc` so, als ob jede Datei im Dateiverzeichnis angegeben wurde, wobei jedoch Nicht-SCCS-Dateien (siehe `admin`(1)) und unlesbare Dateien kommentarlos ignoriert werden. Bei Angabe von - wird die Standardeingabe gelesen und jede Zeile der Standardeingabe wird als Name einer zu bearbeitenden SCCS-Datei angesehen.

Die Argumente für `cdc`, die in beliebiger Reihenfolge erscheinen können, bestehen aus Optionen und Dateinamen. Alle beschriebenen Optionsargumente gelten unabhängig für jede angegebene Datei:

`-r`*SID* wird zur Angabe der SID eines Deltas verwendet, für das der Deltakommentar geändert werden soll

`-m`*MR-Liste* Wenn die Option `v` in der SCCS-Datei gesetzt wurde (siehe `admin`(1)), dann kann eine Liste der MR-Nummern angegeben werden, die im Deltakommentar der bei der Option `-r` angegebenen SID hinzugefügt und/oder gelöscht werden sollen. Eine leere *MR-Liste* hat keine Auswirkung. MR-Nummern werden in der gleichen Weise zur Liste der MRs hinzugefügt wie bei `delta`. Zum Löschen einer MR müssen Sie der MR-Nummer das Zeichen `!` voranstellen (siehe Abschnitt BEISPIELE). Wenn sich die zu löschende MR in der Liste der MRs befindet, wird sie entfernt und in eine Kommentarzeile geändert. Eine Liste aller gelöschten MRs wird in den Kommentarteil des Deltakommentars gesetzt, dem eine Kommentarzeile mit der entsprechenden Löschungsmeldung vorangestellt wird.
Wird `-m` nicht benutzt und ist die Standardeingabe die Tastatur, wird der Prompt `MRs?` auf der Standardausgabe ausgegeben, bevor die Standardeingabe gelesen wird; wenn die Standardeingabe nicht die Tastatur ist, wird kein Prompt ausgegeben. Der Prompt `MRs?` steht immer vor dem Prompt `comments?` (siehe `-y`).
Einträge in der *MR-Liste* werden mit Leerzeichen und/oder Tabulatorzeichen voneinander getrennt. Die MR-Liste wird mit einem nicht entwerteten Neue-Zeile-Zeichen abgeschlossen.
Wenn das Flag `v` ein Argument hat (siehe `admin`(1)), wird dieses als Name eines Programms (oder einer Shell-Prozedur) verwendet, mit dem die Richtigkeit der MR-Nummern validiert wird. Wenn ein Endestatus ungleich Null

vom Validierungsprogramm zurückgegeben wird, bricht `cdc` ab, und der Deltakommentar bleibt unverändert.

`-y`[*Kommentar*]
: beliebiger Text, der die Kommentare ersetzt, die bisher für das mit der Option `-r` angegebene Delta vorhanden waren.

  Die vorhergehenden Kommentare werden beibehalten, und ihnen wird eine Kommentarzeile mit dem Hinweis vorangestellt, daß sie geändert wurden. Ein leerer *Kommentar* hat keine Auswirkungen.

  Wenn `-y` nicht angegeben wurde und die Standardeingabe die Tastatur ist, wird der Prompt `comments?` auf der Standardausgabe ausgegeben, bevor die Standardeingabe gelesen wird; ist die Standardeingabe nicht die Tastatur, wird kein Prompt ausgegeben. Ein nicht entwertetes Neue-Zeile-Zeichen beendet den Kommentar-Text.

Wenn Sie das Delta erstellt haben und über die entsprechenden Zugriffsrechte verfügen, können Sie seinen Deltakommentar ändern.

BEISPIELE

Mit der Eingabe

```
cdc -r1.6 -m"bl88-12345 !bl87-54321 bl89-00001" -ytrouble s.file
```

werden `bl88-12345` und `bl89-00001` zur MR-Liste hinzugefügt, `bl87-54321` von der MR-Liste entfernt und der Kommentar `trouble` zu Delta 1.6 von `s.file` hinzugefügt.

Die folgende Eingabe erfüllt dieselbe Aufgabe:

```
cdc -r1.6 s.file
MRs? !bl87-54321 bl88-12345 bl89-00001
comments? trouble
```

DATEIEN

x-Datei	siehe `delta`(1)
z-Datei	siehe `delta`(1)

SIEHE AUCH

`admin`(1), `delta`(1), `get`(1), `help`(1), `prs`(1), `sccsfile`(4).

HINWEIS

Werden SCCS-Dateinamen über die Standardeingabe (- auf der Kommandozeile) an das Kommando `cdc` geliefert, müssen auch die Optionen `-m` und `-y` verwendet werden.

cflow - C-Flußdiagramm erstellen

```
cflow [-r] [-ix] [-i_] [-dnum] Datei...
```

Das Kommando `cflow` analysiert eine Sammlung von `yacc`, `lex`, Assembler-, C- und Objektdateien und erstellt eine grafische Darstellung auf Grundlage der externen Verweise. Dateien mit dem Zusatz `.y`, `.l` und `.c` werden entsprechend mit `yacc`, `lex` und dem C-Übersetzungssystem bearbeitet. Das Ergebnis der vorübersetzten Dateien und die Dateien mit dem Zusatz `.i` werden dann mit dem ersten Durchgang von `lint` bearbeitet. Die Dateien mit dem Zusatz `.s` werden assembliert. Die Symboltabellen der assemblierten Dateien und der Dateien mit nachgestelltem `.o` werden gelesen. Die Ergebnisse werden gesammelt und in eine grafische Darstellung externer Verweise umgewandelt, die in die Standardausgabe geschrieben wird.

In jeder Ausgabezeile stehen nacheinander eine Zeilennummer, der Name des globalen Symbols, ein Doppelpunkt und die Definition des Typs. Je nach Schachtelungstiefe der Referenz wird der Name um die entsprechenden Tabulatorpositionen eingerückt. Normalerweise werden nur Funktionsnamen aufgelistet, die nicht mit einem Unterstrichsymbol beginnen (siehe `-i`). Bei Informationen aus dem C-Quellcode besteht die Definition aus einer Typbezeichnung, dem durch spitze Klammern begrenzten Namen der Quelldatei und der Nummer der Zeile, in der die Definition gefunden wurde. Definitionen aus Objektdateien zeigen den Dateinamen und den Positionszähler an, unter dem das Symbol (z.B. *Text*) erschienen ist. Führende Unterstriche in externen Namen werden gelöscht. Wenn die Definition eines Namens bereits ausgegeben ist, enthalten alle nachfolgenden Verweise auf diesen Namen nur noch die Bezugsnummer der Zeile, in der die Definition zu finden ist. Bei undefinierten Verweisen wird nur <> ausgegeben.

Außer den Optionen `-D`, `-I` und `-U`, die wie bei `cc` interpretiert werden, werden auch die folgenden Optionen von `cflow` interpretiert:

`-r` die Beziehung 'Aufrufer:Aufgerufener' umkehren, wodurch eine umgekehrte Auflistung mit den Aufrufern jeder Funktion angezeigt wird.
Die Liste wird in der lexikografischen Reihenfolge der Aufgerufenen sortiert.

`-ix` externe und statische Datensymbole einschließen.
Standardmäßig werden nur Funktionen in das Diagramm aufgenommen.

`-i_` Namen einschließen, die mit einem Unterstrich beginnen.
Standardmäßig werden diese Funktionen (und Daten bei Verwendung von `-ix`) nicht eingeschlossen.

`-d`*num* *num* zeigt die maximale Verschachtelungstiefe für das Diagramm an. Standardmäßig ist diese Zahl sehr groß. Versuche, die Verschachtelungstiefe auf eine negative ganze Zahl zu setzen, werden ignoriert.

BEISPIEL

Gegeben sei folgende Datei `file.c`:

```
int   i;

main()
{
      f();
      g();
      f();
}

f()
{
      i = h();
}
```

Hieraus erzeugt das Kommando

```
cflow -ix file.c
```

die Ausgabe

```
1      main: int(), <file.c 4>
2              f: int(), <file.c 11>
3                      h: <>
4                      i: int, <file.c 1>
5              g: <>
```

Wenn die Verschachtelungstiefe zu kompliziert wird, kann die Ausgabe von `cflow` über eine Pipe (mit der Option `-e`) an das Kommando `pr` geschickt werden, um die Tabulatorexpandierung auf weniger als acht Leerstellen zu setzen.

SIEHE AUCH

`cc`(1), `lex`(1), `lint`(1), `nm`(1), `yacc`(1).
`pr`(1) in "SINIX V5.41 Kommandos"

ENDESTATUS

`cflow` meldet mehrfache Definitionen. Nur die erste wird anerkannt.

HINWEIS

Durch `lex` und `yacc` erstellte Dateien bewirken die Umordnung von Zeilennummer-Deklarationen, was die Bearbeitung durch `cflow` stören kann. Um richtige Ergebnisse zu erhalten, müssen Sie `cflow` mit der Eingabe von `yacc` oder `lex` versorgen.

cof2elf - COFF-Objektdatei in ELF-Objektdatei umsetzen

cof2elf [-iqV] [-Qarg] [-s Verzeichnis] Datei...

`cof2elf` (COFF to ELF) konvertiert eine oder mehrere COFF-Objektdateien in ELF-Objektdateien. Diese Umsetzung verändert den Inhalt der angegebenen Dateien. Wenn es sich bei einer Eingabedatei um ein Archiv handelt, wird jede Komponente des Archivs umgesetzt. Das Archiv wird dann in der ursprünglichen Reihenfolge neu erzeugt. Eingabedateien, die sich nicht im COFF-Objektformat befinden, werden von `cof2elf` nicht verändert.

Die Optionen haben folgende Bedeutung:

`-i` Die Datei wird auf jeden Fall übersetzt. Fehler werden ignoriert. Normalerweise werden die Dateien nur verändert, wenn eine komplette Übersetzung durchgeführt wird. Nicht erkannte Daten, wie zum Beispiel unbekannte Relokationstypen, werden als Fehler behandelt, und die Übersetzung wird nicht durchgeführt.

`-q` Üblicherweise gibt `cof2elf` für jede Datei, die bearbeitet wird, eine Meldung aus, die anzeigt, ob eine Datei übersetzt oder ignoriert wurde. Die Option `-q` unterdrückt diese Meldungen.

`-Q`*arg* Wenn *arg* gleich `y` ist, werden zusätzliche Identifikationsinformationen bezüglich `cof2elf` in die Ausgabedateien geschrieben. Dies kann für die Verwaltung der einzelnen Dateien sinnvoll sein. Wenn *arg* gleich `n` ist, werden keine Informationen in die Dateien geschrieben. Dies ist das voreingestellte Verhalten.

`-s`*Verzeichnis*
Wie oben bereits erwähnt wurde, modifiziert `cof2elf` die Eingabedateien. Diese Option sichert die Eingabedateien im angegebenen *Verzeichnis*, welches vorhanden sein muß. Findet keine Modifikation der Dateien statt, sichert `cof2elf` die Dateien auch nicht.

`-V` die Versionsnummer des SCDE wird über die Standard-Fehlerausgabe ausgegeben.

SIEHE AUCH

`ld`(1), `elf`(3E), `a.out`(4), `ar`(4).

HINWEIS

Debuginformation wird z.T. gelöscht. Obwohl sich das Verhalten eines ausführbaren Programms dadurch nicht ändert, kann Symbolinformation verändert sein.

`cof2elf` übersetzt nur verschiebbare (relocatable) COFF-Dateien. Ausführbare (Binaries) oder statische gemeinsam benutzbare Bibliotheksdateien (shared libraries) werden nicht übersetzt. Dies geschieht aus dem Grund, daß das Betriebssystem ausführbare Dateien und statische, gemeinsam benutzbare Bibliotheken unterstützt, was eine Übersetzung überflüssig macht. Außerdem besitzen diese Dateien spezifische Adressen- und Ausrichtungsbeschränkungen, die vom Dateiformat vorgegeben sind. Die Übersetzung der Ausrichtungsbeschränkungen in ein anderes Dateiformat ist sehr problematisch.

Sofern möglich, sollten Programmierer den Quelltext neu kompilieren, um neue Objektdateien zu generieren. `cof2elf` sollte nur dann verwendet werden, wenn der Quelltext nicht verfügbar ist.

comb - SCCS-Deltas zusammenfassen

comb [-o] [-s] [-pSID] [-cListe] Datei...

comb (combine) erstellt eine Shell-Prozedur, mit der die angegebenen SCCS-*Dateien* umgeformt werden können. Dabei werden i.d.R. die umgeformten Dateien kleiner als die Originaldateien! Die Argumente können in beliebiger Reihenfolge angegeben werden. Jedoch gelten alle Optionen für alle angegebenen SCCS-Dateien. Wird ein Dateiverzeichnis angegeben, verhält sich comb so, als ob jede Datei im Dateiverzeichnis angegeben wurde. Hiervon ausgenommen sind jedoch Dateien, die keine SCCS-Dateien sind, d.h. bei denen der Dateiname ohne Pfadangabe nicht mit s. beginnt und unlesbare Dateien, die kommentarlos ignoriert werden. Bei Angabe von - wird von Standardeingabe gelesen; jede Zeile der Eingabe wird als Name einer zu verarbeitenden SCCS-Datei angenommen. Die erzeugte Shell-Prozedur wird auf Standardausgabe geschrieben.

Die Optionen sind nachstehend angeführt. Sie gelten unabhängig für jede angegebene Datei.

-o Für jedes get -e bewirkt dieses Argument den Zugriff auf die umgeformte Datei mit der Version des zu erstellenden Deltas. Sonst würde beim Vorgänger auf die umgeformte Datei zugegriffen. Die Größe der umgeformten SCCS-Datei kann mit der Option -o verringert werden. Sie kann auch die Gestalt des Deltabaums der Originaldatei verändern.

-s comb erzeugt eine Shell-Prozedur, bei deren Ausführung eine Meldung ausgegeben wird, die für jede Datei den Dateinamen, die Größe in Blökken nach der Verwendung von comb, die ursprüngliche Größe in Blöcken und den prozentualen Anteil der Änderung angibt. Dieser wird folgendermaßen berechnet:

100 * (*ursprüngliche Größe - neue Größe*) / *ursprüngliche Größe*

Diese Option sollten Sie vor der tatsächlichen Zusammenfassung von SCCS-Dateien verwenden, um den dadurch eingesparten Platz ermitteln zu können.

-p*SID* SID (SCCS-Identifikationszeichenkette) des ältesten Deltas, das bewahrt werden soll wird angegeben. Alle älteren Deltas werden aus der umgeformten Datei gelöscht.

-c*Liste* *Liste* von Deltas, die bewahrt werden sollen. Alle anderen Deltas werden gelöscht (siehe get(1) zur Syntax einer *Liste*).

Wenn keine Optionen angegeben sind, bewahrt comb nur Blattdeltas und die für die Bewahrung des Baums benötigte Mindestanzahl an Vorgängern auf.

DATEIEN

`s.COMB`	umgeformte SCCS-Datei
`comb?????`	Temporär-Datei

SIEHE AUCH

`admin`(1), `delta`(1), `get`(1), `prs`(1), `sccsfile`(4).
`sh`(1) in "SINIX V5.41 Kommandos"

HINWEIS

`comb` kann die Form des Deltabaums ändern. Es wird u.U. kein Platz gespart; tatsächlich ist es möglich, daß die neukonstruierte Datei größer als die Originaldatei wird!

convert - Format einer Archivdatei umformen

convert [-x] Eingabedatei Ausgabedatei

Das Kommando `convert` liest aus der Datei *Eingabedatei*, transformiert und erzeugt die Datei *Ausgabedatei*. *Eingabedatei* muß ein UNIX System V Release 1.0-Archiv sein, *Ausgabedatei* ist das äquivalente Archiv im UNIX System V Release 2.0 Format. Alle andere Arten von Eingabedateien werden durch `convert` unverändert von der Eingabedatei in die Ausgabedatei kopiert, begleitet von einer entsprechenden Warnmeldung.

Die Option `-x` wird zur Konvertierung von XENIX-Archivdateien benötigt (XENIX ist ein geschütztes Warenzeichen der Firma Microsoft). Diese Option konvertiert das Archiv, beläßt die einzelnen Elemente des Archivs jedoch unverändert.

Eingabedatei und *Ausgabedatei* dürfen nicht gleich sein.

DATEIEN

TMPDIR`/conv*` Zwischendateien

Üblicherweise ist *TMPDIR* gleich `/usr/tmp`, kann jedoch durch Verwendung der Umgebungsvariablen `TMPDIR` geändert werden (siehe `tempnam` in `tmpnam`(3S)).

SIEHE AUCH

`ar`(1), `tmpnam`(3S), `a.out`(4), `ar`(4).

cscope - C-Programme interaktiv untersuchen

cscope [Option...] Datei...

cscope ist ein interaktives, bildschirmorientiertes Programm, mit dem sich Benutzer einen C-Quelltext ansehen können.

Standardmäßig untersucht cscope C-, lex- und yacc-Quellprogramme im aktuellen Verzeichnis. cscope kann auch für Quelldateien, die in der Kommandozeile angegeben sind, aufgerufen werden. In jedem Fall durchsucht cscope die Standardverzeichnisse nach #include-Dateien, die es im aktuellen Verzeichnis nicht findet. cscope verwendet eine Querverweistabelle der Symbole, standardmäßig cscope.out, um Funktionen, Funktionsaufrufe, Makros, Variablen und Symbole des C-Präprozessors in den Dateien aufzufinden.

cscope baut die Querverweistabelle für die Symbole auf, wenn es das erste Mal mit den Quelldateien des untersuchten Programms benutzt wird. Bei späteren Aufrufen wird die Querverweistabelle nur dann neu aufgebaut, wenn eine der Quelldateien geändert wurde oder die Liste der Quelldateien unterschiedlich ist. In diesem Fall werden die Daten der unveränderten Dateien aus der alten Querverweistabelle kopiert, wodurch der Neuaufbau der Querverweistabelle wesentlich schneller als der erstmalige Aufbau wird.

Die folgenden Optionen können in beliebigen Kombinationen angegeben werden:

-b Es werden nur die Querverweise aufgebaut.

-C Beim Suchen wird Groß- und Kleinschreibung nicht berücksichtigt.

-c In der Querverweistabelle werden nur ASCII-Zeichen verwendet, d.h. die Daten werden nicht komprimiert.

-d Die Querverweistabelle wird nicht aktualisiert.

-e Der Kommandoprompt zwischen den Dateien wird unterdrückt.

-f *Verweisdatei*
Anstatt cscope.out wird *Verweisdatei* als Dateiname für die Querverweistabelle benutzt.

-I *Incverz* Vor der Suche in *INCDIR*, dem Verzeichnis für Include-Dateien (normalerweise /usr/include), soll in *Incverz* nach #include-Dateien gesucht werden, deren Name nicht mit / beginnt und die nicht auf der Kommandozeile oder in *Namensdatei* (s.u.) angegeben wurden. Die #include-Dateien können entweder in doppelten Hochkommata oder in spitzen Klammern angegeben werden. Das Verzeichnis *Incverz* wird zusätzlich zum aktuellen Verzeichnis und den Standardverzeichnissen durchsucht. Wird die Option -I mehrmals angegeben, werden die Verzeichnisse in der auf der Kommandozeile angegebenen Reihenfolge durchsucht.

`-i` *Namensdatei*
: Statt der in der Standarddatei (`cscope.files`) aufgelisteten Dateien werden alle Quelldateien untersucht, deren Namen in *Namensdatei* aufgelistet sind. Dateinamen sind durch Leerzeichen, Tabulatoren oder Zeilenendezeichen getrennt. Wird diese Option angegeben, ignoriert `cscope` alle auf der Kommandozeile angegebenen Dateien.

`-L`
: Die Datei wird einmal mit zeilenorientierter Ausgabe durchsucht, wenn diese Option zusammen mit *-Num Muster* verwendet wird.

`-l`
: Zeilenorientierte Schnittstelle (siehe Abschnitt "Zeilenorientierte Schnittstelle").

-Num Muster
: Zum Eingabefeld *Num* gehen und *Muster* finden. Es wird von 0 an gezählt.

`-P` *Pfad*
: erstellt eine Querverweisdatei, in der jedem relativen Dateinamen *Pfad* vorangestellt wird. Dadurch ersparen Sie sich einen Wechsel in das Verzeichnis, in dem die Querverweisdatei erstellt wurde. Diese Option ist nur zusammen mit der Option `-d` zulässig.

`-p` *n*
: Anstatt des Standards (1) werden die letzten *n* Pfadkomponenten angezeigt. Verwenden Sie 0, um gar keine Dateinamen anzuzeigen.

`-s` *Verz*
: sucht in *Verz* nach weiteren Quelldateien. Diese Option wird ignoriert, wenn auf der Kommandozeile Quelldateien angegeben werden.

`-T`
: Zum Vergleich mit C-Symbolen werden nur die ersten acht Zeichen verwendet. Ein regulärer Ausdruck, der andere Sonderzeichen als Punkte enthält, paßt auf kein Symbol, wenn er bei minimaler Länge weniger als acht Zeichen enthält.

`-U`
: Die Zeitstempel der Dateien sollen nicht überprüft werden. Es wird davon ausgegangen, daß die Dateien nicht verändert wurden.

`-u`
: Die Querverweisdatei wird auf jeden Fall aufgebaut. Es wird davon ausgegangen, daß alle Dateien verändert wurden.

`-V`
: Auf der ersten Zeile des Bildschirms wird die Versionsnummer von `cscope` ausgegeben.

Die Optionen `-I`, `-p` und `-T` können auch in der Datei `cscope.files` stehen.

Eingabe des ersten Suchbefehls

Nachdem die Querverweistabelle erzeugt wurde, zeigt cscope das folgende Menü an:

```
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```

Drücken Sie die Taste TAB so lange, bis das gewünschte Eingabefeld erreicht ist, geben Sie den gesuchten Text ein, und drücken Sie die Taste RETURN.

Eingabe weiterer Befehle

Wenn die Suche erfolgreich war, können Sie folgende Tastenkombinationen verwenden

```
1-9         Zeile mit eingegebener Nummer editieren
SPACE       die nächsten gefundenen Zeilen ausgeben
+           die nächsten gefundenen Zeilen ausgeben
^v          die nächsten gefundenen Zeilen ausgeben
-           die vorherigen gefundenen Zeilen ausgeben
^e          alle ausgegebenen Dateien der Reihe nach editieren
>           Zeilenliste an eine Datei anhängen
|           alle angezeigten Zeilen an ein Shell-Kommando übergeben
```

Mit folgenden Tastenkombinationen können Sie das cscope-Menü steuern und andere Befehle ausführen:

```
TAB         nächstes Eingabefeld
RETURN      nächstes Eingabefeld
^n          nächstes Eingabefeld
^p          vorheriges Eingabefeld
^y          zuletzt eingegebenen Text suchen
^b          nächstes Eingabefeld und Begriff suchen
^f          nächstes Eingabefeld und Begriff suchen
^c          bei der Suche wird Groß-/Kleinschreibung berücksichtigt
^r          Querverweistabelle neu generieren
!           interaktive Shell aufrufen (mit ^d kommen Sie zu cscope zurück)
^l          Bildschirm neu aufbauen
?           Hilfsinformationen über cscope-Kommandos ausgeben
^d          cscope verlassen
```

Wenn das erste Zeichen eines gesuchten Textes einer der angegebenen Tastenkombinationen entspricht, geben Sie vor diesem Zeichen ein \ ein.

Ersetzen von altem durch neuen Text

Nachdem der zu ändernde Text eingegeben wurde, fordert `cscope` den neuen Text an; anschließend werden die Zeilen mit dem alten Text angezeigt. Wählen Sie die zu ändernden Zeilen mit den folgenden Tasten aus:

```
1-9         eine Zeile auswählen
*           alle Zeilen auswählen
SPACE       nächste Zeilen ausgeben
+           nächste Zeilen ausgeben
-           vorherige Zeilen ausgeben
a           alle Zeilen auswählen
^d          ausgewählte Zeilen ändern und verlassen
ESCAPE      Änderungsmodus ohne Änderung der markierten Zeilen verlassen
```

Besondere Tasten

Wenn Ihr Terminal Pfeiltasten hat, die im `vi`(1) funktionieren, können Sie diese benutzen, um sich innerhalb der Eingabefelder zu bewegen. Der Aufwärtspfeil ist sinnvoll, um zum vorigen Eingabefeld zu gelangen, statt mehrfach die `TAB`-Taste zu betätigen. Wenn Sie die Tasten `CLEAR`, `NEXT` oder `PREV` zur Verfügung haben, arbeiten diese wie die Kommandos `^l`, `+` und `-`.

Zeilenorientierte Schnittstelle

Mit der Option `-l` können Sie `cscope` auch da verwenden, wo eine bildschirmorientierte Schnittstelle nicht sinnvoll wäre, z.B. von einem anderen bildschirmorientierten Programm aus.

`cscope` meldet sich mit `>>` und erwartet eine Feldnummer (von 0 an gezählt), unmittelbar gefolgt vom Suchmuster. `1main` findet z.B. die Definition der Funktion `main`.

Wenn Sie die Datei nur einmal durchsuchen lassen wollen, sollten Sie die Option `-L` und *-Num Muster* verwenden.

Für `-l` gibt `cscope` die Anzahl der Verweiszeilen aus:

```
cscope: 2 lines
```

Für jeden gefundenen Verweis gibt `cscope` eine Zeile aus, die aus dem Dateinamen, dem Funktionsnamen, der Zeilennummer und dem Zeileninhalt besteht, jeweils durch Leerzeichen getrennt, z.B.

```
main.c main 161 main(argc, argv)
```

Beachten Sie, daß der Editor, anders als bei der bildschirmorientierten Schnittstelle, nicht aufgerufen wird, um einen einzigen Verweis anzuzeigen.

`cscope` wird verlassen, wenn ein Dateiende-Zeichen entdeckt wird oder wenn das erste Zeichen einer Eingabezeile `^d` oder `q` ist.

Umgebungsvariablen

EDITOR	bevorzugter Editor
INCLUDEDIRS	durch Doppelpunkte getrennte Liste von Verzeichnissen, die nach #include-Dateien durchsucht werden sollen
HOME	HOME-Verzeichnis wird beim Start einer Sitzung automatisch gesetzt.
SHELL	bevorzugte Shell, voreingestellt ist sh(1).
SOURCEDIRS	durch Doppelpunkte getrennte Liste von Verzeichnissen, die nach zusätzlichen Quelldateien durchsucht werden sollen
TERM	Terminaltyp Es muß sich um ein bildschirmorientiertes Terminal handeln.
TERMINFO	vollständiger Pfadname des Terminalinformations-Verzeichnisses Wenn Ihr Terminal-Typ nicht in dem Standardverzeichnis terminfo steht, sehen Sie unter curses(3X) und terminfo(4) nach, um Ihre eigene Terminalbeschreibung zu erstellen.
TMPDIR	Verzeichnis für temporäre Dateien mit dem Standardwert /var/tmp
VIEWER	Programm zur Anzeige von Dateien (wie z.B. pg(1)) auswählen Diese Angabe hat Vorrang vor EDITOR.
VPATH	eine durch Doppelpunkte getrennte Liste von Verzeichnissen, von denen jedes dieselbe Verzeichnisstruktur hat. Wenn VPATH gesetzt ist, sucht cscope in den angegebenen Verzeichnissen nach Quelldateien; sonst wird nur im aktuellen Verzeichnis gesucht.

DATEIEN

cscope.files	Standarddatei, die die Optionen -I, -p und -T und die Liste der Quelldateien enthält; kann durch die Option -i anders gewählt werden.
cscope.out	Querverweistabelle, die in das HOME-Verzeichnis geschrieben wird, falls sie im aktuellen Verzeichnis nicht angelegt werden kann.
ncscope.out	temporäre Datei, die die neue Querverweistabelle enthält, bevor sie die alte Querverweistabelle ersetzt.
INCDIR	Standardverzeichnis für #include-Dateien (gewöhnlich /usr/include).

SIEHE AUCH

Kapitel "cscope" in "Leitfaden und Werkzeuge für die Programmierung mit C"
curses und terminfo in "Programmer's Guide: Character User Interface".

HINWEIS

`cscope` erkennt Funktionsdefinitionen der Form

fname blank (args) white arg_decs white {

fname	Funktionsname
blank	enthält beliebig viele Leerzeichen oder Tabulatoren, allerdings keine neue Zeile
args	Zeichenkette ohne " und Neue-Zeile-Zeichen
white	enthält beliebig viele Leerzeichen, Tabulatoren, Neue-Zeile-Zeichen
arg_decs	enthält beliebig viele Argumentdeklarationen, wobei Kommentar und Leerzeichen enthalten sein können.

Es ist nicht notwendig, daß eine Funktionsdeklaration am Anfang einer Zeile beginnt. Der Rückgabetyp kann dem Funktionsnamen vorausgehen; `cscope` erkennt die Deklaration weiterhin. Funktionsdeklarationen, die von dieser Form abweichen, werden von `cscope` nicht erkannt.

In der `function`-Spalte der Menüoption `Find functions called by this function` wird nur die erste Funktion angezeigt, die in dieser Zeile aufgerufen wird. Für die folgende Funktion

```
e()
{
        return (f() + g());
}
```

würde die Ausgabe demnach folgendermaßen aussehen:

```
Functions called by this function: e

File  Function        Line
a.c   f               3 return(f() + g());
```

Gelegentlich kann es vorkommen, daß eine Funktionsdefinition oder ein Funktionsaufruf aufgrund von geschweiften Klammern innerhalb von `#if`-Anweisungen nicht erkannt wird. Genauso kann die Verwendung einer Variablen fälschlicherweise als eine Definition gewertet werden.

Ein `typedef`-Name, der vor einer Präprozessor-Anweisung steht, wird fälschlicherweise als eine globale Definition gewertet, z.B.

```
LDFILE *
#if AR16WR
```

Präprozessor-Anweisungen können auch das Erkennen einer globalen Definition verhindern, z.B.

```
char flag
#ifdef ALLOCATE_STORAGE
        = -1
#endif
;
```

Eine Funktionsdeklaration innerhalb einer Funktion wird fälschlicherweise als Funktionsaufruf gewertet, z.B. wird folgende Funktionsdeklaration fälschlich als Aufruf von `g()` erkannt:

```
f()
{
        void g();
}
```

`cscope` erkennt C++-Klassen an ihrem Schlüsselwort. Es erkennt jedoch nicht, daß auch `struct` eine Klasse ist. Genausowenig erkennt es die Definitionen von 'inline member'-Funktionen innerhalb einer Struktur. Es erwartet außerdem kein Klassen-Schlüsselwort in einem `typedef` und so wertet es fälschlicherweise in der folgenden Definition `X` als Definition:

```
typedef class X * Y;
```

Es erkennt auch keine Operator-Funktionsdefinitionen:

```
Bool Feature::operator==(const Feature & other)
{
        ...
}
```

ctrace - Debugger für C-Programme

ctrace [Option...] [Datei]

Mit dem Kommando `ctrace` wird ein C-Programm für die Ablaufverfolgung zur Laufzeit vorbereitet. Die Auswirkung ist ähnlich wie bei der Ausführung einer Shell-Prozedur mit der Option `-x`. `ctrace` liest das C-Programm aus *Datei* oder von der Standardeingabe, wenn *Datei* nicht angegeben wurde. Jeder ausführbaren Anweisung werden weitere Anweisungen hinzugefügt, die zur Laufzeit den Text dieser ausführbaren Anweisung sowie den Wert aller von ihr veränderten oder referenzierten Variablen ausgeben. `ctrace` schreibt das modifizierte Programm auf die Standardausgabe. Die Ausgabe von `ctrace` muß in eine temporäre Datei geschrieben werden, weil das Kommando `cc`(1) die Verwendung von Pipes nicht zuläßt. Anschließend kann diese Datei dann übersetzt und ausgeführt werden.

Jede Anweisung im Programm wird während ihrer Ausführung auf dem Bildschirm aufgelistet. Anschließend werden Name und Wert der Variablen angezeigt, auf die in der Anweisung verwiesen wurde oder die verändert wurden, und danach folgt die ggf. von diesem Statement veranlaßte Ausgabe. Schleifen in der Ablaufverfolgung werden festgestellt, und die Ablaufverfolgung wird gestoppt, bis die Schleife verlassen oder eine andere Anweisungsreihenfolge innerhalb der Schleife ausgeführt wird. Bei jedem tausendsten Durchlaufen der Schleife wird eine Warnung ausgegeben, um dem Benutzer zu helfen, unendliche Schleifen zu entdecken. Die Ausgabe der Ablaufverfolgung wird auf die Standardausgabe geschrieben, die in eine Datei umgelenkt werden kann, damit der Benutzer diese mit Hilfe eines Editors oder anderer Kommandos untersuchen kann.

Normalerweise werden folgende Optionen eingesetzt:

`-f` *Funktionen* nur den Ablauf dieser *Funktionen* verfolgen

`-v` *Funktionen* außer diesen *Funktionen* den Ablauf aller Funktionen verfolgen

Die Standardformate für die Ausgabe von Variablen können ergänzt werden. Variablen vom Typ `long` und Zeiger-Variablen werden immer als ganze Zahlen mit Vorzeichen ausgegeben. Zeiger auf Zeichenfelder werden auch als Zeichenketten ausgegeben, wenn dies sinnvoll ist. `Char-`, `short-` und `int`-Variablen werden ebenfalls als ganze Zahlen mit Vorzeichen und gegebenenfalls auch als Zeichen ausgegeben. Variablen vom Typ `double` werden in wissenschaftlicher Notation als Gleitkommazahlen ausgegeben.

Mit den folgenden Optionen kann die Ausgabe von Variablen auch in zusätzlichen Formaten erfolgen:

`-o`	oktal
`-x`	hexadezimal
`-u`	vorzeichenlos
`-e`	Gleitkomma

Diese Optionen werden nur unter besonderen Umständen verwendet:

`-l` *n*
: statt der standardmäßig vorgegebenen 20 Anweisungen *n* aufeinanderfolgend ausgeführte Anweisungen für die spezielle Schleifenablaufverfolgung prüfen
Mit 0 erhält man die gesamte Ablaufverfolgung von Schleifen.

`-s`
: die redundante Ablaufverfolgung von einfachen Zuweisungen und Kopierfunktionen für Zeichenketten unterdrücken
Diese Option kann einen Fehler verdecken, der durch die Anwendung des Operators = anstelle des Operators == verursacht wird.

`-t` *n*
: anstelle der standardmäßig vorgegebenen 10 Variablen (Höchstzahl ist 20) *n* Variablen pro Anweisung verfolgen.

`-P`
: den C-Präprozessor vor der Ablaufverfolgung ausführen
Die Optionen `-D`, `-I` und `-U` von `cc`(1) sind ebenfalls verwendbar.

`-p` *Zeichenkette*
: die Ausgabefunktion für die Ablaufverfolgung umdefinieren (Standard: `printf`)
Zum Beispiel wird mit `fprintf(stderr)` die Ablaufverfolgung auf die Standard-Fehlerausgabe geschrieben.

`-r` *f*
: die Datei *f* anstelle des `runtime.c`-Laufzeit-Überwachungspaketes verwenden
Mit dieser Option läßt sich die gesamte Ausgabefunktion ändern, nicht nur der Name und die führenden Argumente (siehe `-p`).

`-V`
: auf Standard-Fehlerausgabe Informationen über die Version von `ctrace` ausgeben

`-Q` *Arg*
: Wenn *Arg* gleich `y` ist, werden zu den Ausgabedateien Informationen zur Identifikation von `ctrace` hinzugefügt. Das kann zur Softwareverwaltung sinnvoll sein. Bei der Angabe von `n` für *Arg* wird explizit angegeben, daß solche Informationen nicht benötigt werden. Das ist das Standardverhalten.

BEISPIEL

Die Datei `lc.c` enthält das folgende fehlerhafte C-Programm:

```
1 #include <stdio.h>
2 main()      /* Zeilen in der Eingabe zählen */
3 {
4     int c, nl;
5
6     nl = 0;
7     while ((c = getchar()) != EOF)
8             if (c = '\n')
9                     ++nl;
10    printf("%d\n", nl);
11 }
```

Geben Sie folgende Kommandos und Testdaten ein:

```
cc lc.c
a.out
1
(^d)
```

Das Programm wird übersetzt und ausgeführt. Die Ausgabe des Programms, die Zahl `2`, ist falsch, weil in den Testdaten nur eine Zeile vorhanden ist. Dieser Programmierfehler ist weit verbreitet, aber oft schwer zu finden. Wird `ctrace` mit diesen Kommandos aufgerufen:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

dann ist die Ausgabe:

```
2 main()
6     nl = 0;
      /* nl == 0 */
7     while ((c = getchar()) != EOF)
```

Das Programm wartet nun auf die Eingabe. Werden dieselben Testdaten wie vorher eingegeben, ist die Ausgabe wie folgt:

```
      /* c == 49 or '1' */
8             if (c = '\n')
              /* c == 10 or '\n' */
9                     ++nl;
                      /* nl == 1 */
7     while ((c = getchar()) != EOF)
      /* c == 10 or '\n' */
8             if (c = '\n')
              /* c == 10 or '\n' */
9                     ++nl;
                      /* nl == 2 */
7     while ((c = getchar()) != EOF)
```

Wird nun ein Dateiende-Zeichen (ctrl-d) eingegeben, ist die endgültige Ausgabe:

```
      /* c == -1 */
10    printf("%d\n", nl);
      /* nl == 2 */2
       return
```

Beachten Sie die Ausgabe am Ende der Kontrollausgabe für die Variable `nl` nach Zeile 10. Beachten Sie weiterhin den von
`ctrace` eingefügten Kommentar `return`. Damit wird die implizite Rückkehr an der schließenden Klammer in der Funktion angezeigt.

Die Ausgabe der Ablaufverfolgung zeigt an, daß der Variablen `c` in Zeile 7 der Wert `1` zugewiesen wird, sie in Zeile 8 jedoch den Wert `\n` hat. Nachdem der Benutzer auf diese `if`-Anweisung hingewiesen worden ist, wird er erkennen, daß der Zuweisungsoperator = statt des Gleichheitsoperators == verwendet wurde. Dieser Fehler kann leicht

beim Lesen des Codes übersehen werden.

Steuerung der Ablaufverfolgung zur Ausführungszeit

Die Standardfunktion von `ctrace` besteht in der Ablaufverfolgung des gesamten Programmes, sofern nicht die Optionen `-f` oder `-v` zur Ablaufverfolgung bestimmter Funktionen eingesetzt werden. Der Benutzer erhält dadurch jedoch keine Möglichkeit den Ablauf einzelner Anweisungen zu verfolgen; auch läßt sich die Ablaufverfolgung nicht während der Programmausführung ein- oder abschalten.

Jedoch kann man durch Einfügen der Funktionsaufrufe `ctroff()` und `ctron()` in das Programm die Ablaufverfolgung während der Ausführungszeit ein- bzw. abschalten. Auf diese Weise kann man beliebig komplexe Kriterien mit `if`-Anweisungen für die Steuerung der Ablaufverfolgung zur Ausführungszeit aktivieren und diesen Code sogar bedingt einfügen, weil `ctrace` die Präprozessor-Variable `CTRACE` definiert.

Die Ablaufverfolgung läßt sich auch durch Setzen der statischen Variablen `tr_ct_` auf `0` bzw. `1` ein- und abschalten. Dies ist nützlich, wenn ein Debugger eingesetzt wird, der diese Funktionen nicht direkt aufrufen kann.

DATEIEN

`/usr/lib/ctrace/runtime.c`	Laufzeit-Überwachungspaket

SIEHE AUCH

`sdb`(1), `ctype`(3C), `fclose`(3S), `printf`(3S), `string`(3C)
`bfs`(1), `tail`(1) in "SINIX V5.41 Kommandos".

ENDESTATUS

Dieser Abschnitt enthält Diagnosemeldungen von `ctrace` und `cc`(1), da das Protokoll der Ablaufverfolgung oft `cc`-Warnungen erhält. In sehr seltenen Fällen kann man auch `cc`-Fehlermeldungen erhalten, die sich jedoch alle vermeiden lassen.

ctrace-Diagnose

`warning: some variables are not traced in this statement`

Nur 10 Variablen werden in einer Anweisung verfolgt, um den C-Übersetzer-Fehler `'out of tree space; simplify expression'` zu vermeiden. Diese Zahl wird mit der Option `-t` erhöht.

`warning: statement too long to trace`

Diese Anweisung ist über 400 Zeichen lang. Achten Sie darauf, daß zum Einrücken

des Codes Tabulatoren und nicht Leerstellen verwendet werden.

`cannot handle preprocessor code, use -P option`

Diese Meldung wird gewöhnlich von den Präprozessor-Anweisungen `#ifdef/#endif` in der Mitte einer C-Anweisung oder von einem Semikolon am Ende einer Präprozessor-Anweisung `#define` verursacht.

`'if ... else if' - sequence too long`

Teilen Sie die Sequenz durch Entfernen eines `else` aus der Mitte auf.

`possible syntax error, try -P option`

Bearbeiten Sie zuerst die `ctrace`-Eingabe mit der Option `-P` und geeigneten Präprozessor-Optionen `-D`, `-I` und `-U`.

HINWEIS

Wird eine Funktion mit demselben Namen wie eine Systemfunktion definiert, kann das bei einer Änderung der Anzahl der Argumente einen Syntaxfehler hervorrufen. Benutzen Sie einfach einen anderen Namen.

`ctrace` nimmt an, daß `BADMAG` ein Präprozessor-Makro ist und daß `EOF` und `NULL` mit `#define` definierte Konstanten sind. Wenn eine von ihnen als Variable erklärt wird, z.B. "`int EOF;`", entsteht dadurch ein Syntaxfehler.

Zeiger werden immer wie Zeiger auf Zeichenketten behandelt.

`ctrace` kennt keine Komponenten eines Verbunds, wie z.B. Strukturen, Unions und Felder. Wenn eine Zuweisung an den gesamten Verbund erfolgt, werden nicht alle einzelnen Komponenten ausgegeben. `ctrace` kann sich in diesem Fall für die Ausgabe der Adresse des Verbundes entscheiden oder das falsche Format verwenden (z.B. `3.149050e-311` für eine Struktur mit zwei ganzzahligen Teilen), wenn der Wert eines Verbunds ausgegeben wird.

Die Eliminierung der Ausgabe einer Schleifenverfolgung erfolgt separat für jede Datei eines Mehrdateien-Programms. Dies kann dazu führen, daß Funktionen von einer Schleife aufgerufen werden, die noch verfolgt wird, oder daß die Ablaufverfolgung einer Funktion in einer Datei unterdrückt wird, bis eine andere Funktion in derselben Datei aufgerufen wird.

cxref - Querverweistabelle für C-Programm erstellen

cxref [Option...] Datei...

Das Kommando cxref (C cross reference) analysiert eine Anzahl von C-Dateien und baut eine Querverweistabelle auf. cxref verwendet eine besondere Version von cc, um Information, die in #define-Anweisungen deklariert ist, in die Symboltabelle aufzunehmen. Es erstellt auf der Standardausgabe eine Liste aller Symbole (auto, static und global) jeder Einzeldatei oder auch kombiniert, wenn die Option -c verwendet wird. Die Tabelle beinhaltet vier Felder: NAME, FILE, FUNCTION und LINE. Das Feld LINE enthält die Zeilennummern, in denen das Symbol NAME vorkommt und die Art, wie es in der jeweiligen Zeile referenziert wird. Folgende drei Zugriffsarten werden dabei markiert:

Zuordnung mit	=
Deklaration mit	-
Definition mit	*

Andere Zugriffsarten werden nicht markiert.

cxref interpretiert die Optionen -D, -I, -U in derselben Art und Weise wie cc. Zusätzlich interpretiert cxref die folgenden Optionen:

-c	die Quelldateien in einem Protokoll kombinieren Ohne Angabe von -c erstellt cxref ein separates Protokoll für jede Datei auf der Kommandozeile.
-d	schaltet die Ausgabe der Deklarationen aus; erleichtert das Lesen des Protokolls
-l	gibt keine lokalen Variablen aus; gibt nur globale und den Gültigkeitsbereich der Datei betreffende Statistiken aus
-o *Datei*	gibt direkt in *Datei* aus
-s	arbeitet kommentarlos; gibt keine Eingabedateinamen aus
-t	formatiert Liste für Breite von 80 Spalten
-w *num*	die Ausgabe wird nicht breiter als *num* (dezimal) Spalten formatiert Standardwert ist 80, wenn *num* nicht angegeben wird oder kleiner ist als 51.
-C	durchläuft nur den ersten Durchgang von cxref, erstellt eine .cx-Datei, welche später zu cxref weitergeleitet werden kann. -C ähnelt der -c-Option von cc oder lint.
-F	gibt die vollständigen Pfade der angegebenen Dateinamen aus
-L *Spalten*	modifiziert die Anzahl der Spalten im Feld LINE Geben Sie keine Anzahl ein, so gibt cxref fünf Spalten vor.

`-V` gibt Versionsinformationen auf die Standard-Fehlerausgabe aus.

`-W`*name, datei, funktion, zeile*
ändert die Breitenvorgabe für wenigstens ein Feld

Die vorgegebenen Breiten sind:

```
Datenfeld     Zeichen
NAME            15
FILE            13
FUNCTION        15
LINE            20 (4 pro Spalte)
```

DATEIEN

TMPDIR`/tcx.*`	temporäre Dateien
TMPDIR`/cx.*`	vorläufige Dateien
LIBDIR`/xref`	zugegriffen durch `cxref`
LIBDIR	gewöhnlich `/usr/ccs/lib`
TMPDIR	gewöhnlich `/var/tmp`, kann durch Setzen der Umgebungsvariablen `TMPDIR` umdefiniert werden (siehe `tempnam` in `tmpnam`(3S))

BEISPIEL

```
a.c

1       main()
2       {
3               int i;
4               extern char c;
5
6               i=65;
7               c=(char)i;
8       }
```

Resultate der Querverweistabelle:

```
NAME  FILE          FUNCTION   LINE
c     a.c           ——         4-      7=
i     a.c           main       3*      6=  7
main  a.c           ——         2*
u3b2  predefined    ——         0*
unix  predefined    ——         0*
```

SIEHE AUCH

`cc`(1), `lint`(1).

ENDESTATUS

Fehlermeldungen bedeuten normalerweise, daß diese Datei nicht übersetzt werden kann.

dbx - Interaktive symbolische Testhilfe

dbx [Option...] [Obj-Name [Sp-Name]]

dbx ist eine interaktive symbolische Testhilfe für Programme in C++, C und FORTRAN77. Mit dbx können Sie ein Programm während der Ausführung beobachten und gegebenenfalls in das Programm eingreifen.

Mit dbx können Sie

- die Stelle erfahren, an der ein Programm abgebrochen wurde
- Objektdateien und Dateien mit Speicherabzügen analysieren
- C-, C++- und FORTRAN77-Programme kontrolliert ablaufen lassen
- den Programmablauf protokollieren
- Variablenveränderungen protokollieren
- die Programmausführung anhalten, z.B. wenn sich der Wert einer Variablen ändert
- Werte in verschiedenen Formaten ausgeben
- Objektdateien auf Maschinenebene bearbeiten

dbx ist ausführlich im Handbuch "DBX (SINIX V5.4x)" beschrieben.

dbx bearbeitet die angegeben Datei *Obj-Name*, die Objekt-Format haben muß.
Wenn *Obj-Name* nicht existiert oder für dbx nicht zugreifbar ist, gibt dbx eine Fehlermeldung aus und bricht ab.
Wird kein *Obj-Name* angegeben, erfragt dbx nach dem Aufruf als erstes die Objektdatei.

Der zugehörige Speicherauszug steht in der Datei *Sp-Name*. Wenn *Sp-Name* nicht existiert oder keinen Speicherauszug enthält, gibt dbx eine Fehlermeldung aus und bricht ab.
Wird kein *Sp-Name* angegeben, erwartet dbx den Speicherauszug in der Datei core im aktuellen Dateiverzeichnis.

Die folgenden Optionen können beim Aufruf von dbx angegeben werden:

`-c` *d_name* dbx führt nach dem Aufruf die dbx-Kommandos in der Datei *d_name* aus. Die benutzereigene Standard-Initialisierungsdatei .dbxinit wird dabei nicht abgearbeitet. Wenn die Datei *d_name* nicht existiert, wird die Option von dbx ohne Meldung ignoriert. Wenn die Datei *d_name* keine korrekten dbx-Kommandos enthält, gibt dbx die entsprechende Zeile und eine Fehlermeldung aus. Ein Kommando quit in der Datei *d_name* hat keine Wirkung.

`-i` `dbx` erwartet seine Standardeingabe vom Terminal. Wenn die Standardeingabe ein Terminal ist, hat die Option keine Wirkung.
Wenn die Standardeingabe umgelenkt wurde, bewirkt die Option folgendes:

1. Die Eingabe wird behandelt wie Eingabe von Tastatur, d.h.
   - Eingabezeilen dürfen keine Kommentare enthalten. Wenn eine Zeile Kommentare enthält, gibt `dbx` eine Fehlermeldung aus und führt die Kommandos in dieser Zeile nicht aus.
   - das Dateiende hat keine Bedeutung.
2. Wenn Sie Ihre Sitzung mittels `record` aufzeichnen, werden die aus der Kommandodatei ausgeführten Kommandos auch aufgezeichnet.
3. `dbx` gibt wie im interaktiven Modus die Eingabeaufforderung aus.

Wenn Sie die Option `-i` setzen, müssen Sie dafür sorgen, daß die Eingabe für `dbx` tatsächlich einer Sitzung am Bildschirm entspricht. Das heißt auch, daß beispielsweise Fortsetzungszeilen in der Kommandodatei nicht zulässig sind. Insbesondere muß ein `quit`-Kommando in der Kommandodatei `dbx` beenden, wenn die Standardeingabe umgelenkt ist.

`-I` *dv* *dv* kann ein relativer oder absoluter Pfadname sein.
Das Dateiverzeichnis *dv* wird in die Liste der Dateiverzeichnisse aufgenommen, die `dbx` standardmäßig nach Quelldateien durchsucht. Sie können die Option `-I` wiederholen, um mehrere Dateiverzeichnisse anzugeben. Geben Sie kein Verzeichnis an, sucht `dbx` im aktuellen Dateiverzeichnis und im Dateiverzeichnis der Objektdatei. Das Dateiverzeichnis *dv* wird nur bei der Suche nach Quelldateien beachtet und nicht, wenn `dbx` beispielsweise `dbx`-Kommandodateien sucht.

`-q` Die `dbx`-Begrüßungszeile mit der Versionsnummer und dem Datum wird nicht ausgegeben. Diese Option ist nur wirksam, wenn sie als erste angegeben wird.

`-r` Die Objektdatei wird sofort ausgeführt.
Eine Umlenkung der Standardeingabe und Argumente im `dbx`-Aufruf nach *Obj-Name* gelten für das Programm *Obj-Name* und nicht für `dbx`:

`dbx -r` *Obj-Name* [*arg1*, ...] < *eingabedatei*

Die Standardeingabe für *Obj-Name* wird umgelenkt. `dbx` liest aus `/dev/tty`, d.h. Sie geben Ihre `dbx`-Eingaben über Tastatur ein: Die Datei `/dev/tty` ist eine Pseudodatei für das Terminal, an dem sie `login` eingegeben haben. Wenn die Ausführung ohne Unterbrechung erfolgreich endet, wird `dbx` beendet. Andernfalls gibt `dbx` zuerst den Grund für den Programmabbruch aus und wird dann initialisiert. Schalter, die Sie für `dbx` gesetzt haben, werden erst jetzt berücksichtigt.

`-u` Großschreibung bei FORTRAN-Namen bleibt erhalten.

HINWEIS

Die Optionen können beliebig kombiniert werden.
Die Option `-q` muß, falls vorhanden, als erste angegeben werden. Ansonsten ist die Reihenfolge der Optionen beliebig.
Ein `use`-Kommando in einer Initialisierungsdatei überschreibt die Dateiverzeichnis-Angaben bei der Option `-I`.

SIEHE AUCH

ausführliche Beschreibung im Handbuch "DBX (SINIX V5.4x)".

delta - Änderung an einer SCCS-Datei vornehmen

delta [-rSID] [-s] [-n] [-gListe] [-m[KMR-Liste]] [-y[Kommentar]] [-p] Datei...

`delta` dient dazu, an einer durch `get -e` erzeugten Datei (g-Datei oder generierte Datei) vorgenommene Änderungen in die entsprechende SCCS-Datei zu übernehmen, also eine neue Version zu erstellen.

Wenn ein Dateiverzeichnis angegeben wird, verarbeitet `delta` alle Dateien des Verzeichnisses, mit Ausnahme von Dateien, die keine SCCS-Dateien sind (Dateiname ohne Pfadangabe beginnt nicht mit `s.`) und unlesbaren Dateien. Diese werden kommentarlos ignoriert. Bei Angabe von - wird von Standardeingabe gelesen (siehe Abschnitt HINWEIS); jede Zeile der Standardeingabe wird als Name einer zu verarbeitenden SCCS-Datei betrachtet.

Abhängig von der Angabe bestimmter Optionen und Flags, die in der SCCS-Datei vorhanden sein können, kann `delta` Prompts auf der Standardausgabe ausgeben (siehe Optionen `-m` und `-y`).

Optionsparameter gelten unabhängig für jede angegebene Datei.

`-r` *SID* — *SID* gibt eindeutig an, welches Delta für die SCCS-Datei erzeugt werden soll. Die Verwendung dieser Option ist nur dann nötig, wenn zwei oder mehrere offenstehende `get`-Aufrufe zum Editieren (`get -e`) in derselben SCCS-Datei von derselben Person (Login-Name) vorgenommen wurden. Die mit der Option `-r` angegebene SID-Nummer kann entweder die in der `get`-Kommandozeile angegebene SID-Nummer oder die vom Kommando `get` gemeldete, auszuführende SID-Nummer sein (siehe `get`(1)). Eine Fehlermeldung erscheint, wenn die angegebene SID-Nummer nicht eindeutig ist oder nicht angegeben wurde.

`-s` — Unterdrückt die Ausgabe der für Delta erstellten SID-Nummer auf der Standardausgabe. Die Anzahl der in der SCCS-Datei eingefügten, gelöschten und unveränderten Zeilen wird ebenfalls nicht ausgegeben.

`-n` — Die editierte g-Datei soll erhalten bleiben. Sie wird normalerweise nach Beendigung von Delta gelöscht.

`-g`*Liste* — *Liste* von Deltas, die ignoriert werden sollen, wenn auf der von diesem Delta erstellten Änderungsebene (SID) auf die Datei zugegriffen werden soll (zur Definition von *Liste* siehe `get`(1)).

`-m`[*MR-Liste*]
Wenn das Flag `v` in der SCCS-Datei gesetzt ist (siehe `admin`(1)), muß eine Änderungsanforderungs-Nummer (MR) als Grund für die Erstellung eines neuen Deltas angegeben werden. Wenn `-m` nicht verwendet wird und die Standardausgabe ein Bildschirm ist, wird der Prompt `MRs?` vor dem Lesen

der Standardeingabe auf der Standardausgabe ausgegeben. Ist die Standardeingabe kein Bildschirm, wird kein Prompt ausgegeben. Der Prompt `MRs?` steht immer vor dem Prompt `comments?` (siehe `-y`). In einer Auflistung werden MRs durch Leerzeichen und/oder Tabulatoren voneinander getrennt. Die MR-Liste wird mit einem nicht entwerteten Neue-Zeile-Zeichen beendet.

Es ist zu beachten, daß das Argument das Flag `v` als Name eines Programms oder einer Shell-Prozedur angesehen wird, das die Richtigkeit der MR-Nummern validiert. Wenn das MR-Validierungsprogramm einen Endestatus liefert, der ungleich Null ist, wird `delta` beendet. Es wird angenommen, daß nicht alle MRs gültig waren.

`-y[`*Kommentar*`]`

Kommentar ist ein beliebiger Text, der den Grund für die Erstellung des Deltas beschreibt. Eine Nullzeichenreihe ist ein gültiger *Kommentar*. Wenn `-y` nicht angegeben ist und die Standardeingabe ein Bildschirm ist, wird der Prompt `comments?` vor dem Lesen der Standardeingabe auf der Standardausgabe ausgegeben; ist die Standardausgabe kein Bildschirm, wird kein Prompt ausgegeben. Der Kommentartext wird mit einem nicht entwerteten Neue-Zeile-Zeichen beendet.

`-p` Bevor und nachdem das Delta erstellt wurde, werden die Unterschiede in der SCCS-Datei in einem Format `diff`(1) auf der Standardausgabe ausgegeben.

DATEIEN

g-Datei	war vor Ausführung von `delta` vorhanden; wird nach Ausführung von `delta` entfernt
p-Datei	war vor Ausführung von `delta` vorhanden; kann nach Ausführung von `delta` vorhanden sein
q-Datei	wird während der Ausführung von `delta` erstellt; wird nach Ausführung von `delta` gelöscht
x-Datei	wird während der Ausführung von `delta` erstellt; wird nach Ausführung von `delta` in eine SCCS-Datei umbenannt
z-Datei	wird während der Ausführung von `delta` erstellt; wird während der Ausführung von `delta` gelöscht
d-datei	wird während der Ausführung von `delta` erstellt; wird nach der Ausführung von `delta` gelöscht
`bdiff`	Programm zur Ermittlung der Differenzen zwischen der 'erhaltenen' Datei und der g-Datei

SIEHE AUCH

`admin`(1), `cdc`(1), `get`(1), `prs`(1), `rmdel`(1), `sccsfile`(4).
`bdiff`(1), `help`(1) in "SINIX V5.41 Kommandos".
Kapitel "SCCS" in "Leitfaden und Werkzeuge für die Programmierung mit C".

HINWEIS

Ein `get` von vielen SCCS-Dateien mit nachfolgendem `delta` bei diesen Dateien ist zu vermeiden, wenn der `get`-Aufruf eine große Menge Daten erzeugt. Statt dessen sind mehrere Aufruffolgen von `get/delta` zu verwenden.

Wenn die Standardeingabe mit - in der `delta`-Kommandozeile angegeben ist, müssen die Optionen `-m` und `-y` ebenfalls vorhanden sein. Das Auslassen dieser Optionen verursacht einen Fehler.

Kommentare sind auf Textfolgen von höchstens 1024 Zeichen begrenzt. Gibt es in einer Zeile mehr als 1000 Zeichen, treten undefinierte Ergebnisse auf.

dump - Teile einer Objektdatei ausgeben

dump [Option...] Datei...

Das Kommando dump gibt ausgewählte Teile von jeder angegebenen Objektdatei aus. Es ist für Objektdateien und Archive von Objektdateien verwendbar. Es verarbeitet jedes Dateiargument entsprechend einer oder mehrerer der nachstehenden Optionen:

-a die Archivköpfe aller Archiv-Elemente ausgeben

-C die Namen in einer C++ Symboltabelle ausgeben

-c die Zeichenketten-Tabelle(n) ausgeben

-D Debug-Informationen ausgeben

-f jeden Dateikopf ausgeben

-g die globalen Symbole in der Symboltabelle eines Archivs ausgeben

-h die Abschnittsköpfe ausgeben

-L soweit verfügbar, die Informationen zum dynamischen Binden und zu gemeinsam benutzten Bibliotheken ausgeben

-l Zeilennummer-Daten ausgeben

-o jeden Vorspann für Programmausführung ausgeben

-r Relokationsdaten ausgeben

-s den Inhalt der Sktionen hexadezimal ausgeben

-T *Index*[,*Index2*]
nur den Eintrag der Symboltabelle ausgeben, auf den *Index* verweist, oder einen Bereich von Einträgen, die zwischen *Index* und *Index2* liegen

-t Symboltabellen-Einträge ausgeben

-u Beim Lesen einer COFF-Objektdatei übersetzt dump die Datei intern nach ELF. Diese Übersetzung beeinflußt nicht den Inhalt der Datei. -u steuert den Umfang der Umsetzung von COFF-Werten nach ELF. Ohne diese Option werden die COFF-Werte soweit wie möglich übernommen, es werden also die aktuellen Byte-Werte in der Datei angezeigt. Wenn -u verwendet wird, aktualisiert dump die Werte, vervollständigt die interne Übersetzung und liefert so eine konsistente ELF-Darstellung des Inhalts. Auch wenn die Bytes, die bei dieser Option angezeigt werden, nicht unbedingt den Werten in der Datei selbst entsprechen, so zeigen sie doch, wie die Datei aussehen würde, wenn sie nach ELF umgewandelt worden wäre (siehe cof2elf(1) für weitere Informationen).

-V Ausgabe der Version von dump

Die folgenden Modifikatoren werden in Verbindung mit den obigen Optionen zur Änderung ihrer jeweiligen Möglichkeiten eingesetzt.

`-d` *Nummer*`[,`*Nummer2*`]`
: Sektion *Nummer* ausgeben, oder Bereich, der bei Sektion *Nummer* beginnt und bei Sektion *Nummer2* endet
Dieser Modifikator kann zusammen mit `-h`, `-s` und `-r` verwendet werden. Wenn `-d` mit `-h` oder `-s` verwendet wird, wird das Argument als die Nummer einer Sektion oder als Sektionsbereich angesehen. Wenn `-d` mit `-r` verwendet wird, wird das Argument als die Nummer der gewünschten Sektion oder als Bereich der Sektionen, angewandt auf die Relokation, angesehen.
Beispiel:
Sie wollen alle Relokationseinträge in der `.text`-Sektion ausgeben. Geben Sie dazu die Nummer der Sektion mit dem Argument `-d` an. Wenn `.text` die zweite Sektion in der Datei ist, so gibt `dump -r -d 2` alle diesbezüglichen Einträge aus. Zur Ausgabe einer speziellen Relokationssektion verwenden Sie `dump -s -n` *Name* für die nicht interpretierte Ausgabe der Daten oder `dump -sv -n` *Name* für die interpretierte Ausgabe.

`-n` *Name*
: Information zum Objekt *Name* ausgeben
Dieser Modifikator gilt für `-h`, `-s`, `-r` und `-t`. Wenn `-n` mit `-h` oder `-s` verwendet wird, wird das Argument als Name einer Sektion behandelt. Wenn `-n` mit `-t` oder `-r` verwendet wird, wird das Argument als Name eines Symbols behandelt. Beispielsweise gibt `dump -t -n .text` den Eintrag in der Symboltabelle zu dem Symbol mit dem Namen `.text` aus, während `dump -h -n .text` die Vorspann-Information für die Sektion `.text` ausgibt.

`-p`
: Ausgabe des Vorspanns unterdrücken

`-v`
: Informationen in einer symbolischen Notation anstelle der numerischen Werte ausgeben
Die Verwendung dieses Modifikators ist möglich zusammen mit
`-a` (Datum, Benutzerkennung, Gruppenkennung),
`-f` (Klasse, Daten, Typ, Rechner, Version, Flags),
`-h` (Typ, Flags),
`-o` (Typ, Flags),
`-r` (Name, Flags),
`-s` (Interpretation der Sektionsinhalte wo immer möglich),
`-t` (Typ, Bindung) und
`-L` (Wert).
Wird `-v` zusammen mit `-s` verwendet, werden alle Sektionen interpretiert, bei denen dies möglich ist, wie die Zeichenketten-Tabelle oder die Symboltabelle.

Beispielsweise erzeugt

```
dump -sv -n .symtab Dateien
```

dieselbe formatierte Ausgabe wie

```
dump -tv Dateien.
```

Durch

```
dump -s -n .symtab Dateien
```

werden jedoch die reinen Daten in hexadezimal ausgegeben. Ohne die Verwendung zusätzlicher Modifikatoren gibt `dump -sv` *Dateien* alle Sektionen in den angegebenen Dateien, bei denen dies möglich ist, in interpretierter Form aus. Der Rest, wie zum Beispiel `.text` oder `.data` wird als reine Daten ausgegeben.

Das Kommando `dump` versucht, die von ihm ausgegebenen Daten in einer sinnvollen Weise zu formatieren und bestimmte Daten nach Bedarf als Zeichen, in Hexadezimal-, Oktal- oder Dezimalform darzustellen.

SIEHE AUCH

`a.out`(4), `ar`(4).

get - auf eine Version einer SCCS-Datei zugreifen

get [Option...] Datei...

`get` erstellt von jeder angegebenen SCCS-Datei eine ASCII-Textdatei gemäß der durch die Optionsargumente spezifizierten Angaben. Die Optionen können in beliebiger Reihenfolge vorgegeben werden, wobei sie für alle angegebenen SCCS-Dateien gültig sind. Wird ein Dateiverzeichnis angegeben, verhält sich `get` so, als ob jede Datei in dem Dateiverzeichnis angegeben wurde. Hiervon ausgenommen sind die Dateien, die keine SCCS-Dateien sind (Dateiname ohne Pfadangabe beginnt nicht mit `s.`) und unlesbare Dateien, die kommentarlos ignoriert werden. Bei Angabe von - wird von Standardeingabe gelesen; jede Zeile der Standardeingabe wird als Name einer zu verarbeitenden SCCS-Datei angesehen.

Der erstellte Text wird normalerweise in eine Datei geschrieben, die als g-Datei bezeichnet wird und deren Name einfach durch Entfernen des vorgestellten `s.` vom Namen der SCCS-Datei abgeleitet wird (siehe Abschnitt DATEIEN).

Jede Option wird nachstehend so erläutert, als ob nur eine SCCS-Datei verarbeitet werden soll. Die Auswirkungen einer Option treffen jedoch auf jede angegebene Datei zu.

`-r`*SID* SCCS-Identifikator (SID) der Version (Delta) einer bereitzustellenden SCCS-Datei
Die Tabelle auf Seite 77 zeigt, welche Version einer SCCS-Datei als Funktion der angegebenen SID bereitgestellt wird.

`-c`*cutoff* Zeitpunkt (Datum/Uhrzeit), von dem ab Änderungen nicht berücksichtigt werden, in folgendem Format:

```
JJ[MM[TT[HH[MM[SS]]]]]
```

In der erstellten ASCII-Textdatei sind also keine Änderungen (Deltas) der SCCS-Datei enthalten, die nach dem vorgegebenen Wert für *cutoff* erstellt wurden. In `cutoff` nicht angegebene Teile werden standardmäßig auf die jeweils möglichen Höchstwerte eingestellt. Das heißt: `-c7502` ist gleichwertig mit `-c750228235959`. Die zweistelligen Teile von *cutoff* Datum/Uhrzeit können durch eine beliebige Anzahl nichtnumerischer Zeichen getrennt sein. Aufgrund dieses Merkmals kann man ein *cutoff*-Datum in folgendem Format angeben: `-c"77/2/2 9:22:25"`.

`-i`*Liste* ist eine *Liste* von Deltas, die bei der Erstellung der generierten Datei eingeschlossen werden. *Liste* hat folgende Syntax:

<Liste> ::= <Bereich> | <Liste> , <Bereich>
<Bereich> ::= SID | SID - SID

SID, die SCCS-Identifikation eines Deltas, kann jedes der in Spalte 'angegebene SID' in der Tabelle angeführten Formate haben.

`-x`*Liste* ist eine *Liste* von Deltas, die bei der Erstellung der generierten Datei ausgeschlossen werden sollen. Das Format von *Liste* ist wie bei `-i`.

`-e` zeigt an, daß `get` zum Editieren oder Ausführen einer Änderung in der SCCS-Datei mit anschließendem `delta`(1) dient. Die in einem `get` für eine bestimmte Version (SID) der SCCS-Datei verwendete Option `-e` verhindert weitere `get`-Aufrufe zum Editieren derselben SID, bis `delta` ausgeführt wird oder die Option `j` (mehrfach Editieren) in der SCCS-Datei gesetzt wird (siehe `admin`(1)). Die gleichzeitige Benutzung von `get -e` für verschiedene SIDs ist stets zulässig.

Wenn die von `get` mit einer Option `-e` generierte g-Datei während des Editierens unbeabsichtigt beschädigt wird, kann sie durch Neuausführen des Kommandos `get` mit der Option `-k` anstelle der Option `-e` wiederhergestellt werden.

Bei Verwendung der Option `-e` wird der Schutz der SCCS-Datei geprüft, der durch den in der SCCS-Datei gespeicherten größten und kleinsten ganzzahligen Wert und die Liste berechtigter Benutzer vorgegeben wird (siehe `admin`(1)).

`-b` wird zusammen mit der Option `-e` verwendet, wenn das neue Delta eine SID in einer neuen Verzweigung haben sollte, wie in Tabelle 1 angegeben. Diese Option wird ignoriert, wenn die Datei nicht das `b`-Flag gesetzt hat (siehe `admin`(1)) oder wenn das wiedergewonnene Delta kein Blatt-Delta ist (ein Blatt-Delta hat keine Nachfolger im SCCS-Dateibaum). Ein Verzweigungsdelta kann immer von einem Nicht-Blatt-Delta erstellt werden. Teil-SIDs werden entsprechend der Angabe in der Spalte 'erhaltene SID' in der Tabelle auf Seite 77 verstanden.

`-k` unterdrückt Ersatz der Schlüsselwörter im wiederhergestellten Text durch ihren jeweiligen Wert. Die Option `-k` wird durch die Option `-e` impliziert.

`-l[p]` Delta-Zusammenfassung wird in eine l-Datei geschrieben. Bei Verwendung von `-lp` wird keine l-Datei erstellt; stattdessen wird die Delta-Zusammenfassung auf Standardausgabe geschrieben. Das Format der l-Datei ist aus dem Abschnitt "Schlüsselwörter" ersichtlich.

`-p` der von der SCCS -Datei zurückgewonnene Text wird auf Standardausgabe geschrieben. Eine g-Datei wird nicht erstellt. Alle Ausgaben, die normalerweise an die Standardausgabe gehen, werden stattdessen an die Standard-Fehlerausgabe gegeben, es sei denn, die Option `-s` wird verwendet. In diesem Falle verschwinden diese Ausgaben.

`-s` unterdrückt alle Ausgaben, die normalerweise auf die Standardausgabe geschrieben werden. Meldungen von schwerwiegenden Fehlern, die immer auf die Standard-Fehlerausgabe gehen, bleiben jedoch unangetastet.

`-m` jeder von der SCCS-Datei zurückgewonnenen Textzeile wird die SID des Deltas vorangestellt, mit der die Textzeile in die SCCS-Datei eingefügt wurde. Das Format ist: SID mit nachgestelltem horizontalen Tabulator und nachfolgender Textzeile.

`-n` jeder generierten Textzeile wird der Wert des Schlüsselworts `%M%` vorangestellt. Das Format ist: `%M%`-Wert mit anschließendem horizontalen Tabulator und nachfolgender Textzeile. Bei Verwendung der beiden Optionen `-m` und `-n` lautet das Format: `%M%-Wert` mit anschließendem horizontalen Tabulator und nachfolgendem, von der Option `-m` generiertem Format.

`-g` unterdrückt die tatsächliche Wiedergewinnung von Text aus der SCCS-Datei. Diese Option wird hauptsächlich zur Erstellung einer I-Datei oder zur Überprüfung des Vorhandenseins einer bestimmten SID verwendet.

`-t` wird für den Zugriff auf das letzte erstellte Delta in einer bestimmten Version (z.B. `-r1`) oder Version und Level (z.B. `-r1.2`) verwendet.

`-w`*Zeichenkette*
ersetzt beim Holen der Datei jedes Auftreten von `%W%` durch *Zeichenkette*. Die Ersetzung findet vor der Schlüsselwort-Erweiterung statt.

`-a`*Folgenr.* die Delta-Folgenummer des Deltas (Version) der SCCS-Datei, das zurückgewonnen werden soll. Diese Option wird vom Kommando `comb` verwendet. Bei Vorgabe der beiden Optionen `-r` und `-a` wird nur die Option `-a` verwendet. Bei der Verwendung der Option `-a` in Verbindung mit der Option `-e` sollten Sie vorsichtig sein, weil die SID des zu erstellenden Deltas möglicherweise nicht die ist, die Sie erwartet haben. Die Option `-r` kann mit den Optionen `-a` und `-e` verwendet werden, um die Benennung der SID des zu erstellenden Deltas zu steuern.

Für jede verarbeitete Datei protokolliert `get` (auf der Standardausgabe) die SID, auf die zugegriffen wird, und die Zahl der von der SCCS-Datei zurückgewonnenen Zeilen.

Bei Verwendung der Option `-e` erscheint die SID des zu erstellenden Deltas nach der SID, auf die zugegriffen wird, und vor der Zahl der generierten Zeilen. Ist mehr als eine angegebene Datei vorhanden oder wird ein Dateiverzeichnis bzw. die Standardeingabe angegeben, wird jeder Dateiname bei Voranstellung eines Neue-Zeile-Zeichens vor der Verarbeitung ausgegeben. Wenn die Option `-i` verwendet wird, werden eingebundene Deltas im Anschluß an die Notation `Included` aufgelistet; bei Verwendung der Option `-x` werden ausgeschlossene Deltas im Anschluß an die Notation `Excluded` aufgelistet.

angegebene SID 1)	Option -b benutzt 5)	andere Bedingungen	erhaltene SID	SID des neuen Deltas
keine 6)	nein	R standardm. auf mR	mR.mL	mR.(mL+1)
keine 6)	ja	R standardm. auf mR	mR.mL	mR.mL.(mB+1).1
R	nein	R > mR	mR.mL	R.1 3)
R	nein	R = mR	mR.mL	mR.(mL+1)
R	ja	R > mR	mR.mL	mR.mL.(mB+1).1
R	ja	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR und R exist. *nicht*	hR.mL 2)	hR.mL.(mB+1).1
R	-	Stamm Nachf 4) in Version > R und R existiert	R.mL	R.mL.(mB+1).1
R.L	nein	Kein Stamm-Nachf	R.L	R.(L+1)
R.L	ja	Kein Stamm-Nachf	R.L	R.L.(mB+1).1
R.L	-	Stamm-Nachf in Version R	R.L	R.L.(mB+1).1
R.L.B	nein	Kein Verzweig.-Nachf	R.L.B.mS	R.L.B.(mS+1)
R.L.B	ja	Kein Verzweig.-Nachf	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	nein	Kein Verzweig.-Nachf	R.L.B.S	R.L.B.(S+1)
R.L.B.S	ja	Kein Verzweig.-Nachf	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Verzweigung-Nachf	R.L.B.S	R.L.(mB+1).1

1) R, L, B, bzw. S sind die Release-, Level-, Verzweigungs- bzw. Folge-Komponenten von SID; m bedeutet Maximum. So bedeutet beispielsweise R.mL die maximale Level-Nummer innerhalb Version R; R.L.(mB+1).1 bedeutet die erste Folgezahl in der neuen Verzweigung (d.h. maximale Verzweigungsanzahl plus eins) von Level L innerhalb Version R. Es ist zu beachten, daß bei Angabe von SID im Format R.L, R.L.B oder R.L.B.S jede der angegebenen Komponenten vorhanden sein muß.

2) hR ist die höchste bestehende Version, die niedriger als die angegebene nicht bestehende Version R ist.

3) Damit muß das erste Delta in einer neuen Version erstellt werden.

4) Nachfolger.

5) Die Option -b ist nur dann wirksam, wenn die Option b (siehe admin(1)) in der Datei gesetzt ist. Der Eintrag - bedeutet 'nicht relevant'.

6) Dieser Fall trifft zu, wenn die Option d Standard-SID) nicht in der Datei vorhanden ist. Wenn die Option d in der Datei vorhanden ist, wird die von der Option d erhaltene SID so verstanden, als ob sie in der Kommandozeile angegeben wurde. Daher trifft einer der anderen Fälle in dieser Tabelle zu.

Schlüsselwörter

Identifikationsinformationen werden in den Text eingefügt, der von der SCCS-Datei gewonnen wurde, indem Schlüsselwörter bei jedem Auftreten durch ihren aktuellen Wert ersetzt werden. Die nachstehenden Schlüsselwörter können im Text einer SCCS-Datei verwendet werden:

```
Schlüsselwort  Wert

%M%            Modulbezeichnung: entweder der Wert der Option
               m in der Datei (siehe admin(1)),
               oder, wenn nicht vorhanden, der Name der SCCS-Datei ohne führendes s.
%I%            SCCS-Identifikation (SID) (%R%.%L%.%B%.%S%) des wiedergewonnenen Textes
%R%            Release (Version)
%L%            Level
%B%            Verzweigung
%S%            Folge
%D%            aktuelles Datum (JJ/MM/TT)
%H%            aktuelles Datum (MM/TT/JJ)
%T%            aktuelle Uhrzeit (HH:MM:SS)
%E%            Erstellungsdatum des neuesten Deltas (JJ/MM/TT)
%G%            Erstellungsdatum des neuesten Deltas (MM/TT/JJ)
%U%            Erstellungszeit des neuesten Deltas (HH:MM:SS)
%Y%            Modultyp: Wert der Option t in der SCCS-Datei (siehe admin(1))
%F%            SCCS-Dateiname
%P%            voll qualifizierter SCCS-Dateiname
%Q%            der Wert der Option q in der Datei (siehe admin(1))
%C%            aktuelle Zeilennummer.
               Dieses Schlüsselwort soll für die Bezeichnung der Meldungen dienen, die vom
               Programm ausgegeben werden, wie z. B. Fehler des Typs 'dies hätte nicht
               geschehen sollen'. Es ist nicht zum Erstellen von Folgenummern in jeder Zeile gedacht!
%Z%            die Vier-Zeichen-Folge @(#), die what erkennt
%W%            eine kurzgefaßte Notation für den Aufbau von
               what-Zeichenketten für Programmdateien des SINIX-Systems:
               %W% = %Z%%M%<Horizontal-Tab>%I%
%A%            eine weitere kurzgefaßte Notation für den Aufbau von what-Zeichenketten
               für Programmdateien außerhalb des SINIX-Systems:
               %A% = %Z%%Y% %M% %I%%Z%
```

Mit `get` können mehrere Hilfsdateien erstellt werden. Diese Dateien werden als g-Datei, l-Datei, p-Datei und z-Datei bezeichnet. Der Buchstabe vor dem Bindestrich ist die Typkennung. Ein Hilfsdateiname wird aus dem SCCS-Dateinamen gebildet: Der Dateiname ohne Pfadangabe jeder SCCS-Datei muß das Format `s.`*Modulname* haben; die Hilfsdateien werden durch Ersetzen des führenden `s` durch die Typkennung angegeben.

Die g-Datei bildet eine Ausnahme von diesem Schema: Der Name der g-Datei wird durch Entfernen des Vorsatzes `s.` gebildet. Zum Beispiel lauten bei `s.xyz.c` die Namen der Hilfsdateien `xyz.c`, `l.xyz.c`, `p.xyz.c` bzw. `z.xyz.c`.

Die g-Datei, die den generierten Text enthält, wird im aktuellen Dateiverzeichnis erstellt, sofern nicht die Option `-p` verwendet wird. Eine g-Datei wird in allen Fällen erstellt, wobei es keine Rolle spielt, ob Textzeilen durch das `get` erstellt wurden oder nicht. Sie gehört dem realen Benutzer. Wenn die Option `-k` verwendet oder impliziert wird, ist ihr Modus 644; andernfalls handelt es sich um den Modus 444. Nur der reale Benutzer muß über eine Schreiberlaubnis im aktuellen Dateiverzeichnis verfügen.

Die l-Datei enthält eine Liste der Deltas, die für die Generierung des wiedergewonnenen Textes verwendet wird. Die l-Datei wird im aktuellen Dateiverzeichnis erstellt, wenn die

Option `-l` verwendet wird; ihr Modus lautet 444, und sie gehört dem realen Benutzer. Nur der reale Benutzer braucht Schreiberlaubnis in der aktuellen Datei.

Die Zeilen in der l-Datei haben folgendes Format:

a. ein Leerzeichen, wenn das Delta angewendet wurde; andernfalls *
b. ein Leerzeichen, wenn das Delta angewendet wurde oder nicht angewendet und ignoriert wurde; *, wenn das Delta nicht angewendet und nicht ignoriert wurde
c. ein Code, mit dem ein 'besonderer' Grund für die Anwendung oder Nichtanwendung des Deltas angezeigt wird: 'I' : eingeschlossen (Included), 'X': ausgeschlossen (Excluded), 'C': abgeschnitten (durch eine Option `-c`).
d. leer
e. SCCS-Identifikation (SID)
f. Tabulatorzeichen
g. Datum und Uhrzeit (im Format *JJ/MM/TT HH:MM:SS*) der jeweiligen Erstellung
h. leer
i. Login-Name der Person, die `delta` erstellt hat

Die Kommentare und MR-Daten stehen in aufeinanderfolgenden Zeilen und sind jeweils um ein horizontales Tabulatorzeichen eingerückt. Jede Eingabe wird mit einer Leerzeile abgeschlossen.

Die p-Datei wird zur Weitergabe von Informationen an `delta` verwendet, die sich aus einem `get` mit der Option `-e` ergeben. Der Inhalt der Datei wird auch zur Verhinderung einer anschließenden Ausführung von `get` mit der Option `-e` für dieselbe SID bis zur Ausführung von `delta` oder zum Setzen des gemeinsamen Editieranzeigers `j` (siehe `admin`(1)) in der SCCS-Datei verwendet. Die p-Datei wird in dem Dateiverzeichnis erstellt, das die SCCS-Datei enthält, und der effektive Benutzer muß eine Schreiberlaubnis in diesem Dateiverzeichnis haben. Ihr Modus ist 644 und sie ist im Besitz des effektiven Benutzers. Eine Zeile in der p-Datei hat folgendes Format:

SID neue_SID login Datum/Uhrzeit_von_get [*argument_zu_*`-i`] [*argument_zu_*`-x`]`\n`

In der p-Datei kann zu jedem Zeitpunkt eine beliebige Anzahl von Zeilen vorhanden sein; dieselbe neue Delta-SID darf nicht in zwei Zeilen gleichzeitig auftreten.

Die z-Datei dient als Sperrmechanismus gegen gleichzeitige Aktualisierungen. Ihr Inhalt besteht aus der binären (2-Byte-) Prozeß-Nummer des Kommandos (d.h. `get`), mit dem sie erstellt wurde. Die z-Datei wird in dem Dateiverzeichnis erstellt, das die SCCS-Datei für die Dauer von `get` enthält. Für die z-Datei gelten dieselben Einschränkungen wie für die p-Datei. Die z-Datei wird im Modus 444 angelegt.

DATEIEN

g-Datei	bei der Ausführung von `get` erzeugt
p-Datei	siehe `delta`(1)
q-Datei	siehe `delta`(1)
z-Datei	siehe `delta`(1)
`bdiff`	Programm zur Ermittlung der Differenzen zwischen der 'erhaltenen' Datei und der g-Datei

SIEHE AUCH

`admin`(1), `delta`(1), `help`(1), `prs`(1), `what`(1).
`bdiff`(1) in "SINIX V5.41 Kommandos".

HINWEIS

Wenn der effektive Benutzer Schreiberlaubnis für das Dateiverzeichnis hat, das die SCCS-Dateien enthält, der reale Benutzer jedoch nicht, darf bei Verwendung der Option `-e` nur eine Datei angegeben werden.

help - Hilfe für Nachrichtennummern oder SCCS-Kommandos

help [Argument...]

`help` gibt weitere Informationen zu einer Meldung eines SCCS-Kommandos oder zu dem Kommando selbst. Kein oder mehrere Argumente können angegeben werden. Sollte kein Argument angegeben sein, fordert `help` zur Eingabe eines solchen auf.

Die Argumente können die SCCS-Kommandos selbst oder die in Klammern stehenden Informationen einer Meldung eines SCCS-Kommandos sein.

Das Programm gibt genauere Information zu *Argument*.

Wenn `help` nicht erfolgreich ausgeführt wird, geben Sie bitte `help stuck` ein.

DATEIEN

LIBDIR`/help`	Verzeichnis, das die Nachrichtentexte enthält
LIBDIR`/help/helploc`	Datei, die die Verzeichnisse von Hilfe-Dateien enthält, die nicht in *LIBDIR*`/help` stehen
LIBDIR	normalerweise `/usr/ccs/lib`

install - Kommandos installieren

/usr/sbin/install [Option...] Datei [Verzeichnis...]

Das Kommando `install` wird meist in 'makefiles' verwendet (siehe `make`(1)), um eine *Datei* (aktuelle Zieldatei) an einem bestimmten Platz im Dateisystem zu installieren. Jede *Datei* wird durch Kopieren in das entsprechende Verzeichnis geschrieben und behält Modus und Eigentümer der ursprünglichen Datei. Das Programm liefert Meldungen, die den Benutzer genau darüber informieren, welche Dateien ersetzt oder erzeugt werden und wo dies geschieht.

Wenn keine *Optionen* oder *Verzeichnisse* angegeben sind, dann durchsucht `install` die Liste mit voreingestellten Verzeichnissen (`/bin`, `/usr/bin`, `/etc`, `/lib` und `/usr/lib` in dieser Reihenfolge) nach einer Datei mit dem Namen *Datei*. Wird eine Datei gefunden, meldet `install`, daß die Datei mit *Datei* überschrieben wird. Wird keine Datei gefunden, meldet das Programm dies und führt keine weitere Operation aus.

Wenn ein *Verzeichnis* oder mehrere *Verzeichnisse* nach *Datei* angegeben sind, werden diese Verzeichnisse vor den Verzeichnissen aus der Liste durchsucht.

Folgende Optionen können beim Aufruf von `install` angegeben werden:

`-c` *dira* installiert ein neues Kommando (*Datei*) im Verzeichnis *dira* nur, wenn es nicht gefunden wird. Wird es gefunden, gibt `install` die Meldung aus, daß die Datei bereits existiert und daher nicht überschrieben wird. Diese Option kann allein oder zusammen mit der Option `-s` verwendet werden.

`-f` *dirb* installiert *Datei* im angegebenen Verzeichnis, unabhängig davon, ob die Datei bereits existiert oder nicht. Wenn die Datei noch nicht existiert, werden Modus und Eigentümer der neuen Datei auf `755` und `bin` gesetzt. Wenn die Datei bereits existiert, wird der Modus und der Eigentümer übernommen. Diese Option kann allein oder zusammen mit den Optionen `-o` oder `-s` verwendet werden.

`-i` ignoriert die voreingestellte Verzeichnisliste und durchsucht nur die angegebenen *Verzeichnisse*. Diese Option kann allein oder zusammen mit anderen Optionen, außer `-c` und `-f`, verwendet werden.

`-n` *dirc* Wenn *Datei* nicht in den durchsuchten Verzeichnissen gefunden wird, wird sie in das Verzeichnis *dirc* geschrieben. Der Modus und der Eigentümer der neuen Datei wird auf `755` und `bin` gesetzt. Diese Option kann alleine oder zusammen mit anderen Optionen, außer `-c` und `-f`, verwendet werden.

`-m` *Modus* Der Modus der neuen Datei wird auf *Modus* gesetzt.

`-u` *Benutzer*
: Der Eigentümer der neuen Datei wird auf *Benutzer* gesetzt.

`-g` *Gruppe*
: Die Gruppennummer der neuen Datei wird auf *Gruppe* gesetzt. Diese Funktion ist nur dem Systemverwalter zugänglich.

`-o`
: Wenn *Datei* gefunden wird, sichert diese Option die gefundene Datei durch Kopieren nach `OLD`*Datei*. Diese Option ist sinnvoll, wenn häufig benutzte Dateien wie zum Beispiel `/bin/sh` oder `/lib/saf/ttymon` installiert werden und die existierende Datei nicht entfernt werden kann. Diese Option kann allein oder zusammen mit anderen Optionen, außer `-c`, verwendet werden.

`-s`
: unterdrückt die Ausgabe von Meldungen, bei denen es sich nicht um Fehlermeldungen handelt. Diese Option kann allein oder zusammen mit anderen Optionen verwendet werden.

SIEHE AUCH

`make`(1).

ld - Binder für Objektdateien

ld [Option...] Datei...

Das Kommando `ld` bindet mehrere Dateien mit verschiebbarem Objektcode zusammen, führt die Adreßauflösung durch und behandelt extern definierte Symbole. `ld` arbeitet in zwei Modi, statisch oder dynamisch. Der zu verwendende Modus wird durch die Option `-d` festgelegt. Im statischen Modus, `-dn`, werden die als Argument gegebenen verschiebbaren Objektdateien zusammengebunden, um eine ausführbare Objektdatei zu erzeugen; wenn die Option `-r` verwendet wird, werden die verschiebbaren Objektdateien zu einer verschiebbaren Objektdatei zusammengebunden. Im dynamischen Modus, `-dy`, dem Standard, werden die als Argument gegebenen verschiebbaren Objektdateien zu einer ausführbaren Datei zusammengebunden, die zur Ausführungszeit mit allen als Argument gegebenen, gemeinsam benutzten Objektdateien geladen wird; wenn die Option `-G` angegeben ist, werden die angegebenen verschiebbaren Objektdateien zu einem gemeinsam benutzten Objekt zusammengebunden. In allen Fällen wird die Ausgabe von `ld` standardmäßig in `a.out` abgelegt.

Wenn ein Argument eine Bibliothek ist, wird sie genau einmal zu dem Zeitpunkt durchsucht, an dem sie in der Argumentliste angetroffen wird. Die Bibliothek kann ein verschiebbares Archiv oder ein gemeinsam benutztes Objekt sein. Bei Archivbibliotheken werden nur solche Routinen geladen, die einen externen noch nicht aufgelösten Verweis definieren. Die Symboltabelle der Archivbibliothek (siehe `ar`(4)) wird sequentiell in so vielen Durchgängen durchsucht, wie zum Auflösen externer Verweise, die durch Elemente der Bibliothek befriedigt werden können, erforderlich sind. Folglich ist die Anordnung der Bibliothekselemente ohne Bedeutung, solange nicht mehrere Bibliothekselemente vorhanden sind, die denselben externen Namen definieren. Ein gemeinsam benutztes Objekt besteht aus einer einzigen Einheit. Alle seine externen Verweise müssen in der aufzubauenden ausführbaren Datei oder in anderen gemeinsam benutzten Objekten, mit denen es zusammengebunden wird, aufgelöst werden.

Folgende Optionen werden von `ld` erkannt:

`-a` nur im statischen Modus:
erzeugt eine ausführbare Objektdatei; liefert Fehlermeldungen für nicht definierte Verweise. Dies ist das standardmäßige Verhalten für den statischen Modus. `-a` darf nicht zusammen mit der Option `-r` verwendet werden.

`-b` nur im dynamischen Modus:
verzichtet bei der Erzeugung einer ausführbaren Datei auf spezielle Maßnahmen für Symbole aus gemeinsam benutzten Objektdateien. Ohne die Option `-b` erzeugt der Binder besondere positionsunabhängige Zugriffe auf Verweise auf Funktionen, die in gemeinsam benutzten Objekten definiert sind. Er sorgt dafür, daß der dynamische Binder Datenobjekte, die in

gemeinsam benutzten Objekten definiert sind, zur Laufzeit in das Speicherabbild der ausführbaren Datei kopiert. Mit der Option `-b` kann der erzeugte Code effizienter werden, was allerdings zu Lasten der gemeinsamen Benutzbarkeit gehen kann.

`-d[y|n]` Bei `-dy` (Standardeinstellung) verwendet `ld` dynamisches Binden; bei `-dn` verwendet `ld` statisches Binden.

`-e` *StartSym*
Die Adresse des Symbols *StartSym* soll die Startadresse für das erzeugte ausführbare Programm werden.

`-h` *Name* nur im dynamischen Modus:
bei der Erstellung eines gemeinsam benutzten Objekts wird *Name* im dynamischen Abschnitt des Objekts vermerkt. In ausführbaren Dateien, die mit diesen Objekten gebunden werden, wird ebenfalls dieser *Name* anstelle des SINIX-Dateinamens des Objekts verwendet. Dementsprechend wird *Name* vom dynamischen Lader bei der Suche zur Laufzeit als Name des gemeinsam benutzten Objekts verwendet.

`-l`*x* sucht eine Bibliothek `lib`*x*`.so` bzw. `lib`*x*`.a`. Dies sind die üblichen Namen für konventionelle bzw. gemeinsam benutzte Bibliotheken. Im dynamischen Modus und solange nicht die Option `-Bstatic` gewählt wurde, sucht `ld` in jedem im Suchpfad für Bibliotheken angegebenen Verzeichnis nach der Datei `lib`*x*`.so` bzw. `lib`*x*`.a`. Die Suche endet, sobald in einem der Verzeichnisse eine der beiden Dateien gefunden wurde. Wenn `-l`*x* zu zwei in Frage kommenden Dateien `lib`*x*`.so` und `lib`*x*`.a` führt, wählt `ld` jene Datei, die auf `.so` endet. Wird keine Datei `lib`*x*`.so` gefunden, dann akzeptiert `ld` auch `lib`*x*`.a`. Im statischen Modus, oder wenn die Option `-Bstatic` verwendet wird, wählt `ld` nur die Datei, die auf `.a` endet. Eine Bibliothek wird zu dem Zeitpunkt durchsucht, zu dem ihr Name angetroffen wird; daher ist die Position einer `-l`-Option in der Kommandozeile von Bedeutung.

`-m` gibt eine Tabelle mit der Speicherzuordnung oder eine Liste der Ein-/Ausgabe-Sektionen auf der Standardausgabe aus.

`-o` *Ausgabedatei*
verwendet für die ausgegebene Objektdatei den Namen *Ausgabedatei*. Der Name der Objektdatei lautet standardmäßig `a.out`.

`-r` kombiniert verschiebbare Objektdateien zu einer einzigen verschiebbaren Objektdatei. Diese Option kann nicht zusammen mit `-a` verwendet werden.

`-s` entfernt Symboltabellen-Informationen aus der Ausgabedatei. Die Abschnitte mit den Zusatzinformationen zur Fehlersuche und mit den Zeilennummern, sowie die dazugehörigen Verschiebe-Informationen werden entfernt. Außer bei verschiebbaren Dateien oder bei Objektdateien zur

gemeinsamen Benutzung werden die Abschnitte mit der Symboltabelle und mit der Zeichenkettentabelle gleichfalls aus der erzeugten Objektdatei entfernt.

`-t` schaltet die Warnung zu mehrfach definierten Symbolen ab, die nicht die gleiche Größe aufweisen.

`-u` *Symbolname*
Symbolname wird als undefiniertes Symbol in der Symboltabelle eingetragen. Dies ist beim Laden einer Bibliothek nützlich, weil die Symboltabelle anfänglich leer ist und ein nicht aufgelöster Verweis benötigt wird, um die erste Funktion zu laden. Die Position dieser Option in der Kommandozeile ist signifikant; sie muß vor die Bibliothek gesetzt werden, die das Symbol definiert.

`-z defs` führt zu einem Abbruchfehler, wenn am Ende des Bindens undefinierte Symbole übrig bleiben. Dies ist der Standard für die Erstellung von ausführbaren Dateien. Diese Option ist auch nützlich bei der Erstellung eines Objekts zur gemeinsamen Benutzung, wenn sichergestellt werden soll, daß das Objekt selbstenthaltend ist, das heißt, daß alle seine symbolischen Referenzen intern aufgelöst werden.

`-z nodefs` erlaubt undefinierte Symbole. Dies ist Standard bei der Erstellung von gemeinsam benutzten Objekten. Es kann bei der Erstellung von ausführbaren Objektdateien im dynamischen Modus verwendet werden, wenn dabei eine gemeinsam benutzte Bibliothek mit nicht aufgelösten Referenzen hinzugebunden wird, und diese Referenzen nur in nicht benutzten Funktionen vorkommen. Diese Option sollte mit Vorsicht behandelt werden.

`-z text` nur im dynamischen Modus:
führt zu einem Fehler, wenn Verschiebungen gegen nicht beschreibbare, anforderbare Abschnitte übrigbleiben.

`-B[dynamic|static]`
steuert die Verwendung von Bibliotheken. `-Bdynamic` ist nur im dynamischen Modus zulässig. Diese Optionen können beliebig oft in einer Kommandozeile verwendet werden und dienen als Umschalter:
wenn `-Bstatic` angegeben ist, werden keine gemeinsam benutzten Objekte akzeptiert, solange bis `-Bdynamic` vorgefunden wird. Siehe dazu auch die Option `-l`.

`-Bsymbolic` nur im dynamischen Modus:
bindet bei der Erstellung eines gemeinsam benutzten Objekts Referenzen auf global sichtbare Symbole an ihre Definitionen innerhalb des Objekts, wenn Definitionen vorhanden sind. Normalerweise werden Referenzen auf global sichtbare Symbole in gemeinsam benutzten Objekten erst zur Laufzeit gebunden, selbst wenn eine Definition verfügbar ist, so daß die Defini-

tion innerhalb des Objekts durch die Definition desselben Symbols in einer ausführbaren Datei oder in einem anderen, gemeinsam benutzten Objekt überschrieben werden kann. `ld` erzeugt Warnungen für undefinierte Symbole, wenn dies nicht durch die Option `-z defs` abgestellt ist.

`-G` nur im dynamischen Modus:
erzeugt ein Objekt zur gemeinsamen Benutzung. Undefinierte Symbole sind erlaubt.

`-I` *Name* verwendet *Name* als Pfadname des Interpreters, der bei der Erstellung einer ausführbaren Datei in den Programmvorspann geschrieben wird. Der Standard im statischen Modus ist kein Interpreter; im dynamischen Modus ist der Standard der Name des dynamischen Binders `/usr/lib/libc.so.1`. In beiden Fällen kann durch `-I` überschrieben werden. `exec` wird beim Laden von `a.out` auch diesen Interpreter laden und die Kontrolle an den Interpreter, anstatt direkt an `a.out`, übergeben.

`-L`*Verz* fügt *Verz* in den Suchpfad für Bibliotheken ein. `ld` sucht Bibliotheken zuerst in allen Verzeichnissen, die durch `-L`-Optionen angegeben wurden, erst danach in den Standardverzeichnissen. Diese Option ist nur wirksam, wenn sie vor der Option `-l` in der Kommandozeile steht.

`-M` *AnwDatei*
nur im statischen Modus:
liest *AnwDatei* als Textdatei mit Anweisungen an `ld` ein. Da diese Anweisungen die Form der von `ld` erzeugten Ausgabedatei ändern, wird von der Verwendung dieser Option dringend abgeraten.

`-Q[y|n]` Mit `-Qy` wird eine `ident`-Zeichenkette zu dem `.comment`-Abschnitt der Ausgabedatei hinzugefügt, um die Version des Binders, der die Datei erzeugt hat, zu vermerken. Dies führt zu mehrfachen `ld-ident`s, wenn das Binden in mehreren Schritten vorgenommen wurde, wie durch die Verwendung von `ld -r`. Dies entspricht der standardmäßigen Vorgehensweise des `cc`-Kommandos. `-Qn` unterdrückt die Versionsinformation.

`-V` gibt eine Meldung mit Informationen über die Version des eingesetzten `ld` aus.

`-YP` *VerzListe*
ändert die Standardliste mit Verzeichnissen, in denen nach Bibliotheken gesucht wird. *VerzListe* ist eine durch Kommata getrennte Liste mit Pfadnamen.

Die Umgebungsvariable LD_LIBRARY_PATH kann zur Angabe von Verzeichnissen verwendet werden, wenn nach Bibliotheken gesucht werden soll. Im allgemeinsten Fall besteht die Umgebungsvariable aus zwei, durch ein Semikolon getrennten, Verzeichnislisten:

VerzListe1;*VerzListe2*

Wenn ld mit einer beliebigen Anzahl von -L-Optionen aufgerufen wird, wie in

ld ... -L*Pfad1* ...-L*PfadN* ...

dann ist die Reihenfolge im Suchpfad:

VerzListe1 Pfad1 ... PfadN VerzListe2 LIBPATH

Beachten Sie, daß LD_LIBRARY_PATH auch vom dynamischen Binder benutzt wird. Ist LD_LIBRARY_PATH in Ihrer Umgebung gesetzt, durchsucht der dynamische Binder die dort aufgeführten Dateiverzeichnisse nach den mehrfach benutzten Objekten, die zur Laufzeit mit Ihrem Programm gebunden werden.

Die Umgebungsvariable LD_RUN_PATH, mit einer Verzeichnisliste, kann ebenfalls dazu verwendet werden, Verzeichnisse für die Bibliotheken dem dynamischen Binder zu übergeben. Wenn sie definiert und ungleich Null ist, wird sie über Daten, die in die ausgegebenen Objektdatei geschrieben werden, von ld an den dynamischen Binder übergeben.

DATEIEN

libx.so	Bibliotheken
libx.a	Bibliotheken
pa.out	Ausgabedatei
LIBPATH	üblicherweise /usr/ccs/lib:/usr/lib

SIEHE AUCH

cc(1), exec(2), exit(2), end(3C), a.out(4), ar(4).

Kapitel "Das C-Übersetzungssystem" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ldd - dynamische Abhängigkeiten ausgeben

ldd [-d | -r] Datei

Das `ldd`-Kommando (list dynamic dependencies) gibt die Pfadnamen aller gemeinsam benutzten Objekte aus, die bei der Ausführung von *Datei* geladen werden. Ist *Datei* eine ausführbare Datei, die nicht auf gemeinsam benutzte Objekte zugreift, läuft `ldd` erfolgreich ab und generiert keine Ausgaben.

`ldd` kann außerdem verwendet werden, um die Kompatibilität der *Datei* mit den gemeinsam benutzten Objektdateien zu prüfen. Können Symbolreferenzen nicht aufgelöst werden, so werden Warnungen ausgegeben. Eine der folgenden Optionen kann beim Aufruf von `ldd` angegeben werden:

`-d` veranlaßt `ldd` zur Prüfung aller Referenzen auf Datenobjekte.

`-r` veranlaßt `ldd` zur Überprüfung der Datenobjekte und Funktionen.

SIEHE AUCH

`cc`(1), `ld`(1).
Kapitel "Das C-Übersetzungssystem" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ENDESTATUS

`ldd` gibt die Pfadnamen der gemeinsam benutzten Objekte über `stdout` aus. Die Ausgabe der nichtauflösbaren Referenzen wird über `stderr` ausgegeben. Wenn *Datei* keine ausführbare Datei ist oder nicht zum Lesen geöffnet werden kann, so liefert das Kommando einen Rückgabewert ungleich Null.

HINWEIS

`ldd` führt keine gemeinsam benutzten Objekte auf, die über `dlopen`(3X) angehängt werden.

`ldd` verwendet zum Auffinden der gemeinsam benutzten Objekte den selben Algorithmus wie der dynamische Binder.

lex - Scanner erstellen

lex [-ctvn -V -Q[y|n]] [Datei]

`lex` erzeugt ein C-Programm aus einer *Datei* ("`lex`-Quelltext"), den Sie für das vorliegende Problem entwickelt haben. Ein `lex`-Quelltext besteht aus höchstens drei Abschnitten: Definitionen, Regeln und Benutzerfunktionen. Die Regeln geben an, welche Muster in einem Eingabetext gesucht und welche Aktionen ausgeführt werden sollen, wenn ein Muster gefunden wurde. Sie müssen angegeben werden. Die Definitionen und Benutzerfunktionen sind optional.

Mehrere Dateien werden wie eine Einzeldatei behandelt. Wenn keine Datei angegeben wird, wird die Standardeingabe verwendet.

`lex` erzeugt eine Datei mit dem Namen `lex.yy.c`. Wenn `lex.yy.c` mit der Lex-Bibliothek übersetzt und gebunden wird, kopiert es die Eingabe auf die Ausgabe, es sei denn, ein in der Datei angegebenes Muster wird gefunden. In diesem Fall wird der entsprechende Programmtext ausgeführt. Das Muster, für das eine Übereinstimmung gefunden wurde, befindet sich in `yytext[]`, einem externen Zeichenfeld. Die Prüfung auf Übereinstimmung wird in der Reihenfolge der Suchmuster in der Eingabedatei durchgeführt.

Die `lex`-Optionen haben folgende Bedeutung:

`-c`	steht für die Verwendung von C-Aktionen und ist der Standard
`-t`	das Programm wird in die Datei `lex.yy.c`, nicht auf Standardausgabe geschrieben
`-v`	liefert eine zweizeilige Statistik-Zusammenfassung
`-n`	verhindert Ausdrucken der Zusammenfassung von `-v`
`-V`	gibt Versionsinformationen auf die Standard-Fehlerausgabe aus
`-Q[y|n]`	gibt im Fall von `-Qy` Versionsinformationen an die Ausgabedatei `lex.yy.c` aus. Die Option `-Qn` gibt keine Versionsinformationen aus und ist der Standard.

Bestimmte Standard-Tabellengrößen sind für einige Benutzer zu klein. Die Tabellengrößen für den erzeugten endlichen Automaten können im Definitionsabschnitt gesetzt werden:

`%p` *n*	Anzahl der Positionen ist *n* (Standard 2500)
`%n` *n*	Anzahl der Zustände ist *n* (500)
`%e` *n*	Anzahl der Knoten des Syntaxbaums ist *n* (1000)
`%a` *n*	Anzahl der Übergänge ist *n* (2000)
`%k` *n*	Anzahl der gepackten Zeichenklassen ist *n* (2500)
`%o` *n*	Größe des Ausgabefelds ist *n* (3000)

Die Verwendung einer oder mehrerer Größen zieht automatisch die Option `-v` nach sich, wenn die Option `-n` nicht verwendet wird.

Der Regelteil der *Datei* beginnt mit dem Begrenzungssymbol `%%`. Sie können im Regelteil lokale Variablen für `yylex()` vereinbaren. Alle Zeilen im Regelteil, die mit einem Leerzeichen oder Tabulator beginnen und vor der ersten Regel stehen, werden an den Anfang der Funktion `yylex()` kopiert, direkt hinter die erste geöffnete Klammer.

Jede Regel besteht aus einem regulären Ausdruck, der ein aufzufindendes Muster beschreibt, und Aktionen, die ausgeführt werden sollen, wenn das Muster gefunden wird. Eingabetext, der keinem aufzufindenden Muster entspricht, wird von `lex` unverändert an die Ausgabedatei weitergegeben.

Ein regulärer Ausdruck besteht aus Textzeichen mit oder ohne zusätzliche Operatoren.

Folgende Operatoren können bei `lex` verwendet werden:

Ausdruck	Bedeutung
\x	x
"xy"	xy, auch wenn x und/oder y lex-Operatoren sind (außer \)
[xy]	x oder y
[x-z]	x, y oder z
[^x]	jedes Zeichen außer x
.	jedes Zeichen außer Neue-Zeile-Zeichen
^x	x am Zeilenanfang
<y>x	x wenn lex sich im Startzustand y befindet
x$	x am Ende einer Zeile
x?	x einmal oder keinmal
x*	leere Zeichenkette oder mehrfaches Vorkommen von x
x+	ein- oder mehrfaches Vorkommen von x
x{m,n}	m bis n Vorkommen von x
xx|yy	xx oder yy
x |	die Aktion von x ist auch die Aktion für die nächste Regel
(x)	x
x/y	x wenn y folgt
{xx}	Ersetzung für xx aus dem Definitionsteil

Im Aktionsteil einer Regel können spezielle Aufgaben durchgeführt werden. Folgende Makros werden dafür von `lex` zur Verfügung gestellt:

`input()`	ein weiteres Zeichen wird vom Eingabestrom gelesen
`unput()`	ein Zeichen wird für einen späteren Lesevorgang zurückgestellt
`output()`	ein Zeichen wird in den Ausgabestrom geschrieben

Sie können diese Makros umdefinieren, wenn Sie die Ein-/Ausgabe selbst steuern möchten. Achten Sie dabei aber auf Konsistenz.

Abgesehen vom Abspeichern gefundener Muster in `yytext[]` gibt es weitere Möglichkeiten, mit `lex`-Funktionen die gefundenen Textmuster zu bearbeiten:

`yymore()`	Neu erkannte Zeichen werden an die bereits in `yytext[]` befindlichen angehängt (normalerweise wird `yytext[]` mit den nächsten gefundenen Zeichen überschrieben).
`yyless(`*n*`)`	Nur die ersten *n* Zeichen in `yytext[]` werden berücksichtigt.
`REJECT`	Zeichenketten, die sich überlappen oder die zum Teil in einer anderen Zeichenkette enthalten sind, werden verarbeitet. `REJECT` springt direkt zur nächsten Regel, ohne den Inhalt von `yytext[]` zu ändern.

SIEHE AUCH

`yacc`(1).
Kapitel "lex" in "Leitfaden und Werkzeuge für die Programmierung mit C".

lint - C-Programme prüfen

lint [Option...] Datei...

`lint` ermittelt Merkmale der C-Programmdateien, die möglicherweise Fehler darstellen, nicht portabel oder nutzlos sind. Auch prüft es die Typbenutzung strenger als der Übersetzer. `lint` gibt Warnungen und Fehlermeldungen aus. Zu den Punkten, die geprüft werden, gehören nicht erreichbarer Code, Schleifen, in die gesprungen wird, automatische Variablen, die erklärt, jedoch nicht verwendet werden, und logische Ausdrücke mit konstantem Wert. `lint` überprüft Funktionen, die Werte an einigen Stellen zurückgeben und an anderen nicht, Funktionen, die mit variierender Argumentenanzahl oder verschiedenen Argument-Typen aufgerufen werden, sowie Funktionen, deren Werte nicht verwendet werden oder deren Werte zwar verwendet, jedoch nicht zurückgegeben werden.

Argumente, deren Namen mit `.c` enden, werden als C-Quelldateien angesehen. Argumente, deren Namen mit `.ln` enden, werden als das Ergebnis eines früheren Aufrufs von `lint` verstanden, bei dem entweder die Option `-c` oder `-o` verwendet wurde. Die `.ln`-Dateien sind ähnlich den `.o`-(Objekt-)Dateien, die vom Kommando `cc`(1) erstellt werden, wenn eine Datei `.c` als Eingabe verwendet wird. Vor Dateien mit anderen Zusätzen wird gewarnt: diese Dateien werden ignoriert.

`lint` nimmt alle Dateien `.c`, `.ln` und `llib-l`*x*`.ln` (angegeben durch die Option `-l`*x*) und verarbeitet sie in der Reihenfolge, wie sie in der Aufrufzeile erscheinen. Standardmäßig hängt `lint` die Standard-C-Lint-Bibliothek `llib-lc.ln` an das Ende der Dateiliste an. Bei Anwendung der Option `-c` werden die Dateien `.ln` und `llib-l`*x*`.ln` ignoriert. Wird die Option `-c` nicht verwendet, prüft der zweite Durchgang von `lint` die Dateiliste `.ln` und `llib-l`*x*`.ln` auf gegenseitige Kompatibilität.

Die `lint`-Optionen können in beliebiger Anzahl und Reihenfolge sowie mit Dateiargumenten vermischt verwendet werden. Die folgenden Optionen werden zur Unterdrükkung bestimmter Arten von Reklamationen benutzt:

`-a` Reklamationen bei Zuweisungen von `long`-Werten an Variablen unterdrükken, die nicht `long` sind

`-b` Reklamationen bei `break`-Anweisungen unterdrücken, die nicht erreicht werden können

`-h` keine Heuristik-Tests durchführen, die versuchen, Fehler zu vermeiden, den Stil zu verbessern und den Aufwand zu reduzieren

`-m` Reklamationen bei externen Symbolen unterdrücken, die als `static` deklariert sein könnten

`-u` Reklamationen bei Funktionen und externen Variablen unterdrücken, die verwendet und nicht definiert oder definiert und nicht verwendet wurden Diese Option ist für das Ausführen von `lint` in einer Untergruppe von Dateien eines größeren Programms geeignet.

`-v` Reklamationen bei unbenutzten Argumenten in Funktionen unterdrücken

`-x` keine Variablen melden, auf die externe Deklarationen verweisen, die jedoch niemals verwendet werden

Die nachstehenden Argumente verändern das Verhalten von `lint`:

`-I`*Verz* in dem angegebenen Verzeichnis *Verz* nach eingebundenen Include-Dateien suchen, bevor im aktuellen Verzeichnis und/oder im Standardverzeichnis gesucht wird

`-l`*x* die lint-Bibliothek `llib-l`*x*`.ln` mit aufnehmen
So kann man beispielsweise eine lint-Version der Arithmetik-Bibliothek `llib-lm.ln` durch Einfügen von `-lm` in die Kommandozeile einschließen. Mit diesem Argument wird die Standardverwendung von `llib-lc.ln` nicht unterdrückt. Diese lint-Bibliotheken müssen sich im dafür vorgesehenen Dateiverzeichnis befinden. Diese Option kann für Verweise auf lokale lint-Bibliotheken verwendet werden und ist nützlich bei der Entwicklung von Mehrdatei-Projekten.

`-L`*Verz* die lint-Bibliotheken in dem Verzeichnis *Verz* suchen, bevor sie im Standardverzeichnis gesucht werden

`-n` nicht auf Kompatibilität bzgl. der Standard-C-Lint-Bibliothek prüfen

`-p` Portabilität auf andere Dialekte von C prüfen
Gleichzeitig mit der strengeren Prüfung bewirkt diese Option auch ein Abschneiden aller interner Namen auf acht Zeichen und aller externen Namen auf sechs Zeichen und einheitliche Groß- oder Kleinschreibung.

`-s` nur eine einzeilige Diagnose erzeugen
`lint` puffert Meldungen gelegentlich, um einen zusammenhängenden Bericht zu erzeugen.

`-k` Verhalten bei /*LINTED [*Meldung*]*/-Anweisungen ändern
Normalerweise unterdrückt `lint` Warnungen zu der Programmzeile, die dieser Anweisungen folgt. Mit dieser Option jedoch gibt `lint` eine zusätzliche Meldung aus, die den Kommentar innerhalb der Anweisung enthält, anstatt die Meldung zu unterdrücken.

`-y` bearbeitete Datei so behandeln, als wäre die Anweisung /*LINTLIBRARY*/ verwendet worden
Normalerweise wird eine lint-Bibliothek durch die Anweisung /*LINTLIBRARY*/ erzeugt.

`-F` Pfadnamen der Dateien angeben
`lint` gibt den Dateinamen normalerweise ohne den Pfad an.

`-c` `lint` zum Erstellen einer Datei `.ln` für jede Datei `.c` in der Kommandozeile veranlassen
Diese `.ln`-Dateien sind nur das Produkt des ersten Durchgangs von `lint`, es wird nicht auf Kompatibilität zwischen den Funktionen geprüft.

`-o`*x* `lint` erstellt eine lint-Bibliothek mit dem Namen `llib-l`*x*`.ln`
Die Option `-c` hebt jede Angabe der Option `-o` auf. Die eingerichtete lint-Bibliothek ist die Eingabe für den zweiten Durchgang von `lint`. Die Option `-o` bewirkt lediglich, daß diese Datei in der angegebenen lint-Bibliothek gesichert wird. Zur Erzeugung einer `llib-l`*x*`.ln` ohne zusätzliche Meldungen wird die Verwendung der Option `-x` vorgeschlagen. Die Option `-v` ist nützlich, wenn die Quelldateien für die lint-Bibliothek nur externe Schnittstellen sind.

`-V` Produktname und Version an Standard-Fehlerausgabe ausgeben

`-W`*Datei* eine `.ln`-Datei nach *Datei* schreiben, damit sie von `cflow`(1) verwendet werden kann

`-R`*Datei* eine `.ln`-Datei nach *Datei* schreiben, damit sie von `cxref`(1) verwendet werden kann

`lint` erkennt viele `cc`-Optionen, wie `-D`, `-U`, `-g`, `-O`, `-Xt`, `-Xa`, und `-Xc`, wobei `-g` und `-O` ignoriert werden. Bei Verwendung anderer Optionen wird eine Warnung ausgegeben, und sie werden ignoriert.

Durch bestimmte konventionelle Kommentare in der C-Quelle kann das Verhalten von `lint` beeinflußt werden:

/*ARGSUSED n */
sorgt dafür, daß `lint` nur die ersten *n* Argumente zur Verwendung überprüft; ein fehlendes *n* wird als 0 gewertet. Diese Option verhält sich wie die Option `-v` für die nächste Funktion.

/*CONSTCOND*/ oder /*CONSTANTCOND*/ oder /*CONSTANTCONDITION*/
unterdrückt für den nächsten Ausdruck Meldungen über konstante Operanden.

/*EMPTY*/ unterdrückt Meldungen über eine leere Anweisung, die auf eine `if`-Anweisung folgt. Diese Angabe sollte direkt nach dem Test-Ausdruck und vor dem Semikolon stehen. Sie wird zur Verfügung gestellt, um leere `if`-Anweisungen zu unterstützen, wenn eine gültige `else`-Anweisung folgt. Damit werden Meldungen über eine folgende leere `else`-Anweisung unterdrückt.

/*FALLTHRU*/ oder /*FALLTHROUGH*/
: unterdrückt Hinweise auf Anweisungen, die auch direkt von der vorhergehenden Anweisung aus erreicht werden können, obwohl sie mit einem `case`- oder einem `default`-Label versehen sind. Diese Angabe sollte direkt vor dem Label stehen.

/*LINTLIBRARY*/
: am Anfang einer Datei unterdrückt Meldungen über unbenutzte Funktionen und Funktionsargumente in dieser Datei. Das gleiche kann auch mit den Optionen `-v` und `-x` erreicht werden.

/*LINTED [*Meldung*]*/
: unterdrückt alle dateiweiten Warnungen, außer solchen, die unbenutzte Variablen oder Funktionen betreffen. Diese Angabe sollte unmittelbar vor der Zeile stehen, in der die lint-Warnung auftrat. Die Option `-k` ändert die Art, in der `lint` diese Angabe behandelt. Statt Meldungen zu unterdrükken, gibt `lint` dann eine zusätzliche Meldung aus, sofern eine im Kommentar vorhanden ist. Diese Angabe ist im Zusammenhang mit der Option `-s` bei der Verwendung von nachgeschalteten Filtern sinnvoll.

/*NOTREACHED*/
: stoppt an geeigneten Stellen die Kommentare über nicht erreichbaren Code. Dieser Kommentar wird normalerweise unmittelbar nach Aufrufen von Funktionen wie `exit`(2) verwendet.

/*PRINTFLIKE *n* */
: sorgt dafür, daß `lint` wie gewöhnlich die ersten *(n-1)* Argumente überprüft. Das *nte* Argument wird als Formatierungsanweisung für `printf` interpretiert, um die verbleibenden Argumente zu überprüfen.

/*PROTOLIB *n* */
: bewirkt, daß `lint` Funktionsdeklarations-Prototypen wie Funktionsdefinitionen behandelt, wenn *n* ungleich Null ist. Diese Angabe kann nur zusammen mit /* LINTLIBRARY */ verwendet werden. Wenn *n* gleich Null ist, werden Funktions-Prototypen normal behandelt.

/*SCANFLIKE *n* */
: bewirkt, daß `lint` wie gewöhnlich die ersten *(n-1)* Argumente überprüft. Das *nte* Argument wird als Formatierungsanweisung für `scanf` interpretiert, um die verbleibenden Argumente zu überprüfen.

/*VARARGS *n* */
: unterdrückt die übliche Prüfung auf eine veränderliche Anzahl der Argumente in der nachfolgenden Funktionsdeklaration. Die Datentypen der ersten *n* Argumente werden geprüft; ein fehlendes *n* wird als 0 interpretiert. Der Gebrauch des Auslassungssymbols (...) in der Definition wird für neuen oder neubearbeiteten Code vorgeschlagen.

`lint` erzeugt die erste Ausgabe pro Quelldatei. Reklamationen hinsichtlich eingebundener Dateien werden gesammelt und nach Verarbeitung aller Quelldateien ausgegeben, wenn `-s` nicht angegeben wurde. Wird die Option `-c` nicht verwendet, so werden die Informationen von allen Eingabedateien gesammelt und auf Übereinstimmung geprüft. Wenn Unklarheit darüber besteht, ob eine Reklamation von einer angegebenen Quelldatei stammt oder von einer der eingebundenen Dateien, wird an diesem Punkt der Name der Quelldatei mit anschließendem Fragezeichen ausgegeben. Das Verhalten der Optionen `-c` und `-o` ermöglicht eine inkrementelle Verwendung von `lint` in einem Satz von C-Quelldateien. Gewöhnlich ruft man `lint` einmal für jede Quelldatei mit der Option `-c` auf. Mit jedem dieser Aufrufe wird eine `.ln`-Datei angelegt, die der `.c`-Datei entspricht und alle Meldungen ausgibt, die genau diese Quelldatei betreffen. Nachdem alle Quelldateien separat durch `lint` bearbeitet worden sind, wird `lint` erneut aufgerufen (ohne die Option `-c`), wobei alle `.ln`-Dateien mit den benötigten Optionen `-l`*x* aufgelistet werden. Dadurch werden alle Inkonsistenzen zwischen den Dateien ausgegeben. Dieses Verfahren funktioniert gut mit `make`; es ermöglicht die Verwendung von `make`, um nur die Quelldateien, die seit der letzten Bearbeitung dieser Dateien verändert worden sind, mit `lint` zu bearbeiten.

DATEIEN

LIBDIR	das Dateiverzeichnis, in dem die in der Option `-l`*x* angegebenen lint-Bibliotheken vorhanden sein müssen
LIBDIR`/lint[12]`	erster und zweiter Durchgang
LIBDIR`/llib-lc.ln`	Deklarationen für C-Bibliotheksfunktionen (Binärformat; Quelle ist in *LIBDIR*`/llib-lc`)
LIBPATH`/llib-lm.ln`	Deklarationen für Arithmetik-Bibliotheks-Funktionen (Binärformat; Quelle ist in *LIBDIR*/llib-lm)
TMPDIR`/*lint*`	temporäre Dateien
TMPDIR	gewöhnlich `/var/tmp`, kann jedoch durch Setzen der Umgebungsvariablen `TMPDIR` neudefiniert werden (siehe `tempnam`() in `tmpnam`(3S))
LIBDIR	gewöhnlich `/ccs/lib`
LIBPATH	gewöhnlich `/usr/ccs/lib:/usr/lib`

SIEHE AUCH

`cc`(1)
Kapitel "lint" in "Leitfaden und Werkzeuge für die Programmierung mit C".

HINWEIS

`lint` ist auf den PCC C-Compiler zugeschnitten. Mit `lint` können aber auch Programme überprüft werden, die mit dem CES C-Compiler übersetzt werden sollen. Allerdings sind dann nicht alle ausgegebenen Warnungen sinnvoll.

lorder - Ordnungsrelation für Objektbibliothek ermitteln

lorder Datei...

Mit `lorder` kann die Ordnungsrelation für eine oder mehrere Objekt- oder Bibliotheksarchiv-Dateien ermittelt werden (siehe `ar`(1)). Die Standardausgabe ist eine Liste von Paaren von Objektdatei- oder Archivelement-Namen; die erste Datei des Paares bezieht sich auf externe Bezeichner, die in der zweiten Datei definiert sind. Zur Ermittlung einer Ordnung für eine Bibliothek, die sich für einen Zugriff durch `pld`(1) in einem Durchgang eignet, kann die Ausgabe durch `tsort`(1) bearbeitet werden. Es ist zu beachten, daß der Binder `ld` im portablen Archivformat mehrere Durchgänge durch ein Archiv ausführen kann (siehe `ar`(4)) und die Verwendung von `lorder` beim Aufbau eines Archivs nicht benötigt. Die Verwendung des Kommandos `lorder` kann jedoch einen etwas effizienteren Zugriff auf das Archiv während des Binderlaufs ermöglichen.

Im folgenden Beispiel wird eine neue Bibliothek aus vorhandenen `.o`-Dateien aufgebaut.

```
ar -cr library 'lorder *.o | tsort'
```

DATEIEN

TMPDIR`/*symref`	temporäre Dateien
TMPDIR`/*symdef`	temporäre Dateien
TMPDIR	ist gewöhnlich `/var/tmp` , kann jedoch durch Einstellung der Umgebungsvariablen `TEMPDIR` neu definiert werden (siehe `tempnam` in `tmpnam`(3S)).

SIEHE AUCH

`ar`(1), `ld`(1), `tsort`(1), `tempnam`(3s), `tmpname`(3s), `ar`(4).

HINWEIS

`lorder` akzeptiert jede Objekt- oder Archivdatei unabhängig von der jeweiligen Erweiterung als Eingabe, vorausgesetzt, es ist mehr als eine Eingabedatei vorhanden. Wenn nur eine einzelne Eingabedatei vorliegt, muß die Erweiterung `.o` lauten.

m4 - Makroprozessor

m4 [Option...] [Datei...]

Das Kommando m4 ist ein Makroprozessor, der für C-, Assembler und andere Sprachen geeignet ist. Alle Argument-Dateien werden der Reihe nach verarbeitet; sind keine Dateien vorhanden oder lautet ein Dateiname -, wird von Standardeingabe gelesen. Der verarbeitete Text wird auf die Standardausgabe geschrieben.

Die Optionen haben folgende Bedeutung:

-e	interaktiver Betrieb Unterbrechungen werden ignoriert, und die Ausgabe ist ungepuffert.
-s	Zeilennummer-Anpassung für den C-Präprozessor einschalten (#line...)
-B*int*	Größe der Rückstell- und Argumentpuffer ändern (Standard 4.096)
-H*int*	Größe des Hash-Feldes für die Symboltabellen ändern (Standard 199) Diese Größe sollte eine Primzahl sein.
-S*int*	Größe des Aufrufpuffers ändern (Standard 100 Einträge) Makros benötigen drei Einträge, Nicht-Makroargumente einen Eintrag.
-T*int*	Größe des Token-Puffers ändern (Standard: 512 Bytes)

Um wirksam zu sein, müssen die oben angeführten Optionen vor den Dateinamen und vor den Optionen -D oder -U erscheinen:

-D*Name*[=*Wert*]	definiert *Name* zu *Wert* oder Null, falls *Wert* nicht angegeben ist
-U*Name*	löscht die Definition von *Name*.

Makroaufrufe haben das Format:

Name(*arg1*,*arg2*, ..., *argn*)

Zwischen Klammer und Makronamen darf kein Leerzeichen stehen. Wenn auf den Namen eines definierten Makros nicht (folgt, wird angenommen, daß es sich um einen Aufruf des Makros ohne Argumente handelt. Makronamen bestehen aus alphanumerischen Ziffern und dem Unterstrich wobei das erste Zeichen keine Ziffer sein darf.

Führende Leerzeichen ohne Anführungszeichen, Tabulatoren und Neue-Zeile-Zeichen werden während der Sammlung von Argumenten ignoriert. Für die Darstellung von Zeichenketten werden einfache Anführungszeichen verwendet. Der Wert einer mit Anführungszeichen umschlossenen Zeichenkette ist die Zeichenkette nach Entfernung der Anführungszeichen.

Wenn ein Makroname erkannt wird, werden seine Argumente beim Suchen nach der zugehörigen rechten Klammer gesammelt. Wenn weniger Argumente geliefert werden als in der Makrodefinition enthalten sind, werden die nachfolgenden Argumente als Null verstanden. Die Makroauswertung läuft normalerweise während der Sammlung der Argumente ab, und Kommata oder rechte runde Klammern, die sich innerhalb des Wertes eines verschachtelten Aufrufs befinden, werden genauso behandelt, wie die entsprechenden Zeichen im ursprünglichen Eingabetext. Nach der Argumentsammlung wird das Makro in die Eingabe zurückgeschrieben und wieder auf Ersetzung geprüft.

`m4` stellt nachstehende vordefinierte Makros zur Verfügung. Diese Makros können neudefiniert werden, dann geht jedoch ihre ursprüngliche Bedeutung verloren. Ihre Werte sind Null, sofern keine anderen Angaben vorliegen.

`define` erwartet zwei Argumente, wobei das erste den Namen und das zweite den Wert des Makros angibt. Jedes `$`*n* im Ersatztext wird durch das *n*-te Argument ersetzt. Fehlende Argumente werden durch die Nullzeichenkette ersetzt; `$#` wird durch die Anzahl der Argumente ersetzt; `$*` wird durch eine Liste aller durch Kommata voneinander getrennten Argumente ersetzt; `$@` hat dieselbe Bedeutung wie `$*`, wobei jedoch jedes Argument in (die aktuellen) Anführungszeichen gesetzt wird.

`undefine` erwartet als Argument ein Makro, dessen Definition gelöscht wird

`defn` gibt die Definition des Arguments in Anführungszeichen zurück. Dies ist für die Umbenennung von Makros, insbesondere vordefinierten, nützlich.

`pushdef` hat dieselbe Bedeutung wie `define`, sichert jedoch die vorhergehende Definition.

`popdef` löscht die aktuelle Definition des Arguments und macht ggf. eine frühere Definition wieder sichtbar.

`ifdef` Wenn das erste Argument definiert ist, ist der Wert das zweite Argument, sonst das dritte. Wenn es kein drittes Argument gibt, ist sein Wert Null.

`shift` gibt alle Argumente außer dem ersten Argument zurück. Die anderen Argumente werden mit Anführungszeichen versehen und nach Einfügen von Kommata zwischen den einzelnen Argumenten zurückgestellt. Durch die Anführungszeichen wird verhindert, daß die Argumente später zusätzlich überprüft werden.

`changequote`
ändert linke und rechte Anführungszeichen-Symbole (zwei Argumente) Die Symbole können bis zu fünf Zeichen lang sein. `changequote` ohne Argumente stellt die ursprünglichen Werte wieder her (d.h. ` ').

`changecom` ändert die Standardeinstellung für linke (#) und rechte (Neue-Zeile) Kommentarmarkierung. Ohne Argumente ist der Kommentarmechanismus abgeschaltet. Bei einem Argument wird die linke Markierung gemäß dem Argument gesetzt, und die rechte Markierung bleibt Neue-Zeile. Bei zwei Argumenten werden beide Markierungen geändert. Kommentarmarkierungen können bis zu fünf Zeichen lang sein.

`divert` `m4` arbeitet mit 10 Ausgabedateien, die von 0 bis 9 numeriert sind. Die Endausgabe ist die Verkettung dieser Ausgabedateien in numerischer Reihenfolge. Am Anfang ist Datei 0 die aktuelle Datei. Das Makro `divert` (umlenken) legt die aktuelle Ausgabedatei auf das angegebene (Ziffernfolge-) Argument. Ausgaben, die in eine Datei umgelenkt werden, die keine Numerierung zwischen 0 und 9 aufweist, werden gelöscht.

`undivert` liefert alle Ausgaben in numerischer Reihenfolge. Werden Argumente angegeben, liefert `undivert` die gewünschten Ausgaben aus den angegebenen Ausgabepuffern in der angegebenen Reihenfolge.

`divnum` gibt den Wert der aktuellen Ausgabedatei zurück.

`dnl` liest und löscht Zeichen einschließlich des nächsten Neue-Zeile-Zeichens

`ifelse` setzt Texte bedingt in den Ausgabetext ein
`ifelse` erwartet drei oder mehr Argumente. Wenn das erste Argument dieselbe Zeichenkette wie das zweite Argument ist, dann wird das dritte Argument zurückgeliefert. Ist dies nicht der Fall und liegen mehr als vier Argumente vor, wird der Prozeß mit den weiteren Argumenten wiederholt. Andernfalls ist der Wert entweder die vierte Zeichenkette oder Null, wenn es diese nicht gibt.

`incr` gibt den um 1 erhöhten Wert des Arguments zurück. Der Wert des Arguments wird berechnet, indem die Anfangsziffernfolge als Dezimalzahl interpretiert wird.

`decr` gibt den um 1 verringerten Wert des Arguments zurück.

`eval` bewertet das Argument unter Anwendung der 32-Bit-Arithmetik als arithmetischen Ausdruck. Zu den Operatoren gehören `+`, `-`, `*`, `/`, `%`, `**` (Potenzierung), bitweises `&`, `|`, `^` und `~`, Relationen, runde Klammern. Oktal- und Hexadezimalzahlen können wie in C spezifiziert werden. Das zweite Argument liefert die Basis für das Ergebnis; der Standardwert ist 10. Das dritte Argument kann zur Angabe der Mindestanzahl von Ziffern im Ergebnis verwendet werden.

`len` gibt die Anzahl der Zeichen des Arguments zurück.

`index` gibt die Position im ersten Argument zurück, an der das zweite Argument (Nullursprung) beginnt, oder -1, wenn das zweite Argument nicht vorhanden ist.

`substr` gibt eine Teilzeichenkette des ersten Arguments zurück. Das zweite Argument ist die Position des ersten Zeichens (von 0 ab gezählt); das dritte Argument zeigt die Länge der Teilzeichenkette an. Fehlt das dritte Argument, endet die Teilzeichenkette mit der Original-Zeichenkette.

`translit` übersetzt die Zeichen des ersten Arguments, indem die Buchstaben der Menge des zweiten Arguments gegen die Buchstaben der Menge des dritten Arguments ausgetauscht werden. Abkürzungen sind nicht zulässig.

`include` gibt den Inhalt der im Argument angegebenen Datei zurück.

`sinclude` arbeitet wie `include` mit folgendem Unterschied: es gibt keine Meldung aus, wenn nicht auf die Datei zugegriffen werden kann.

`syscmd` führt das SINIX-Systemkommando aus, das im ersten Argument angegeben wird. Ein Wert wird nicht zurückgegeben.

`sysval` ist der Rückgabewert des letzten Aufrufs von `syscmd`.

`maketemp` füllt die Zeichenkette `XXXXX` im Argument mit der aktuellen Prozeßnummer auf.

`m4exit` `m4` wird sofort beendet. Das erste Argument ist, falls angegeben, der Endestatus; der Standard ist 0.

`m4wrap` das erste Argument wird an das endgültige Dateiende zurückgestellt.

`errprint` gibt sein Argument an die Fehler-Ausgabedatei.

`dumpdef` gibt aktuelle Namen und Definitionen für die angegebenen Einheiten aus oder für alle, wenn keine Argumente angegeben werden.

`traceon` schaltet die Ablaufverfolgung für die angegebenen Makros ein
Werden keine Makros angegeben, wird die Ablaufverfolgung für alle Makros, einschließlich eingebauter, eingeschaltet.

`traceoff` schaltet die Ablaufverfolgung global und für angegebene Makros ab. Bei Makros, die explizit mit `traceon` verfolgt werden, kann die Ablaufverfolgung nur durch explizite Aufrufe von `traceoff` abgeschaltet werden.

SIEHE AUCH

`cc`(1).
Kapitel "m4" in "Leitfaden und Werkzeuge für die Programmierung mit C".

make - Gruppen von Dateien verwalten

make [Option...] [Makro-Definition...] [Ziel...]

Bei der modularen Programmierung bestehen Programme meist aus mehreren Dateien. `make` bietet eine Methode zum Aktualisieren solcher Programme.

`make` verwendet ein *makefile*, in dem Sie Ziele festlegen und angeben können, wie ein Ziel von anderen Zielen abhängt. Wurde eine Quelldatei geändert, erzeugt `make` das Programm neu, indem es genau die Teile neu übersetzt, die direkt oder indirekt von der geänderten Datei abhängen.

`make` erstellt das Ziel neu, wenn es älter ist als mindestens eine der Dateien, von denen es abhängt. `make` berücksichtigt die Abhängigkeitsbeziehungen der Dateien untereinander und ermittelt Datum und Uhrzeit der letzten Änderung einer Datei.

Das *makefile* heißt normalerweise `makefile`, `Makefile`, `s.makefile` oder `s.Makefile`. Folgt man dieser Benennungskonvention, können Sie `make` ohne Angabe von Argumenten aufrufen. `make` sucht nach dem *makefile* im aktuellen Dateiverzeichnis und generiert das Ziel neu, wenn mindestens eine Änderung stattgefunden hat.

Die folgenden vier Anweisungen können in einem *makefile* angegeben werden, um die `make`-Optionen zu erweitern:

`.DEFAULT` Wenn eine Datei angelegt werden muß, jedoch keine ausdrücklichen Kommandos oder relevante Standardregeln für die Endungen vorliegen, werden die mit dem Namen `.DEFAULT` in Zusammenhang stehenden Kommandos verwendet.

`.IGNORE` Es tritt dieselbe Wirkung wie bei der Option `-i` auf.

`.PRECIOUS` Von diesem Ziel abhängige Dateien werden nicht gelöscht, falls das Quit- oder Interrupt-Signal empfangen wird.

`.SILENT` Es tritt dieselbe Wirkung wie bei der Option `-s` auf.

Die `make`-Optionen haben folgende Bedeutung:

`-e` überschreibt die Definitionen in *makefiles* durch Umgebungsvariablen

`-f`*makefile* *makefile* wird als Name einer Beschreibungsdatei angesehen.

`-i` ignoriert Fehlermeldungen von Kommandos, die in einer Kommandofolge stehen. `make` setzt die Bearbeitung auch nach Fehlern fort. `.IGNORE` im *makefile* bewirkt dasselbe.

`-k` bricht die Kommandofolge ab, wenn in einem Eintrag des *makefiles* ein Fehler auftritt. Die Bearbeitung wird an anderer Stelle fortgesetzt.

-n	dient als Testhilfe. Alle Kommandos werden ausgegeben, auch die mit vorangestelltem @, sie werden aber nicht ausgeführt.
-p	gibt eine vollständige Liste der Makrodefinitionen und der Beschreibung der Zieldateien aus.
-q	gibt den Status Null zurück, wenn die Zieldatei aktuell ist. Ein Status ungleich Null gibt an, daß die Zieldatei neu erstellt werden muß.
-r	ignoriert die vordefinierten Regeln
-s	unterdrückt die Ausgabe der ausgeführten Kommandos. .SILENT als Ziel in dem *makefile* bewirkt dasselbe
-t	gibt den Zielen des *makefiles* einen neuen Zeitstempel und diese werden als aktuell angenommen. Die Kommandofolge wird nicht ausgeführt.

Erzeugen eines makefiles

Das mit der Option -f angegebene *makefile* ist eine sorgfältig strukturierte Datei mit expliziten Anweisungen zum Aktualisieren von Programmen. Die Datei enthält eine Reihe von Einträgen, die Abhängigkeiten angeben. Die erste Zeile eines Eintrags ist eine durch Leerzeichen getrennte, nicht leere Liste von Zielen, dann folgt ein :, danach eine (möglicherweise leere) Liste der erforderlichen Dateien oder Abhängigkeiten. Auf ein : folgender Text und alle nachfolgenden Zeilen, die mit einem Tabulator beginnen, sind Shell-Kommandos, die zur Aktualisierung des Ziels ausgeführt werden sollen. Die erste nicht leere Zeile, die nicht mit einem Tabulator oder # anfängt, beginnt eine neue Abhängigkeit oder Makrodefinition. Shell-Kommandos können durch die Angabe von <Backslash><Neue-Zeile-Zeichen> über mehrere Zeilen gehen. Alles, was von make ausgegeben wird (mit Ausnahme des ersten Tabulators), geht direkt und unverändert an die Shell. Daher erzeugt

```
echo a\
b
```

die Ausgabe von ab genau wie dies auch mit der Shell erfolgen würde.

Kommentare werden von # und dem Neue-Zeile-Zeichen umschlossen.

Das folgende *makefile* besagt, daß pgm von den beiden Dateien a.o und b.o abhängig ist, die ihrerseits von ihren jeweiligen Quelldateien (a.c und b.c) sowie von der gemeinsamen Datei incl.h abhängen:

```
pgm: a.o b.o
	cc a.o b.o -o pgm
a.o: incl.h a.c
	cc -c a.c
b.o: incl.h b.c
	cc -c b.c
```

Kommandozeilen werden nacheinander jeweils von einer eigenen Shell ausgeführt. Mit der SHELL-Umgebungsvariablen kann die von make zur Ausführung von Kommandos verwendete Shell angegeben werden. Der Standard ist /usr/bin/sh. Das erste bzw. die beiden ersten Zeichen in einem Kommando können wie folgt lauten: -, @, -@ oder @-. Wenn @ vorhanden ist, wird die Ausgabe des Kommandos unterdrückt. Wenn - vorhanden ist, ignoriert make einen Fehler. Eine Zeile wird ausgegeben, wenn sie ausgeführt wird, es sei denn, die Option -s ist vorhanden, oder der Eintrag .SILENT ist in der Datei *makefile* oder die Anfangszeichenkette enthält ein @. Option -n gibt das Kommando aus, ohne es auszuführen; wenn die Kommandozeile jedoch die Zeichenkette $(MAKE) enthält, wird die Zeile immer ausgeführt (siehe Erläuterung von Makro-MAKEFLAGS unter "Umgebung"). Die Option -t aktualisiert das geänderte Datum einer Datei ohne Ausführen von Kommandos.

make wird normalerweise durch Kommandos beendet, die einen Status ungleich Null zurückgeben. Wenn die Option -i vorhanden ist, oder der Eintrag .IGNORE in *makefile* erscheint, oder die erste Zeichenkette des Kommandos - enthält, wird der Endestatus ignoriert. Wenn die Option -k vorhanden ist, wird die Arbeit am aktuellen Eintrag verlassen, jedoch in anderen Verzweigungen, die nicht von diesem Eintrag abhängen, fortgeführt.

Das Ziel wird durch Unterbrechung und Verlassen gelöscht, wenn es nicht von .PRECIOUS abhängt.

Umgebung

Die Umgebung wird von make gelesen. Alle Variablen werden als Makrodefinitionen verstanden und als solche verarbeitet. Die Umgebungsvariablen werden vor jedem *makefile* und unmittelbar nach den vordefinierten Regeln verarbeitet. Daher werden Umgebungsvariablen von Makrozuweisungen in einem *makefile* überschrieben. Die Option -e bewirkt, daß die Umgebung die Makrozuweisungen in einem *makefile* überschreibt. Dateinamen-Zusätze und ihre entsprechenden Regeln in *makefile* überschreiben vordefinierte Regeln für diese Erweiterungen.

Die Umgebungsvariable MAKEFLAGS wird von make verarbeitet, als ob sie zulässige Eingabeoptionen (außer -f und -p) enthält, die für die Kommandozeile gültig sind. Wenn die Variable nicht definiert ist, wird sie automatisch bei Aufruf von make mit den aktuellen Optionen belegt und bei Aufruf der Kommandos weitergereicht. Auf diese Weise enthält MAKEFLAGS stets die aktuellen Eingabeoptionen. Dies erweist sich als sehr nützlich für "super-makes". Tatsächlich wird $(MAKE) bei Verwendung der Option -n auf jeden Fall ausgeführt. Daher kann man ein make -n rekursiv bei einem ganzen Softwaresystem ausführen, um zu sehen, was ausgeführt worden wäre. Dies ist deshalb möglich, weil das -n in MAKEFLAGS eingetragen wird und an weitere Aufrufe von $(MAKE) weitergeleitet wird. Dies ist eine Möglichkeit der Fehlersuche in allen *makefiles* für ein Softwareprojekt, ohne tatsächlich etwas auszuführen.

Include-Dateien

Wenn `include` am Anfang einer Zeile in *makefile* erscheint und danach ein Leerzeichen oder ein Tabulator folgt, wird der Rest der Zeile als Dateiname interpretiert, dessen zugehörige Datei vom aktuellen Aufruf nach Ersetzen von Makros gelesen wird.

Makro-Definitionen

Einträge der Form *Zeichenkette1 = Zeichenkette2* sind Makrodefinitionen. *Zeichenkette2* wird als Folge aller Zeichen bis zu einem Kommentarzeichen oder einem nicht entwerteten Neue-Zeile-Zeichen definiert. Bei jedem anschließenden Auftreten wird *$(Zeichenkette1[:Ersatz1=[Ersatz2]])* durch *Zeichenkette2* ersetzt. Die runden Klammern sind optional, wenn ein Makroname mit einem Zeichen verwendet wird und keine Ersatz-Regel vorhanden ist. Das optionale *:Ersatz1=Ersatz2* ist eine Ersatz-Regel. Bei Angabe dieser Regel wird jedes sich nicht überschneidende Auftreten von *Ersatz1* im angegebenen Makro durch *Ersatz2* ersetzt. Zeichenketten für diese Art von Ersetzung werden durch Leerzeichen, Tabulatorzeichen, Neue-Zeile-Zeichen und Zeilenanfänge begrenzt. Ein Beispiel für die Verwendung der Ersatz-Regeln wird im Abschnitt "Bibliotheken" gezeigt.

Interne Makros

Es gibt insgesamt fünf intern verwaltete Makros, die zum Schreiben von Regeln für den Aufbau von Zielen nützlich sind.

`$*` stellt den Stamm des Dateinamens der aktuellen abhängigen Datei dar, wobei der Zusatz gelöscht ist. Die Auswertung erfolgt nur bei Ausführung der Abhängigkeitsregeln.

`#@` stellt den vollen Zielnamen des aktuellen Ziels dar. Die Auswertung erfolgt nur für ausdrücklich angegebene Abhängigkeiten.

`$<` wird nur bei Anwendung der Abhängigkeitsregeln oder der `.DEFAULT`-Regel ausgewertet. Es ist das Modul, das in Bezug auf das Ziel veraltet ist, d.h. der "angefertigte" abhängige Dateiname. Auf diese Weise würde Makro `$<` in der Regel `.c.o` als `.c`-Datei ausgewertet werden. Ein Beispiel für die Erstellung von optimierten `.o`-Dateien aus `.c`-Dateien ist:

```
.c.o:
        cc -c -O $*.c
```

oder:

```
.c.o:
        cc -c -O $<
```

`$?` wird ausgewertet, wenn die expliziten Regeln des *makefiles* ausgewertet werden. Es ist die Liste der Vorbedingungen, die in Hinblick auf das Ziel veraltet sind.

`$%` wird nur ausgewertet, wenn das Ziel ein Teil einer Archivbibliothek in der Form `lib(datei.o)` ist. In diesem Fall wird `$@` als `lib` ausgewertet, und `$%` als die Bibliotheksdatei `datei.o`.

Vier der fünf Makros können alternative Formen aufweisen. Wenn an einen der vier Makros ein Großbuchstabe `D` oder `F` angehängt wird, ändert sich die Bedeutung in 'Dateiverzeichnis-Teil' für `D` und 'Dateiteil' für `F`. Daher bezieht sich `$(@D)` auf den Dateiverzeichnis-Teil der Zeichenkette `$@`. Ist kein Dateiverzeichnis-Teil vorhanden, wird `./` erstellt. Als einziges Makro ist `$?` von dieser Alternativform ausgeschlossen.

Dateinamen-Zusätze (Suffixe)

Bestimmte Namen, wie z. B. solche, die auf `.o` enden, haben ableitbare Vorbedingungen wie z. B. `.c`, `.s` usw. Wenn für eine solche Datei keine Aktualisierungskommandos in *makefile* erscheinen und wenn eine ableitbare Vorbedingung vorhanden ist, wird diese Vorbedingung übersetzt, um das Ziel zu erzeugen. Für diesen Fall hat `make` Abhängigkeitsregeln, die durch Prüfung der Suffixe und Bestimmung einer zur Anwendung geeigneten Abhängigkeitsregel den Aufbau von Dateien aus anderen Dateien ermöglichen. Die aktuellen Standard-Abhängigkeitsregeln sind:

```
.c      .c~     .f      .f~     .s      .s~     .sh     .sh~    .C      .C~
.c.a    .c.o    .c~.a   .c~.c   .c~.o   .f.a    .f.o    .f~.a   .f~.f   .f~.o
.h~.h   .l.c    .l.o    .l~.c   .l~.l   .l~.o   .s.a    .s.o    .s~.a   .s~.o
.s~.s   .sh~.sh .y.c    .y.o    .y~.c   .y~.o   .y~.y   .C.a    .C.o    .C~.a
.C~.C   .C~.o   .L.C    .L.o    .L~.C   .L~.L   .L~.o   .Y.C    .Y.o    .Y~.C
.Y~.o   .Y~.Y
```

Die internen Regeln für `make` sind in der Quelldatei `rules.c` für das Programm `make` enthalten. Diese Regeln können lokal verändert werden. Um die in `make` eingebauten Regeln auf einem beliebigen Gerät in einer für eine Neuübersetzung geeigneten Form auszugeben, wird nachstehendes Kommando verwendet:

```
make -pf - 2>/dev/null </dev/null
```

Eine Tilde in den obigen Regeln bezieht sich auf eine SCCS-Datei (siehe `sccsfile`(4)). Daher würde die Regel `.c~.o` eine SCCS-C-Quelldatei in eine Objektdatei (`.o`) umwandeln. Da das `s.` der SCCS-Dateien ein Dateiname-Präfix ist, ist es nicht mit dem Begriff von Suffix im Sinn von `make` vereinbar. Daher stellt die Tilde eine Möglichkeit zur Änderung eines Dateiverweises in einen SCCS-Dateiverweis dar.

Eine Regel mit nur einem Zusatz (z.B. `.c:`) ist die Definition dafür, wie man x aus x`.c` aufbaut. Tatsächlich ist das andere Suffix leer. Dies ist nützlich für den Aufbau von Zielen aus nur einer Quelldatei (z. B. Shell-Prozeduren, einfache C-Programme).

Zusätzliche Suffixe können als Abhängigkeitsliste für `.SUFFIXES` definiert werden. Die Reihenfolge ist von Bedeutung; der erste mögliche Name, für den eine Datei und eine Regel vorhanden sind, wird als Vorbedingung abgeleitet. Die Standardliste lautet:

```
.SUFFIXES .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~ .C .C~ .Y .Y~ .L .L~
```

Auch in diesem Fall wird das obige Kommando zur Ausgabe der internen Regeln die Liste der auf der aktuellen Maschine implementierten Zusätze anzeigen. Mehrfach-Suffix-Listen sind akkumulierend; `.SUFFIXES:` ohne Abhängigkeiten löscht die Suffix-Liste.

Abhängigkeitsregeln

Das oben angegebene Beispiel läßt sich kürzer formulieren:

```
pgm: a.o b.o
     cc a.o b.o -o pgm
a.o b.o: incl.h
```

Dies ergibt sich daraus, daß `make` einen Satz interner Regeln zum Aufbauen von Dateien hat. Der Benutzer kann zu dieser Liste Regeln hinzufügen, indem diese einfach in die Datei *makefile* geschrieben werden.

Die Standard-Abhängigkeitsregeln verwenden gewisse Makros, um die Aufnahme von optionalen Teilen in die erzeugte Kommandofolge zu ermöglichen. Zum Beispiel werden `CFLAGS`, `LFLAGS` und `YFLAGS` für Übersetzeroptionen von `cc`(1), `lex`(1) bzw. `yacc`(1) verwendet. Auch in diesem Fall wird die vorherige Methode zum Prüfen der aktuellen Regeln empfohlen.

Die Ableitung von Vorbedingungen kann gesteuert werden. Die Regel zur Erstellung einer Datei mit dem Suffix `.o` aus einer Datei mit dem Suffix `.c` ist als Eintrag mit `.c.o:` als Ziel und ohne Abhängigkeiten spezifiziert. Mit dem Ziel in Zusammenhang stehende Shell-Kommandos definieren die Regel zur Erstellung einer `.o`-Datei aus einer `.c`-Datei. Ein Ziel, das keine Schrägstriche aufweist und mit einem Punkt beginnt, wird als Regel angesehen und nicht als echtes Ziel.

Bibliotheken

Wenn eine Ziel- oder eine Abhängigkeitsbezeichnung Klammern enthält, wird sie als Archivbibliothek angesehen, wobei die in Klammern stehende Zeichenkette auf einen Eintrag in der Bibliothek verweist. So verweisen `lib(datei.o)` und `$(LIB)(datei.o)` auf eine Archivbibliothek, die `datei.o` enthält. Hierbei wird davon ausgegangen, daß Makro `LIB` vorher definiert wurde. Der Ausdruck `$(LIB)(datei1.o datei2.o)` ist nicht zulässig. Zu Archivbibliotheken gehörende Regeln weisen die Form .*XX*`.a` auf, wobei *XX* das Suffix der Datei ist, aus der der Archiveintrag erstellt werden soll. Leider ist es bei der gegenwärtigen Implementierung erforderlich, daß *XX* sich vom Suffix des Archiveintrags unterscheiden muß. Daher kann man nicht angeben, daß `lib(datei.o)` von `datei.o` abhängig ist.

Es folgt die Beschreibung der häufigsten Anwendung der Archivschnittstelle. Hier soll davon ausgegangen werden, daß alle Quelldateien C-Quellen sind:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
       @echo lib is now up-to-date
.c.a:
      $(CC) -c $(CFLAGS) $<
      $(AR) $(ARFLAGS) $@ $*.o
      rm -f $*.o
```

Tatsächlich ist diese Regel `.c.a` in `make` vordefiniert und hier überflüssig. Ein interessanteres, allerdings stärker begrenztes Beispiel für eine Archivdatei-Wartung:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      $(CC) -c $(CFLAGS) $(?:.o=.c)
      $(AR) $(ARFLAGS) lib $?
      rm $?
      @echo lib is now up-to-date
.c.a::
```

In diesem Fall wird der Makro-Ersetzungsmechanismus verwendet. Die Liste `$?` ist als der Satz von Objektdateinamen definiert (innerhalb von `lib`), deren entsprechende C-Quelldateien veraltet sind. Der Makro-Ersetzungsmechanismus ersetzt `.o` durch `.c`. Leider kann eine Umwandlung in `.c` noch nicht vorgenommen werden, könnte aber in Zukunft implementiert werden. Auch ist die Abschaltung der Regel `.c.a:` zu beachten, die jede Objektdatei der Reihe nach erstellt hätte. Diese besondere Konstruktion beschleunigt die Wartung der Archivbibliothek beträchtlich. Sie wird recht umständlich, wenn die Archivbibliothek eine Mischung von Assembler- und C-Programmen enthält.

DATEIEN

`[Mm]akefile` und `s.[Mm]akefile`
`/usr/bin/sh`

SIEHE AUCH

`cc`(1), `lex`(1), `yacc`(1), `printf`(3S), `sccsfile`(4).
`cd`(1), `sh`(1) in "SINIX V5.41 Kommandos".
Kapitel "make" in "CES V5.41 Leitfaden und Werkzeuge für die Programmierung mit C".

HINWEIS

Einige Kommandos geben ungerechtfertigt einen Status ungleich Null zurück; zur Lösung dieses Problems verwendet man `-i` oder die Option - in der Kommandozeile.

Dateinamen mit den Zeichen `= : @` können nicht verarbeitet werden. Kommandos, die direkt von der Shell ausgeführt werden, insbesondere `cd`(1), sind in `make` über Neue-Zeile-Zeichen hinweg wirkungslos. Die Syntax `lib(datei1.o datei2.o datei3.o)` ist unzulässig. Es ist nicht möglich, `lib(datei.o)` aus `datei.o` aufzubauen.

mcs - Kommentarteil einer Objektdatei verwalten

mcs [-a Zeichenkette] [-c] [-d] [-n Name] [-p] [-V] Datei...

Das Kommando `mcs` (manipulate comment section) wird dazu benutzt, einen Abschnitt, standardmäßig den `.comment`-Abschnitt, in einer ELF-Objektdatei zu verwalten. Das Kommando wird zum Hinzufügen, Löschen, Ausdrucken und Komprimieren eines Abschnittsinhalts in einer ELF-Objektdatei und nur zum Ausdrucken des Abschnittsinhalts in einer COFF-Objektdatei verwendet. `mcs` muß eine oder mehrere der nachstehend beschriebenen Optionen erhalten. Das Kommando wendet jede der Optionen der Reihe nach auf die Objektdateien an.

Folgende Optionen stehen zur Verfügung:

`-a` *Zeichenkette*
: *Zeichenkette* an den Kommentarteil der ELF-Objektdateien anhängen
Wenn die *Zeichenkette* Leerzeichen enthält, muß sie in Anführungszeichen eingeschlossen sein.

`-c`
: Inhalt des Kommentarteils der ELF-Objektdateien komprimieren
Alle doppelten Einträge werden entfernt. Die Reihenfolge der übrigen Einträge wird nicht geändert.

`-d`
: Inhalt des Kommentarteils aus den ELF-Objektdateien löschen
Auch der Abschnittskopf des Kommentarteils der Objektdatei wird entfernt.

`-n` *Name*
: *Name* des Kommentarteils angeben, auf den zugegriffen werden soll
Standardmäßig bearbeitet `mcs` den Abschnitt, der mit `.comment` bezeichnet ist.

`-p`
: Inhalt des Kommentarteils auf der Standardausgabe ausgeben
Jeder ausgedruckte Abschnitt wird mit dem Namen jener Datei versehen, aus der er stammt; dabei wird das Format *Dateiname*`[`*Mitglieds_name*`]:` für Archivdateien und *Dateiname*`:` für andere Dateien benutzt.

`-V`
: Versionsnummer von `mcs` auf der Standardfehlerausgabe ausdrucken

Ist die Eingabedatei ein Archiv (siehe `ar`(4)), wird das Archiv als eine Reihe individueller Dateien behandelt. Wenn beispielsweise die Option `-a` angegeben ist, wird die Zeichenkette an den Kommentarteil jeder ELF-Objektdatei im Archiv angehängt; ist das Archivmitglied keine ELF-Objektdatei, bleibt sie unverändert.

Wird `mcs` auf einer Archivdatei durchgeführt, wird die Symboltabelle des Archivs entfernt, außer wenn nur die Option `-p` angegeben wurde. Bevor das Archiv durch das Kommando `ld` verknüpft werden kann, muß die Symboltabelle des Archivs wiederhergestellt werden, indem das Kommando `ar` mit der Option `-s` ausgeführt wird. `mcs` erzeugt entsprechende Warnmeldungen, wenn diese Situation eintritt.

BEISPIELE

```
mcs -p Datei                    # Kommentarteil der Datei ausgeben
mcs -a Zeichenkette Datei       # Zeichenkette an den Kommentarteil der Datei anhängen
```

DATEIEN

TMPDIR/mcs*	temporäre Dateien
TMPDIR	üblicherweise /var/tmp, kann aber durch die Umgebungsvariable TMPDIR neu definiert werden (siehe tempnam in tmpnam(3S)).

SIEHE AUCH

ar(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4).

HINWEIS

mcs kann den Inhalt eines Abschnitts, der in einem Segment enthalten ist, weder hinzufügen noch löschen oder komprimieren.

nm - Symboltabelle einer Objektdatei ausgeben

```
nm [Option...] Datei...
```

Das Kommando nm (name list) zeigt die Symboltabelle jeder ELF- oder COFF-Objektdatei an, die durch *Datei* angegeben ist. Die Datei kann eine verschiebbare oder eine absolute ELF- oder COFF-Objektdatei sein; oder es kann sich auch um ein Archiv von verschiebbaren oder absoluten ELF- oder COFF-Objektdateien handeln. Für jedes Symbol werden folgende Informationen ausgegeben:

`Index` Index des Symbols, erscheint in Klammern

`Value` Wert des Symbols ist der Offset eines definierten Symbols in einer verschiebbaren Datei; es gibt Beschränkungen für die Ausrichtung von Symbolen mit einem Sektionsindex gleich SHN_COMMON sowie einer virtuellen Adresse in ausführbaren und dynamischen Bibliotheksdateien.

`Size` Größe in Bytes des assoziierten Objekts

`Type` Typ des Symbols

NOTYPE	es wurde kein Typ angegeben
OBJECT	ein Datenobjekt, z.B. Feld oder Variable
FUNC	eine Funktion oder anderer ausführbarer Code
SECTION	ein Symbol für einen Abschnitt
FILE	Name der Quelldatei

`Bind` Attribute des Gültigkeitsbereichs eines Symbols
Bei LOCAL-Symbolen ist der Gültigkeitsbereich auf die Objektdatei, die ihre Defintion enthält, beschränkt. GLOBAL-Symbole sind in allen kombinierten Objektdateien sichtbar. WEAK-Symbole sind im Grunde genommen globale Symbole, die aber einen geringeren Vorrang als GLOBAL-Symbole haben.

`Other` Feld, das für zukünftige Verwendung reserviert ist, zur Zeit enthält es 0.

`Shndx` Mit Ausnahme von drei speziellen Werten, ist dies der Index in der Abschnittskopftabelle, in der das Symbol definiert ist. Die folgenden speziellen Werte existieren: ABS gibt an, daß sich der Symbolwert bei einer Relokation nicht ändert; COMMON gibt einen freien Block und den Wert für eine Ausrichtung an. UNDEF gibt ein undefiniertes Symbol an.

`Name` Name des Symbols

Die Ausgabe von nm kann mit folgenden Optionen gesteuert werden:

-o Wert und Größe eines Symbols in Oktal- statt in Dezimalform ausgeben

-x Wert und Größe eines Symbols in Hexadezimal- statt in Dezimalform ausgeben

-h Daten des Ausgabekopfs nicht anzeigen

-v externe Symbole vor der Ausgabe nach ihrem Wert sortieren

-n externe Symbole vor der Ausgabe nach ihren Namen sortieren

-e siehe Abschnitt HINWEIS

-f siehe Abschnitt HINWEIS

-u nur undefinierte Symbole ausgeben

-r Name der Objektdatei oder des Archivs vor jede Ausgabezeile stellen

-p leicht analysierbare kurze Ausgabe erstellen
Jeder Symbolbezeichnung wird ihr Wert (Leerzeichen, wenn undefiniert) und einer der Buchstaben U (undefiniert), N (Symbol hat keinen Typ), D (Datenobjekt-Symbol), T (Textobjekt-Symbol), S (Segmentsymbol) oder F (Dateisymbol) vorangestellt. Hat das Symbol das Attribut lokal (nicht extern), wird der Buchstabe für den Typ kleingeschrieben. Hat das Symbol das Attribut WEAK, so wird der Buchstabe groß geschrieben. Wird zusätzlich die Option -l angegeben, so folgt dem Buchstaben ein *. Der Buchstabe wird groß ausgegeben, wenn das Symbol das Attribut GLOBAL hat.

-l Damit zwischen WEAK- und GLOBAL- Attributen der Symbole unterschieden werden kann, wird ein * an den Buchstaben bei WEAK-Symbolen angehängt.

-V Die Version des Kommandos nm wird auf der Standard-Fehlerausgabe ausgeben.

-T siehe Abschnitt HINWEIS

Die Optionen können in beliebiger Reihenfolge einzeln oder in Kombinationen verwendet werden. Wenn Optionen angegeben werden, die sich widersprechen (wie z.B. nm -v -n), wird die erste Option verwendet und die zweite ignoriert. Es wird eine Warnung ausgegeben, daß die zweite Option ignoriert wurde.

SIEHE AUCH

cc(1), dump(1), ld(1), a.out(4), ar(4).

HINWEIS

Die folgenden Optionen sind aufgrund von Änderungen beim Objektformat überflüssig geworden. Sie werden in zukünftigen Versionen nicht mehr erscheinen.

-e ausschließliche Ausgabe von externen und statischen Symbolen Automatische Symbole erscheinen nicht mehr in der Symboltabelle. Sie stehen in den Informationen für den Debugger, die mit dem Kommando cc -g erzeugt werden. Diese können dann mit dump(1) angesehen werden.

-f erzeugt eine vollständige Ausgabe. Redundante Symbole (wie z.B. .text, .data, etc) existieren nicht mehr und eine vollständige Ausgabe entspricht der Standardausgabe.

-T Standardmäßig gibt nm den vollständigen Namen der angezeigten Symbole aus. Da die Symbolnamen jetzt in der letzten Spalte ausgegeben werden, ist das Problem der zu langen Bezeichner verschwunden und es ist nicht länger notwendig, den Symbolnamen zu kürzen.

prof - Ausführungsprofil ausgeben

prof [-t | c | a | n] [-o | x] [-g | l] [-z] [-h] [-s] [-m Meßdatei] [-V] [Prog]

Das Kommando `prof` (profile) interpretiert eine Datei mit Meßwerten, die von der Funktion `monitor` angelegt wird. Die Symboltabelle in der Objektdatei *Prog* (standardmäßig `a.out`) wird gelesen und in Beziehung gesetzt zu einer entsprechenden Datei mit Meßwerten, standardmäßig `mon.out`. Für jedes externe Textsymbol wird der prozentuale Wert der Zeit, die für die Ausführung zwischen der Adresse dieses Symbols und der Adresse des nächsten Symbols benötigt wurde, sowie die Häufigkeit, mit der diese Funktion aufgerufen wurde, und die durchschnittliche Dauer je Aufruf in Millisekunden ausgegeben.

Die sich gegenseitig ausschließenden Optionen `-t`, `-c`, `-a` und `-n` legen die Sortierung der Ausgabezeilen fest:

`-t` nach abnehmendem Prozentanteil der Gesamtzeit sortieren (Standard)

`-c` nach abnehmender Anzahl der Aufrufe sortieren

`-a` nach zunehmender Symboladresse sortieren

`-n` lexikalisch nach Symbolnamen sortieren

Die sich gegenseitig ausschließenden Optionen `-o` und `-x` legen die Form der Ausgabe der Adresse jedes überwachten Symbols fest:

`-o` jede Symboladresse in Oktalform zusammen mit dem Symbolnamen ausgeben

`-x` jede Symboladresse in Hexadezimalform zusammen mit dem Symbolnamen ausgeben

Die sich gegenseitig ausschließenden Optionen `-g` und `-l` steuern die Typen der Symbole, über die Bericht erstattet werden soll. Die Option `-l` muß mit Vorsicht verwendet werden; sie schlägt die Zeit, die in einer Funktion der Speicherklasse `static` verbracht wird, der im Speicher vorhergehenden globalen Funktion zu, anstatt der Funktion der Speicherklasse `static` im Bericht einen getrennten Eintrag zu geben. Wenn alle Funktionen der Speicherklasse `static` an der richtigen Stelle untergebracht sind, kann diese Option sehr sinnvoll sein (siehe Beispiel). Wenn das aber nicht der Fall ist, kann der Bericht irreführend werden.

Nehmen Sie an, daß `A` und `B` globale Funktionen sind und nur `A` die Funktion `S` der Speicherklasse `static` aufruft. Wenn `S` im Quellcode unmittelbar hinter `A` steht (das heißt, wenn `S` an der richtigen Stelle liegt), dann kann mit der Option `-l` die Zeit, die in `A` verbracht wurde, einschließlich der Zeit, die in `S` verbracht wurde, leicht bestimmt werden. Wenn jedoch beide, `A` und `B`, `S` aufrufen und die Option `-l` verwendet wird, wird der Bericht irreführend. Die von `S` beim Aufruf von `B` verbrachte Zeit wird zu `A` mitge-

rechnet, und dann scheint es so, als ob in A mehr Zeit verbracht wurde, als es in Wirklichkeit der Fall war. In diesem Fall kann die Funktion S gar nicht an einer richtigen Stelle untergebracht werden.

-g lokale Funktionen der Speicherklasse static einschließen

-l lokale Funktionen der Speicherklasse static nicht einschließen (Standard)

Die folgenden Optionen können in beliebiger Kombination verwendet werden:

-z alle Symbole im Meßbereich einschließen, selbst wenn diese nicht aufgerufen werden und keine Zeit verbrauchen

-h die Überschrift unterdrücken, die normalerweise für die Aufstellung ausgegeben wird. Dies ist nützlich, wenn die Liste weiter verarbeitet werden soll.

-s eine Zusammenfassung mehrerer Überwachungsparameter und -Statistiken auf der Standard-Fehlerausgabe ausgeben

-m *Meßdatei*
Meßdatei anstelle von mon.out als Eingabedatei mit Meßwerten verwenden

-V Versionsinformationen von prof auf der Standard-Fehlerausgabe ausgeben

Ein Programm erstellt eine Datei mit Meßwerten, wenn es mit der Option -p von cc gebunden wurde. Diese Option für das Kommando cc sorgt für den Aufruf von monitor am Anfang und Ende der Ausführung. Mit dem Aufruf von monitor am Ende der Ausführung wird das Schreiben der Meßwert-Datei ausgelöst. Die Anzahl der Aufrufe einer Funktion wird gezählt, wenn die Option -p bei der Übersetzung der Datei verwendet wurde, die diese Funktion enthält.

Der Name der Datei, die von einem profilierten Programm erstellt wird, wird von der Umgebungsvariablen PROFDIR gesteuert. Wenn PROFDIR nicht gesetzt ist, wird mon.out in dem Dateiverzeichnis erstellt, das bei Programmende aktuell ist. PROFDIR=*Zeichenkette* erstellt *Zeichenkette/pid.Progname*, wobei *Progname* aus argv[0] durch Entfernen eines eventuellen Pfadpräfixes entsteht und *pid* die Prozeßnummer des Programms ist. Wenn PROFDIR gesetzt, aber Null ist, wird keine Meßwert-Ausgabe erstellt.

Eine einzelne Funktion kann für die Profilierung mit Hilfe des Makros MARK (siehe prof(5)) in Unterfunktionen unterteilt werden.

DATEIEN

`mon.out`	für Ausführungsprofil
`a.out`	Standard-Objektdatei für die Symboltabelle

SIEHE AUCH

`cc`(1), `exit`(2), `profil`(2), `monitor`(3C), `prof`(5).

HINWEIS

Die für aufeinanderfolgende identische Abläufe gemeldeten Zeiten können voneinander abweichen. Dies ist auf die variierenden Cache-Trefferraten infolge der Mitbenutzung des Cache-Speichers durch andere Prozesse zurückzuführen. Selbst wenn es den Anschein hat, daß ein Programm den Rechner allein benutzt, können dennoch Hintergrund- oder asynchrone Prozesse die Daten verwischen. In seltenen Fällen kann es zu Konflikten zwischen dem Taktimpuls, der die Aufzeichnung des Programmzählers steuert, und Schleifen im Programm kommen, wodurch die Messungen stark verzerrt werden. Aufrufe werden jedoch immer genau gezählt.

Nur für Programme, die `exit` aufrufen, oder von `main` zurückkehren, wird garantiert, daß sie eine Datei mit Meßwerten erzeugen, es sei denn am Ende des Programms wird explizit `monitor` aufgerufen.

Die Zeiten für Funktionen der Speicherklasse `static` werden dem vorhergehenden externen Text-Symbol zugeschrieben, wenn die Option `-g` nicht verwendet wird. Jedoch werden die Aufrufe für die vorhergehende Funktion dennoch richtig gezählt, d.h. die Aufrufe von Funktionen der Speicherklasse `static` werden nicht zu den Aufrufen der externen Funktion hinzugefügt.

Wenn mehr als eine der Optionen `-t`, `-c`, `-a` und `-n` angegeben wird, wird die zuletzt angegebene Option verwendet, und der Benutzer wird gewarnt.

Die Erstellung von Profilen kann mit dynamisch gebundenen ausführbaren Dateien verwendet werden, aber dann ist Vorsicht angebracht. Zur Zeit können von gemeinsam benutzten Objekten keine Profile mit `prof` erstellt werden. Wenn `prof` also mit einem dynamisch gebundenen Programm verwendet wird, wird bei der Ausführung nur der 'Haupt'-Teil des Abbilds ausgewertet. Das heißt, daß die ganze Zeit, die außerhalb des 'Haupt'-Objekts, also in einem gemeinsam benutzten Objekt verbracht wird, nicht bei der Meßwert-Ausgabe erscheint; die für das Programm angegebene Gesamtzeit kann geringer sein als die Zeit, die das Programm wirklich verbrauchte.

Da die Zeit, die in einem gemeinsam benutzten Objekt verbracht wird, nicht mitgerechnet werden kann, sollten Sie weniger gemeinsam benutzte Objekte verwenden, wenn die Meßwert-Erstellung eines Programms mit `prof` erfolgt. Wenn möglich, sollte das Programm vor der Profil-Erstellung statisch gebunden werden.

Betrachten wir einen Extremfall:
Ein Programm mit Profileinstellung, das dynamisch mit der gemeinsam benutzten C-Bibliothek gebunden wurde, verbringt 100 Zeiteinheiten in einer `libc`-Routine, z.B. `malloc`. Nehmen wir an, daß `malloc` nur von der Routine `B` aufgerufen wird und daß `B` nur eine Zeiteinheit benötigt. Nehmen wir weiter an, daß Routine `A` 10 Zeiteinheiten braucht, mehr als alle anderen Routinen in dem profilierten 'Haupt'-Teil des Abbilds. In diesem Fall wird `prof` zu dem Schluß gelangen, daß die meiste Zeit in `A` und gar keine Zeit in `B` verbracht wird. Daraus ist es dann fast unmöglich, festzustellen, daß die größte Verbesserung in Routine `B` und nicht in Routine `A` erreicht wurde. Der Wert des Profil-Erzeugers ist in diesem Fall stark gesunken. Verwenden Sie bei der Erstellung von Profilen möglichst viele Archive.

prs - SCCS-Datei ausgeben

prs [-d[Datenspez]] [-r[SID]] [-e] [-l] [-c[Datum-Zeit]] [-a] Datei...

`prs` (print SCCS-file) gibt Teile einer oder eine gesamte SCCS-Datei in einem vom Benutzer angegebenen Format auf der Standardausgabe aus (siehe `sccsfile`(4)). Wird ein Dateiverzeichnis angegeben, gibt `prs` die Dateien in dem Verzeichnis aus, Nicht-SCCS-Dateien (letzte Komponente des Pfadnamens beginnt nicht mit `s.`) und unlesbare Dateien werden dabei nicht berücksichtigt. Bei Angabe von - wird von Standardeingabe gelesen; jede Zeile der Standardeingabe wird als Name einer SCCS-Datei oder eines Dateiverzeichnisses interpretiert. `prs` ignoriert kommentarlos Nicht-SCCS-Dateien und unlesbare Dateien.

Argumente für `prs`, die in beliebiger Reihenfolge erscheinen können, bestehen aus Optionen und Dateinamen.

Alle beschriebenen Optionen gelten unabhängig für jede angegebene Datei:

`-d[`*Datenspez*`]`
: gibt das Ausgabeformat an. *Datenspez* ist eine Zeichenkette, die aus Daten-Schlüsselwörtern der SCCS-Datei besteht (siehe Abschnitt "Daten-Schlüsselwörter"), zwischen denen ein optionaler, vom Benutzer definierter Text steht.

`-r[`*SID*`]`
: wird zur Angabe der Zeichenkette für die SCCS-Identifikation (SID) eines Deltas verwendet, über das Informationen angefordert werden. Standard ist das neueste Delta.

`-e`
: fordert Informationen über alle Deltas an, die vor dem Delta erstellt wurden, das durch die Option `-r` angegeben wird, oder vor dem Datum, das durch die Option `-c` angegeben wird, einschließlich des angegebenen Deltas.

`-l`
: fordert Informationen über alle Deltas an, die seit dem Delta erstellt wurden, das durch die Option `-r` angegeben wird, oder seit dem Datum, das durch die Option `-c` angegeben wird, einschließlich des angegebenen Deltas.

`-c`[*Datum-Zeit*]
: Das Stichdatum/Uhrzeit hat das Format:

  JJ[MM[TT[HH[MM[SS]]]]]

  Für nicht angegebene Einheiten gelten standardmäßig die zulässigen Höchstwerte, z.B. ist `-c7502` gleichwertig mit `-c750228235959`. Eine beliebige Anzahl nicht-numerischer Zeichen kann die verschiedenen Felder des Stichdatums trennen; z.B. "`-c77/2/2 9:22:25`".

`-a`
: bewirkt die Ausgabe von Informationen über gelöschte Deltas, d.h. Deltatyp = R (siehe `rmdel`(1)), und vorhandene Deltas, d.h. Deltatyp = D. Wird die Option `-a` nicht angegeben, werden Informationen nur für die vorhandenen Deltas geliefert.

Daten-Schlüsselwörter

Daten-Schlüsselwörter geben an, welcher Teil einer SCCS-Datei extrahiert und ausgegeben werden soll. Alle Teile einer SCCS-Datei (siehe `sccsfile`(4)) haben ein zugehöriges Daten-Schlüsselwort. Ein Daten-Schlüsselwort darf beliebig oft in *Datenspez* erscheinen.

Die von `prs` ausgegebenen Informationen bestehen aus dem vom Benutzer gelieferten Text und den entsprechenden aus der SCCS-Datei geholten Werten, die die erkannten Daten-Schlüsselwörter in der Reihenfolge des Auftretens in *Datenspez* ersetzen. Das Format des Wertes eines Daten-Schlüsselworts ist entweder 'einfach' (S), wobei die Schlüsselwort-Ersetzung direkt erfolgt, oder 'mehrzeilig' (M), wobei nach der Schlüsselwort-Ersetzung ein RETURN folgt.

Vom Benutzer gelieferter Text ist jeder Text, der nicht als Daten-Schlüsselwort erkannt wird. Ein Tabulator wird mit `\t` und das Zeichen für eine neue Zeile mit `\n` angegeben. Die Standard-Daten-Schlüsselwörter sind:

S.-Wort	Beschreibung	D.-Abschn.	Wert	Form.
:Dt:	Delta-Daten	Delta-Tab.	Siehe unten*	S
:DL:	Delta-Zeilenstatistik	"	:Li:/:Ld:/:Lu:	S
:Li:	Von Delta eingefügte Zeilen	"	*nnnnn*	S
:Ld:	Von Delta gelöschte Zeilen	"	*nnnnn*	S
:Lu:	Von Delta unveränderte Zeilen	"	*nnnnn*	S
:DT:	Deltatyp	"	D *oder* R	S
:I:	SCCS ID Zeichenkette (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Versions(Release)-Nummer	"	*nnnn*	S
:L:	Level-Nummer	"	*nnnn*	S
:B:	Verzweigungsnummer	"	*nnnn*	S
:S:	Folgenummer	"	*nnnn*	S
:D:	Erstellungsdatum von Delta	"	:Dy:/:Dm:/:Dt:	S
:Dy:	Erstellungsjahr von Delta	"	*nn*	S
:Dm:	Erstellungsmonat von Delta	"	*nn*	S
:Dd:	Erstellungstag von Delta	"	*nn*	S
:T:	Erstellungsuhrzeit von Delta	"	:Th:.:Tm:.:Ts:	S
:Th:	Stunde der Erstellung von Delta	"	*nn*	S
:Tm:	Minute der Erstellung von Delta	"	*nn*	S
:Ts:	Sekunde der Erstellung von Delta	"	*nn*	S
:P:	Programmierer, der Delta erstellt hat	"	logname	S
:DS:	Lfd. Deltanummer	"	*nnnn*	S
:DP:	Deltanummer des Vorgängers	"	*nnnn*	S
:DI:	Lfd. Nr. von Deltas inkl., exkl., ignoriert	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Eingeschlossene Deltas (Lfd. Nr.)	"	:DS: :DS:...	S
:Dx:	Ausgeschlossene Deltas (Lfd. Nr.)	"	:DS: :DS:...	S
:Dg:	Ignorierte Deltas (Lfd. Nr.)	"	:DS: :DS:...	S
:MR:	Änderungsanforderungs-Nr. für Delta	"	*Text*	M
:C:	Kommentare für Delta	"	*Text*	M
:UN:	Benutzernamen	Benutzernamen	*Text*	M
:FL:	Optionsliste	Optionen	*Text*	M
:Y:	Modultyp-Option	"	*Text*	S
:MF:	Validierungsanzeiger für Änderungsanforderung	"	ja oder nein	S
:MP:	MR-Validierungsprogramm-Name	"	*text*	S
:KF:	Schlüsselwortfehler/-Warnanzeige	"	ja oder nein	S
:KV:	Schlüsselwort-Validierungs-zeichenkette	"	*Text*	S
:BF:	Verzweigungsanzeiger	"	ja oder nein	S
:J:	Anzeiger für gemeinsames Editieren	"	ja oder nein	S
:Q:	benutzerdefiniertes Schlüsselwort	"	*Text*	S
:M:	Modulname	"	*Text*	S
:FB:	Untergrenze	"	:R:	S
:CB:	Obergrenze	"	:R:	S
:Ds:	Standard-SID	"	:I:	S
:ND:	Nulldelta-Option	"	ja oder nein	S
:FD:	Dateibeschreibungstext	Kommentare	*Text*	M
:BD:	Hauptteil	Hauptteil	*Text*	M
:GB:	Erhaltener Hauptteil	"	*Text*	M
:W:	Eine Form der what(1)-Zeichenkette	N/A	:Z::M:\t:I:	S
:A:	Eine Form der what(1)-Zeichenkette	N/A	:Z::Y: :M: :I::Z:	S
:Z:	what(1) Zeichenkettenbegrenzer	N/A	@(#)	S
:F:	SCCS-Dateiname	N/A	Text	S
:PN:	SCCS-Datei-Pfadname	N/A	Text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

BEISPIELE

Das Kommando

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

kann auf der Standardausgabe folgendes erzeugen:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

Das Kommando

```
prs -d"Newest delta for pgm :m:: :I: Created :D: By :P:" -r s.file
```

kann auf der Standardausgabe folgendes erzeugen:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

Der Standardfall:

```
prs s.file
```

erzeugt auf der Standardausgabe für jeden Deltatabellen-Eintrag des Typs 'D':

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
b178-12345
b179-54321
COMMENTS:
this is the comment line for s.file initial delta
```

Die einzige Option, die mit dem 'Sonderfall' verwendet werden darf, ist die Option `-a`.

DATEIEN

`/var/tmp/pr?????`

SIEHE AUCH

`admin`(1), `delta`(1), `get`(1), `help`(1), `sccsfile`(4).

regcmp - Übersetzen von regulären Ausdrücken

regcmp [-] Datei...

Das Kommando regcmp (regular expression compile) übersetzt die regulären Ausdrücke aus *Datei* und schreibt die Ausgabe nach *Datei*.i. Wenn die Option - verwendet wird, wird die Ausgabe nach *Datei*.c geschrieben. Das Format der Einträge aus *Datei* besteht aus einem Namen (einer C-Variablen), einem oder mehreren Leerzeichen und einem oder mehreren regulären Ausdrücken in Anführungszeichen. regcmp generiert C-Quelltext. Übersetzte reguläre Ausdrücke werden als Vektoren vom Typ extern char dargestellt. *Datei*.i-Dateien können über #include in C-Programme eingebunden werden; *Datei*.c-Dateien können übersetzt und später geladen werden. Verwendet ein C-Programm die Ausgabe von regcmp, so kann mit regex(abc,zeile) der reguläre Ausdruck abc auf zeile angewendet werden. Die Diagnosen sind selbsterklärend.

BEISPIELE

```
name    "([A-Za-z][A-Za-z0-9\_]*)$0"

telno   "\({0,1}([2-9][01][1-9])$0\){0,1} *"
        "([2-9][0-9]{2})$1[ -]{0,1}"
        "([0-9]{4})$2"
```

Die drei Argumente für telno müssen in einer Zeile angegeben werden.
Verwendet ein C-Programm die Ausgabe von regcmp, so wendet

```
regex(telno, zeile, bereich, exch, rest)
```

den regulären Ausdruck telno auf zeile an.

SIEHE AUCH

regcmp(3G).

rmdel - Delta aus einer SCCS-Datei entfernen

```
rmdel -rSID Datei...
```

`rmdel` (remove delta) entfernt das durch *SID* (SCCS identification string) angegebene Delta aus jeder angegebenen SCCS-Datei *Datei*. Das zu entfernende Delta muß das letzte Delta in der jeweiligen Verzweigung der Deltakette der jeweiligen SCCS-Datei sein. Außerdem darf die angegebene *SID* nicht die Version sein, die gerade editiert wird, um ein Delta zu erstellen. Wenn also für die SCCS-Datei eine p-Datei vorhanden ist (siehe `get`(1)), darf die *SID* nicht in einem Eintrag der p-Datei erscheinen.

Durch die Option `-r` wird die Version mit der Nummer *SID* gelöscht.

Bei Angabe eines Dateiverzeichnisses verhält sich `rmdel` so, als ob jede Datei im Dateiverzeichnis angegeben wurde. Nicht-SCCS-Dateien (letzte Komponente des Pfadnamens beginnt nicht mit `s.`) und unlesbare Dateien werden kommentarlos ignoriert. Bei Angabe von - wird von Standardeingabe gelesen; jede Zeile der Standardeingabe wird als Name einer SCCS-Datei interpretiert, die bearbeitet werden soll.

Bei der Entfernung eines Deltas gelten folgende Regeln: Wenn Sie ein Delta erstellt haben und die entsprechenden Dateizugriffsrechte besitzen, dürfen Sie dieses Delta auch entfernen; wenn Sie der Eigentümer der Datei und des Dateiverzeichnisses sind, in denen sich ein neues Delta befindet, können Sie das Delta immer entfernen.

DATEIEN

x.Datei	siehe `delta`(1)
z.Datei	siehe `delta`(1)

SIEHE AUCH

`delta`(1), `get`(1), `prs`(1), `sccsfile`(4).

sact - Editieraktivitäten einer SCCS-Datei ausgeben

sact Datei...

`sact` (SCCS-file activity) informiert den Benutzer über bevorstehende Deltas der SCCS-Datei *Datei*. Diese Situation tritt ein, wenn `get`(1) mit der Option `-e` ohne anschließendes `delta` ausgeführt wurde. Wenn ein Dateiverzeichnis in der Kommandozeile angegeben wird, verhält sich `sact` so, als ob jede Datei im Dateiverzeichnis angegeben wurde. Nicht-SCCS-Dateien und unlesbare Dateien werden kommentarlos ignoriert. Bei Angabe von - wird jede Zeile der Standardeingabe als Name einer zu bearbeitenden SCCS-Datei interpretiert.

Die Ausgabe für jede angegebene Datei besteht aus fünf Feldern, die mit Leerzeichen voneinander getrennt sind.

Feld 1 gibt die SID eines Deltas an, das gegenwärtig in der SCCS-Datei vorhanden ist und das für die Erstellung des neuen Deltas geändert wird.

Feld 2 gibt die SID für das neue zu erstellende Delta an.

Feld 3 enthält den Login-Namen des Benutzers, der das Delta ausführen wird, d.h. der ein `get` zur Editierung ausgeführt hat.

Feld 4 enthält das Datum, an dem `get -e` ausgeführt wurde.

Feld 5 enthält die Uhrzeit, zu der `get -e` ausgeführt wurde.

SIEHE AUCH

`delta`(1), `get`(1), `help`(1), `unget`(1).

sccsdiff - zwei Versionen einer SCCS-Datei vergleichen

sccsdiff -rSID1 -rSID2 [-p] [-sn] Datei...

`sccsdiff` vergleicht zwei Versionen einer SCCS-Datei und erstellt die Differenzen zwischen den beiden Versionen. Eine beliebige Anzahl von SCCS-Dateien kann angegeben werden, die Argumente werden auf alle Dateien angewendet.

`-r`*SID1* `-r`*SID2* — *SID1* und *SID2* geben die Deltas einer SCCS-Datei an, die verglichen werden sollen. Die Versionen werden in der angegebenen Reihenfolge an `bdiff`(1) weitergegeben.

`-p` — Die Ausgabe für jede Datei wird an `pr` geschickt.

`-s`*n* — Datei-Segmentgröße, die `bdiff` an `diff` weiterreicht. Dies ist nützlich, wenn `diff` wegen hoher Systembelastung nicht erfolgreich ausgeführt werden kann.

DATEIEN

`/var/tmp/get?????` temporäre Dateien

SIEHE AUCH

`get`(**1**), `help`(**1**).
`diff`(**1**), `bdiff`(**1**), `pr`(1) in "SINIX V5.41 Kommandos".

sdb - Symbolischer Debugger

sdb [-Option...] [Objektdatei [Speicherabzugsdatei [Verzeichnisliste]]]

`sdb` ist der symbolische Debugger für C- und Assembler-Programme. `sdb` kann dazu benutzt werden, ausführbare Programmdateien und Speicherabzugsdateien zu überprüfen. Es kann auch dazu verwendet werden, Prozesse in einer kontrollierten Umgebung zu überprüfen.

Das Argument *Objektdatei* ist der Name einer ablauffähigen Programmdatei. Um die Vorteile der symbolischen Fähigkeiten von `sdb` voll zu nutzen, sollte diese Datei mit der (Fehlersuch-)Option `-g` übersetzt werden. *Objektdatei* kann auch ein Pfadname im Verzeichnis `/proc` sein, in dem Fall wird der ausführende Prozeß, der durch den Pfadnamen bezeichnet wird, kontrolliert.

Eine *Speicherabzugsdatei* wird bei einem außerplanmäßigen Abbruch von *Objektdatei* oder beim Gebrauch von `gcore` erstellt. Sie enthält eine Kopie der Segmente eines Programms. Der Standard für *Speicherabzugsdatei* ist `core`. Eine Speicherabzugsdatei muß für die Verwendung von `sdb` nicht verfügbar sein. Die Verwendung von - statt *Speicherabzugsdatei* bewirkt, daß `sdb` eine existierende Speicherabzugsdatei ignoriert.

Das Argument *Verzeichnisliste* ist eine durch Doppelpunkte getrennte Liste von Verzeichnissen, die von `sdb` verwendet wird, um Quelldateien zum Aufbau von `Objektdatei` zu lokalisieren. Wenn keine Verzeichnisliste angegeben ist, schaut `sdb` im aktuellen Verzeichnis nach.

Die folgenden Optionen werden von `sdb` erkannt:

`-e` symbolische Informationen ignorieren und nichtsymbolische Adressen als Datei-Offsets betrachten

`-s` *Signal* laufende Prozesse, die unter Kontrolle von `sdb` laufen und *Signal* erhalten, nicht stoppen. *Signal* ist eine Dezimalzahl, die einer Signal-Nummer entspricht. Die Option kann mehrfach in der Kommandozeile von `sdb` angegeben werden.

`-V` Versionsinformationen ausgeben
Wenn kein Argument *Objektdatei* auf der Kommandozeile angegeben ist, endet `sdb` nach der Ausgabe der Versionsinformationen.

`-W` Warnungen, die die *Speicherabzugsdatei* oder die Quelldatei betreffen und älter als die *Objektdatei* sind, unterdrücken

`-w` dem Benutzer erlauben, auf die *Objektdatei* oder *Speicherabzugsdatei* zu schreiben

`sdb` erkennt eine aktuelle Zeile und eine aktuelle Datei. Wenn `sdb` eine ausführbare Programmdatei ohne Speicherabzugsdatei untersucht, werden die aktuelle Zeile und aktuelle Datei auf die entsprechenden Angaben der Datei gesetzt, die das erste `main` ent-

hält. Wenn die *Speicherabzugsdatei* vorhanden ist, werden die aktuelle Zeile und Datei zuerst auf die Zeile und Datei gesetzt, die die Quellanweisung, an der der Prozeß beendete, enthält. Die aktuelle Zeile und Datei ändern sich automatisch bei der Ausführung eines Prozesses. Sie können auch mit Prüfkommandos in der Quelldatei verändert werden.

Die Namen der Variablen werden genau wie in C geschrieben. Auf Variablen, die für eine Prozedur lokal sind, kann durch Verwendung der Form *Prozedur*:*Variable* zugegriffen werden. Wenn kein Prozedurname angegeben ist, wird die Prozedur, die die aktuelle Zeile enthält, standardmäßig verwendet.

Auf Strukturelemente kann mit *Variable*.*Element* zugegriffen werden, auf Zeiger auf Strukturelemente mit *Variable*->*Element*, und auf Feldelemente mit *Variable*[*Nummer*]. Zeiger können auch durch die Form *Zeiger*[*Nummer*] dereferenziert werden. Kombinationen dieser Formen können auch verwendet werden. Es können auch die Formen *Nummer*->*Element* verwendet werden, wobei *Nummer* die Adresse auf einen Zeiger ist, und *Nummer*.*Element*, wobei *Nummer* als Adresse der Instanz einer Struktur interpretiert wird. Die Schablone des in diesem Fall verwendeten Strukturtyps ist der letzte Strukturtyp, auf den verwiesen wurde. Wenn `sdb` den Wert einer Struktur anzeigt, werden die Werte aller Elemente der Struktur angezeigt. Die Adresse der Struktur ist die Adresse der Instanz der Struktur und nicht die der einzelnen Elemente.

Auf Elemente in einem mehrdimensionalen Feld kann mit *Variable* [*Nummer*]... oder mit *Variable* [*Nummer*,*Nummer*,...] zugegriffen werden. Statt *Nummer* kann auch *Nummer*;*Nummer* verwendet werden, um einen Bereich von Werten anzuzeigen. * kann verwendet werden, um alle legitimen Werte für den Index anzuzeigen. Der Index kann auch ganz weggelassen werden, wenn es sich um den letzten Index handelt, und der volle Bereich der Werte benötigt wird. Wenn keine Indizes angegeben sind, zeigt `sdb` die Werte von allen Elementen des Feldes an.

Eine bestimmte Instanz einer Variablen auf dem Stack kann man mit *Prozedur*:*Variable*,*Nummer* erreichen. Die *Nummer* ist die Nummer des Auftretens der angegebenen Prozedur im Stack, wobei das oberste Auftreten die Nummer 1 hat. Die Standardprozedur ist die, die die aktuelle Zeile enthält.

In `sdb`-Kommandos können auch Adressen verwendet werden. Sie werden mit Dezimal-, Oktal- oder Hexadezimalzahlen angegeben.

Zeilennummern des Quellprogramms werden mit *Dateiname*:*Nummer* oder *Prozedur*:*Nummer* angegeben. In jedem Fall ist *Nummer* relativ zum Anfang der Datei und entspricht der Zeilennummer, die von Text-Editoren verwendet wird oder der Ausgabe von `pr` entspricht. Eine Zahl für sich entspricht einer Zeile in der aktuellen Datei.

Wenn ein Prozeß unter `sdb` läuft, beziehen sich alle Adressen und Bezeichner auf den laufenden Prozeß. Wenn `sdb` nicht einen laufenden Prozeß untersucht, beziehen sich die Adressen und Bezeichner auf *Objektdatei* oder *Speicherabzugsdatei*.

Kommandos

Folgende Kommandos können verwendet werden, um Daten in einem Programm zu untersuchen:

`t` gibt ein Laufzeitkeller-Protokoll des beendeten oder unterbrochenen Programms aus. Die zuletzt aufgerufene Funktion befindet sich oben auf dem Keller. Bei C-Programmen endet der Keller mit `_start`. Das ist die Startroutine, die `main` aufruft.

`T` gibt die oberste Zeile des Laufzeitkeller-Protokolls aus.

Variable/clm

gibt den Wert von *Variable* gemäß der Länge *l* und des Formats *m* aus. Der Zähler *c* gibt an, daß ein Speicherbereich, der an der durch *Variable* implizit angegebenen Adresse beginnt, angezeigt werden soll. Die Längenangaben sind:

- `b` ein Byte
- `h` zwei Bytes (halbes Wort)
- `l` vier Bytes (langes Wort)

Zulässige Werte für *m* sind:

- `c` Zeichen
- `d` dezimal
- `u` dezimal, vorzeichenlos
- `o` oktal
- `x` hexadezimal
- `f` 32-Bit Gleitkomma, mit einfacher Genauigkeit
- `g` 64-Bit Gleitkomma, mit doppelter Genauigkeit
- `s` *Variable* wird als Zeiger auf eine Zeichenkette gewertet, und es werden Zeichen von der Adresse an ausgeben, auf die die Variable zeigt.
- `a` Es werden Zeichen ab der Adresse der Variablen ausgeben. Dieses Format darf nicht mit Registervariablen verwendet werden.
- `p` Zeiger auf eine Prozedur
- `i` disassembliert eine Maschinensprachen-Anweisung, wobei die Adressen numerisch und symbolisch erscheinen.
- `I` disassembliert eine Maschinensprachen-Anweisung, wobei die Adressen nur numerisch ausgegeben werden.

Längenangaben sind mit den Formaten `c`, `d`, `u`, `o` und `x` wirksam. Die Längenangabe bestimmt die Ausgabelänge des Wertes, der angezeigt werden soll. Dieser Wert kann abgeschnitten werden. Der Zähler *c* zeigt entsprechend viele Speichereinheiten an, wobei bei der Adresse von *Variable* begonnen wird. Die Anzahl der Bytes in einer Speichereinheit werden von *l* oder durch die zu der Variablen gehörende Größe bestimmt. Wenn die

Angaben *c*, *l* und *m* weggelassen werden, verwendet `sdb` Standardwerte. Wenn ein Zähler mit dem Kommando `s` oder `a` verwendet wird, werden so viele Zeichen angezeigt, wie die Zahl, die er enthält. Sonst werden aufeinanderfolgende Zeichen angezeigt, bis entweder ein Null-Byte erreicht wird oder 128 Zeichen ausgegeben worden sind. Die letzte Variable kann mit dem Kommando `./` noch einmal angezeigt werden.

Für einen eingeschränkten Mustervergleich kann man die `sh` Metazeichen * und `?` innerhalb von Prozedur- und Variablennamen verwenden. (`sdb` akzeptiert diese Metazeichen nicht in Dateinamen, als Funktionsnamen bei der Angabe der Zeilennummer für einen Unterbrechungspunkt, bei einem Funktionsaufruf-Kommando oder als Argument zu dem Kommando `e`.) Wenn kein Prozedurname hinzugefügt wird, führt `sdb` den Mustervergleich mit dem Namensmuster bei lokalen und globalen Variablen durch. Wenn der Prozedurname angegeben ist, führt `sdb` diesen Vergleich nur bei lokalen Variablen durch. Um diesen Vergleich nur bei globalen Variablen durchzuführen, verwendet man *:Muster*. Um alle Variablen auszugeben, verwendet man `*:*`.

Zeilennummer`?`*lm*
Variable`:?`*lm*

gibt den Wert an der angegebenen Adresse der ausführbaren Datei oder des Adreßraums aus, der durch *Zeilennummer* oder *Variable* (Prozedurname) gegeben ist, und zwar im Format *lm*. Das Standardformat ist `i`.

Variable`=`*lm*
Zeilennummer`=`*lm*
Nummer`=`*lm*

gibt die Adresse von *Variable* oder *Zeilennummer* oder den Wert von *Nummer* aus. *l* spezifiziert die Länge und *m* das Format. Wenn kein Format angegeben ist, verwendet `sdb` `1x` (Vier-Byte Hex). *m* ermöglicht bequemes Umwandeln zwischen Dezimal-, Oktal- und Hexadezimalwerten.

Variable`!`*Wert*

setzt *Variable* auf den angegebenen *Wert*. Dieser Wert kann eine Zahl, eine Zeichenkonstante oder eine Variable sein. Der Wert muß eindeutig definiert sein. Strukturen sind nur bei solchen Zuweisungen erlaubt, die eine andere Strukturvariable desselben Typs betreffen. Zeichenkonstanten werden als *'Zeichen* gekennzeichnet. Zahlen werden als ganzzahlig betrachtet, wenn kein Dezimalpunkt oder Exponent benutzt wird. In diesem Fall wird der Typ `double` angenommen. Register, mit Ausnahme der Gleitkomma-Register, werden als ganze Zahlen betrachtet. Registernamen sind mit denen identisch, die in Assembler verwendet werden (z.B. *%Regname*, wobei *Regname* der Name eines Registers ist). Wenn die Adresse einer Variablen angegeben ist, wird sie als die Adresse einer

Variablen des Typs `int` betrachtet. C-Konventionen werden in allen für die Ausführung der angezeigten Zuweisung benötigten Typenumwandlungen verwendet.

`x` gibt die Register und die aktuelle Maschinensprachen-Anweisung aus.

`X` gibt die aktuelle Maschinensprachen-Anweisung aus.

Die Kommandos für die Prüfung der Quelldateien sind:

`e`
`e` *Prozedur*
`e` *Dateiname*
`e` *Verzeichnis/*
Ohne Argumente gibt `e` den Namen der aktuellen Datei aus. Das zweite Format setzt die aktuelle Datei auf die Datei, die die *Prozedur* enthält. Das dritte Format setzt die aktuelle Datei auf *Dateiname*. Die aktuelle Zeile wird auf die erste Zeile der angegebenen Prozedur oder Datei gesetzt. Es wird angenommen, daß sich Quelldateien in den Verzeichnissen der Verzeichnisliste befinden. Das vierte Format fügt *Verzeichnis* ans Ende der Verzeichnisliste an.

/regulärer Ausdruck/
Wie in `ed` wird von der aktuellen Zeile vorwärts nach einer Zeile gesucht, die eine zum *regulären Ausdruck* passende Zeichenreihe enthält. Das nachgestellte `/` kann weggelassen werden, außer wenn es mit einem Haltepunkt zusammen auftritt.

?regulärer Ausdruck?
Wie in `ed` wird rückwärts von der aktuellen Zeile aus nach einer Zeile gesucht, die eine zum *regulären Ausdruck* passende Zeichenreihe enthält. Das nachgestellte `?` kann weggelassen werden, außer wenn es mit einem Haltepunkt zusammen auftritt.

`p` gibt die aktuelle Zeile aus.

`z` gibt die aktuelle Zeile und die nächsten neun Zeilen aus und setzt die aktuelle Zeile auf die letzte gedruckte Zeile.

`w` gibt 10 Zeilen aus, die um die aktuelle Zeile liegen.

Nummer gibt die aktuelle Zeile an, dann wird die neue aktuelle Zeile ausgegeben.

Anzahl+ schiebt die aktuelle Zeile um `Anzahl` Zeilen vor; gibt die neue aktuelle Zeile aus.

Anzahl– schiebt die aktuelle Zeile um *Anzahl* Zeilen zurück; gibt die neue aktuelle Zeile aus.

Die Kommandos zur Steuerung der Ausführung des Quellprogramms sind:

Anzahl `r` *Argumente*
Anzahl `R`

führt das Programm mit den angegebenen Argumenten aus. Das Kommando `r` ohne Argumente verwendet die vorhergehenden Argumente des Programms erneut. Das Kommando `R` führt das Programm ohne Argumente aus. Ein mit < oder > beginnendes Argument lenkt die Standardeingabe bzw. -ausgabe um. Es wird die ganze Syntax von `sh` akzeptiert. Wenn *Anzahl* angegeben ist, bestimmt sie die Anzahl der zu ignorierenden Haltepunkte.

Zeilennummer `c` *Anzahl*
Zeilennummer `C` *Anzahl*

setzt die Ausführung fort. `sdb` hält an, wenn es *Anzahl* Haltepunkte antrifft. Das Signal, das den Programmstopp veranlaßt hat, wird durch das Kommando `C` wieder aktiviert und durch das Kommando `c` ignoriert. Wenn eine *Zeilennummer* angegeben ist, wird ein temporärer Haltepunkt in die Zeile gesetzt und mit der Ausführung fortgefahren. Der Haltepunkt wird nach Ausführung des Kommandos entfernt.

Zeilennummer `g` *Anzahl*

setzt die Ausführung an der angegebenen Zeile fort. Wenn der Zählwert *Anzahl* angegeben ist, bestimmt er die Anzahl der zu ignorierenden Haltepunkte.

`s` *Anzahl*
`S` *Anzahl*

`s` durchläuft das Programm in Einzelschritten durch *Anzahl* Zeilen, oder, wenn *Anzahl* nicht angegeben ist, wird eine Zeile des Programms ausgeführt. `s` läuft von einer Funktion in eine aufgerufene Funktion. `S` durchläuft ein Programm auch schrittweise, aber es läuft nicht in eine aufgerufene Funktion. Es läuft über die aufgerufene Funktion hinweg.

`i` *Anzahl*
`I` *Anzahl*

durchläuft das Programm in Einzelschritten *Anzahl* Maschinensprachen-Anweisungen weit. Das Signal, das den Programmstopp veranlaßt hat, wird durch das Kommando `I` wieder aktiviert und durch das Kommando `i` ignoriert.

Variable`$m` *Anzahl*
Adresse`:m` *Anzahl*

durchläuft das Programm in Einzelschritten (wie bei `s`), bis die angegebene Position durch einen neuen Wert modifiziert wird. Wenn *Anzahl* ausgelassen wird, läuft das Programm effektiv unendlich. Auf die *Variable* muß von der aktuellen Prozedur aus zugegriffen werden können. Dieses Kommando kann sehr langsam sein.

Level v
: stellt den Anzeigemodus bei Einzelschritt-Verfahren S, s oder m ein. Wenn *Level* ausgelassen wird, dann werden nur die aktuelle Quelldatei und/oder der Funktionsname ausgegeben, wenn sich eine(r) der beiden ändert. Wenn *Level* 1 oder größer ist, wird jede C-Quellzeile vor der Ausführung erst ausgegeben; wenn *Level* 2 oder größer ist, wird auch jede Assembleranweisung ausgegeben. Ein v schaltet den Anzeigemodus ab.

k
: bricht das Programm, in dem Fehler gesucht werden, ab.

Prozedur(Argument1,Argument2,...)
Prozedur(Argument1,Argument2,...)/m
: führt die Prozedur mit den angegebenen Argumenten aus. Argumente können Registernamen, ganze Zahlen, Zeichen- und Zeichenkettekonstanten oder Namen von Variablen sein, auf die von der aktuellen Prozedur aus zugegriffen werden kann. Das zweite Format veranlaßt, daß der durch die Prozedur zurückgegebene Wert entsprechend dem Format *m* ausgegeben wird. Wenn kein Format angegeben ist, wird standardmäßig d genommen.

Zeilennummer b *Kommandos*
: setzt einen Haltepunkt auf die angegebene Zeile. Wenn ein Prozedurname ohne eine Zeilennummer angegeben ist (z.B. *Proz*:), wird ein Haltepunkt an die erste Zeile der Prozedur gesetzt, selbst wenn sie nicht mit der Option -g übersetzt wurde. Wenn keine *Zeilennummer* angegeben ist, wird ein Haltepunkt in die aktuelle Zeile gesetzt. Wenn keine *Kommandos* angegeben sind, stoppt die Ausführung direkt vor dem Haltepunkt, und die Steuerung wird an sdb zurückgegeben. Andernfalls werden die *Kommandos* ausgeführt, wenn der Haltepunkt angetroffen wird, und danach läuft die Ausführung weiter. Mehrere Kommandos werden durch Trennung mit Semikolon angegeben. Die Schachtelung von verknüpften Kommandos ist nicht erlaubt; das Setzen von Haltepunkten innerhalb der verknüpften Umgebungen ist erlaubt.

B
: gibt eine Liste der momentan aktiven Haltepunkte aus.

Zeilennummer d
: löscht einen Haltepunkt an der angegebenen Zeile. Wenn keine Zeilennummer angegeben ist, werden die Haltepunkte interaktiv gelöscht. Jede Haltepunktposition wird angegeben, und eine Zeile wird von der Standardeingabe gelesen. Wenn die Zeile mit einem y oder einem d beginnt, wird der Haltepunkt gelöscht.

D
: löscht alle Haltepunkte.

l
: gibt die zuletzt ausgeführte Zeile aus.

Zeilennummer `a`
: zeigt eine Zeilennummer an. Wenn die *Zeilennummer* die Form *Proz*`:`*Nummer* hat, führt das Kommando *Zeilennummer*`:b 1;c` aus. Wenn *Zeilennummer* die Form *Proz*`:` hat, führt das Kommando *Proz*`:b T;c` aus.

Verschiedene Kommandos:

#Rest-der-Zeile
: Der *Rest-der-Zeile* steht für Kommentare, die von `sdb` ignoriert werden.

`!`*Kommando* *Kommando* wird von `sh` interpretiert.

`Neue-Zeile` Wenn das vorhergehende Kommando eine Quellzeile ausgegeben hat, wird die aktuelle Zeile um eine Zeile vorgeschoben und die neue aktuelle Zeile ausgegeben. Wenn das vorhergehende Kommando eine Speicherposition angezeigt hat, wird die nächste Speicherposition angezeigt. Wenn das vorhergehende Kommando einen Befehl disassembliert hat, wird der nächste Befehl disassembliert.

`Dateiende-Zeichen`
: zeigt die nächsten 10 Anweisungszeilen an, Quelle oder Daten, je nachdem, was zuletzt ausgegeben wurde. Normalerweise ist `^d` das Dateiende-Zeichen.

`<` *Dateiname*
: liest Kommandos aus der Datei *Dateiname*, bis das Dateiende erreicht wird, und nimmt dann Kommandos weiter von der Standardeingabe an. Direkt vor der Ausführung werden die Kommandos ausgegeben, wobei ihnen zwei Sternzeichen vorangestellt werden. Dieses Kommando darf nicht geschachtelt werden. < darf nicht als Kommando in einer Datei erscheinen.

`M` gibt die Adreßabbildungen aus.

`"` *Zeichenkette*
: gibt die angegebene Zeichenkette aus. Die C-Fluchtzeichen der Form *\Zeichen*, *\Oktalziffern*, oder *\xHexadezimalziffern* werden erkannt. Dabei ist *Zeichen* ein nichtnumerisches Zeichen. Das abschließende Anführungszeichen kann weggelassen werden.

`q` verläßt den Debugger.

`V` gibt Versionsinformationen aus.

SIEHE AUCH

cc(1), dbx(1), signal(2), a.out(4), core(4), syms(4)
Kapitel "sdb" in "Leitfaden und Werkzeuge für die Programmierung mit C".

HINWEIS

Wenn *Objektdatei* eine dynamisch gebundene, ausführbare Datei ist, kann auf Variablen, Funktionsnamen, usw., die in gemeinsam benutzten Objekten definiert sind, nicht verwiesen werden bis das entsprechende gemeinsam benutzte Objekt an den Prozeß angehängt ist. Bei gemeinsam benutzten Objekten, die beim Start angehängt werden (z.B. libc.so.1, die Standard-C-Bibliothek), bedeutet das, daß auf solche Variablen nicht zugegriffen werden kann, bevor main aufgerufen wird.

Auf das Argument *Objektdatei* wird für Debug-Informationen direkt während der Erzeugung des Prozesses mit der PATH-Variablen zugegriffen.

Verwenden Sie zur Fehlersuche in Ihrem mit CES erstellten Programm den symbolischen Debugger dbx(1). Eine fehlerfreie Zusammenarbeit von CES mit sdb wird nicht garantiert.

size - Größe einer Objektdatei ausgeben

```
size [-F -f -n -o -V -x] Datei...
```

`size` gibt die Größe einer Objektdatei, d.h. eines Objektmoduls oder eines ablauffähigen Programms, in Bytes aus.

Auf Standardausgabe wird die Anzahl der Bytes ausgegeben, die vom Textsegment, von Datensegment und vom bss-Segment belegt wird.

`size` verarbeitet ELF- und COFF-Objektdateien, die auf der Kommandozeile eingegeben wurden. Wird eine Archivdatei mit dem Kommando `size` eingegeben, wird die Information für jede Objektdatei im Archiv ausgegeben.

Beim Berechnen der Segmentinformation gibt das Kommando `size` die Dateigröße der schreibbaren und nichtschreibbaren Segmente und die Speichergröße der schreibbaren Segmente abzüglich ihrer Dateigröße aus.

Falls `size` keine Segmentinformationen berechnen kann, berechnet es die Abschnittsinformation neu. Dabei druckt es die Größe der Abschnitte aus, die belegbar, nichtschreibbar und nicht-NOBITS sind, zudem die Größe der Abschnitte, die belegbar, schreibbar und nicht-NOBITS sind und die Größe der schreibbaren Abschnitte vom Typ NOBITS. (NOBITS-Abschnitte nehmen tatsächlich keinen Platz in der *Datei* ein.)

Wenn `size` weder Segments- noch Abschnittsinformation neu berechnen kann, gibt es eine Fehlermeldung aus und unterbricht die Dateiverarbeitung.

`-F` gibt die Größe und Zugriffsmöglichkeiten jedes ladbaren Segments und die Summe der ladbaren Segmentgrößen aus. Falls keine Segmentdaten existieren, gibt `size` eine Fehlermeldung aus und unterbricht die Dateiverarbeitung.

`-f` gibt die Größe jedes belegbaren Abschnitts, den Abschnittsnamen und die Summe der Abschnittsgrößen aus. Falls keine Abschnittsdaten existieren, gibt `size` eine Fehlermeldung aus und unterbricht die Dateiverarbeitung.

`-n` gibt nichtladbare Segment- oder nichtbelegbare Abschnittsgrößen aus. Falls Segmentdaten existieren, gibt `size` die Speichergröße jedes ladbaren Segments oder die Dateigröße jedes nichtladbaren Segments, die Zugriffsmöglichkeiten und die Größe der Segmente aus. Falls keine Segmentdaten existieren, gibt `size` für jeden zuweisbaren und nichtzuweisbaren Abschnitt die Speichergröße, den Abschnittsnamen und die Gesamtgröße der Abschnitte aus. Falls keine Segments- oder Abschnittsdaten existieren, gibt `size` eine Fehlermeldung aus und unterbricht die Verarbeitung.

`-o` gibt Oktal-, aber keine Dezimalnummern aus.

`-V` gibt die Versionsinformation für das Kommando `size` auf der Standard-Fehlerausgabe aus.

`-x` gibt hexadezimale, aber keine dezimalen Nummern aus.

BEISPIELE

Die unten aufgeführten Beispiele sind typisch für die `size`-Ausgabe:

```
size Datei          2724 + 88 + 0 = 2812
size -f Datei       26(.txt) + 5(.init) + 5(.fini)  = 36
size -F Datei       2724(r-x) + 88(rwx) = 2812
```

SIEHE AUCH

`cc`(1), `ld`(1), `a.out`(4), `ar`(4).

HINWEIS

Da die Größe von bss-Abschnitten vor dem Binden nicht bekannt ist, gibt das Kommando `size` nicht die korrekte Gesamtgröße vorgebundener Objektdateien aus.

strip - Symboltabelle entfernen

```
strip [-blrVx] Datei...
```

Das Kommando `strip` entfernt die Symboltabellen, Information zur Fehlersuche und zu Zeilennummern von ELF-Objektdateien; COFF-Objektdateien können nicht mehr mit `strip` bearbeitet werden. Nach der Ausführung dieses Kommandos können Sie auf die symbolische Fehlersuchinformation nicht mehr zugreifen. Daher wird dieses Kommando normalerweise nur bei Produktmodulen eingesetzt, die auf Fehler durchsucht und getestet worden sind.

Wurde `strip` auf einer allgemeinen Archivdatei durchgeführt (siehe `ar`(1)) und die einzelnen Teile verarbeitet, entfernt es zusätzlich die Archivsymboltabelle. Diese muß mit Hilfe des Kommandos `ar`(1) mit der Option `-s` wiederhergestellt werden, bevor das Archiv durch das Kommando `ld`(1) verarbeitet werden kann. `strip` erzeugt entsprechende Warnmeldungen, sobald diese Situation eintritt.

Der Umfang der von der ELF-Objektdatei entfernten Daten kann durch die folgenden Optionen gesteuert werden:

`-b` hat denselben Effekt wie das Standardverhalten. Diese Option ist veraltet und wird im nächsten Release nicht mehr vorhanden sein.

`-l` entfernt nur Informationen zur Zeilennummer; entfernt nicht die Symboltabelle oder Information zur Fehlersuche.

`-r` hat denselben Effekt wie das Standardverhalten. Diese Option ist veraltet und wird im nächsten Release entfernt.

`-V` gibt auf die Standard-Fehlerausgabe die Versionsnummer von `strip` aus.

`-x` entfernt nicht die Symboltabelle; Informationen zu Fehlersuche und Zeilennummer können entfernt werden.

`strip` wird dazu benutzt, den Dateispeicherplatz der Objektdatei zu reduzieren.

DATEIEN

TMPDIR`/strp*`
temporäre Dateien

TMPDIR ist im allgemeinen `/var/tmp`, kann jedoch durch Setzen der Umgebungsvariablen `TMPDIR` neu definiert werden (siehe `tempnam`() in `tmpnam`(3S)).

SIEHE AUCH

`ar`(1), `cc`(1), `ld`(1), `tmpnam`(3S), `a.out`(4), `ar`(4).

HINWEIS

Der Abschnitt Symboltabelle wird nicht entfernt, wenn er in einem Segment enthalten ist oder wenn die Datei entweder ein verschiebbares oder ein dynamisch gemeinsam benutztes Objekt ist.

Die Abschnitte zu Zeilennummer und Fehlersuche werden nicht entfernt, wenn sie in einem Segment enthalten sind oder wenn ihr zugehöriger Relokations-Abschnitt in einem Segment enthalten ist.

tsort - topologisch sortieren

tsort [Datei]

Das Kommando `tsort` (topological sort) schreibt auf die Standardausgabe eine vollständig geordnete Liste von Wörtern, die aus einer teilweise geordneten Liste von Wörtern der *Datei* hervorgeht. Wenn keine *Datei* angegeben ist, wird die Standardeingabe genommen.

Die Eingabe besteht aus Paaren von Wörtern (nichtleere Zeichenketten), die durch Leerzeichen voneinander getrennt sind. Paare mit verschiedenen Wörtern erfordern Einordnung gemäß der Sortierordnung. Paare mit zwei gleichen Wörtern bedeuten, daß das Wort ohne Einordnung aufgenommen werden soll.

SIEHE AUCH

`lorder`(1).

ENDESTATUS

`Odd data:` Es liegt eine ungerade Anzahl von Feldern in der Eingabedatei vor.

unget - get einer SCCS-Datei rückgängig machen

unget [-rSID] [-s] [-n] Datei

`unget` (undo get) hebt die Wirkung eines `get -e` wieder auf, das vor der Erstellung eines neuen Deltas ausgeführt wurde. `unget` muß von demselben Benutzer aufgerufen werden, der das Kommando `get -e` ausgeführt hat. Wenn ein Dateiverzeichnis angegeben wird, verhält sich `unget` so, als ob jede Datei im Dateiverzeichnis angegeben ist, wobei jedoch Nicht-SCCS-Dateien und unlesbare Dateien kommentarlos ignoriert werden. Wenn - angegeben ist, wird die Standardeingabe so gelesen, daß jede Zeile als Name einer zu bearbeitenden SCCS-Datei aufgefaßt wird.

Die Optionen gelten unabhängig für jede angegebene Datei.

`-r`*SID* identifiziert eindeutig das nicht mehr beabsichtigte Delta. (Dies wäre durch `get` als das 'neue Delta' angegeben worden). Die Verwendung dieser Option ist nur notwendig, wenn zwei oder mehr noch nicht abgespeicherte `get` zum Editieren in derselben SCCS-Datei von derselben Person (Login-Name) ausgeführt wurden. Es gibt eine Fehlermeldung, wenn die angegebene *SID* nicht eindeutig ist oder notwendig, aber auf der Kommandozeile ausgelassen wurde.

`-s` unterdrückt die Ausgabe der SID des Deltas auf der Standardausgabe.

`-n` erhält die g-Datei, die normalerweise aus dem aktuellen Dateiverzeichnis entfernt würde.

DATEIEN

p-Datei	siehe `delta`(1)
q-Datei	siehe `delta`(1)
z-Datei	siehe `delta`(1)

SIEHE AUCH

`delta`(1), `get`(1), `help`(1), `sact`(1).

val - SCCS-Datei validieren

```
val -
val [-s] [-rSID] [-mName] [-yTyp] Datei...
```

`val` (validate) bestimmt, ob die angegebene *Datei* eine SCCS-Datei ist und den von der optionalen Argumentliste vorgegebenen Merkmalen entspricht. Argumente für `val` können in beliebiger Reihenfolge erscheinen.

Die Option - hat beim Kommando `val` eine von den anderen SCCS-Kommandos abweichende Bedeutung. Es erlaubt `val`, die Argumente von der Standardeingabe statt von der Kommandozeile zu lesen.

`val` generiert Diagnosemeldungen auf der Standardausgabe und gibt außerdem beim Verlassen einen 8-Bit-Code zurück.

Die Wirkung jedes Optionsarguments trifft unabhängig auf jede in der Kommandozeile angegebenen Dateien zu.

`-s` unterdrückt die Diagnosemeldung, die normalerweise auf der Standardausgabe für jeden Fehler ausgegeben wird.

`-r`*SID* prüft die *SID* auf Mehrdeutigkeit (z.B. ist `-r1` mehrdeutig, weil sie physisch nicht vorhanden ist, jedoch 1.1, 1.2 usw. impliziert, die vorhanden sein können) oder Gültigkiet (z.B. ist `-r1.0` oder `-r1.1.0` ungültig, weil es keine eine gültige Deltanummer sein kann). Wenn *SID* gültig und nicht mehrdeutig ist, wird geprüft, ob dieser Wert tatsächlich existiert.

`-m`*Name* *Name* wird mit dem SCCS-Schlüsselwort `%M%` in *Datei* verglichen.

`-y`*Typ* *Typ* wird mit dem SCCS-Schlüsselwort `%Y%` in *Datei* verglichen.

Der von `val` zurückgegebene 8-Bit-Code ist eine ODER-Verknüpfung der möglichen Fehler; er kann als eine Bit-Zeichenkette interpretiert werden, in der die gesetzten Bits von links nach rechts fortschreitend wie folgt interpretiert werden:

Bit 0 = Dateiname *Datei* fehlt
Bit 1 = unbekanntes oder doppeltes Schlüsselkennwort im Argument
Bit 2 = beschädigte SCCS-Datei
Bit 3 = Datei kann nicht eröffnet werden, oder Datei ist nicht SCCS
Bit 4 = *SID* ist ungültig oder mehrdeutig
Bit 5 = *SID* ist nicht vorhanden
Bit 6 = `%Y%`, `-y` paßt nicht
Bit 7 = `%M%`, `-m` paßt nicht

`val` kann zwischen zwei und 50 Dateien auf einer angegebenen Kommandozeile und jeweils mehrere Kommandozeilen beim Lesen der Standardeingabe verarbeiten. In diesen Fällen wird ein Summencode zurückgegeben: ein logisches ODER des für jede verarbeitete Kommandozeile und Datei generierten Codes.

SIEHE AUCH

`admin`(1), `delta`(1), `get`(1), `help`(1), `prs`(1).

vc - Versionskontrolle

vc [Option...] [Schlüsselwort=Wert...]

vc (version control) ist veraltet und wird im nächsten Release nicht mehr vorhanden sein.

vc kopiert Zeilen von der Standardeingabe auf die Standardausgabe, gesteuert von seinen Argumenten und Kontrollanweisungen in der Eingabe. Im Verlaufe dieses Kopiervorgangs können vom Benutzer angegebene *Schlüsselwörter* durch ihre Zeichenkette *Wert* ersetzt werden, wenn sie im laufenden Text und/oder in den Steueranweisungen erscheinen.

Das Kopieren von Zeilen von der Standardeingabe zur Standardausgabe erfolgt bedingt, basierend auf Prüfungen der Schlüsselwort-Werte, die in Steueranweisungen oder als Kommandoargumente für vc angegeben werden.

Eine Steueranweisung besteht aus einer mit einem Kontrollzeichen beginnenden Einzelzeile. Eine Ausnahme bildet -t. Das Standardkontrollzeichen ist ein Doppelpunkt :, wobei jedoch eine Änderung durch die Option -c möglich ist. Mit einem Backslash \ und nachfolgendem Kontrollzeichen anfangende Zeilen sind keine Steuerzeilen und werden ohne Backslash in die Standardausgabe kopiert. Zeilen, die mit einem Backslash und ohne folgendes Kontrollzeichen beginnen, werden in vollem Umfang kopiert.

Ein Schlüsselwort besteht aus neun oder weniger alphanumerischen Zeichen; das erste Zeichen muß ein Buchstabe sein. Ein Wert ist jede ASCII-Zeichenkette, die mit ed erstellt werden kann; ein numerischer Wert ist eine vorzeichenlose Folge von Ziffern. Schlüsselwort-Werte dürfen keine Leerzeichen oder Tabulatoren enthalten.

Schlüsselwörter werden stets durch Werte ersetzt, wenn sie in einer Steueranweisung auf ein Schlüsselwort treffen, das in Kontrollzeichen eingeschlossen ist. Mit der Option -a werden die Schlüsselwörter im gesamten Text ersetzt. Ein nichtinterpretiertes Kontrollzeichen kann durch Voranstellen eines Backslashs \ in einen Wert aufgenommen werden. Wenn \ selbst erwünscht ist, muß diesem ebenfalls ein \ vorangestellt werden.

Die folgenden Optionen sind gültig:

-a ersetzt in Kontrollzeichen eingeschlossene Schlüsselwörter durch die zugeordneten Werte in allen Textzeilen und nicht nur in vc-Anweisungen.

-t Alle Zeichen vom Anfang einer Zeile bis einschließlich des ersten Tabulator-Zeichens werden zwecks Feststellung einer Steueranweisung ignoriert. Wenn eine Steueranweisung gefunden wird, werden alle Zeichen bis zum Tabulator-Zeichen einschließlich gelöscht.

-c*Zeichen* gibt ein Kontrollzeichen an, das anstelle des Standards : verwendet werden soll.

`-s` unterdrückt die normalerweise in der Diagnose-Ausgabe angegebenen Warnungen; aber nicht die Fehlermeldungen.

`vc` erkennt die folgenden Versionssteueranweisungen:

`:dcl` *Schlüsselwort*[, ..., *Schlüsselwort*]
wird zum Deklarieren von Schlüsselwörtern verwendet. Alle Schlüsselwörter müssen deklariert werden.

`:asg` *Schlüsselwort=Wert*
wird für die Zuweisung von Werten an Schlüsselwörter verwendet. Eine `asg`-Anweisung hebt die Zuweisung für das entsprechende Schlüsselwort auf der `vc`-Kommandozeile und alle vorhergehenden `asg`-Anweisungen für dieses Schlüsselwort auf. Deklarierte Schlüsselwörter, denen noch kein Wert zugewiesen ist, haben einen Nullwert.

`:if` *Bedingung*
...
`:end`
wird zum Überspringen von Zeilen in der Standardeingabe verwendet. Wenn die Bedingung wahr ist, werden alle Zeilen zwischen der `if`-Anweisung und der entsprechenden `end`-Anweisung in die Standardausgabe kopiert. Ist die Bedingung unwahr, werden alle dazwischenliegenden Zeilen einschließlich der Steueranweisungen gelöscht. Es ist zu beachten, daß dazwischenliegende `if`-Anweisungen und die entsprechenden `end`-Anweisungen lediglich zur Erhaltung der korrekten Entsprechung von `if-end` erkannt werden.

Die Syntax einer Bedingung lautet:

```
<Bed>     ::= [ 'not' ] <oder>
<oder>    ::= <und> | <und> '|' <oder>
<und>     ::= <Ausdr> | <Ausdr> '&' <und>
<Ausdr>   ::= '(' <oder> ')' | <Wert> <op> <Wert>
<op>      ::= '=' | '!=' | '<' | '>'
<Wert>    ::= <beliebige ASCII-Zeichenkette> | <Ziffernfolge>
```

Die zur Verfügung stehenden Operatoren und ihre Bedeutung sind:

```
=     gleich
!=    ungleich
&     und
|     oder
>     größer als
<     kleiner als
()    wird für logische Zusammenfassungen verwendet
not   kann nur unmittelbar hinter einem if stehen, und dreht dann den
      Wert der gesamten Bedingung um
```

Die Vergleichsoperatoren > und < wirken nur auf vorzeichenlose ganze Werte (z.B. `: 012 > 12` ist unwahr). Alle anderen Operatoren erwarten Zeichenketten als Argumente (z.B. `: 012 != 12` ist wahr).

Die Priorität der Operatoren ist wie folgt:

```
= != > <        alle mit gleichen Rang
&
|
```

Die Rangfolge kann mit Hilfe von Klammern geändert werden.

Werte müssen von Operatoren oder Klammern durch mindestens ein Leer- oder Tabulatorzeichen getrennt werden.

`::`*Text* ersetzt Schlüsselwörter in Zeilen, die auf die Standardausgabe kopiert werden. Die beiden führenden Kontrollzeichen werden entfernt. In Kontrollzeichen eingeschlossene Schlüsselwörter im Text werden durch ihren Wert ersetzt, bevor die Zeile in die Ausgabedatei kopiert wird. Dieser Vorgang ist unabhängig von der Option `-a`.

`:on`
`:off` Schlüsselwort-Ersetzung in allen Zeilen ein- bzw. ausschalten

`:ctl` *Zeichen*
Kontrollzeichen auf *Zeichen* umändern

`:msg` *Meldung*
gibt die angegebene *Meldung* auf der Fehlerausgabe aus.

`:err` *Meldung*
gibt die angegebene *Meldung* auf der Fehlerausgabe aus:

```
ERROR: err statement on line ... (915)
```

`vc` beendet die Ausführung und gibt den Endestatus 1 zurück.

SIEHE AUCH

`help`(1).
`ed`(1) in "SINIX V5.41 Kommandos".

what - Dateien identifizieren

what [-s] Datei...

`what` durchsucht die angegebenen *Dateien* nach jedem Vorkommen des Musters, das `get`(1) für `%Z%` einsetzt (zur Zeit `@(#)`) und gibt die darauffolgenden Zeichen bis zum ersten ", >, Neue-Zeile Zeichen, \ oder Nullzeichen aus.

Wenn beispielsweise das C-Programm in der Datei `f.c` folgende Anweisungszeile enthält:

```
#ident "@(#)Identifikationsinformation";
```

und `f.c` so übersetzt wurde, daß `f.o` und `a.out` erstellt wurden, gibt das Kommando

```
what f.c f.o a.out
```

folgendes aus:

```
f.c:
     Identifikationsinformation
f.o:
     Identifikationsinformation
a.out:
     Identifikationsinformation
```

`what` wird zusammen mit dem `get`-Kommando verwendet, das automatisch Identifikationsdaten einfügt. Es kann jedoch auch verwendet werden, wenn die Identifikationsdaten manuell eingefügt werden.

Die Option hat folgende Bedeutung:

`-s` Suche beenden, sobald das Muster zum ersten Mal gefunden wurde.

SIEHE AUCH

`get`(1), `help`(1), `mcs`(1).

ENDESTATUS

Wenn das gesuchte Muster gefunden wird, ist der Endestatus 0, andernfalls 1.

yacc - Parser erstellen

yacc [Option...] Datei

Das Kommando yacc (yet another compiler-compiler) wandelt eine kontextfreie Grammatik in eine Menge von Tabellen für einen einfachen Automaten um, der einen LALR(1) Syntaxanalyse-Algorithmus ausführt. Die Grammatik kann mehrdeutig sein; für die Auflösung dieser Mehrdeutigkeiten werden bestimmte Regeln angewendet.

Die Ausgabedatei y.tab.c muß zur Erstellung eines Programms yyparse vom C-Übersetzer übersetzt werden. Dieses Programm muß mit dem lexikalischen Analyse-Programm yylex sowie mit main und yyerror, einer Fehlerbehandlungsroutine, gebunden werden. Diese Routinen sind vom Benutzer bereitzustellen. Das Kommando lex(1) ist zum Erstellen lexikalischer Analysatoren, die von yacc verwendet werden können, nützlich.

Folgende Optionen können beim Aufruf von yacc verwendet werden:

-v stellt die Datei y.output bereit. Sie enthält eine Beschreibung der Syntaxanalyse-Tabellen und eine Meldung über Konflikte, die aus Mehrdeutigkeiten in der Grammatik entstanden sind.

-d erzeugt die Datei y.tab.h mit den #define-Anweisungen, die die von yacc zugewiesenen Token-Codes mit den vom Benutzer angegebenen Token-Namen in Verbindung setzen. Damit können auch andere Quelldateien als y.tab.c auf die Token-Codes zugreifen.

-l gibt an, daß der in y.tab.c erzeugte Code keine #line-Anweisungen enthält. Verwenden Sie diese Option nur, wenn Grammatik und zugehörige Aktionen völlig ausgetestet, d.h. fehlerfrei sind.

-Q[y|n] Die Option -Qy schreibt die Versionsinformation in y.tab.c. Dadurch erfahren Sie, welche yacc-Version erstellt wurde. Die Option -Qn (Standard) schreibt keine Versionsinformation.

-t übersetzt standardmäßig Code zur Unterstützung der Fehlersuche zur Laufzeit. Der Fehlersuchcode wird in y.tab.c immer generiert, steht jedoch unter bedingter Übersetzungssteuerung. Standardmäßig wird dieser Code bei der Übersetzung von y.tab.c nicht berücksichtigt. Unabhängig von der Verwendung der Option -t wird die Bereitstellung des Codes für die Laufzeit-Fehlersuche durch das Präprozessor-Symbol YYDEBUG gesteuert. Wenn YYDEBUG einen Wert ungleich Null hat, wird der Fehlersuchcode eingebunden. Ist der Wert gleich Null, wird der Code nicht eingebunden. Größe und Ablaufzeit eines ohne Fehlersuchcode erstellten Programms ist kleiner bzw. kürzer.

-V druckt die Versionsinformation für yacc auf die Standard-Fehlerausgabe.

DATEIEN

`y.output, y.tab.c, y.tab.h`	Definitionen von Token-Namen
`yacc.tmp, yacc.debug, yacc.acts`	temporäre Dateien
LIBDIR`/yaccpar`	Analysealgorithmus-Prototyp für C-Programme
LIBDIR	im allgemeinen `/usr/ccs/lib`

SIEHE AUCH

`lex`(1).
Kapitel "yacc" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ENDESTATUS

Die Anzahl der reduziere/reduziere- und lies/reduziere-Konflikte wird auf der Standard-Fehlerausgabe gemeldet; ein detaillierterer Bericht ist in der Datei `y.output` zu finden. Eine Meldung erfolgt auch dann, wenn einige Regeln nicht vom Startsymbol aus erreichbar sind.

HINWEIS

Da die Dateinamen festgelegt sind, kann jeweils nur höchstens ein `yacc`-Prozeß zu einem gegebenen Zeitpunkt in einem gegebenen Dateiverzeichnis aktiv sein.

Systemaufrufe

intro - Einführung in Systemaufrufe und Fehlernummern

```
#include <errno.h>
```

In diesem Kapitel sind alle Systemaufrufe beschrieben. Für die meisten dieser Aufrufe gibt es eine oder mehrere Fehlerrückgabewerte. Ein Fehler wird durch einen Wert angezeigt, der normalerweise nicht möglich ist. Dies ist fast immer der Wert -1 oder der Nullzeiger; Einzelheiten sind in den entsprechenden Beschreibungen zu finden. Außerdem wird eine Fehlernummer in der externen Variablen `errno` zur Verfügung gestellt. `errno` wird bei erfolgreichen Aufrufen nicht rückgesetzt und soll deshalb nur nach dem Auftreten eines Fehlers getestet werden.

In jeder Beschreibung eines Systemaufrufs wird versucht, alle möglichen Fehlernummern aufzulisten. Nachstehend wird eine komplette Liste der Fehlernummern und ihrer Bezeichnungen gegeben, wie sie in `errno.h` definiert sind.

`1 EPERM` Nicht Systemverwalter
: Typischerweise tritt dieser Fehler auf, wenn versucht wurde, eine Datei auf eine Weise zu ändern, die außer für den Eigentümer oder Systemverwalter verboten ist. Bei normalen Benutzern kann diese Situation auch auftreten, wenn sie versuchen, Maßnahmen auszuführen, die dem Systemverwalter vorbehalten sind.

`2 ENOENT` Datei oder Dateiverzeichnis unbekannt
: Es wurde der Name einer Datei verwendet, die vorhanden sein sollte, dies jedoch nicht ist, oder aber eines der Dateiverzeichnisse in einem Pfadnamen ist nicht vorhanden.

`3 ESRCH` Prozeß unbekannt
: Es kann kein Prozeß gefunden werden, der dem durch PID in der Routine `kill` oder `ptrace` angegebenen Prozeß entspricht.

`4 EINTR` Unterbrochener Systemaufruf
: Ein asynchrones Signal (z.B. `interrupt` oder `quit`), das nicht ignoriert werden soll, ist während eines Systemaufrufs aufgetreten. Bei Wiederaufnahme der Ausführung nach Verarbeitung des Signals wird der unterbrochene Systemaufruf anstelle seines normalen Resultats diesen Fehler zurückgegeben.

5 EIO Ein-/Ausgabe-Fehler
Ein physikalischer Ein-/Ausgabe-Fehler ist aufgetreten. In einigen Fällen ist es möglich, daß der Fehler erst bei dem nächsten Systemaufruf auftritt (z.B. kann ein Fehler bei `write` nach `close` gemeldet werden).

6 ENXIO Gerät oder Adresse nicht verfügbar
Eine Ein-/Ausgabe-Operation verwendet ein Gerät, auf das nicht zugegriffen werden kann.

7 E2BIG Argumentliste zu lang
Einem `exec`-Aufruf wurde eine Argumentliste übergeben, die länger als `ARG_MAX` Bytes ist. Die Obergrenze für die Länge einer Argumentliste ist die Summe aus der Größe der Argumentliste und der Größe aller von der Umgebung exportierten Shell-Variablen.

8 ENOEXEC Ungültiges Format der Binärdatei
Es wurde versucht, eine Datei auszuführen, für die zwar die entsprechenden Berechtigungen gesetzt sind, die jedoch nicht mit einem gültigen Format beginnt (siehe auch `a.out`(4)).

9 EBADF Ungültige Dateinummer
Ein Dateideskriptor bezieht sich nicht auf eine offene Datei.

10 ECHILD Keine Sohnprozesse vorhanden
Ein Aufruf von `wait` wurde von einem Prozeß ausgeführt, der keine Sohnprozesse hat, oder keine, auf die er noch warten könnte.

11 EAGAIN Keine weiteren Prozesse
Der `fork`-Aufruf war erfolglos, weil die Prozeßtabelle des Systems voll ist oder der Benutzer keine weiteren Prozesse mehr starten darf. Es ist auch möglich, daß ein Systemaufruf wegen unzureichendem Speicher- oder Swap-Bereich gescheitert ist.

12 ENOMEM Hauptspeicher erschöpft
Während der Ausführung einer `exec`-, `brk`- oder `sbrk`-Routine fordert ein Programm mehr Platz an, als das System bereitstellen kann. Die maximal verwendbare Speicherplatzgröße wird durch einen Systemparameter vergeben. Der Fehler kann auch dann auftreten, wenn die Anordnung von Text, Daten und Kellersegmenten zuviele Segmentregister erfordert, oder wenn während der Ausführung einer `fork`-Routine kein ausreichender Swap-Bereich zur Verfügung steht. Wenn dieser Fehler bei einem Gerät in Zusammenhang mit gemeinsamer Benutzung von fernen Dateien (RFS) auftritt, wird dadurch ein Mangel an Speicherplatz angezeigt, der in Abhängigkeit von der Systemaktivität zum Zeitpunkt des Systemaufrufs vorübergehend sein kann.

13 EACCES Zugriff untersagt
Auf eine Datei wurde in einer Weise zugegriffen, die durch die Dateizugriffsrechte untersagt ist.

14 EFAULT Adresse oder Parameter ungültig
Das System hat beim Versuch, ein Argument einer Routine zu benutzen, einen Hardwarefehler erzeugt. errno kann diesen Wert beispielsweise dann erhalten, wenn bei einer Routine, die ein Zeigerargument vorsieht, eine ungültige Adresse übergeben wird, und zwar zu jedem beliebigen Zeitpunkt, zu dem das System diese Bedingung erkennen kann. Da sich Systeme in ihrer Fähigkeit, zuverlässig fehlerhafte Adressen zu erkennen, unterscheiden, gehört das Übergeben von fehlerhaften Adressen an eine Routine zu den Situationen mit einem undefinierten Verhalten.

15 ENOTBLK Nur bei blockorientierten Geräten möglich
Eine nicht blockorientierte Datei wurde angegeben, während ein blockorientiertes Gerät benötigt wird (z.B. bei einem Aufruf der mount-Routine).

16 EBUSY Gerät nicht verfügbar
Es wurde versucht, ein Gerät einzuhängen, das bereits eingehängt ist, oder es wurde versucht, ein Gerät auszuhängen, auf dem sich eine aktive Datei (offene Datei, aktuelle Datei, eingehängte Datei, aktives Textsegment) befindet. Diese Fehlersituation tritt auch dann auf, wenn versucht wird, die Prozesse zur Benutzerabrechnung zu aktivieren, obwohl diese bereits aktiv sind.

17 EEXIST Datei oder Dateiverzeichnis bereits vorhanden
Eine vorhandene Datei wurde in einem falschen Zusammenhang angegeben (z.B. beim Aufruf der link-Routine).

18 EXDEV Verweis über Gerätegrenzen hinaus
Es wurde versucht, einen Verweis auf eine Datei auf einem anderen Gerät zu erzeugen.

19 ENODEV Gerät existiert nicht
Es wurde versucht, eine falsche Operation auf ein Gerät anzuwenden (z.B. Lesen von einem Gerät, auf das nur geschrieben werden darf).

20 ENOTDIR Ist kein Verzeichnis
Es wurde kein Dateiverzeichnis angegeben, wo ein Dateiverzeichnis benötigt wird (z.B. in einem Pfad-Präfix oder als Argument für chdir(2)).

21 EISDIR Ist ein Verzeichnis
Es wurde versucht, direkt in ein Dateiverzeichnis zu schreiben.

22 EINVAL Ungültiges Argument
Ein ungültiges Argument wurde verwendet (z.B. Aushängen eines nichteingehängten Geräts); ein undefiniertes Signal wurde in signal(2) oder kill(2) verwendet.

23 ENFILE Tabelle der offenen Dateien voll
Die Systemtabelle für Dateien ist voll, d.h. SYS_OPEN Dateien sind geöffnet, und vorübergehend können keine weiteren Dateien geöffnet werden.

24 EMFILE Zuviele offene Dateien
Ein Prozeß darf höchstens OPEN_MAX Dateideskriptoren gleichzeitig verwenden.

25 ENOTTY Nur bei zeichenorientierten Gerätedateien möglich
Es wurde versucht, ein ioctl auf eine Datei anzuwenden, die keine zeichenorientierte Gerätedatei ist.

26 ETXTBSY Programm aktiv
Es wurde versucht, ein direkt ablauffähiges Programm auszuführen, das zu diesem Zeitpunkt zum Schreiben geöffnet war. Dies gilt ebenfalls bei einem Versuch, ein ausführbares Programm während seiner Ausführung zu löschen oder zu überschreiben.

27 EFBIG Datei zu lang
Die Länge einer Datei hat die maximale Dateigröße oder FCHR_MAX überschritten (siehe auch getrlimit(2)).

28 ENOSPC Plattenplatz erschöpft
Während des Schreibens ist kein freier Platz mehr auf dem Datenträger übrig. Bei fcntl ist das Setzen oder Aufheben von Datensatzsperren in einer Datei nicht möglich, weil das System keine weiteren Datensatz-Einträge übrig hat.

29 ESPIPE Ungültige Positionieranweisung
Ein lseek wurde in einer Pipeline verwendet.

30 EROFS Dateisystem nur zum Lesen
Es wurde versucht, eine Datei oder ein Dateiverzeichnis mit Schreibschutz zu ändern.

31 EMLINK Zu viele Verweise
Es wurde versucht, mehr als die maximal zulässige Anzahl von Verweisen (LINK_MAX) auf eine Datei zu legen.

32 EPIPE Pipeline unterbrochen
Es wurde versucht, in eine Pipeline zu schreiben, für die es keinen Prozeß zum Lesen der Daten gibt. Diese Bedingung erzeugt normalerweise ein Signal; der Fehler wird zurückgegeben, wenn dieses Signal ignoriert wird.

33 EDOM Mathematisches Argument außerhalb des Definitionsbereichs
Das Argument einer mathematischen Funktion (3M) liegt außerhalb des Definitionsbereichs.

34 ERANGE Resultat nicht darstellbar
Der Wert einer mathematischen Funktion (3M) kann innerhalb der Gerätegenauigkeit nicht dargestellt werden.

35 ENOMSG Keine Nachricht des gewünschten Typs
Es wurde versucht, eine Nachricht eines Typs zu empfangen, der in der angegebenen Nachrichten-Warteschlange nicht vorhanden ist (siehe auch msgop(2)).

36 EIDRM Bezeichner gelöscht
Dieser Fehler wird an Prozesse zurückgegeben, die die Ausführung aufgrund der Entfernung eines Bezeichners aus dem Wertebereich der Namen des Dateisystems wiederaufnehmen (siehe msgctl(2), semctl(2) und shmctl(2)).

37 ECHRNG Kanalnummer nicht im zulässigen Bereich

38 EL2NSYNC Ebene 2 nicht synchronisiert

39 EL3HLT Ebene 3 beendet

40 EL3RST Reset auf Ebene 3

41 ELNRNG Verweisnummer nicht im zulässigen Bereich

42 EUNATCH Protokolltreiber nicht eingerichtet

43 ENOCSI Keine CSI-Struktur verfügbar

44 EL2HLT Ebene 2 beendet

45 EDEADLK Gefahr eines Deadlocks
Ein Deadlock wurde festgestellt und vermieden. Dieser Fehler bezieht sich auf Datei- und Datensatzsperren.

46 ENOLCK Keine Datensatzsperren verfügbar
Es sind keine weiteren Sperren verfügbar. Die Systemtabelle für Sperren ist voll (siehe fcntl(2)).

47–49 Reserviert

58–59 Reserviert

60 ENOSTR Kein Stream
Es wurde versucht, die Systemaufrufe putmsg oder getmsg auf eine Datei anzuwenden, die keine STREAM-Datei ist.

61 ENODATA Keine Daten verfügbar

62 ETIME Zeituhr abgelaufen
Die für einen STREAMS-Systemaufruf ioctl gesetzte Zeituhr ist abgelaufen. Die Ursache für diesen Fehler ist gerätespezifisch und kann entweder einen Hardware- oder einen Software-Fehler anzeigen oder möglicherweise auch einen Zeitwert, der für die bestimmte Funktion zu klein ist. Der Status der ioctl-Funktion ist unbestimmt.

63 ENOSTR Keine weiteren Betriebsmittel für STREAMS verfügbar
Während eines STREAMS open standen entweder keine STREAMS-Warteschlangen oder keine STREAMS-Kopfdatenstrukturen zur Verfügung. Dies ist eine vorübergehende Fehlersituation; sobald andere Prozesse Ressourcen freigeben, ist der Fehler behoben.

64 ENONET Rechner nicht an das Netzwerk angeschlossen
Dieser Fehler bezieht sich speziell auf die gemeinsame Benutzung ferner Dateien (RFS). Er tritt auf, wenn der Benutzer versucht, ferne Betriebsmittel anzumelden, abzumelden, ein- oder auszuhängen, und der Rechner hat die Startprozeduren zum Anschluß des Netzwerks (noch) nicht durchgeführt.

65 ENOPKG Paket nicht installiert
Dieser Fehler tritt auf, wenn der Benutzer die Verwendung eines Systemaufrufs versucht, der Teil eines nicht installierten Pakets ist.

66 EREMOTE Betriebsmittel ist nicht lokal
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt auf, wenn der Benutzer die Anmeldung eines Betriebsmittels versucht, das sich nicht im lokalen Rechner befindet, oder wenn das Einhängen/Aushängen eines Geräts (oder Pfadnamens) versucht wird, das (der) sich auf einem fernen Rechner befindet.

67 ENOLINK Virtuelle Verbindung verloren
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt auf, wenn die (virtuelle) Verbindung mit einem fernen Rechner verlorengeht.

68 EADV Anmeldefehler
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt dann auf, wenn der Benutzer ein Betriebsmittel anzumelden versucht, das bereits angemeldet worden ist, oder wenn versucht wird, RFS zu stoppen, während noch Betriebsmittel angemeldet sind, oder wenn das Aushängen eines Gerätes versucht wird, während dieses noch angemeldet ist.

69 ESRMNT Srmount-Fehler
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt dann auf, wenn ein Benutzer versucht, RFS anzuhalten, während noch Betriebsmittel auf fernen Rechnern eingehängt sind, oder wenn ein Betriebsmittel wieder angemeldet wird, und dabei eine Benutzerliste verwendet, die einen der fernen Rechner nicht enthält, der derzeit das Betriebsmittel eingehängt hat.

70 ECOMM Kommunikationsfehler beim Senden
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt auf, wenn der aktuelle Prozeß auf eine Nachricht von einem fernen Rechner wartet und die virtuelle Verbindung scheitert.

71 EPROTO Protokollfehler
Dieser Fehler ist gerätespezifisch, steht im allgemeinen jedoch nicht in Zusammenhang mit einem Hardware-Ausfall.

74 EMULTIHOP Kein Überspringen möglich
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Er tritt dann auf, wenn Benutzer einen Zugriff auf ferne Betriebsmittel versuchen, auf die man nicht direkt zugreifen kann.

76 EDOTDOT
Dieser Fehler betrifft die gemeinsame Benutzung ferner Dateien (RFS). Dies ist ein Weg für den Server, dem Client mitzuteilen, daß ein Prozeß von einem Einhängepunkt zurückübertragen wurde.

77 EBADMSG Keine zulässige Daten-Nachricht
Bei einem Systemaufruf `read`, `getmsg` oder `ioctl I_RECVFD` auf einem STREAMS-Gerät ist etwas an den Kopf der Warteschlange gelangt, das nicht verarbeitet werden kann. Was dies ist, hängt jeweils vom verwendeten Systemaufruf ab:

`read`	– Steuerinformationen oder ein weitergereichter Dateideskriptor
`getmsg`	– ein weitergereichter Dateideskriptor
`ioctl`	– Steuer- oder Dateninformationen

78 ENAMETOOLONG Dateiname zu lang
Die Länge eines Pfadnamen-Arguments überschreitet `PATH_MAX` Zeichen, oder die Länge überschreitet `NAME_MAX`, und `_POSIX_NO_TRUNC` wirkt; (siehe `limits`(4)).

79 EOVERFLOW Wert ist für den definierten Datentyp zu groß

80 ENOTUNIQ Name im Netzwerk nicht eindeutig

81 EBADFD Dateideskriptor in falschem Zustand
Entweder verweist ein Dateideskriptor nicht auf eine geöffnete Datei, oder ein Lesezugriff wurde auf eine Datei versucht, die nur zum Schreiben geöffnet wurde.

82 EREMCHG Adresse für fernen Zugriff geändert

83 ELIBACC Kein Zugriff auf benötigte, gemeinsam benutzte Bibliothek möglich
Es wurde versucht, mit `exec` ein `a.out` zu starten, das eine gemeinsam benutzte Bibliothek benötigt, und die gemeinsam benutzte Bibliothek ist nicht vorhanden, bzw. der Benutzer hat keine Erlaubnis für ihre Benutzung.

84 ELIBBAD Zugriff auf beschädigte, gemeinsam benutzte Bibliothek
Es wurde versucht, mit `exec` ein `a.out` zu starten, das eine gemeinsam benutzte Bibliothek benötigt (um dazugebunden zu werden), und `exec` konnte die gemeinsam benutzte Bibliothek nicht laden. Die gemeinsam benutzte Bibliothek ist wahrscheinlich beschädigt.

85 ELIBSCN `.lib`-Abschnitt in `a.out` beschädigt
Es wurde versucht, mit `exec` ein `a.out` auszuführen, das eine gemeinsam benutzte Bibliothek (zum Einbinden) benötigt, und in dem `.lib`-Abschnitt von `a.out` waren fehlerhafte Daten vorhanden. Dem `.lib`-Abschnitt entnimmt `exec` die Information, welche gemeinsam benutzten Bibliotheken benötigt werden.

86 ELIBMAX Versuch, mehr gemeinsam benutzte Bibliotheken als zulässig einzubinden
Es wurde versucht, mit `exec` ein `a.out` auszuführen, das mehr gemeinsam benutzte Bibliotheken benötigt als auf Grundlage der aktuellen Konfiguration des Systems zulässig sind.

87 ELIBEXEC Kein direktes Ausführen einer gemeinsam benutzten Bibliothek möglich
Es wurde versucht, eine gemeinsam benutzte Bibliothek direkt mit exec auszuführen.

88 EILSEQ Unzulässige Bytesequenz
Behandlung mehrerer Zeichen als einzelnes Zeichen

89 ENOSYS Operation nicht anwendbar

90 ELOOP
Anzahl der symbolischen Verweise während der Traversierung eines Pfades überschreitet MAXSYMLINKS

91 ESTART
Unterbrochener Systemaufruf sollte erneut gestartet werden.

92 ESTRPIPE Fehler in Streams-Pipe (nicht extern sichtbar).

93 ENOTEMPTY Verzeichnis nicht leer

94 EUSERS Zu viele Benutzer.

95 ENOTSOCK Socket-Operation auf nicht-Socket

96 EDESTADDRREQ Zieladresse benötigt
Eine benötigte Adresse für eine Operation auf einem Endpunkt einer Transportverbindung war nicht angegeben. Eine Zieladresse wird benötigt.

97 EMSGSIZE Nachricht zu lang
Eine Nachricht, die an den Diensterbringer der Transportverbindung gesendet wurde, war größer als der interne Nachrichtenpuffer oder irgend eine andere Begrenzung im Netzwerk.

98 EPROTOTYPE Falscher Protokolltyp für Socket
Es wurde ein Protokoll angegeben, das die Semantik des geforderten Socket-Typs nicht unterstützt.

99 ENOPROTOOPT Protokoll nicht verfügbar
Eine falsche Version oder eine falsche Ebene wurde beim Abfragen oder beim Setzen von Protokolloptionen verwendet.

120 EPROTONOSUPPORT Protokoll nicht unterstützt
Das Protokoll ist im aktuellen System nicht konfiguriert, oder es existiert dafür keine Unterstützung.

121 ESOCKTNOSUPPORT Socket-Typ nicht unterstützt
Für den angegebenen Socket-Typ ist das aktuelle System nicht konfiguriert, oder es existiert dafür keine Unterstützung.

122 EOPNOTSUPP Operation am Endpunkt der Transportverbindung nicht unterstützt
Beispielsweise der Versuch, eine Verbindung auf einem Endpunkt einer datagram-Transportverbindung zu akzeptieren.

123 EPFNOSUPPORT Protokollfamilie nicht unterstützt
Die Protokollfamilie ist im aktuellen System nicht konfiguriert, oder es exisitiert dafür keine Implementierung. Der Fehler tritt bei Internet-Protokollen auf.

124 EAFNOSUPPORT Adreßfamile durch Protokollfamilie nicht unterstützt
Eine mit dem angeforderten Protokoll nicht kompatible Adresse wurde verwendet.

125 EADDRINUSE Adresse bereits verwendet
Der Benutzer versuchte, eine Adresse zu verwenden, die bereits verwendet wurde, obwohl das Protokoll dies nicht zuläßt.

126 EADDRNOTAVAIL Angegebene Adresse kann nicht verwendet werden
Es wurde versucht, einen Endpunkt für eine Transportverbindung mit einer Adresse einzurichten, die nicht auf dem aktuellen Rechner liegt.

127 ENETDOWN Netzwerk ist nicht aktiv
Eine Operation fand ein inaktives Netzwerk vor.

128 ENETUNREACH Netzwerk nicht erreichbar
Eine Operation wurde an ein nicht erreichbares Netzwerk gerichtet.

129 ENETRESET Verbindung wurde aufgrund eines Rücksetzens abgebrochen
Der Host, mit dem Sie verbunden sind, ist abgestürzt und wurde neu geladen.

130 ECONNABORTED Abbruch der Verbindung durch Software
Aus internen Gründen Ihres Hosts bricht die Verbindung ab.

131 ECONNRESET Verbindung durch Kommunikationspartner rückgesetzt
Der Kommunikationspartner hat die Verbindung abgebrochen. Normalerweise wird dies aufgrund einer Zeitüberschreitung oder aufgrund eines Neustarts des Rechners ausgelöst.

132 ENOBUFS Kein Speicherplatz für Datenpuffer verfügbar
Eine Operation auf einem Endpunkt einer Transportverbindung konnte aufgrund mangelnden Speicherplatzes für Datenpuffer oder aufgrund einer vollen Warteschlange nicht ausgeführt werden.

133 EISCONN Endpunkt einer Transportverbindung ist bereits verbunden
Es wurde eine Verbindung angefordert oder mit `sendto` oder `sendmsg` eine Zieladresse angegeben, obwohl der Endpunkt bereits verbunden war.

134 ENOTCONN Endpunkt einer Transportverbindung nicht verbunden
Die Anforderung nach Senden oder Empfangen von Daten konnte nicht erfüllt werden, da der Kommunikationsendpunkt nicht verbunden war und (beim Senden von datagrams) keine Adresse angegeben wurde.

143 ESHUTDOWN Kein Senden nach Schließen des Endpunkts einer Transportverbindung
Das geforderte Senden von Daten war nicht möglich, da der Kommunikationsendpunkt bereits geschlossen wurde.

144 ETOOMANYREFS Zu viele Referenzen: Verbinden nicht möglich

145 ETIMEDOUT Zeitüberschreitung für Verbindung
Ein Verbindungsaufbau oder eine Sendeanforderung scheiterte, weil der Kommunikationspartner nicht ordnungsgemäß innerhalb der vorgegebenen Zeit antwortete. Das Zeitintervall ist abhängig vom Kommunikationsprotokoll.

146 ECONNREFUSED Verbindung verweigert
Der Verbindungsaufbau war nicht möglich, weil der Zielrechner diese verweigert. Dieser Fehler entsteht üblicherweise beim Versuch, eine Verbindung zu einem Dienst herzustellen, der auf dem fernen Rechner nicht aktiv ist.

147 EHOSTDOWN Host ist nicht aktiv
Eine Operation des Transportdienstgebers ist gescheitert, weil der Zielrechner nicht aktiv ist.

148 EHOSTUNREACH Keine Verbindung zu Host bekannt
Eine Operation des Transportdienstgebers versuchte, einen nicht erreichbaren Host zu verwenden.

149 EALREADY Operation bereits aktiv
Es wurde versucht, eine weitere Operation auf einem nichtblockierenden Objekt zu starten, das bereits eine Operation bearbeitet.

150 EINPROGRESS Operation jetzt aktiv
Es wurde versucht, eine Operation, die eine lange Zeit benötigt (wie zum Beispiel `connect`), auf einem nichtblockierenden Objekt auszuführen.

151 ESTALE Unwirksamer NFS-Dateibezeichner

Besondere Prozesse

Die Prozesse mit den Prozeßnummern 0 und 1 sind Sonderprozesse, sie werden als `proc0` und `proc1` bezeichnet; siehe auch `kill`(2). `proc0` ist der Prozeß-Scheduler. `proc1` ist der Initialisierungsprozeß (`init`). `proc1` ist der Ur-Vaterprozeß jedes anderen Prozesses im System und wird zur Steuerung der Prozeßstruktur verwendet.

Bezeichner für gemeinsam benutzten Speicher

Ein Bezeichner für gemeinsam benutzten Speicher (*shmid* - shared memory identifier) ist eine positive, ganze Zahl, die von einem Systemaufruf `shmget` bereitgestellt wird. Jeder *shmid* repräsentiert ein Speichersegment (das als gemeinsam benutztes Speichersegment bezeichnet wird) und eine zugehörige Datenstruktur. Es ist zu beachten, daß diese gemeinsam benutzten Speichersegmente durch den Benutzer explizit entfernt werden müssen, nachdem der letzte Verweis in diese gelöscht worden ist. Die Datenstruktur wird mit `shmid_ds` bezeichnet und enthält folgende Elemente:

```
struct ipc_perm   shm_perm;     /* Struktur für Zugriffsrechte der Operationen */
int               shm_segsz;    /* Größe des Segments */
struct anon-map   *shm_amp;     /* Zeiger auf Bereichsstruktur */
ushort            shm_lkcnt;    /* Anzahl der gleichzeitigen Sperren */
pid_t             shm_lpid;     /* Prozeßnummer der letzten Operation */
pid_t             shm_cpid;     /* Prozeßnummer des Erstellers */
ulong             shm_nattch;   /* Anzahl der aktuellen Anschlüsse */
ulong             shm_cnattch;  /* Verwendung nur für shminfo */
time_t            shm_atime;    /* Zeitpunkt des letzten Anschlusses */
long              shm_pad1;     /* reserviert */
time_t            shm_dtime;    /* Zeitpunkt der letzten Trennung */
long              shm_pad2;     /* reserviert */
time_t            shm_ctime;    /* Zeitpunkt der letzten Änderung */
                                /* Zeiten gemessen in Sekunden seit */
                                /* 00:00:00 GMT, 1. Januar 1970 */
long              shm_pad3;     /* reserviert */
long              shm_pad4[4];  /* reserviert */
```

Es folgt eine Beschreibung der einzelnen Felder der `shmid_ds`-Struktur:
`shm_perm` ist eine `ipc_perm`-Struktur zur Beschreibung der Zugriffsrechte für die Operationen auf dem gemeinsam benutzten Speicher (siehe unten). Diese Struktur besteht aus folgenden Elementen:

```
uid_t             cuid;         /* Benutzernummer des Erstellers */
gid_t             cgid;         /* Gruppennummer des Erstellers */
uid_t             uid;          /* Benutzernummer */
gid_t             gid;          /* Gruppennummer */
mode_t            mode;         /* Schreib-/Lese-Erlaubnis */
ulong             seq;          /* Abfolgenummer für Platzbenutzung # */
key_t             key;          /* Schlüssel */
long              pad[4];       /* reserviert */
```

`shm_segsz` die Größe des gemeinsam benutzten Speichersegments in Bytes an.

`shm_cpid` die Prozeßnummer des Prozesses, der den Bezeichner des gemeinsam benutzten Speichers bereitgestellt hat.

`shm_lpid` Prozeßnummer des letzten Prozesses, der eine `shmop`-Operation ausgeführt hat.

`shm_nattch` Anzahl der Prozesse, die mit diesem Segment momentan verbunden sind.

`shm_atime` Zeitpunkt der letzten `shmat`-Operation (siehe auch `shmop`(2)).

`shm_dtime` Zeitpunkt der letzten `shmdt`-Operation (siehe auch `shmop`(2)).

`shm_ctime` Zeitpunkt der letzten `shmctl`-Operation, bei der eines der Elemente der obigen Struktur geändert wurde.

Bezeichner für Nachrichten-Warteschlange

Der Bezeichner einer Nachrichten-Warteschlange (msqid) ist eine positive ganze Zahl, die von einem Systemaufruf msgget zurückgeliefert wird. Jede msqid hat eine zugehörige Nachrichten-Warteschlange und eine Datenstruktur. Die Datenstruktur wird als msqid_ds bezeichnet und enthält folgende Teile:

```
struct ipc_perm   msg_perm;
struct msg        *msg_first;
struct msg        *msg_last;
ulong             msg_cbytes;
ulong             msg_qnum;
ulong             msg_qbytes;
pid_t             msg_lspid;
pid_t             msg_lrpid;
time_t            msg_stime;
long              msg_pad1;       /* reserviert */
time_t            msg_rtime;
long              msg_pad2;       /* reserviert */
time_t            msg_ctime;
long              msg_pad3;       /* reserviert */
long              msg_pad4[4];    /* reserviert */
```

Es folgt eine Beschreibung der Komponenten der msgid_ds-Struktur:

msg_perm ist eine ipc_perm-Struktur, die die Rechte für den Nachrichtenbetrieb (siehe unten) angibt. Diese Struktur weist folgende Elemente auf:

```
uid_t             cuid;           /* Benutzernummer des Erstellers */
gid_t             cgid;           /* Gruppennummer des Erstellers */
uid_t             uid;            /* Benutzernummer */
gid_t             gid;            /* Gruppennummer */
mode_t            mode;           /* Schrein-/Lese-Erlaubnis */
ulong             seq;            /* Belegungsliste der Einträge */
key_t             key;            /* Schlüssel */
long              pad[4];         /* reserviert */
```

*msg_first	Zeiger auf die erste Nachricht in der Warteschlange.
*msg_last	Zeiger auf die letzte Nachricht in der Warteschlange.
msg_cbytes	aktuelle Anzahl der Bytes in der Warteschlange.
msg_qnum	Anzahl der sich zum jeweiligen Zeitpunkt in der Warteschlange befindenden Nachrichten.
msg_qbytes	Höchstzahl der in einer Warteschlange zugelassener Bytes.
msg_lspid	Prozeßnummer des letzten Prozesses, der eine msgsnd-Operation ausgeführt hat.
msg_lrpid	Prozeßnummer des letzten Prozesses, der eine msgrcv-Operation ausgeführt hat.
msg_stime	Zeitpunkt der letzten msgsnd-Operation.
msg_rtime	Zeitpunkt der letzten msgrcv-Operation.
msg_ctime	Zeitpunkt der letzten msgctl-Operation, die eines der Elemente der obigen Struktur geändert hat.

Dateideskriptor

Ein Dateideskriptor ist eine kleine, ganze Zahl, die zur Ausführung von E/A-Operationen auf einer Datei verwendet wird. Der Wert eines Dateideskriptors liegt im Bereich von 0 bis (NOFILES-1). Ein Prozeß darf gleichzeitig nicht mehr als NOFILES Dateideskriptoren geöffnet haben. Ein Dateideskriptor wird von Systemaufrufen, wie z.B. open oder pipe als Ergebnis zurückgegeben. Dateideskriptoren werden als Argument in Funktionen wie read, write, ioctl und close verwendet.

Dateiname

Dateinamen bestehen aus 1 bis NAME_MAX Zeichen. Sie können zur Bezeichnung einer normalen Datei, einer Gerätedatei oder eines Verzeichnisses dienen.

Diese Zeichen können aus dem Satz aller Zeichenwerte mit Ausnahme von \0 (Null) und dem ASCII-Code für / (Schrägstrich) ausgewählt werden.

Bitte beachten Sie, daß im Normalfall die Zeichen * , ? , [oder] nicht als Teil von Dateinamen verwendet werden sollten, weil die Shell diesen Zeichen jeweils eine besondere Bedeutung zumißt (siehe sh (1)). Auch die Verwendung nicht druckbarer Zeichen in Dateinamen sollte vermieden werden.

Ein Dateiname wird manchmal als Komponente eines Pfadnamens verwendet. Die Interpretation einer Komponente eines Pfadnamens ist abhängig von den Werten von NAME_MAX und _POSIX_NO_TRUNC, angewendet auf den Präfix einer solchen Komponente. Wenn irgendeine Komponente länger als NAME_MAX Zeichen ist, und _POSIX_NO_TRUNC für den Pfad-Präfix dieser Komponente effektiv ist (siehe dazu fpathconf(2) und limits(4)), so sollte dies als Fehlerbedingung angesehen werden. Andernfalls sollte die Implementierung die ersten NAME_MAX Bytes der Komponente des Pfadnamens verwenden.

Dateizugriffsrechte

Die Erlaubnis zum Lesen, Schreiben und Ausführen/Durchsuchen einer Datei wird einem Prozeß erteilt, wenn einer oder mehrere der nachstehenden Punkte erfüllt sind:

- Die effektive Benutzernummer des Prozesses ist die des Systemverwalters.
- Die effektive Benutzernummer des Prozesses entspricht der Benutzernummer des Eigentümers der Datei, und das entsprechende Zugriffsbit des 'Eigentümer'-Teils (0700) des Dateimodus ist gesetzt.
- Die effektive Benutzernummer des Prozesses entspricht nicht der Benutzernummer des Eigentümers der Datei, aber die effektive Gruppennummer des Prozesses entspricht der Dateigruppe, und das entsprechende Zugriffsbit des 'Gruppen'-Teils (0070) des Dateimodus ist gesetzt.
- Die effektive Benutzernummer des Prozesses entspricht nicht der Benutzernummer des Eigentümers der Datei, aber die effektive Gruppennummer des Prozesses entspricht nicht der Gruppennummer der Datei, und das entsprechende Zugriffsbit des 'Andere'-Teils (0007) des Dateimodus ist gesetzt.

Effektive Benutzernummer und effektive Gruppennummer

Ein aktiver Prozeß hat eine effektive Benutzernummer und eine effektive Gruppennummer. Sie dienen zur Bestimmung der Rechte für Dateizugriffe (siehe unten). Die effektive Benutzernummer und die effektive Gruppennummer entsprechen der realen Benutzernummer des Prozesses bzw. der realen Gruppennummer, wenn der Prozeß oder einer seiner Vorgänger nicht aus einer Datei entstanden ist, bei der das s-Bit (das Bit für die Benutzereinstellung oder für die Gruppennummerneinstellung) gesetzt wurde (siehe auch `exec`(2)).

Hintergrund-Prozeßgruppe

Jede Prozeßgruppe, die keine Vordergrund-Prozeßgruppe mit einer Verbindung zu einem steuernden Terminal ist.

Lebensdauer eines Prozesses

Die Lebensdauer eines Prozesses beginnt mit seiner Erzeugung durch `fork` und endet mit seinem Ende mit `exit`, sobald seine Beendigung durch seinen Vaterprozeß bestätigt wurde. Siehe dazu auch `wait`(2).

Lebensdauer einer Prozeßgruppe

Die Lebensdauer einer Prozeßgruppe beginnt mit ihrer Erzeugung durch ihren Prozeßgruppenleiter und endet mit dem Ende der Lebensdauer des letzten Prozesses der Prozeßgruppe, oder wenn der letzte Prozeß die Gruppe verläßt.

Lebensdauer einer Sitzung

Die Lebensdauer einer Sitzung beginnt mit der Erzeugung der Sitzung durch den Sitzungsleiter und endet mit dem Ende der Lebensdauer des letzten Prozesses der Sitzung, oder wenn der letzte Prozeß die Sitzung verläßt.

Lese-Warteschlange

Die Nachrichten-Warteschlange in einem Modul oder einem Treiber eines Streams, die Nachrichten enthält, die sich stream-aufwärts bewegen.

Modul

Ein Modul ist eine Einheit, die Verarbeitungsroutinen für Eingabe- und Ausgabedaten enthält. Es existiert immer in der Mitte eines Streams zwischen dem Kopf des Streams und einem Treiber. Ein Modul ist das STREAMS-Gegenstück zu den Kommandos in einer Shell-Pipeline, wobei ein Modul jedoch ein Funktionspaar enthält, das einen unabhängigen bidirektionalen Datenfluß (stream-abwärts und stream-aufwärts) sowie die Bearbeitung der Daten ermöglicht.

Multiplexer

Ein Multiplexer ist ein Treiber, der den Anschluß von Streams, die mit mehreren Benutzerprozessen verbunden sind, an einen einzigen Treiber, oder den Anschluß von mehreren Treibern an einen einzigen Benutzerprozeß ermöglicht. STREAMS liefert keinen allgemeinen Multiplexer-Treiber, stellt jedoch Einrichtungen für dessen Konstruktion und für den Anschluß von Stream-Konfigurationen in Multiplexschaltungen zur Verfügung.

Nachricht

Einer oder mehrere Daten- oder Informationsblöcke mit den zugehörigen STREAMS-Steuerstrukturen in einem Stream. Nachrichten können von verschiedenen definierten Typen sein, die jeweils den Inhalt der Nachricht kennzeichnen. Mit Nachrichten werden Kommunikations-Daten innerhalb eines Streams übertragen.

Nachrichten-Warteschlange

Eine verkettete Liste von Nachrichten in einem Stream, die auf die Verarbeitung durch ein Modul oder einen Treiber warten.

Pfadname

Ein Pfadname ist eine mit einem Nullzeichen endende Zeichenkette, die mit einem optionalen Schrägstrich /, beginnt, gefolgt von Null oder mehr Dateiverzeichnisnamen, die durch Schrägstriche voneinander getrennt sind. Wahlweise kann danach noch ein Dateiname folgen.

Wenn ein Pfadname mit einem Schrägstrich beginnt, beginnt die Pfadsuche beim Root-Verzeichnis. Andernfalls beginnt die Suche beim aktuellen Dateiverzeichnis.

Ein alleinstehender Schrägstrich bezeichnet das Root-Dateiverzeichnis.

Wenn nicht ausdrücklich anders angegeben, wird ein leerer Pfadname so behandelt, als ob er eine nicht vorhandene Datei benennen würde.

Privilegien

Über ausreichende Privilegien zu verfügen, bedeutet, die Fähigkeit zu haben, Beschränkungen des Systems ignorieren zu können.

Prozeßgruppe

Jeder Prozeß im System ist Mitglied einer Prozeßgruppe, die durch eine Prozeßgruppennummer gekennzeichnet ist. Jeder Prozeß, der nicht Prozeßgruppenleiter ist, kann eine neue Prozeßgruppe erzeugen und deren Leiter werden. Jeder Prozeß, der nicht Prozeßgruppenleiter ist, kann sich einer existierenden Prozeßgruppe anschließen, die derselben Sitzung angehört wie der Prozeß. Ein neu erzeugter Prozeß gehört zu der Prozeßgruppe seines Vaterprozesses.

Prozeßgruppenleiter

Ein Prozeßgruppenleiter ist jeder Prozeß, dessen Prozeßgruppennummer gleich seiner Prozeßnummer ist.

Prozeßgruppennummer

Jeder aktive Prozeß ist Mitglied einer Prozeßgruppe, dargestellt durch eine positive, ganze Zahl, die Prozeßgruppennummer. Diese Nummer ist die Prozeßnummer des Prozeßgruppenleiters. Mit dieser Gruppierung ist das Senden von Signalen an zusammengehörige Prozesse möglich (siehe auch `kill(2)`).

Prozeßnummer

Jeder aktive Prozeß im System wird eindeutig durch eine positive ganze Zahl gekennzeichnet, die Prozeßnummer. Eine Prozeßnummer kann durch das System nicht wieder verwendet werden, bevor nicht die Lebensdauer des Prozesses, der Prozeßgruppe und der Sitzung für jede Prozeßnummer, Prozeßgruppe und für jede Sitzungsnummer gleich der fraglichen Prozeßnummer beendet ist.

Prozeßnummer des Vaterprozesses

Ein neuer Prozeß wird von einem gegenwärtig aktiven Prozeß erzeugt (siehe auch `fork`(2)). Die Vaterprozeßnummer eines Prozesses ist die Prozeßnummer des jeweiligen Erzeugers.

Reale Benutzerkennung und reale Gruppenkennung

Jeder am System zugelassene Benutzer wird durch eine postive, ganze Zahl (0 bis `MAXUID`) identifiziert, der sogenannten realen Benutzerkennung.

Jeder Benutzer ist auch Mitglied einer Gruppe. Die Gruppe wird durch eine positive ganze Zahl gekennzeichnet, die als reale Gruppennummer bezeichnet wird.

Ein aktiver Prozeß hat eine reale Benutzernummer und eine reale Gruppennummer, die auf die reale Benutzernummer bzw. die reale Gruppennummer des Benutzers gesetzt werden, der für die Erstellung des Prozesses verantwortlich ist.

Root-Dateiverzeichnis und aktuelles Arbeitsverzeichnis

Zu jedem Prozeß gehört das Konzept eines Root-Dateiverzeichnisses und ein aktuelles Dateiverzeichnis, um Pfadnamen auflösen zu können. Das Root-Dateiverzeichnis eines Prozesses muß nicht das Root-Dateiverzeichnis des Root-Dateisystems zu sein.

Schreib-Warteschlange

Die Nachrichten-Warteschlange in einem Modul oder einem Treiber eines Streams, die Nachrichten enthält, die sich stream-abwärts bewegen.

Semaphor-Bezeichner

Ein Semaphor-Bezeichner (semid) ist eine positive ganze Zahl, die von einem Systemaufruf semget bereitgestellt wird. Jede semid verfügt über einen Satz Semaphore und eine damit verbundene Datenstruktur. Die Datenstruktur wird mit semid_ds bezeichnet und enthält folgende Elemente:

```
struct ipc_perm   sem_perm;      /* Struktur für Zugriffsrechte von Operationen */
struct sem        *sem_base;     /* Zeiger auf ersten Semaphor im Satz */
ushort            sem_nsems;     /* Anzahl der Semaphoren im Satz */
time_t            sem_otime;     /* Zeitpunkt der letzten Operation */
long              sem_pad1;      /* reserviert */
time_t            sem_ctime;     /* Zeitpunkt der letzten Änderung */
                                 /* Zeiten gemessen in Sekunden seit */
                                 /* 00:00:00 GMT, 1. Jan. 1970 */
long              sem_pad2;      /* reserviert */
long              sem_pad3[4];   /* reserviert */
```

sem_perm ist eine ipc_perm-Struktur, die die Zugriffsrechte für Semaphor-Operationen spezifiziert (siehe unten). Diese Struktur besteht aus folgenden Elementen:

```
uid_t             cuid;          /* Benutzernummer des Erstellers */
gid_t             cgid;          /* Gruppennummer des Erstellers */
uid_t             uid;           /* Benutzernummer */
gid_t             gid;           /* Gruppennummer */
mode_t            mode;          /* Schrein-/Lese-Erlaubnis */
ulong             seq;           /* Belegungsliste der Einträge */
key_t             key;           /* Schlüssel */
long              pad[4];        /* reserviert */
```

sem_nsems — Anzahl der Semaphore im Satz. Jeder Semaphor im Satz wird durch eine positive, ganze Zahl referenziert, die mit sem_num bezeichnet wird. sem_num-Werte laufen sequentiell von 0 bis zum Wert von sem_nsems minus 1.

sem_otime — Zeitpunkt der letzten semop-Operation.

sem_ctime — Zeitpunkt der letzten semctl-Operation, mit der ein Element der obigen Struktur verändert wurde.

Ein Semaphor ist eine Datenstruktur namens sem und enthält folgende Elemente:

```
ushort            semval;        /* Wert des Semaphors */
pid_t             sempid;        /* pid der letzten Operation */
ushort            semncnt;       /* # wartend auf semval > cval */
ushort            semzcnt;       /* # wartend auf semval = 0 */
```

semval — nichtnegative ganze Zahl, die den aktuellen Wert des Semaphors darstellt.

sempid — Prozeßnummer des letzten Prozesses, der eine Semaphor-Operation auf diesem Semaphor ausgeführt hat.

semncnt — Anzahl der Prozesse, die zum gegebenen Zeitpunkt zurückgestellt sind und darauf warten, daß semval dieses Semaphors größer als der aktuelle Wert wird.

semzcnt — Anzahl der Prozesse, die zum gegebenen Zeitpunkt vorübergehend zurückgestellt sind und darauf warten, daß semval dieses Semaphors gleich Null wird.

Sitzung

Eine Sitzung ist eine Gruppe von Prozessen, die durch eine gemeinsame Kennung, die Sitzungsnummer, gekennzeichnet ist. Sie ist in der Lage, eine Verbindung zu einem steuernden Terminal aufzubauen. Jeder Prozeß, der nicht Prozeßgruppenleiter ist, kann eine neue Sitzung und eine neue Prozeßgruppe erzeugen und wird deren Sitzungsleiter und Prozeßgruppenleiter. Ein neu erzeugter Prozeß gehört zu der Sitzung seines Erzeugers.

Sitzungsleiter

Ein Sitzungsleiter ist ein Prozeß, dessen Sitzungsnummer gleich seiner Prozeßnummer und seiner Prozeßgruppennummer ist.

Sitzungsnummer

Jede Sitzung im System wird durch eine positive ganze Zahl gekennzeichnet, die Sitzungsnummer ist gleich der Prozeßnummer des Sitzungsleiters.

Steuernder Prozeß

Ein Sitzungsleiter, der die Verbindung zu dem steuernden Terminal hergestellt hat.

Steuerndes Terminal

Ein Terminal, das mit einer Sitzung verbunden ist. Jede Sitzung muß mindestens ein steuerndes Terminal zugeordnet haben, und ein steuerndes Terminal kann maximal einer Sitzung zugeordnet sein. Bestimmte Eingabefolgen vom steuernden Terminal führen dazu, daß Signale an die Prozeßgruppe der Sitzung gesendet werden, die dem steuernden Terminal zugeordnet ist; siehe dazu `termio`(7).

STREAMS

Ein Satz von Mechanismen im Systemkern zur Unterstützung der Realisierung der Netzwerkdienste und Datenkommunikations-Treiber. Er definiert Schnittstellen-Standards für Zeichen-Ein-/Ausgabe innerhalb des Systemkerns und zwischen dem Systemkern und den Prozessen auf Benutzerebene. Der STREAMS-Mechanismus setzt sich aus Dienstprogramm-Routinen, Systemkern-Einrichtungen und einem Satz von Datenstrukturen zusammen.

Stream

Ein Stream ist ein Vollduplex-Datenpfad innerhalb des Systemkerns zwischen einem Benutzerprozeß und Treiberroutinen. Die Hauptkomponenten sind ein Stream-Kopf, ein Treiber und Null oder mehr Module zwischen dem Stream-Kopf und dem Treiber. Ein Stream entspricht einer Shell-Pipeline, wobei jedoch Datenfluß und Verarbeitung bidirektional sind.

Stream-Kopf

In einem Stream bildet der Stream-Kopf jenes Ende des Streams, das die Schnittstelle zwischen dem Stream und einem Benutzerprozeß darstellt. Die Hauptfunktionen des Stream-Kopfes bestehen in der Verarbeitung von STREAMS-bezogenen Systemaufrufen und der Weitergabe von Daten und Informationen zwischen einem Benutzerprozeß und dem Stream.

Stream-abwärts

In einem Stream die Richtung vom Stream-Kopf zum Treiber.

Stream-aufwärts

In einem Stream die Richtung vom Treiber zum Stream-Kopf.

Systemverwalter

Ein Prozeß wird als Systemverwalter-Prozeß anerkannt und erhält besondere Privilegien, wie zum Beispiel Unabhängigkeit von den Dateizugriffsrechten, wenn die effektive Benutzernummer gleich 0 ist.

Treiber

In einem Stream bildet der Treiber die Schnittstelle zwischen der peripheren Hardware und dem Stream. Ein Treiber kann auch ein Pseudo-Treiber sein, wie z.B. ein Multiplexer oder ein Protokoll-Treiber (siehe auch `log`(7)), der nicht direkt mit einer Geräteschnittstelle in Verbindung steht.

Verzeichnis

Dateiverzeichnisse organisieren Dateien in ein hierarchisches Dateisystem, wobei die Verzeichnisse die Knoten im System darstellen. In einem Dateiverzeichnis werden alle Dateien, einschließlich Verzeichnissen, die sich direkt unter ihm in der Hierarchie befinden, katalogisiert. Dateiverzeichnis-Einträge werden als Verweise bezeichnet. Es ist allgemein üblich, daß ein Dateiverzeichnis wenigstens zwei Verweise, '.' und '..' enthält. Der Verweis '.' verweist auf das Dateiverzeichnis selbst, und '..' verweist auf das übergeordnete Dateiverzeichnis. Das Root-Verzeichnis als oberster Knoten der Hierarchie verweist auf sich selbst als übergeordnetes Verzeichnis. Der Pfadname des Root-Verzeichnisses ist `/` und das übergeordnete Verzeichnis des Root-Verzeichnisses ist `/.`

Vordergrund-Prozeßgruppe

Jede Sitzung mit einem zugeordneten steuernden Terminal unterscheidet eine Prozeßgruppe der Sitzung als die Vordergrund-Prozeßgruppe des steuernden Terminals. Diese Gruppe hat einige Vorrechte bei Zugriffen auf das steuernde Terminal, die den Hintergrund-Prozeßgruppen untersagt sind.

Zugriffsrechte auf gemeinsam benutzten Speicher

Bei den Systemaufrufen `shmop` und `shmctl` wird die für eine Operation benötigte Erlaubnis wie folgt interpretiert:

```
00400   Lesen durch Benutzer
00200   Schreiben durch Benutzer
00040   Lesen durch Gruppe
00020   Schreiben durch Gruppe
00004   Lesen durch Andere
00002   Schreiben durch Andere
```

Lese- und Schreiberlaubnis auf einem *shmid* werden einem Prozeß gewährt, wenn einer oder mehrere der nachstehenden Punkte gegeben sind:

- Die effektive Benutzernummer des Prozesses entspricht der des Systemverwalters.
- Die effektive Benutzernummer des Prozesses entspricht `shm_perm.cuid` oder `shm_perm.uid` in der Datenstruktur zu *shmid*, und das entsprechende Bit des 'Benutzer'-Teils (0600) von `shm_perm.mode` ist gesetzt.
- Die effektive Gruppennummer des Prozesses entspricht `shm_perm.cgid` oder `shm_perm.gid`, und das entsprechende Bit des 'Gruppe'-Teils (060) von `shm_perm.mode` ist gesetzt.
- Das entsprechende Bit des 'Andere'-Teils (06) von `shm_perm.mode` ist gesetzt.
- Andernfalls wird die entsprechende Erlaubnis verweigert.

Zugriffsrechte für Nachrichten-Warteschlangen

In den Beschreibungen der Systemaufrufe `msgop` und `msgctl` wird die für eine Operation erteilte Erlaubnis wie folgt interpretiert wird:

```
00400   Lesen durch Benutzer
00200   Schreiben durch Benutzer
00040   Lesen durch Gruppe
00020   Schreiben durch Gruppe
00004   Lesen durch Andere
00002   Schreiben durch Andere
```

Die Schreib- und Lese-Erlaubnis auf einem `msqid` werden einem Prozeß gewährt, wenn einer oder mehrere der nachfolgenden Punkte gegeben sind:

- Die effektive Benutzernummer des Prozesses ist gleich der des Systemverwalters.
- Die effektive Benutzernummer des Prozesses ist gleich `msg_perm.cuid` oder `msg_perm.uid` in der Datenstruktur, die zu `msqid` gehört, und das entsprechende Bit des 'Benutzer'-Teils (0600) von `msg_perm.mode` ist gesetzt.
- Die effektive Gruppennummer des Prozesses entspricht `msg_perm.cgid` oder `msg_perm.gid`, und das entsprechende Bit des 'Gruppen'-Teils (060) von `msg_perm.mode` ist gesetzt.
- Das entsprechende Bit des 'Andere'-Teils (006) von `msg_perm.mode` ist gesetzt.

Andernfalls wird die entsprechende Erlaubnis nicht erteilt.

Zugriffsrechte für Semaphor-Operationen

In der Beschreibung der Systemaufrufe `semop` und `semctl` wird die für eine Operation benötigte Erlaubnis wie folgt interpretiert:

```
00400   Lesen durch Benutzer
00200   Ändern durch Benutzer
00040   Lesen durch Gruppe
00020   Ändern durch Gruppe
00004   Lesen durch Andere
00002   Ändern durch Andere
```

Die Erlaubnis, lesend und ändernd auf einen `semid` zuzugreifen, wird einem Prozeß gewährt, wenn einer oder mehrere der nachstehenden Punkte gegeben sind:

- Die effektive Benutzernummer des Prozesses entspricht der des Systemverwalters.
- Die effektive Benutzernummer des Prozesses entspricht `sem_perm.cuid` oder `sem_perm.uid` in der Datenstruktur zu `semid`, und das entsprechende Bit des 'Benutzer'-Teils (0600) von `sem_perm.mode` ist gesetzt.
- Die effektive Gruppennummer des Prozesses entspricht `sem_perm.cgid` oder `sem_perm.gid`, und das entsprechende Bit des 'Gruppe'-Teils (060) von `sem_perm.mode` ist gesetzt.
- Das entsprechende Bit des 'Andere'-Teils (006) von `sem_perm.mode` ist gesetzt.

Andernfalls wird die entsprechende Erlaubnis verweigert.

access - Zugriffsrechte auf eine Datei feststellen

```
#include <unistd.h>
int access(const char *Pfad, int amode);
```

Pfad weist auf einen Pfadnamen, der eine Datei benennt. access prüft die angegebene Datei auf Zugreifbarkeit entsprechend dem Bitmuster in *amode*, wobei die reale Benutzernummer anstelle der effektiven Benutzernummer und die reale Gruppennummer anstelle der effektiven Gruppennummer verwendet werden. Das in *amode* enthaltene Bitmuster wird durch eine ODER-Verknüpfung der folgenden Konstanten (definiert in unistd.h) konstruiert:

R_OK Prüfe auf Lese-Erlaubnis

W_OK Prüfe auf Schreib-Erlaubnis

X_OK Prüfe auf Ausführungs-(Durchsuch-)Erlaubnis

F_OK Prüfe die Existenz der Datei

Der Zugriff auf die Datei wird verweigert, wenn einer oder mehrere der folgenden Punkte zutreffen:

EACCES Eine Komponente des Pfades darf nicht durchsucht werden.

EACCES Die Schutzbiteinstellung der Datei läßt den geforderten Zugriff nicht zu.

EFAULT *Pfad* liegt nicht im Adreßraum des Prozesses.

EINTR Ein Signal wurde während des access-Systemaufrufs abgefangen.

ELOOP Es wurden zu viele symbolische Verweise im übersetzten *Pfad* vorgefunden.

EMULTIHOP Einige Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner.

ENAMETOOLONG
Die Länge des *Pfad*-Arguments ist größer als PATH_MAX, oder die Länge einer *Pfad*-Komponente übertrifft NAME_MAX, während _POSIX_NO_TRUNC gültig ist.

ENOTDIR Eine Komponente des Pfades ist kein Dateiverzeichnis.

ENOENT Eine Lese-, Schreib- oder Ausführ- (Durchsuch-) Erlaubnis wird für einen leeren Pfadnamen angefordert.

ENOENT	Die angegebene Datei ist nicht vorhanden.
ENOLINK	*Pfad* verweist auf ein fernes System, und die Verbindung dorthin ist nicht mehr aktiv.
EROFS	Schreibzugriff wurde für eine Datei auf einem Nur-Lesen-Dateisystem angefordert.

SIEHE AUCH

chmod(2), stat(2).
Abschnitt "Dateizugriffsrechte" in intro(2).

ERGEBNIS

Bei Erlaubnis des angeforderten Zugriffs wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird gesetzt, um den Fehler anzuzeigen.

acct - Prozeßabrechnung ein- oder ausschalten

```
#include <unistd.h>
int acct(const char *Pfad);
```

acct schaltet die System-Prozeßabrechnungsroutine ein oder aus. Bei eingeschalteter Routine wird ein Abrechnungssatz für jeden beendeten Prozeß in eine Abrechnungsdatei geschrieben. Die Beendigung kann durch zwei Möglichkeiten verursacht werden: durch einen exit-Aufruf oder ein Signal (siehe exit(2) und signal(2)). Die effektive Benutzernummer des Prozesses, der acct aufruft, muß die Nummer des Systemverwalters sein, damit dieser Aufruf ausgeführt werden kann.

Pfad weist auf einen Pfadnamen, der die Abrechnungsdatei benennt. Das Format der Abrechnungsdatei wird in acct(4) angegeben.

Die Abrechnungsroutine wird eingeschaltet, wenn *Pfad* ungleich (char *)NULL ist und während des Systemaufrufs keine Fehler auftreten. Sie wird abgeschaltet, wenn *Pfad* Null ist und während des Systemaufrufs keine Fehler auftreten.

acct ist erfolglos, wenn einer oder mehrere der folgenden Punkte zutreffen:

EACCES — Die in *Pfad* angegebene Datei ist keine normale Datei.

EBUSY — Ein Versuch zum Einschalten der Prozeßabrechnung wird unternommen, während diese bereits eingeschaltet ist.

EFAULT — *Pfad* weist auf eine ungültige Adresse.

ELOOP — Es wurden zu viele symbolische Verweise im übersetzten *Pfad* vorgefunden.

ENAMETOOLONG — Die Länge des *Pfad*-Arguments ist größer als PATH_MAX, oder die Länge einer *Pfad*-Komponente übertrifft NAME_MAX, während _POSIX_NO_TRUNC gültig ist.

ENOTDIR — Eine Komponente des Pfades ist kein Dateiverzeichnis.

ENOENT — Eine oder mehrere Komponenten des Pfadnamens der Abrechnungsdatei existieren nicht.

EPERM — Der effektive Benutzer des aufrufenden Prozesses ist nicht der Systemverwalter.

EROFS — Die angegebene Datei steht in einem schreibgeschützten Dateisystem.

SIEHE AUCH

`exit`(2), `signal`(2).
`acct`(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Nach erfolgreicher Ausführung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

adjtime - Synchronisation mit Systemuhr korrigieren

```
#include <sys/time.h>
int adjtime(struct timeval *delta, struct timeval *olddelta);
```

`adjtime` korrigiert die aktuelle Zeit, die von `gettimeofday`(3C) zurückgeliefert wird. Die Zeit wird um den Zeitraum, der in `struct timeval` angegeben ist, auf den *delta* zeigt, verändert.

Die Veränderung wird dadurch erreicht, daß die Systemuhr beschleunigt oder verzögert wird, abhängig davon, ob der Zeitraum positiv oder negativ ist. Diese Beschleunigung bzw. Verzögerung macht jedoch nur einen geringen Prozentsatz aus, im allgemeinen einen Bruchteil eines Prozents. Die Zeit ist immer eine monoton wachsende Funktion. Eine Zeitkorrektur durch einen vorhergehenden Aufruf von `adjtime` kann nicht beendet werden, wenn `adjtime` nochmals aufgerufen wird. Wenn *delta* `0` ist, dann liefert *olddelta* den Status der Wirkung des letzten `adjtime`-Aufrufs zurück; dieser Aufruf hat keine Korrekturwirkung. Wenn *olddelta* kein Nullzeiger ist, dann enthält die referenzierte Struktur nach Beendigung des Aufrufs die Anzahl der Sekunden und/oder Mikrosekunden, die vom vorhergehenden Aufruf korrigiert wurden. Wenn *olddelta* ein Nullzeiger ist, wird die entsprechende Information nicht zurückgeliefert.

Dieser Aufruf kann für Zeit-Server verwendet werden, welche die Uhren von Rechnern in lokalen Netzwerken synchronisieren. Solche Zeit-Server verlangsamen die Uhren auf einigen Maschinen und beschleunigen die Uhren auf anderen, um eine gemittelte Netzwerk-Zeit einzurichten.

Nur der Systemverwalter kann die Zeit korrigieren.

Der Anpassungswert wird auf die Auflösung der Systemuhr gerundet.

`adjtime` ist erfolglos, wenn einer oder mehrere der folgenden Punkte zutreffen:

`EFAULT` *delta* oder *olddelta* zeigt außerhalb des Prozeßbereichs, oder *olddelta* zeigt auf einen Prozeßbereich, der nicht beschreibbar ist.

`EPERM` Die effektive Benutzernummer des Prozesses ist nicht die des Systemverwalters.

SIEHE AUCH

`gettimeofday`(3C)
`date`(1) in "SINIX V5.41 Kommandos"

ERGEBNIS

Ein Rückgabewert von 0 zeigt an, daß der Aufruf erfolgreich war. -1 zeigt einen Fehler an, dessen Fehlercode aus der globalen Variablen `errno` gelesen werden kann.

alarm - Prozeß-Alarmuhr setzen

```
#include <unistd.h>
unsigned alarm(unsigned sec)
```

`alarm` weist die Alarmuhr des aufrufenden Prozesses an, das Signal `SIGALRM` an den aufrufenden Prozeß zu senden, nachdem die Anzahl der in *sec* angegebenen Echtzeit-Sekunden abgelaufen ist (siehe `signal`(2)).

Alarmanforderungen werden nicht gekellert; aufeinanderfolgende Aufrufe setzen die Alarmuhr des aufrufenden Prozesses zurück.

Wenn *sec* 0 ist, werden alle vorher angegebenen Alarmanforderungen aufgehoben.

`fork` setzt die Alarmuhr eines neuen Prozesses auf 0 (siehe auch `fork`(2)). Ein Prozeß, der durch die Familie der `exec`-Kommandos erzeugt wurde, übernimmt die noch vorhandene Zeit der Alarmuhr des alten Prozesses.

SIEHE AUCH

`exec`(2), `fork`(2), `pause`(2), `signal`(2), `sigset`(2).

ERGEBNIS

`alarm` gibt die Zeit zurück, die vorher in der Alarmuhr des aufrufenden Prozesses geblieben war.

brk, sbrk - Größe des Datensegments verändern

```
#include <unistd.h>
int brk(void *endds);
void *sbrk(int incr);
```

brk und sbrk werden zum dynamischen Ändern des Speicherplatzes verwendet, der dem Datensegment des aufrufenden Prozesses zugewiesen ist (vgl. exec(2)). Die Änderung wird durch Rücksetzen des Speichergrenzwerts ('break value') des Prozesses und Zuweisen eines entsprechenden Bereichs vorgenommen. Der Speichergrenzwert ('break value') ist die erste nicht belegte Adresse oberhalb des Datensegments. Der Umfang des zugewiesenen Speicherplatzes erhöht sich mit der Vergrößerung des Speichergrenzwerts. Neu zugewiesener Speicherplatz wird auf Null gesetzt. Wenn jedoch derselbe Speicherplatz demselben Prozeß wieder zugewiesen wird, ist sein Inhalt undefiniert.

brk setzt den Grenzwert auf *endds* und ändert den zugewiesenen Platz entsprechend.

sbrk fügt *incr* Bytes zum Grenzwert hinzu und ändert den zugewiesenen Platz entsprechend. *incr* kann negativ sein. In diesem Fall wird der Umfang des zugewiesenen Speicherplatzes verringert.

brk und sbrk sind erfolglos und ändern den zugewiesenen Speicherplatz nicht, wenn einer oder mehrere der folgenden Punkte zutreffen:

ENOMEM — Eine derartige Änderung würde dazu führen, daß mehr Speicherplatz zugewiesen wird, als durch die systembedingte maximale Prozeßgröße zulässig ist (siehe ulimit(2)).

EAGAIN — Der Gesamtbetrag des für einen read-Aufruf während physikalischer Ein-/Ausgabe zur Verfügung stehenden Systemspeichers ist vorübergehend unzureichend (siehe shmop(2)). Dies kann sogar dann auftreten, wenn der angeforderte Speicherplatz kleiner als die systembedingte maximale Prozeßgröße ist (siehe ulimit(2)).

SIEHE AUCH

exec(2), shmop(2), ulimit(2), end(3C).

ERGEBNIS

Nach erfolgreicher Beendigung gibt brk 0 und sbrk den alten Speichergrenzwert zurück. Andernfalls wird -1 zurückgegeben und errno zur Anzeige des Fehlers gesetzt.

chdir, fchdir - aktuelles Dateiverzeichnis ändern

```
#include <unistd.h>
int chdir(const char *path);
int fchdir(int fildes);
```

`chdir` und `fchdir` machen das Verzeichnis, auf das *path* oder *fildes* verweist, zum aktuellen Arbeitsverzeichnis, dem Startpunkt für Pfadsuchen nach Pfadnamen, die nicht mit / beginnen. *path* zeigt auf den Pfadnamen eines Verzeichnisses. Das *fildes*-Argument ist ein offener Dateideskriptor eines Verzeichnisses.

Um ein Verzeichnis zum aktuellen zu machen, muß ein Prozeß Zugriffsrechte zum Ausführen (Suchen) auf das Verzeichnis haben.

`chdir` ist erfolglos und das aktuelle Dateiverzeichnis bleibt unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Für eine Komponente des Pfadnamens gibt es keine Durchsuchertaubnis.

`EFAULT` *path* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während des Systemaufrufs `chdir` abgefangen.

`EIO` Es trat während des Lesens oder Schreibens vom Dateisystem ein Ein- oder Ausgabefehler auf.

`ELOOP` Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

`ENAMETOOLONG`
Die Länge des Arguments *path* ist größer als `PATH_MAX`, oder die Länge einer Komponente von *path* ist größer als `NAME_MAX`, während `POSIX_NO_TRUNC` aktiv ist.

`ENOTDIR` Eine Komponente des Pfadnamens ist kein Dateiverzeichnis.

`ENOENT` Eine Komponente von *path* existiert nicht oder ist ein leerer Pfadname.

`ENOLINK` *path* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

`EMULTIHOP` Die Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner.

`fchdir` ist erfolglos und das aktuelle Dateiverzeichnis bleibt unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES`	Für *fildes* gibt es keine Durchsucherlaubnis.
`EBADF`	*fildes* ist kein Dateideskriptor für eine offene Datei.
`EINTR`	Ein Signal wurde während des Systemaufrufs `fchdir` abgefangen.
`EIO`	Es trat während des Lesens oder Schreibens vom Dateisystem ein Ein- oder Ausgabefehler auf.
`ENOLINK`	*fildes* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.
`ENOTDIR`	Der offene Dateideskriptor zeigt nicht auf ein Dateiverzeichnis.

SIEHE AUCH

`chroot`(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

chmod, fchmod - Dateimodus ändern

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

int chmod(const char *path, mode_t mode);

int fchmod(int fildes, mode_t mode);
```

chmod und fchmod setzen den Zugriffserlaubnis-Teil des Modus der durch *path* angegebenen oder durch den Deskriptor *fildes* ausgewiesenen Datei, entsprechend dem im *mode* enthaltenen Bitmuster. Die Zugriffserlaubnis-Bits werden wie folgt interpretiert:

```
S_ISUID   04000   Benutzernummer bei Ausführung setzen
S_ISGID   020#0   Gruppennummer bei Ausführung setzen, wenn # die Werte
                  7, 5, 3 oder 1 hat
                  Aufhebung der obligatorischen Sperre von Dateien
                  und Dateisätzen, wenn # 6, 4, 2 oder 0 ist
S_ISVTX   01000   Textsegment nach Ausführung sichern
S_IRWXU   00700   Lesen, Schreiben, Ausführen (Durchsuchen) durch Eigentümer
S_IRUSR   00400   Lesen durch Eigentümer
S_IWUSR   00200   Schreiben durch Eigentümer
S_IXUSR   00100   Ausführen durch Eigentümer (Durchsuchen, wenn es sich um
                  ein Dateiverzeichnis handelt)
S_IRWXG   00070   Lesen, Schreiben, Ausführen (Durchsuchen) durch Gruppe
S_IRGRP   00040   Lesen durch Gruppe
S_IWGRP   00020   Schreiben durch Gruppe
S_IXGRP   00010   Ausführen durch Gruppe
S_IRWXO   00007   Lesen, Schreiben, Ausführen (Durchsuchen) durch andere
S_IROTH   00004   Lesen durch andere
S_IWOTH   00002   Schreiben durch andere
S_IXOTH   00001   Ausführen durch andere
```

Andere Modi werden durch bitweise ODER-Verküpfung der Zugriffsmodi erzeugt.

Die effektive Benutzernummer des Prozesses muß mit der des Eigentümers der Datei übereinstimmen, oder der Prozeß muß das entsprechende Privileg haben, damit der Modus einer Datei geändert werden kann.

Wenn der Prozeß kein privilegierter Prozeß und die Datei kein Dateiverzeichnis ist, wird das Modusbit 01000 (Textsegment nach Ausführung sichern) gelöscht.

Wenn weder der Prozeß, noch ein Mitglied der anhängenden Gruppenliste privilegiert ist und die effektive Gruppennummer des Prozesses nicht mit der Gruppennummer der Datei übereinstimmt, wird das Modusbit 02000 (Gruppennummer bei Ausführung setzen) gelöscht.

Wenn bei einer ausführbaren Datei des Typs 0410 das Sticky-Bit (Modusbit 01000) gesetzt ist, löscht das Betriebssystem den Programmtext nicht im Swap-Bereich, wenn der letzte Benutzerprozeß beendet wird. Wenn das Sticky-Bit bei einer ausführbaren Datei des Typs 0413 oder ELF gesetzt ist, löscht das Betriebssystem den Programmtext nicht aus dem Speicher, wenn der letzte Benutzerprozeß beendet wird. In beiden Fällen ist beim Setzen des Sticky-Bits der Text bereits (entweder in einem Swap-Bereich oder im Speicher) vorhanden, wenn der nächste Benutzer diese Datei ausführt. Die Ausführung wird dadurch beschleunigt.

Wenn ein Verzeichnis beschrieben werden kann und das Sticky-Bit gesetzt ist, können Dateien in diesem Verzeichnis nur dann entfernt (gelöscht) oder umbenannt werden, wenn mindestens eine der folgenden Bedingungen zutrifft (siehe unlink(2) und rename(2)):

die Datei gehört dem Benutzer
das Verzeichnis gehört dem Benutzer
der Benutzer hat das Recht, auf die Datei zu schreiben
der Benutzer ist ein privilegierter Benutzer

Wenn das Modusbit 02000 (Gruppennummer bei Ausführung setzen) gesetzt und das Modusbit 00010 (Ausführen oder Suchen durch Gruppe) nicht gesetzt ist, liegt das obligatorische Sperren von Dateien und Dateisätzen bei einer normalen Datei vor. Dies kann zukünftige Aufrufe von open(2), creat(2), read(2) und write(2) auf diese Datei beeinflussen.

Bei einer erfolgreichen Beendigung markieren chmod und fchmod das st_ctime-Feld der Datei, um dieses zu aktualisieren.

chmod ist erfolglos und die Datei bleibt unverändert, wenn einer oder mehrere der folgenden Punkte zutreffen:

EACCES Eine Komponente von *path* darf nicht durchsucht werden.

EFAULT *path* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EINTR Ein Signal wurde während der Ausführung des Systemaufrufs abgefangen.

EIO Ein E/A-Fehler trat während des Lesens oder Schreibens auf das Dateisystems auf.

ELOOP Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

EMULTIHOP Die Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt es nicht.

ENAMETOOLONG
Die Länge des *path*-Arguments überschreitet PATH_MAX, oder die Länge einer *path*-Komponente überschreitet NAME_MAX, während _POSIX_NO_TRUNC wirksam ist.

ENOTDIR Eine Komponente des Pfades *path* ist kein Dateiverzeichnis.

ENOENT Entweder eine Komponente des Pfades oder die Datei, auf welche durch *path* verwiesen wurde, existiert nicht oder ist ein Null-Pfadname.

ENOLINK *fildes* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EPERM — Die effektive Benutzernummer entspricht nicht der des Dateieigentümers und der Prozeß ist nicht entsprechend privilegiert.

EROFS — Die durch *path* angegebene Datei steht in einem schreibgeschützten Dateisystem.

`fchmod` ist erfolglos und der Dateimodus wird nicht verändert, wenn

EBADF — *fildes* kein offener Dateideskriptor ist.

EIO — ein E/A-Fehler während des Lesens oder Schreibens auf das Dateisystems auftrat,

EINTR — ein Signal während der Ausführung des `fchmod` Systemaufrufs abgefangen wurde,

ENOLINK — *path* auf einen fernen Rechner weist, und die Verbindung zu diesem Rechner nicht mehr aktiv ist,

EPERM — die effektive Benutzernummer nicht der des Dateieigentümers entspricht und der Prozeß nicht entsprechend privilegiert ist,

EROFS — die durch *fildes* angegebene Datei in einem schreibgeschützten Dateisystem steht.

SIEHE AUCH

`chown`(2), `creat`(2), `fcntl`(2), `mknod`(2), `open`(2), `read`(2), `stat`(2), `write`(2), `mkfifo`(3C), `stat`(5).
`chmod`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

chown, lchown, fchown - Eigentümer/Gruppe einer Datei ändern

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

int chown(const char *path, uid_t owner, gid_t group);

int lchown(const char *path, uid_t owner, gid_t group);

int fchown(int fildes, uid_t owner, gid_t group);
```

Die Eigentümernummer und die Gruppennummer der Datei, die durch *path* spezifiziert wird, oder auf die *fildes* verweist, werden auf *owner* bzw. *group* gesetzt. Wenn *owner* oder *group* mit -1 spezifiziert ist, wird die der Datei zugehörige ID nicht geändert.

Die Funktion `lchown` setzt den Eigentümer und die Gruppenzugehörigkeit der angegebenen Datei genau wie `chown`, es sei denn, die Datei besteht aus einem symbolischen Verweis. In diesem Fall ändert `lchown` die Zugehörigkeit der Verweisdatei, wohingegen `chown` die Zugehörigkeit der Datei oder des Verzeichnisses ändert, auf das sich der Verweis bezieht.

Wenn `chown`, `lchown` oder `fchown` von einem Prozeß aufgerufen wird, der nicht den Systemverwalterstatus hat, dann wird das Bit zum Setzen der Benutzer- und Gruppennummer bei Ausführung, beziehungsweise `S_ISUID` und `S_ISGID`, gelöscht (siehe `chmod`(2)).

Das Betriebssystem hat die Konfigurationsoption `_POSIX_CHOWN_RESTRICTED`, um Zugehörigkeitsänderungen für `chown`-, `lchown`- und `fchown`-Systemaufrufe zu verhindern. Wenn `_POSIX_CHOWN_RESTRICTED` nicht aktiv ist, muß die effektive Benutzernummer des Prozesses die des Eigentümers der Datei sein, oder der Prozeß muß Systemverwalter sein, um den Besitz der Datei zu ändern. Wenn `_POSIX_CHOWN_RESTRICTED` aktiv ist, bewahren die `chown`-, `lchown`- und `fchown`- Systemaufrufe den Eigentümer einer Datei davor, daß die Eigentümernummer seiner Dateien geändert wird und beschränkt den Gruppenwechsel der Datei auf die Liste der ergänzenden Gruppennummern.

Nach erfolgreichem Abschluß markieren `chown`, `lchown` und `fchown` das `ST_CTIME`-Feld der Datei, um sie auf den neuesten Stand zu bringen.

`chown` und `lchown` sind erfolglos, und Eigentümer und Gruppe der angegebenen Datei bleiben unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Eine Komponente von *path* darf nicht durchsucht werden.

`EFAULT` *path* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während des `chown`- oder `lchown`-Systemaufrufs abgefangen.

`EINVAL` *group* oder *owner* sind außerhalb des zulässigen Bereichs.

EIO — Es trat während des Lesens oder Schreibens vom Dateisystem ein Ein- oder Ausgabefehler auf.

ELOOP — Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

EMULTIHOP — Die Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dieses nicht.

ENAMETOOLONG — Die Länge des *path*-Arguments überschreitet PATH_MAX, oder die Länge einer *path*-Komponente überschreitet NAME_MAX, während _POSIX_NO_TRUNC wirksam ist.

ENOLINK — *path* weist auf einen fernen Rechner, zu dem die Verbindung nicht mehr aktiv ist.

ENOTDIR — Eine Komponente des Pfades *path* ist kein Dateiverzeichnis.

ENOENT — Eine Komponente des Pfades oder die Datei, auf die *path* verweist, existiert nicht oder ist ein leerer Pfadname.

EPERM — Die effektive Benutzernummer entspricht nicht dem Eigentümer der Datei, und die effektive Benutzernummer ist nicht der Systemverwalter, wenn gleichzeitig _POSIX_CHOWN_RESTRICTED anzeigt, daß ein solches Vorrecht verlangt wird.

EROFS — Die Datei steht in einem schreibgeschützten Dateisystem.

fchown ist erfolglos, und Eigentümer und Gruppe der angegebenen Datei bleiben unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EABDF — verweist auf keine offene Datei.

EINVAL — *group* oder *owner* sind außerhalb des zulässigen Bereichs.

EPERM — Die effektive Benutzernummer entspricht nicht dem Eigentümer der Datei, und die effektive Benutzernummer ist nicht der Systemverwalter, wenn gleichzeitig _POSIX_CHOWN_RESTRICTED anzeigt, daß ein solches Vorrecht verlangt wird.

EROFS — Die Datei steht in einem schreibgeschützten Dateisystem.

EINTR — Ein Signal wurde während des chown- oder lchown-Systemaufrufs abgefangen.

EIO — Es trat während des Lesens oder Schreibens vom Dateisystem ein Ein- oder Ausgabefehler auf.

ENOLINK — *path* weist auf einen fernen Rechner, zu dem die Verbindung nicht mehr aktiv ist.

SIEHE AUCH

chmod(2).
chown(1), chgrp(1) in "SINIX V5.41 Kommandos"

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben, andernfalls wird -1 zurückgegeben und errno zur Anzeige des Fehlers gesetzt.

chroot - Root-Dateiverzeichnis ändern

```
#include <unistd.h>
int chroot(const char *path);
```

path weist auf einen Pfadnamen, der ein Dateiverzeichnis benennt. chroot bewirkt, daß das angegebene Dateiverzeichnis zum Root-Dateiverzeichnis wird, d.h. zum Ausgangspunkt für die Pfadsuche nach Pfadnamen, die mit / beginnen. Das aktuelle Dateiverzeichnis des Benutzers wird durch den Systemaufruf chroot nicht beeinflußt. Die effektive Benutzernummer des Prozesses muß die des Systemverwalters sein, damit das Root-Dateiverzeichnis geändert werden kann.

Der Eintrag '..' im Root-Dateiverzeichnis wird so verstanden, daß es sich um das Root-Dateiverzeichnis selbst handelt. Daher kann '..' nicht für den Zugriff auf Dateien außerhalb des Unterbaums verwendet werden, dessen Wurzel das Root-Dateiverzeichnis ist.

chroot ist erfolglos, und das Root-Dateiverzeichnis bleibt unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

ELOOP — Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

ENAMETOOLONG — Die Länge des Arguments *path* ist größer als PATH_MAX oder NAME_MAX während POSIX_NO_TRUNC aktiv ist.

EFAULT — *path* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EINTR — Ein Signal wurde während des Systemaufrufs chroot abgefangen.

EMULTIHOP — Die Komponenten von *path* erfordern den Sprung auf ferne Rechner.

ENOLINK — *path* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOTDIR — Eine Komponente des Pfadnamens ist kein Dateiverzeichnis.

ENOENT — Das angegebene Dateiverzeichnis existiert nicht.

EPERM — Die effektive Benutzernummer entspricht nicht der des Systemverwalters.

SIEHE AUCH

chdir(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

close - Dateideskriptor schließen

```
#include <unistd.h>
int close (int fildes)
```

fildes ist ein Dateideskriptor, der von einem `creat`-, `open`-, `dup`-, `fcntl`- oder `pipe`-Systemaufruf geliefert wurde. `close` schließt die durch *fildes* angegebene Datei. Alle ausstehenden Datensatzsperren (in der von *fildes* angegebenen Datei) des Prozesses werden aufgehoben.

Wenn alle Dateideskriptoren, die mit der offenen Datei verbunden sind, geschlossen worden sind, wird die Verwaltungsstruktur für diese Datei freigegeben.

Der von der Datei belegte Speicherplatz wird freigegeben, wenn die Anzahl der Verweise auf diese Datei Null ist und alle dazugehörenden Dateideskriptoren geschlossen wurden. Auf die Datei kann anschließend nicht mehr zugegriffen werden.

Wenn eine STREAMS-Datei (siehe `intro`(2)) geschlossen wird und für den aufrufenden Prozeß vorher registriert wurde, daß ihm ein SIGPOLL-Signal (siehe `signal`(2) und `sigset`(2)) bei Ereignissen in Zusammenhang mit dieser Datei (siehe I_SETSIG in `streamio`(7)) geschickt wird, wird die Registrierung des aufrufenden Prozesses für Ereignisse in Zusammenhang mit dieser Datei aufgehoben. Das letzte `close` auf eine Datei bewirkt, daß die zu *fildes* gehörende Datei beseitigt wird. Wenn `O_NDELAY` und `O_NONBLOCK` nicht gesetzt sind und keine Signale für die Datei abgesetzt wurden, wartet `close` bis zu 15 Sek auf jedes Modul und jeden Treiber, damit in der Warteschlange stehende Ausgaben vor Beseitigen der Datei gemacht werden können. Die Zeitverzögerung kann durch einen `ioctl`-Aufruf mit dem Parameter `I_SETCLTIME` geändert werden (siehe `streamio`(7)). Wenn die Option `O_NDELAY` oder `O_NONBLOCK` gesetzt wurde oder wenn Signale vorliegen, wartet `close` nicht auf die Beendigung der Ausgabe und beseitigt die Datei sofort.

Ist *fildes* das Ende einer Pipe, so bewirkt der letzte `close`-Aufruf, daß ein "hangup" am anderen Ende der Pipe erzeugt wird. Wurde dem anderen Ende der Pipe ein Name zugewiesen (siehe `fattach`(3C)), so bewirkt der letzte `close`, daß diese Zuweisung aufgehoben wird (siehe `fdetach`(3C)). Sollte am anderen Ende der Pipe kein Prozeß mehr vorhanden sein und die Namenszuweisung wird aufgehoben, so wird der entsprechende STREAM ebenfalls abgebaut.

Die angegebene Datei wird geschlossen, außer wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EBADF` *fildes* ist kein gültiger offener Dateideskriptor.

`EINTR` Ein Signal wurde während des Systemaufrufs `close` abgefangen.

`ENOLINK` *fildes* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

`creat`(2), `dup`(2), `exec`(2), `fcntl`(2), `intro`(2), `open`(2), `pipe`(2), `signal`(2), `signal`(5), `streamio`(7).
`fattach`(3C), `fdetach`(3C) in "Leitfaden für Programmierer: Netzwerk-Schnittstellen".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

creat - neue Datei erstellen oder vorhandene Datei überschreiben

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

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

creat erstellt eine neue Datei oder bereitet eine vorhandene Datei vor, neu beschrieben zu werden. Die Datei wird durch den Pfadnamen angegeben, auf den *pfad* zeigt.

Wenn die Datei vorhanden ist, wird ihre Länge auf 0 abgeschnitten, und Modus und Eigentümer bleiben unverändert.

Wenn die Datei nicht existiert, wird die Datei-Eigentümernummer auf die effektive Benutzernummer des Prozesses gesetzt. Die Gruppennummer der Datei wird auf die effektive Gruppennummer des Prozesses gesetzt; wenn aber das S_ISGID-Bit im übergeordneten Verzeichnis gesetzt ist, dann wird die Gruppennummer der Datei vom übergeordneten Verzeichnis geerbt. Die Zugriffserlaubnis-Bits des Dateimodus werden auf den Wert von *modus* wie folgt geändert:

- Wenn die Gruppennummer der neuen Datei nicht zur effektiven oder einer der zusätzlichen Gruppennummern paßt, wird das S_ISGID-Bit gelöscht.
- Alle Bits, die in der Dateityp-Erstellungsmaske des Prozesses gesetzt sind, werden gelöscht (siehe umask(2)).
- Das Bit für die Sicherung des Textsegments nach der Ausführung wird gelöscht (siehe chmod(2)).

Nach erfolgreicher Beendigung wird ein Dateideskriptor mit reiner Schreiberlaubnis zurückgegeben, und die Datei wird zum Schreiben geöffnet, auch wenn der Modus das Schreiben nicht zuläßt. Der Schreib-/Lesezeiger wird auf den Anfang der Datei gesetzt. Die Datei bleibt standardmäßig bei exec-Systemaufrufen geöffnet (siehe fcntl(2)). Eine neue Datei kann mit einem Modus eröffnet werden, der Schreiben nicht zuläßt.

Der Aufruf creat(*pfad*, *modus*) entspricht dem Aufruf von

open(*pfad*, O_WRONLY | O_CREAT | O_TRUNC, *modus*)

creat ist erfolglos, wenn einer oder mehrere der folgenden Punkte zutreffen:

EACCES Eine Komponente des Pfades darf nicht durchsucht werden.

EACCES Die Datei ist nicht vorhanden, und das Dateiverzeichnis, in dem die Datei angelegt werden soll, läßt das Schreiben nicht zu.

EACCES Die Datei ist vorhanden, und die Schreiberlaubnis wird verweigert.

EAGAIN Die Datei ist vorhanden, obligatorisches Sperren von Dateien und Dateisätzen ist gesetzt, und in der Datei sind noch Dateisatzsperren vorhanden (siehe chmod(2)).

EFAULT *pfad* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EISDIR Die angegebene Datei ist ein Dateiverzeichnis.

EINTR Ein Signal wurde während des Systemaufrufs `creat` abgefangen.

ELOOP Während der Übersetzung von *pfad* wurden zu viele symbolische Verweise angetroffen.

EMFILE Der Prozeß hat zu viele Dateien geöffnet (siehe `getrlimit`(2)).

ENAMETOOLONG
Die Länge des *pfad*-Arguments überschreitet `PATH_MAX`, oder die Länge einer *pfad*-Komponente überschreitet `NAME_MAX`, während `_POSIX_NO_TRUNC` wirksam ist.

ENOTDIR Eine Komponente des Pfadnamens ist kein Dateiverzeichnis.

ENOENT Eine Komponente des Pfadnamens existiert nicht.

ENOENT Der Pfadname ist Null.

EROFS Die angegebene Datei steht in einem schreibgeschützten Dateisystem.

ETXTBSY Die Datei ist eine reine Programmdatei, die gerade ausgeführt wird.

ENFILE Die Systemdatei-Tabelle ist voll.

ENOLINK *pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EMULTIHOP Die Komponenten von *pfad* erfordern den Sprung auf mehrere ferne Rechner.

ENOSPC Das Dateisystem hat keine Indexeinträge mehr.

SIEHE AUCH

`chmod`(2), `close`(2), `dup`(2), `fcntl`(2), `getrlimit`(2), `lseek`(2), `open`(2), `read`(2), `umask`(2), `write`(2), `stat`(5).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nicht negative ganze Zahl, d.h. der Dateideskriptor, zurückgegeben. Andernfalls wird -1 zurückgegeben, es wird keine Datei geöffnet oder modifiziert und `errno` wird zur Anzeige des Fehlers gesetzt.

dup - Dateideskriptor duplizieren

```
#include <unistd.h>
int dup(int fildes);
```

fildes ist ein Dateideskriptor, der von einem Systemaufruf `creat`, `open`, `dup`, `fcntl` oder `pipe` geliefert wurde. `dup` gibt einen neuen Dateideskriptor zurück, der mit dem Original-Dateideskriptor folgendes gemein hat:

- dieselbe offene Datei (oder Pipe)
- denselben Schreib-/Lesezeiger (d.h. beide Dateideskriptoren benutzen denselben Schreib-/Lesezeiger)
- denselben Zugriffsmodus (Lesen, Schreiben oder Schreiben/Lesen)

Der neue Dateideskriptor bleibt standardmäßig bei `exec`-Systemaufrufen geöffnet (siehe `fcntl`(2)).

Der zurückgegebene Dateideskriptor ist die kleinste Zahl, die zur Verfügung steht.

`dup` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EBADF` *fildes* ist kein gültiger offener Dateideskriptor.

`EINTR` Ein Signal wurde während des Systemaufrufs `dup` abgefangen.

`EMFIL` Der Prozeß hat zu viele offene Dateien (siehe `getrlimit`(2)).

`ENOLINK` *fildes* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

`close`(2), `creat`(2), `exec`(2), `fcntl`(2), `getrlimit`(2), `open`(2), `pipe`(2), `dup2`(3C), `lockf`(3C).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nicht negative ganze Zahl, d.h. der Dateideskriptor, zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

exec - Datei ausführen

```
#include <unistd.h>
int execl (const char *Pfad, const char *arg0, ..., const char *argn, (char *)0);
int execv (const char *Pfad, char *const *argv);
int execle (const char *Pfad, const char *arg0, ..., const char *argn,
        (char *0), const char *envp[]);
int execve (const char *Pfad, const char *argv, const char *envp[]);
int execlp (const char *Datei, const char *arg0, ..., const char *argn, (char *)0);
int execvp (const char *Datei, char *const *argv);
```

exec überlagert in allen seinen Formen einen alten Prozeß mit einem neuen Prozeß. Der neue Prozeß wird von einer normalen ausführbaren Datei erzeugt. Diese Datei ist entweder eine ausführbare Objektdatei oder eine Datendatei für einen Interpreter. Von einem erfolgreichen exec-Aufruf kann nicht zurückgekehrt werden, da das Abbild des aufrufenden Prozesses vom neuen überlagert wird.

Eine Interpreter-Datei beginnt mit einer Zeile der Form

#! *Pfadname* [*arg*]

wobei der *Pfadname* der Pfad des Interpreters und *arg* ein optionales Argument ist. Wenn eine Interpreter-Datei ausgeführt wird, führt das System den angegebenen Interpreter aus. Der angegebene Pfadname wird als *arg0* an den Interpreter übergeben. Wenn *arg* in der Interpreter-Datei angegeben wurde, wird es als *arg1* an den Interpreter übergeben. Die verbleibenden Argumente für den Interpreter sind *arg0* bis *argn* der ursprünglich ausgeführten Datei.

Wird ein C-Programm ausgeführt, wird es wie folgt aufgerufen:

```
int main (int argc, char *argv[], *envp[]);
```

Hierbei ist *argc* der Argumentzähler, *argv* ein Feld von Zeigern auf die Argumente selbst und *envp* ein Feld von Zeigern auf die Umgebungsvariablen. *argc* ist mindestens 1, und das erste Element des Feldes weist auf eine Zeichenkette, die den Namen der Programmdatei enthält.

Pfad weist auf einen Pfadnamen, der die neue Programmdatei des Prozesses bezeichnet.

Datei weist auf die neue Programmdatei des Prozesses. Wenn *Datei* keinen Schrägstrich enthält, wird der Pfad für diese Datei durch Durchsuchen der Dateiverzeichnisse erhalten, die in der Umgebungsvariablen PATH definiert werden (siehe environ(5)). Die Umgebung wird typischerweise von der Shell bereitgestellt (siehe sh(1)).

Wenn die neue Prozeßdatei keine ausführbare Objektdatei ist, verwenden `execlp` und `execvp` den Inhalt dieser Datei als Standardeingabe für `sh`(1).

Die Argumente *arg0*, ..., *argn* zeigen auf Zeichenketten, die mit dem Null-Byte abgeschlossen sind. Diese Zeichenketten stellen die Argumentliste dar, die für das neue Prozeßabbild zur Verfügung steht. Wenigstens *arg0* muß vorhanden sein. Es wird der Name des Prozesses, wie er vom Kommando `ps` angezeigt wird. Üblicherweise zeigt *arg0* auf eine Zeichenkette, die dieselbe wie *Pfad* (oder die letzte Komponente von *Pfad*) ist. Die Liste der Argumentzeichenketten wird durch ein Argument `(char *)0` beendet.

argv ist ein Feld von Zeigern auf Zeichenketten, die mit dem Null-Byte abgeschlossen sind. Diese Zeichenketten stellen die Argumentliste dar, die dem neuen Prozeßabbild zur Verfügung steht. Es ist üblich, daß *argv* wenigstens ein Element hat, das auf eine Zeichenkette weisen muß, die dieselbe wie *Pfad* (oder ihre letzte Komponente) ist. *argv* wird mit einem Nullzeiger abgeschlossen.

envp ist ein Feld von Zeigern auf Zeichenketten, die mit dem Null-Byte abgeschlossen sind. Diese Zeichenketten stellen die Umgebung für das neue Prozeßabbild dar. *envp* wird mit einem Nullzeiger abgeschlossen. Bei `execl`, `execv`, `execvp`, und `execlp` setzt die C-Laufzeit-Startroutine einen Zeiger auf die Umgebung des aufrufenden Prozesses in dem globalen Objekt `extern char **environ`, und diese wird zum Weiterreichen der Umgebung des aufrufenden Prozesses an den neuen Prozeß verwendet.

Die Dateien, die im aufrufenden Prozeß offen sind, bleiben auch im neuen Prozeß offen. Eine Ausnahme bilden dabei jene Dateien, deren Status-Bit für 'Schließen-bei-exec' gesetzt ist (siehe `fcntl`(2)). Bei Dateideskriptoren, die offen bleiben, wird der Schreib-/Lesezeiger nicht verändert.

Für jene Signale, die im aufrufenden Prozeß abgefangen werden, wird im neuen Prozeßabbild die Standardeinstellung eingestellt (siehe `signal`(2)). In den anderen Fällen erbt das neue Prozeßabbild die Einstellungen bezüglich der Signale vom aufrufenden Prozeß.

Wenn das Bit zum Setzen der Benutzernummer bei Ausführung in der neuen Programmdatei des Prozesses gesetzt ist (siehe `chmod`(2)), setzt `exec` die effektive Benutzernummer des neuen Prozesses auf die Eigentümernummer der neuen Prozeßdatei. Entsprechend wird bei gesetztem Bit zum Setzen der Gruppennummer bei Ausführung die effektive Gruppennummer des neuen Prozesses auf die Gruppennummer der Prozeßdatei gesetzt. Die reale Benutzernummer und die reale Gruppennummer des neuen Prozesses bleiben dieselben wie beim aufrufenden Prozeß.

Ist die effektive Benutzerkennung gleich `root` oder gleich Systemverwalter, so werden die 'set-user-ID'- und 'set-group-ID'-Bits berücksichtigt, vorausgesetzt, der Prozeß wird durch `ptrace` überwacht.

Die gemeinsam benutzten Speichersegmente (Shared Memory Segments), die dem aufrufenden Prozeß zugeordnet waren, werden dem neuen Prozeß nicht zugeordnet (siehe

shmop(2)). Für den neuen Prozeß wird das Profilieren abgeschaltet (siehe profil(2)). Außerdem übernimmt der neue Prozeß noch folgende Attribute vom aufrufenden Prozeß:

nice-Wert (siehe nice(2))
Prozeßklasse und Priorität (siehe priocntl(2))
Prozeßnummer
Prozeßnummer des Vaters
Prozeßnummer der Gruppe
zusätzliche Gruppennummern
semadj-Werte (siehe semop(2))
Sitzungsnummer (siehe exit(2) und signal(2))
Ablaufverfolgungsanzeiger (siehe ptrace(2) request 0)
Restzeit, bis zu einem Alarmuhr-Signal (siehe alarm(2))
aktuelles Dateiverzeichnis
'root'-Dateiverzeichnis
Schutzbitmaske des Prozesses (siehe umask(2))
Betriebsmittel-Grenzwerte (siehe getrlimit(2))
utime, stime, cutime und cstime (siehe times(2))
Dateisperren (siehe fcntl(2) und lockf(3C))
steuerndes Terminal
Prozeß-Signalmaske (siehe sigprocmask(2))
unbearbeitete Signale (siehe sigpending(2))

Bei erfolgreicher Ausführung sieht exec die Veränderung des st_atime-Feldes der Datei vor. Sollte exec erfolgreich ausgeführt worden sein, so wird die Datei des Prozeßabbilds als mit open() geöffnet betrachtet. Das zugehörige close() erfolgt konzeptionell nach dem Öffnen, aber vor dem Prozeßende oder einer erfolgreichen Bearbeitung eines weiteren Aufrufs von exec. exec ist erfolglos und kehrt zum aufrufenden Prozeß zurück, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EACCES — In einem der Verzeichnisse, die im Pfad der Datei des neuen Prozesses aufgelistet sind, ist keine Erlaubnis zum Durchsuchen gegeben.

E2BIG — Die Anzahl der Bytes in der Argumentliste des neuen Prozesses ist größer als die systembedingte Obergrenze von 5120 Bytes. Die Obergrenze der Argumentliste ist die Summe der Größe der Argumentliste plus die Größe der von der Umgebung exportierten Shell-Variablen.

EACCES — Die neue Prozeßdatei ist keine normale Datei.

EACCES — Der Modus der neuen Prozeßdatei erlaubt keine Ausführung.

EAGAIN — Die Gesamtmenge des Systemspeichers, die während des Lesens über zeichenorientierte E/A verfügbar ist, reicht zeitweise nicht aus.

EFAULT — Angeforderte Hardware ist nicht vorhanden.

EFAULT Ein `a.out`, das mit MAU oder dem 32B Schalter übersetzt wurde, läuft auf einem Rechner ohne ein MAU oder 32B.

EFAULT Ein Argument zeigt auf eine ungültige Adresse.

EINTR Ein Signal wurde während des Systemaufrufs `exec` abgefangen.

ELIBACC Angeforderte, gemeinsam benutzte Bibliothek hat keine Ausführerlaubnis.

ELIBEXEC Es wird versucht, eine gemeinsam benutzte Bibliothek direkt mit `exec` auszuführen.

ELOOP Beim Übersetzen von *Pfad* oder *Datei* wurden zuviele symbolische Verweise angetroffen.

EMULTIHOP Die Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt das nicht.

ENAMETOOLONG
Die Länge des *Datei-* oder *Pfad*-Arguments übertrifft `PATH_MAX`, oder die Länge einer *Datei-* oder *Pfad*-Komponente übertrifft `NAME_MAX`, während `_POSIX_NO_TRUNC` aktiv ist.

ENOENT Eine oder mehrere Komponenten des Pfadnamens des neuen Prozesses der Datei existieren nicht, oder er ist gleich Null.

ENOTDIR Eine Komponente des neuen Prozeßpfades ist kein Dateiverzeichnis.

ENOEXEC Der `exec`-Aufruf ist kein `execlp` oder `execvp`, und die neue Prozeßdatei hat die entsprechende Zugriffserlaubnis, jedoch eine ungültige magische Nummer in ihrem Dateikopf.

ETXTBSY Die Datei ist eine reine Programmdatei, die gerade von einem anderen Prozeß beschrieben wird.

ENOMEM Der neue Prozeß erfordert mehr Speicherplatz als vom systembedingten Maximum MAXMEM zugelassen wird.

ENOLINK *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

alarm(2), exit(2), fcntl(2), fork(2), getrlimit(2), nice(2), priocntl(2), ptrace(2), semop(2), signal(2), sigpending(2), sigprocmask(2), times(2), umask(2), lockf(3C), system(3S), a.out(4), environ(5).
sh(1), ps(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Wenn exec zu dem aufrufenden Prozeß zurückkehrt, ist ein Fehler aufgetreten; der Rückgabewert ist dann -1, und errno wird zur Anzeige des Fehlers gesetzt.

exit, _exit - Prozeß beenden

```
#include <stdlib.h>
void exit(int status);
#include <unistd.h>
void _exit(int status);
```

`exit` beendet den aufrufenden Prozeß mit nachstehenden Folgen:

- Alle im aufrufenden Prozeß offenen Dateien, Verzeichnisse und Nachrichtenverzeichnisse werden geschlossen.
- An den Vaterprozeß des aufrufenden Prozesses wird ein `SIGCHLD`-Signal gesendet.
- Wenn der Vaterprozeß des aufrufenden Prozesses nicht die Option `SA_NOCLDWAIT` gesetzt hat (siehe `sigaction`(2)), wird der aufrufende Prozeß in einen Zombie-Prozeß umgewandelt. Ein Zombie-Prozeß ist ein Prozeß, der nur einen Platz in der Prozeßtabelle einnimmt. Ihm ist weder im Benutzer- noch im Systembereich weiterer Speicherplatz zugewiesen. Der von ihm belegte Platz in der Prozeßtabelle wird teilweise mit Zeitabrechnungsdaten überlagert (siehe `sys/proc.h`), die vom Systemaufruf `times` benutzt werden.
- Die Vaterprozeßnummer aller vorhandenen Sohn- und Zombie-Prozesse des aufrufenden Prozesses wird auf 1 gesetzt. Dies bedeutet, daß der Initialisierungsprozeß (siehe `intro`(2)) jeden dieser Prozesse erbt.
- Jedes angeschlossene, gemeinsam benutzte Speichersegment (Shared Memory Segment) wird abgehängt, und der Wert von `shm_nattach` in der Datenstruktur, die der Kennung dieses Segments zugeordnet ist, wird um 1 verringert.
- Für jedes Semaphor, für das der aufrufende Prozeß einen `semadj`-Wert gesetzt hat (siehe `semop`(2)), wird dieser `semadj`-Wert zum `semval` des angegebenen Semaphors hinzugefügt.
- Wenn der Prozeß eine Prozeß-, Text- oder Datensperre hat, wird ein UNLOCK ausgeführt (siehe `plock`(2)).
- Wenn die Prozeßabrechnungsroutine des Systems eingeschaltet ist, wird ein Abrechnungssatz in die Abrechnungsdatei geschrieben (siehe `acct`(2)).
- Wenn der Prozeß ein steuernder Prozeß ist, wird `SIGHUP` an die Prozeßgruppe im Vordergrund seines steuernden Terminals gesendet, und sein steuerndes Terminal wird freigegeben.

- Wenn der aufrufende Prozeß irgendwelche angehaltenen Sohnprozesse hat, deren Prozeßgruppe verwaist, wenn der aufrufende Prozeß beendet wird, oder wenn der aufrufende Prozeß Element einer Prozeßgruppe ist, die verwaist, wenn der aufrufende Prozeß beendet wird, werden dieser Prozeßgruppe die Signale `SIGHUP` und `SIGCONT` gesendet.

Die C-Funktion `exit`(3C) ruft alle Funktionen auf, die durch die Funktion `atexit` eingetragen wurden, und zwar in der umgekehrten Reihenfolge ihrer Eintragung. Mit der Funktion `_exit` werden all solche Funktionen und Bereinigungsmaßnahmen umgangen.

Die Symbole `EXIT_SUCCESS` und `EXIT_FAILURE` sind in `stdlib.h` definiert und können als Wert von *status* verwendet werden, um erfolgreiches oder nicht erfolgreiches Beenden anzuzeigen.

SIEHE AUCH

`acct`(2), `intro`(2), `plock`(2), `semop`(2), `sigaction`(2), `signal`(2), `times`(2), `wait`(2), `atexit`(3C).

HINWEIS

Siehe `signal`(2), Abschnitt HINWEIS.

fcntl - geöffnete Datei steuern

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
int fcntl(int fildes, int cmd, ... /* arg */);
```

`fcntl` ermöglicht die Steuerung von offenen Dateien. *fildes* ist ein offener Dateideskriptor (siehe `intro`(2)).

An `fcntl` kann ein drittes Argument übergeben werden, dessen Datentyp und Wert vom übergebenen Kommando *cmd* abhängen. *cmd* spezifiziert die Operation, die von `fcntl` ausgeführt wird, und kann einer der folgenden Werte sein:

`F_DUPFD` Ein neuer Dateideskriptor wird wie folgt zurückgegeben:

- Dateideskriptor mit der niedrigsten verfügbaren Nummer, die größer als oder gleich dem ganzzahligen Wert ist, der als drittes Argument übergeben wird,
- dieselbe offene Datei (oder Pipe) wie die ursprüngliche Datei,
- derselbe Schreib-/Lesezeiger wie der der ursprünglichen Datei (d.h. beide Dateideskriptoren teilen sich denselben Schreib-/Lesezeiger),
- derselbe Zugriffsmodus (Lesen, Schreiben oder Lesen/Schreiben) wie die ursprüngliche Datei,
- dieselben Dateistatus-Bits wie die ursprüngliche Datei,
- das Bit 'Schließen-bei-exec' (siehe `F_GETFD`), das zum neuen Dateideskriptor gehört, so setzen, daß die Datei bei `exec`(2)-Aufrufen geöffnet bleibt.

`F_GETFD` ruft das Bit 'Schließen-bei-exec' ab, das zu dem Dateideskriptor *fildes* gehört. Wenn das niederwertige Bit `0` ist, bleibt die Datei bei `exec` offen, andernfalls wird die Datei bei Aufruf von `exec` geschlossen.

`F_SETFD` setzt das zu *fildes* gehörende Bit 'Schließen-bei-exec' auf das niederwertige Bit des ganzzahligen Wertes, der als drittes Argument übergeben wird (`0` oder `1` wie oben).

`F_GETFL` ruft das Dateistatus-Bit für *fildes* ab.

`F_SETFL` setzt das Dateistatus-Bit für *fildes* auf den ganzzahligen Wert, der als drittes Argument übergeben wird. Nur bestimmte Bits können gesetzt werden (siehe `fcntl`(5)).

F_FREESP gibt Speicherplatz, der mit einem Abschnitt der normalen *fildes*-Datei verbunden ist, frei. Dieser Abschnitt wird von einer Variablen des Datentyps `struct flock`, auf die das dritte Argument *arg* zeigt, spezifiziert. Der Datentyp `struct flock` ist in der Include-Datei `fcntl.h` definiert (siehe `fcntl`(5)) und beinhaltet folgende Mitglieder: `l_whence` ist 0, 1 oder 2, um anzuzeigen, daß der relative Offset `l_start` vom Anfang der Datei, von der momentanen Position, oder vom Ende der Datei gemessen wird. `l_start` ist der Offset von der Position aus, die in `l_whence` spezifiziert wird. `l_len` ist die Länge dieses Abschnitts. Eine Länge von 0 gibt bis zum Ende der Datei alles frei; in diesem Fall wird das Ende der Datei auf den Anfang des freigegebenen Teils gesetzt. Auf die Daten, die vorher in diesen Teil geschrieben wurden, kann nicht mehr zugegriffen werden.

Die folgenden Kommandos werden für Dateisperren und Datensatzsperren benutzt. Sperren können auf eine ganze Datei oder auf Segmente einer Datei gelegt werden.

F_SETLK Eine Sperre in einem der Dateisegmente ist entsprechend der Variablen des Typs `struct flock`, auf die *arg* zeigt, zu setzen oder zu löschen (siehe `fcntl`(5)). Die Aktion F_SETLK wird zum Einrichten der Lesesperre (F_RDLCK) und der Schreibsperre (F_WRLCK) sowie für die Aufhebung beider Sperrtypen (F_UNLCK) verwendet. Läßt sich eine Lese- oder Schreibsperre nicht setzen, gibt `fcntl` sofort den Fehlerwert -1 zurück.

F_SETLKW Dieses *cmd* ist dasselbe wie F_SETLK, außer daß der Prozeß schläft, bis das Segment frei zum Sperren ist, wenn die Sperranforderung durch andere Sperren blockiert wird.

F_GETLK Wenn die durch die `flock`-Struktur angegebene Sperranforderung, auf die *arg* zeigt, erzeugt werden könnte, wird diese Struktur unverändert zurückgegeben, außer daß der Sperrtyp auf F_UNLCK und das Feld `l_whence` auf SEEK_SET gesetzt wird. Wird eine Sperre gefunden, die eine Erzeugung dieser Sperre verhindern würde, dann wird die Struktur mit der Beschreibung der ersten Sperre überschrieben.

Dieses Kommando erzeugt niemals eine Sperre; es testet nur, ob einzelne Sperren eingerichtet werden könnten.

F_RSETLK Dieses Kommando wird vom Netzwerkdämon `lockd`(3N) benutzt, um mit dem NFS-Server NFS-Dateien zu sperren.

F_RSETLKW Dieses Kommando wird vom Netzwerkdämon `lockd`(3N) benutzt, um mit dem NFS-Server NFS-Dateien zu sperren.

F_RGETLK Dieses Kommando wird vom Netzwerkdämon `lockd`(3N) benutzt, um mit dem NFS-Server NFS-Dateien zu sperren.

Eine Lesesperre verhindert, daß ein Prozeß den geschützten Bereich mit einer Schreibsperre belegen kann. Für ein bestimmtes Segment einer Datei kann zu einem Zeitpunkt mehr als eine Lesesperre vorhanden sein. Der Dateideskriptor, auf den die Lesesperre gesetzt wird, muß mit der Lese-Erlaubnis geöffnet worden sein.

Eine Schreibsperre verhindert, daß ein Prozeß den geschützten Bereich mit einer Schreib- oder eine Lesesperre belegt. Für ein bestimmtes Segment einer Datei kann zu einem gegebenen Zeitpunkt jeweils nur eine Schreib- oder Lesesperre vorliegen. Der Dateideskriptor, auf den eine Schreibsperre gesetzt wird, muß mit Schreiberlaubnis geöffnet worden sein.

Die Struktur `flock` beschreibt Typ (`l_type`), Startpunkt-Offset (`l_whence`), relativen Offset (`l_start`), Größe (`l_len`), Prozeßnummer (`l_pid`) und Systemnummer (`l_sysid`) des betroffenen Segments in der Datei. Die Prozeß- und Systemnummer werden nur bei `F_GETLK` zur Rückgabe der Werte für eine blockierende Sperre verwendet. Die Sperren können hinter dem aktuellen Ende einer Datei beginnen und sich über das Dateiende hinaus erstrecken. Sie dürfen jedoch nicht negativ in Relation zum Anfang der Datei sein. Durch Setzen von `l_len` auf Null (0) kann eine Sperre so gesetzt werden, daß sie stets bis zum Ende der Datei geht. Wenn bei einer derartigen Sperre auch `l_whence` und `l_start` auf Null (0) gesetzt sind, wird die gesamte Datei gesperrt. Durch Ändern oder Freigeben eines Segments aus der Mitte eines größeren gesperrten Segments bleiben zwei kleinere Segmente an jedem Ende. Durch Sperren eines Segments, das bereits vom aufrufenden Prozeß gesperrt ist, wird die alte Sperre aufgehoben und die neue gesetzt. Alle mit einer Datei bei einem bestimmten Prozeß verbundenen Sperren werden aufgehoben, wenn ein Dateideskriptor für diese Datei durch den Prozeß geschlossen oder wenn der Prozeß beendet wird, der diesen Dateideskriptor enthält. Sperren werden nicht von einem Sohnprozeß in einem Systemaufruf `fork`(2) geerbt.

Wenn ein obligatorisches Sperren von Dateien und Dateisätzen in einer Datei aktiv ist (siehe `chmod`(2)), werden `open`(2)-, `read`(2)- und `write`(2)-Systemaufrufe auf die Datei durch die eingeschalteten Dateisatzsperren beeinflußt.

`fcntl` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` — *cmd* ist `F_SETLK`, der Typ der Sperre (`l_type`) ist eine Lesesperre (`F_RDLCK`), und das Segment einer zu sperrenden Datei ist bereits von einem anderen Prozeß schreibgesperrt.

`EACCES` — Der Typ ist eine Schreibsperre (`F_WRLCK`), und das Segment einer zu sperrenden Datei wird bereits von einem anderen Prozeß lese- oder schreibgesperrt.

`EAGAIN` — *cmd* ist `F_FREESP`, die Datei existiert, obligatorisches Datei-/Datensatzsperren ist gesetzt, und es gibt noch ausstehende Datensatzsperren in der Datei.

`EAGAIN` — *cmd* ist `F_SETLK` oder `F_SETLKW`, und die Datei wird momentan mit `mmap` (siehe `mmap`(2)) in den virtuellen Speicher abgebildet.

EBADF	*fildes* ist kein gültiger offener Dateideskriptor.
EBADF	*cmd* ist F_SETLK oder SETLKW, die Sperre (l_type) ist eine Lesesperre (F_RDLCK), und *fildes* ist kein gültiger, zum Lesen geöffneter Dateideskriptor.
EBADF	*cmd* ist F_SETLK oder SETLKW, die Sperre (l_type) ist eine Schreibsperre (F_WRLCK), und *fildes* ist kein gültiger, zum Schreiben geöffneter Dateideskriptor.
EBADF	*cmd* ist F_FREESP, und *fildes* ist kein gültiger, zum Schreiben geöffneter Dateideskriptor.
EDEADLK	*cmd* ist F_SETLKW, die Sperre ist durch eine Sperre von einem anderen Prozeß blockiert, und ein Deadlock würde verursacht, wenn der Prozeß schlafend gesetzt wird, um auf die Aufhebung dieser Sperre zu warten.
EDEADLK	*cmd* ist F_FREESP, obligatorisches Datensatzsperren ist möglich, O_NDELAY und O_NONBLOCK sind gelöscht, und es wurde eine Situation entdeckt, in der es zu einem Deadlock kommen könnte.
EFAULT	*cmd* ist F_FREESP, und der Wert, auf den *arg* zeigt, befindet sich in einer Adresse außerhalb des Adreßraums, der vom Prozeß belegt wird.
EFAULT	*cmd* ist F_GETLK, F_SET_LK oder F_SETLKW und der Wert, auf den *arg* zeigt, befindet sich in einer Adresse außerhalb des Adreßraums, der vom Prozeß belegt wird.
EINTR	Ein Signal wurde während des Systemaufrufs fcntl abgefangen.
EIO	Während des Lesens oder Schreibens vom Dateisystem trat ein Ein-/Ausgabefehler auf.
EMFILE	*cmd* ist F_DUPFD, und im aufrufenden Prozeß ist die Anzahl der offenen Dateideskriptoren gleich dem in der Konfiguration angegebenen Maximalwert der offenen Dateien für jeden Benutzer.
EINVAL	*cmd* ist F_DUPFD. *arg* ist entweder negativ, größer oder gleich dem Wert für die maximale Anzahl der jedem Benutzer zur Verfügung stehenden offenen Dateideskriptoren.
EINVAL	*cmd* besitzt keinen gültigen Wert.
EINVAL	*cmd* ist F_GETLK, F_SETLK oder SETLKW, und *arg* oder die Daten, auf die verwiesen wird, sind nicht gültig; oder *fildes* gibt eine Datei an, die Sperren nicht unterstützt.

ENOLCK *cmd* ist F_SETLK oder F_SETLKW, der Typ der Sperre ist eine Lese- oder Schreibsperre, und keine weiteren Dateisatzsperren stehen zur Verfügung (zuviele Dateisegmente gesperrt), weil das Maximum des Systems überschritten wurde.

ENOLINK *fildes* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOLINK *cmd* ist F_FREESP, die Datei ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EOVERFLOW *cmd* ist F_GETLK, und die Prozeßnummer des Prozesses, der die verlangte Sperre enthält, ist zu groß, um im l_pid-Feld gespeichert zu werden.

SIEHE AUCH

close(2), creat(2), dup(2), exec(2), fork(2), open(2), pipe(2), fcntl(5).

ERGEBNIS

Nach erfolgreicher Beendigung hängt der jeweils zurückgegebene Wert wie folgt von *cmd* ab:

F_DUPFD	ein neuer Dateideskriptor
F_GETFD	Wert des Status-Bit für 'Schließen-bei-exec' (nur das niederwertige Bit wird definiert)
F_SETFD	Wert, der nicht -1 ist
F_FREESP	Wert 0
F_GETFL	Wert des Dateistatus-Bits, nie negativ
F_SETFL	Wert, der nicht -1 ist
F_GETLK	Wert, der nicht -1 ist
F_SETLK	Wert, der nicht -1 ist
F_SETLKW	Wert, der nicht -1 ist

Bei einem Fehler wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Da die Variable errno in Zukunft auf EAGAIN und nicht auf EACCES gesetzt wird, wenn ein Teil einer Datei bereits von einem anderen Prozeß gesperrt ist, sollten portable Anwenderprogramme beide Werte erwarten und beide Werte prüfen.

fork - neuen Prozeß erzeugen

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

pid_t fork(void);
```

`fork` bewirkt die Erzeugung eines neuen Prozesses. Der neue Prozeß (Sohnprozeß) ist eine genaue Kopie des aufrufenden Prozesses (Vaterprozeß). Dies bedeutet, daß der Sohnprozeß nachstehende Attribute vom Vaterprozeß erbt:

reale Benutzernummer, reale Gruppennummer, effektive Benutzernummer, effektive Gruppennummer
Umgebung
'Schließen-bei-exec'-Bit (siehe `exec`(2))
Einstellungen für Signalbehandlung (`SIG_DFL`, `SIG_IGN`, `SIG_HOLD`, Funktionsadresse)
zusätzliche Gruppennummern
s-Bit für Eigentümer
s-Bit für Gruppe
Ein-/Aus-Status für Profilierung
nice-Wert (siehe `nice`(2))
Prozeßklasse (siehe `priocntl`(2))
alle zugeteilten gemeinsam benutzten Speichersegmente (Shared Memory) (siehe `shmop`(2))
Prozeßgruppennummer
Terminalnummer (siehe `exit`(2))
Aktuelles Dateiverzeichnis
Root-Dateiverzeichnis
Dateimodus-Erstellungsmaske (siehe `umask`(2))
Betriebsmittel-Grenzwerte (siehe `getrlimit`(2))
steuerndes Terminal

Prozeßpriorität und jegliche prozeßspezifischen Prioritätsparameter, die für die gegebene Prozeßklasse bezeichnend sind, können abhängig von Vereinbarungen dieser speziellen Klasse ererbt sein (siehe `priocntl`(2)).

Der Sohnprozeß unterscheidet sich vom Vaterprozeß in folgenden Punkten:

- Der Sohnprozeß hat eine eindeutige Prozeßnummer, die ungleich allen anderen Prozeßgruppennummern ist.
- Der Sohnprozeß hat eine andere Vaterprozeßnummer (d.h. die Prozeßnummer des Vaterprozesses).

- Der Sohnprozeß hat eine eigene Kopie des Dateideskriptors und des Verzeichnis-Streams des Vaterprozesses. Jeder Sohn-Dateideskriptor hat einen gemeinsamen Schreib-/Lesezeiger mit dem entsprechenden Dateideskriptor des Vaters.
- Alle `semadj`-Werte werden aufgehoben (siehe `semop`(2)).
- Prozeßsperren, Textsperren und Datensperren werden vom Sohnprozeß nicht geerbt (siehe `plock`(2)).
- Die `tms`-Struktur des Sohnprozesses ist wie folgt deklariert: `tms_utime`, `stime`, `cutime`, und `cstime` werden auf 0 gesetzt (siehe `times`(2)).
- Die Zeitspanne bis zum Rücksetzen eines Alarmuhr-Signals wird auf 0 gesetzt.
- Die Menge der Signale, die an den Prozeß ausgeliefert werden sollen, wird auf eine leere Menge initialisiert.

Vom Vaterprozeß gesetzte Datensatzsperren werden nicht vom Sohnprozeß geerbt (siehe `fcntl`(2)).

`fork` ist erfolglos, und ein Sohnprozeß wird nicht erstellt, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EAGAIN` Die systembedingte Begrenzung der Gesamtanzahl der für einen Benutzer ablaufenden Prozesse würde überschritten.

`EAGAIN` Der Systemspeicher, der beim Lesen über die ungepufferte ('raw') Ein/Ausgabeschnittstelle zur Verfügung steht, ist vorübergehend unzureichend.

`ENOMEM` Der Swap-Bereich ist zu klein.

SIEHE AUCH

`alarm`(2), `exec`(2), `fcntl`(2), `getrlimit`(2), `nice`(2), `plock`(2), `priocntl`(2), `ptrace`(2), `semop`(2), `shmop`(2), `signal`(2), `times`(2), `umask`(2), `wait`(2), `system`(3S).

ERGEBNIS

Nach erfolgreicher Beendigung gibt `fork` 0 an den Sohnprozeß und die Prozeßnummer des Sohnprozesses an den Vaterprozeß zurück. Andernfalls wird `(pid_t)-1` an den Vaterprozeß zurückgegeben, kein Sohnprozeß erstellt und `errno` zur Anzeige des Fehlers gesetzt.

fpathconf, pathconf - konfigurierbare Pfadnamenvariablen lesen

```
#include <unistd.h>

long fpathconf (int fildes, int name);
long pathconf (char *path, int name);
```

Die Funktionen `fpathconf` und `pathconf` liefern den aktuellen Wert einer konfigurierbaren Option oder eines Grenzwerts, der mit einer Datei oder einem Verzeichnis verknüpft ist. Das Argument *path* zeigt auf den Pfadnamen einer Datei oder eines Verzeichnisses; *fildes* ist ein geöffneter Dateideskriptor und *name* ist die symbolische Konstante (definiert in `unistd.h`), welche das konfigurierbare Systemlimit oder die Option angibt, die zurückgeliefert werden soll.

Der zurückgegebene Wert von `pathconf` und `fpathconf` hängt vom Typ der Datei, der durch *path* oder *fildes* festgelegt ist, ab. Die folgende Tabelle enthält die symbolischen Konstanten, welche von `pathconf` und `fpathconf` zusammen mit den von POSIX definierten Rückgabewerten erkannt werden.

Wert von name	Siehe Bemerkungen
_PC_LINK_MAX	1
_PC_MAX_CANNON	2
_PC_MAX_INPUT	2
_PC_NAME_MAX	3,4
_PC_PATH_MAX	4,5
_PC_PIPE_BUF	6
_PC_CHOWN_RESTRICTED	7
_PC_NO_TRUNC	3,4
_PC_VDISABLE	2

Bemerkungen:

1 Wenn *path* oder *fildes* sich auf ein Verzeichnis bezieht, so gilt der zurückgelieferte Wert für das Verzeichnis.

2 Das Verhalten ist undefiniert für den Fall, daß *path* oder *fildes* sich nicht auf eine endgültige Datei bezieht.

3 Wenn *path* oder *fildes* sich auf ein Verzeichnis bezieht, so gilt der Rückgabewert für die Dateien innerhalb des Verzeichnisses.

4 Das Verhalten ist undefiniert, wenn *path* oder *fildes* sich nicht auf ein Verzeichnis bezieht.

5 Wenn sich *path* oder *fildes* auf ein Verzeichnis bezieht, dann entspricht der zurückgelieferte Wert der maximalen Länge eines relativen Pfadnamens, wenn das angegebene Verzeichnis das Arbeitsverzeichnis wird.

6 Wenn *path* oder *fildes* sich auf eine Pipe oder eine FIFO-Datei bezieht, dann entspricht der zurückgegebene Wert der FIFO-Datei selbst. Entspricht *path* oder *fildes* einem Verzeichnis, dann gilt der zurückgegebene Wert irgendeiner FIFO-Datei, der innerhalb des Verzeichnisses existiert oder erzeugt werden kann. Wenn sich *path* oder *fildes* auf einen anderen Dateityp bezieht, dann ist das Verhalten undefiniert.

7 Wenn sich *path* oder *fildes* auf ein Verzeichnis bezieht, dann gilt der zurückgegebene Wert für irgendeine Datei, die innerhalb des Verzeichnisses existiert oder erzeugt werden kann.

Der Wert des konfigurierbaren Systemlimits oder der Option aus *name* ändert sich während der Lebensdauer des aufrufenden Prozesses nicht.

`fpathconf` schlägt fehl, wenn die folgende Bedingung erfüllt ist:

`EBADF` *fildes* ist kein gültiger Dateideskriptor.

`pathconf` schlägt fehl, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

`EACCES` Suchrechte für eine Komponente des Pfadnamens sind nicht gegeben.

`ELOOP` Es gibt zu viele symbolische Verweise beim Übersetzen von *path*.

`EMULTIHOP` Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

`ENAMETOOLONG`
Die Länge eines Pfadnamens überschreitet `PATH_MAX`, oder eine Komponente des Pfadnamens ist länger als `NAME_MAX`, während `(_POSIX_NO_TRUNC)` wirkt.

`ENOENT` *path* wird für das Kommando benötigt, und die benannte Datei existiert nicht, oder das Argument *path* zeigt auf eine leere Zeichenkette.

`ENOLINK` *path* zeigt auf einen fernen Rechner, und der Verweis auf diesen Rechner ist nicht mehr aktiv.

`ENOTDIR` Eine Komponente des Pfadpräfixes ist kein Verzeichnis.

Sowohl `fpathconf` als auch `pathconf` schlagen fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` *name* enthält einen ungültigen Wert.

SIEHE AUCH

`sysconf`(3C), `limits`(4)

ERGEBNIS

Wenn `fpathconf` oder `pathconf` mit einer ungültigen symbolischen Konstante aufgerufen wird oder die symbolische Konstante einem konfigurierbaren Systemlimit oder einer Option entspricht, die auf dem System nicht unterstützt wird, so wird der Wert -1 an den aufrufenden Prozeß zurückgeliefert. Wenn die Funktion fehlschlägt, weil das konfigurierbare Systemlimit oder die Option *name* nicht vom System unterstützt wird, ändert sich der Wert von `errno` nicht.

fsync - Dateizustand synchronisieren

```
#include <unistd.h>
int fsync(int fildes);
```

`fsync` schreibt alle modifizierten Daten und Attribute von *fildes* auf das Speichermedium. Wenn `fsync` ausgeführt ist, wurden alle Kopien von *fildes*, die sich in Puffern befanden, auf das physikalische Speichermedium geschrieben. `fsync` unterscheidet sich von `sync` in dem Sinne, daß `sync` die Zustände aller gepufferten Dateien aktualisiert und zurückkehrt, bevor die Aktualisierung beendet ist.

`fsync` sollte von Programmen verwendet werden, welche voraussetzen, daß sich eine Datei in einem bekannten Zustand befindet. Enthält ein Programm beispielsweise eine einfache Transaktionsmöglichkeit, kann mit `fsync` sichergestellt werden, daß alle transaktionsbedingten Änderungen an einer oder mehreren Dateien durchgeführt worden sind.

`fsync` schlägt fehl, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

`EBADF` *fildes* ist kein gültiger Dateideskriptor, der zum Schreiben geöffnet ist.

`ENOLINK` *fildes* befindet sich auf einem fernen Rechner, und der Verweis auf diesem Rechner ist nicht länger aktiv.

`EINTR` Während der Ausführung von `fsync` wurde ein Signal empfangen.

`EIO` Ein E/A-Fehler trat während des Lesens oder Schreibens im Dateisystem auf.

ERGEBNIS

Nach der erfolgreichen Beendigung wird der Wert 0 zurückgeliefert. Ansonsten wird der Wert -1 zurückgegeben und `errno` gesetzt, um den Fehler zu bestimmen.

HINWEIS

Der Weg, auf den die Daten das physikalische Medium erreichen, hängt von der Implementierung und der Hardware ab. `fsync` kehrt zurück, wenn der Gerätetreiber anzeigt, daß der Schreibvorgang stattgefunden hat.

SIEHE AUCH

`sync`(2).

getcontext, setcontext - Benutzerkontext ändern

```
#include <ucontext.h>
int getcontext(ucontext_t *ucp);
int setcontext(ucontext_t *ucp);
```

Diese Funktionen dienen im Zusammenhang mit den in `makecontext`(3C) definierten Funktionen zur Implementierung der Kontextwechsel auf Benutzerebene zwischen mehreren Kontrollflüssen eines Prozesses.

`getcontext` initialisiert die Struktur, auf die *ucp* zeigt, als aktuellen Benutzerkontext des aufrufenden Prozesses. Der Benutzerkontext wird durch `ucontext`(5) definiert und enthält die Inhalte der Maschinenregister, der Signalmaske und des Stapels des aufrufenden Prozesses.

`setcontext` restauriert den Benutzerkontext, auf den *ucp* zeigt. Der Aufruf von `setcontext` kehrt nicht zurück; die Programmausführung fährt an der Stelle fort, auf die die Kontextstruktur aus `setcontext` zeigt. Die Kontextstruktur sollte durch einen vorhergehenden Aufruf von `getcontext` oder `makecontext` erzeugt werden oder als drittes Argument an eine Signalbehandlungsroutine (siehe `sigaction`(2)) gegeben werden. Wenn die Kontextstruktur mit `getcontext` erzeugt wurde, wird die Programmausführung wieder aufgenommen, als ob der entsprechende Aufruf von `getcontext` zurückgekehrt wäre. Wenn die Kontextstruktur mit `makecontext` erzeugt wurde, wird die Programmausführung mit der mit `makecontext` angegebenen Funktion wieder aufgenommen.

HINWEIS

Wenn eine Signalbehandlungsroutine ausgeführt wird, wird der Benutzerkontext gespeichert und vom Kernel ein neuer Kontext erzeugt. Wenn der Prozeß die Signalbehandlungsroutine über `longjmp`(3C) verläßt, so wird der ursprüngliche Kontext nicht restauriert, und zukünftige Aufrufe von `getcontext` sind nicht mehr zuverlässig. Signalbehandlungsroutinen sollten deshalb `siglongjmp`(3C) oder `setcontext` verwenden.

`getcontext` und `setcontext` schlagen fehl, wenn folgende Bedingung erfüllt ist:

`EFAULT` *ucp* zeigt an eine ungültige Adresse.

ERGEBNIS

Nach erfolgreicher Ausführung kehrt `setcontext` nicht zurück, und `getcontext` liefert 0. Ansonsten wird -1 geliefert, und `errno` enthält eine Fehlernummer.

SIEHE AUCH

`sigaction`(2), `sigaltstack`(2), `sigprocmask`(2), `makecontext`(3C), `ucontext`(5).

getdents - Dateiverzeichnis-Einträge umwandeln

```
#include <sys/dirent.h>
int getdents (int fildes, struct dirent *buf, size_t nbyte);
```

fildes ist ein Dateideskriptor, der von einem open(2)- oder dup(2)-Systemaufruf geliefert wird.

getdents versucht, *nbyte* Bytes aus dem zu *fildes* gehörenden Dateiverzeichnis zu lesen und diese als vom Dateisystem unabhängige Dateiverzeichnis-Einträge in den Puffer zu bringen, auf den *buf* zeigt. Da die vom Dateisystem unabhängigen Dateiverzeichnis-Einträge unterschiedlich lang sind, ist die tatsächliche Anzahl zurückgegebener Bytes in den meisten Fällen wesentlich kleiner als *nbyte*. Sehen Sie in dirent(4) nach, um die Anzahl der Bytes zu berechnen.

Der dateisystemunabhängige Dateiverzeichnis-Eintrag wird durch die Struktur dirent angegeben. Eine Beschreibung hiervon ist in dirent(4) zu finden.

Bei Geräten, die positionieren können, beginnt getdents an der Stelle in der Datei, die durch den *fildes* zugeordneten Schreib-/Lesezeiger angegeben wird. Nach Rückkehr von getdents wird der Schreib-/Lesezeiger erhöht, damit er auf den nächsten Dateiverzeichnis-Eintrag zeigt.

Dieser Systemaufruf wurde für die Implementierung der Funktion readdir entwickelt (eine Beschreibung ist in directory(3C) zu finden) und sollte daher nicht für andere Zwecke verwendet werden.

getdents ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EBADF — *fildes* ist kein zum Lesen geöffneter, gültiger Dateideskriptor.

EFAULT — *buf* weist über den zugewiesenen Adreßraum hinaus.

EINVAL — *nbyte* ist für einen Dateiverzeichnis-Eintrag nicht groß genug.

ENOENT — Der aktuelle Schreib-/Lesezeiger für das Dateiverzeichnis befindet sich nicht auf einem gültigen Eintrag.

ENOLINK — *fildes* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOTDIR — *fildes* ist kein Dateiverzeichnis.

EIO — Während des Zugriffs auf das Dateisystem ist ein E/A-Fehler aufgetreten.

SIEHE AUCH

`directory`(3C).
`dirent`(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nicht negative ganze Zahl zurückgegeben, die die Anzahl der tatsächlich gelesenen Bytes angibt. Der Wert 0 zeigt an, daß das Ende des Dateiverzeichnisses erreicht wurde. War der Systemaufruf erfolglos, wird -1 zurückgegeben und `errno` zur Anzeige des Fehlers gesetzt.

getgroups, setgroups - Gruppennummern lesen/schreiben

```
#include <unistd.h>
int getgroups(int gidsetsize, gid_t *grouplist)
int setgroups(int ngroups, const gid_t *grouplist)
```

getgroups liest die aktuelle Gruppenzugriffsliste des aufrufenden Prozesses und speichert das Ergebnis in dem Feld der Gruppennummern, welches durch *grouplist* angegeben wird. Dieses Feld hat *gidsetsize*-Einträge und muß groß genug sein, um die komplette Liste aufzunehmen. Diese Liste kann nicht größer als NGROUPS_MAX sein. Wenn *gidsetsize* gleich 0 ist, liefert getgroups die Anzahl der Gruppen, zu denen der aufrufende Prozeß gehört, ohne daß das Feld *grouplist* verändert wird.

setgroups setzt die Gruppenzugriffsliste des aufrufenden Prozesses aus dem Feld der Gruppennummern. Die Anzahl der Einträge wird durch *ngroups* angegeben und darf nicht größer als NGROUPS_MAX sein. Diese Funktion kann nur vom Systemverwalter aufgerufen werden.

getgroups schlägt fehl, wenn:

EINVAL Der Wert von *gidsetsize* ist ungleich Null und kleiner als die Anzahl der Gruppennummern für den aufrufenden Prozeß.

setgroups schlägt fehl, wenn:

EINVAL Der Wert *ngroups* ist größer als NGROUPS_MAX.

EPERM Die effektive Benutzernummer ist nicht die des Systemverwalters.

Beide Aufrufe schlagen fehl, wenn:

EFAULT Ein referenzierter Teil des Feldes *grouplist* befindet sich außerhalb des allokierten Adreßbereichs des Prozesses.

SIEHE AUCH

chown(2), getuid(2), setuid(2), initgroups(3C).
groups(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Ausführung liefert getgroups die Anzahl der Gruppennummern zurück, die für den aufrufenden Prozeß gesetzt sind, und setgroups liefert den Wert 0. Ansonsten wird der Wert -1 zurückgegeben und errno gesetzt, um den Fehler anzuzeigen.

getmsg - Nachricht aus einem Stream abrufen

```
#include <stropts.h>
int getmsg(int fd, struct strbuf *ctlptr, struct strbuf *dataptr, int *flagsp);
int getpmsg(int fd, struct strbuf *ctlptr, struct strbuf *dataptr, int *bandp, int *flagsp);
```

`getmsg` holt den Inhalt einer Nachricht (siehe `intro`(2)), die in der Leseschlange des Stream-Kopfs einer STREAMS-Datei steht, und schreibt den Inhalt in einen vom Benutzer angegebenen Puffer. Die Nachricht muß entweder einen Datenteil, einen Steuerteil oder beide Teile enthalten. Der Daten- und der Steuerteil der Nachricht werden, wie nachstehend beschrieben, in separate Puffer geschrieben. Die Semantik jedes Teils wird durch das STREAMS-Modul definiert, das die Nachricht generiert hat.

Die Funktion `getpmsg` führt das gleiche aus wie `getmsg`, aber sie liefert eine genauere Kontrolle über die Priorität der erhaltenen Meldungen. Außer, wenn es speziell vermerkt wurde, gelten alle Informationen, die `getmsg` betreffen, auch für `getpmsg`.

fd gibt einen Dateideskriptor an, der auf einen offenen Stream zeigt. *ctlptr* und *dataptr* verweisen je auf eine `strbuf`-Struktur, die nachstehende Elemente aufweist:

```
int maxlen;    /* Maximum Puffergröße */
int len;       /* Länge der Daten */
char *buf;     /* Zeiger auf den Puffer */
```

`buf` weist auf einen Puffer, in den die Daten bzw. Steuerinformationen geschrieben werden sollen, und `maxlen` zeigt die größtmögliche Anzahl Bytes an, die dieser Puffer aufnehmen kann. Bei der Rückgabe enthält `len` die Byte-Anzahl der tatsächlich empfangenen Daten bzw. Steuerinformationen, oder der Wert ist 0, wenn der Steuer- oder Datenteil eine Nullänge aufweist, oder der Wert ist -1, wenn die Nachricht keine Daten- oder Steuerinformationen enthält. *flagsp* sollte auf eine Ganzzahl verweisen, welche die Art der Nachricht, die der Benutzer erhalten kann, anzeigt. Dieses wird später beschrieben.

ctlptr wird zur Aufnahme des Steuerteils der Nachricht und *dataptr* zur Aufnahme des Datenteils der Nachricht verwendet. Wenn *ctlptr* (oder *dataptr*) `NULL` ist oder das `maxlen`-Feld -1 ist, wird der Steuer- (bzw. Daten-)teil der Nachricht nicht verarbeitet und bleibt in der Leseschlange des Stream-Kopfes. Wenn *ctlptr* (oder *dataptr*) nicht `NULL` ist und es keinen korrespondierenden Steuer- (oder Daten-)teil der Nachricht in der Leseschlange des Stream-Kopfes gibt, wird `len` auf -1 gesetzt. Wenn das `maxlen`-Feld auf 0 gesetzt ist und ein Steuer- (oder Daten-)teil mit einer Nullänge vorliegt, wird dieser Nullängenteil aus der Leseschlange entfernt und `len` auf 0 gesetzt. Wenn das `maxlen`-Feld auf 0 gesetzt ist und mehr als 0 Byte Steuer- (oder Daten-) Informationen vorhanden sind, bleiben diese Informationen in der Leseschlange, und `len` wird auf 0 gesetzt. Wenn das `maxlen`-Feld in *ctlptr* bzw. *dataptr* kleiner als der Steuer- oder Datenteil der Nachricht ist, werden `maxlen` Bytes geholt. In diesem Fall wird der Rest der Nachricht in der Leseschlange des Stream-Kopfes gelassen und ein Rückgabewert von

ungleich Null geliefert, wie nachstehend im Abschnitt ERGEBNIS beschrieben.

Standardmäßig verarbeitet getmsg die erste Meldung, die in der Leseschlange zur Verfügung steht. Der Benutzer kann sich jedoch durch Setzen von der Ganzzahl, auf die *flagsp* zeigt, auf RS_HIPRI, dazu entscheiden, nur Meldungen hoher Priorität zu empfangen. In diesem Fall verarbeitet getmsg die nächste Nachricht nur, wenn diese eine Nachricht hoher Priorität ist. Wenn die Ganzzahl, auf die durch *flagsp* verwiesen wird, 0 ist, bringt getmsg jede verfügbare Nachricht in der Leseschlange des Stream-Kopfes. In diesem Fall wird bei Rückkehr die Ganzzahl, auf die durch *flagsp* verwiesen wird, auf RS_HIPRI gesetzt, wenn eine Nachricht hoher Priorität angetroffen wurde, anderenfalls auf 0.

Für getpmsg sind die Optionen unterschiedlich. *flagsp* verweist auf eine Bitmaske mit den folgenden Optionen, die sich gegenseitig ausschließen: MSG_HIPRI, MSG_BAND und MSG_ANY. Ebenso wie getmsg verarbeitet getpmsg die als nächste zur Verfügung stehende Nachricht in der Leseschlange des Stream-Kopfes. Der Benutzer erhält nur Nachrichten hoher Priorität, indem er die Ganzzahlen, auf die mit *flagsp*=MSG_HIPRI und *bandp*=0 verwiesen wird, auf 0 setzt. In diesem Fall verarbeitet getpmsg nur dann die nächste Nachricht, wenn es eine Nachricht hoher Priorität ist. In ähnlicher Weise kann der Benutzer eine Nachricht aus einem speziellen Prioritätsbereich aufrufen, indem er die Ganzzahl, auf die durch *flagsp* verwiesen wird, auf MSG_BAND setzt, und die Ganzzahl, auf die durch *bandp* verwiesen wird, auf den gewünschten Prioritätsbereich setzt. In diesem Fall verarbeitet getpmsg nur dann die nächste Nachricht, wenn sie sich in einem Prioritätsbereich befindet, welcher gleich oder größer als die Ganzzahl ist, auf welche durch *bandp* verwiesen wird, oder wenn es sich um eine Nachricht hoher Priorität handelt. Wenn ein Benutzer lediglich die erste Meldung der Schlange abrufen möchte, sollte die Ganzzahl, auf welche durch *flagsp* verwiesen wird, auf MSG_ANY gesetzt sein, und die Ganzzahl, auf welche durch *bandp* verwiesen wird, sollte auf 0 gesetzt sein. Bei Rückkehr wird, in dem Fall, daß die erhaltene Nachricht eine Nachricht hoher Priorität war, die Ganzzahl, auf welche durch *flagsp* verwiesen wird, auf MSG_HIPRI, und die Ganzzahl, auf welche durch *bandp* verwiesen wird, auf 0 gesetzt. Anderenfalls wird die Ganzzahl, auf welche durch *flagsp* verwiesen wird, auf MSG_BAND gesetzt, und die Ganzzahl, auf die durch *bandp* verwiesen wird, wird auf den Prioritätsbereich der Nachricht gesetzt.

Wenn O_NDELAY und O_NONBLOCK nicht gesetzt wurden, blockiert getmsg, bis eine Nachricht des mit *flagsp* angegebenen Typs in der Leseschlange des Stream-Kopfes vorhanden ist. Wenn O_NDELAY oder O_NONBLOCK gesetzt wurde und keine Nachricht des angegebenen Typs in der Leseschlange vorhanden ist, bleibt getmsg erfolglos und setzt errno auf EAGAIN.

Wenn auf dem Stream, aus dem die Nachrichten geholt werden sollen, ein Verbindungsabbruch auftritt, arbeitet getmsg normal weiter, wie oben beschrieben, bis die Leseschlange entleert ist. Danach gibt es 0 in den len-Feldern von *ctlptr* und *dataptr* zurück.

`getmsg` oder `getpmsg` sind erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EAGAIN` `O_NDELAY` oder `O_NONBLOCK` ist gesetzt, und es stehen keine Nachrichten zur Verfügung.

`EBADF` *fd* ist kein zum Lesen offener, gültiger Dateideskriptor.

`EBADMSG` Die zu lesende Nachricht in der Schlange ist für `getmsg` nicht gültig.

`EFAULT` *ctlptr*, *dataptr*, *bandp*, oder *flagsp* weisen auf eine Stelle außerhalb des zugewiesenen Adreßraums.

`EINTR` Ein Signal wurde während des Systemaufrufs `getmsg` abgefangen.

`EINVAL` Ein ungültiger Wert wurde in *flagsp* angegeben, oder der durch *fd* angegebene Stream ist mit einem Multiplexer verbunden.

`ENOSTR` Ein Stream ist keiner *fd* zugeordnet.

`getmsg` kann auch dann erfolglos sein, wenn vor dem Aufruf von `getmsg` eine STREAMS-Fehlermeldung am Stream-Kopf empfangen wurde. Der zurückgegebene Fehler ist der Wert, der in der STREAMS-Fehlermeldung enthalten ist.

SIEHE AUCH

`intro`(2), `poll`(2), `putmsg`(2), `read`(2), `write`(2).
"Leitfaden für Programmierer: STREAMS".

ERGEBNIS

Nach erfolgreicher Beendigung wird ein nicht negativer Wert zurückgegeben. Der Wert 0 zeigt an, daß eine vollständige Nachricht erfolgreich gelesen wurde. Der Rückgabewert `MORECTL` zeigt an, daß weitere Steuerinformationen auf einen Abruf warten. Der Rückgabewert `MOREDATA` zeigt an, daß weitere Daten auf den Abruf warten. Der Rückgabewert `MORECTL | MOREDATA` zeigt an, daß noch beide Arten übrig sind. Der Rest der Nachricht wird mit anschließenden `getmsg`-Aufrufen geholt. Wenn jedoch eine Nachricht hoher Priorität in dem Stream-Kopf der Leseschlange eingetroffen ist, wird der nächste `getmsg`-Aufruf die Nachricht hoher Priorität vorrangig bearbeiten, bevor der Rest der vorher empfangenen Teilnachricht bearbeitet wird.

getpid, getpgrp, getppid getpgid - Prozeßnummern abfragen

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

pid_t getpid(void);

pid_t getpgrp(void);

pid_t getppid(void);

pid_t getpgid(pid_t pid);
```

`getpid` gibt die Prozeßnummer des aufrufenden Prozesses zurück.

`getpgrp` gibt die Prozeßgruppennummer des aufrufenden Prozesses zurück.

`getppid` gibt die Vaterprozeßnummer des aufrufenden Prozesses zurück.

`getpgid` gibt die Prozeßgruppennummer des Prozesses zurück, dessen Prozeßnummer gleich *pid* ist, oder die Prozeßgruppennummer des aufrufenden Prozesses, falls *pid* 0 ist.

`getpgid` schlägt fehl, wenn eine oder mehrere der folgenden Bedingungen zutreffen:

`EPERM` Der Prozeß, dessen Prozeßnummer gleich *pid* ist, befindet sich nicht in der gleichen Sitzung wie der aufrufende Prozeß, und die Implementierung erlaubt keinen Zugriff auf die Prozeßgruppennummer dieses Prozesses vom aufrufenden Prozeß aus.

`ESRCH` Es gibt keinen Prozeß mit einer Prozeßnummer *pid*.

SIEHE AUCH

`exec`(2), `fork`(2), `getpid`(2), `getsid`(2), `intro`(2), `setpgid`(2), `setsid`(2), `setpgrp`(2), `signal`(2).

ERGEBNIS

Nach erfolgreicher Ausführung liefert `getpgid` eine Prozeßgruppennummer zurück. Andernfalls wird (`pid_t`)-1 zurückgeliefert, und `errno` wird entsprechend gesetzt, um den Fehler anzuzeigen.

getrlimit, setrlimit - Betriebsmittelverbrauch kontrollieren

```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);

int setrlimit(int resource, const struct rlimit *rlp);
```

Dieser Aufruf limitiert die Benutzung einer Vielzahl von Betriebsmitteln durch einen Prozeß und aller seiner Sohnprozesse; mit `getrlimit` werden die Grenzwerte gelesen und mit `setrlimit` gesetzt.

Jeder Aufruf von `getrlimit` oder `setrlimit` gibt ein bestimmtes Betriebsmittel und einen bestimmten Grenzwert dafür an. Der Grenzwert setzt sich aus einem Wertepaar zusammen. Ein Wert gibt dabei den weichen Grenzwert und der andere Wert den maximalen, harten Grenzwert an. Weiche Grenzwerte können von einem Prozeß auf einen Wert gesetzt werden, der kleiner oder gleich dem harten Grenzwert ist. Ein Prozeß kann seinen harten Grenzwert verringern (nicht umkehrbar), so daß er größer oder gleich dem weichen Grenzwert wird. Nur ein Prozeß mit einer effektiven Benutzernummer gleich 0 oder der Systemverwalter kann einen harten Grenzwert erhöhen. Sowohl harter als auch weicher Grenzwert können durch einen einzigen Aufruf von `setrlimit` verändert werden, abhängig von den oben beschriebenen Beschränkungen. Grenzwerte dürfen einen unendlich großen Wert `RLIM_INFINITY` besitzen. *rlp* ist ein Zeiger auf `struct rlimit`, welche die folgenden Komponenten besitzt:

```
rlim_t  rlim_cur;       /* aktueller weicher Grenzwert  */
rlim_t  rlim_max;       /* harter Grenzwert     */
```

`rlim_t` ist ein arithmetischer Datentyp, in den Objekte des Typs `int`, `size_t` und `off_t` konvertiert werden können, ohne daß Informationen verlorengehen.

Die möglichen Betriebsmittel, deren Beschreibungen und die resultierenden Maßnahmen beim Überschreiten eines Grenzwertes werden in der folgenden Tabelle zusammengefaßt:

Betriebsmittel	Beschreibung	Maßnahme
RLIMIT_CORE	Die maximale Größe einer Speicherabzugsdatei in Bytes, die von einem Prozeß erzeugt werden darf. Eine Größe von 0 verhindert die Erzeugung von Speicherabzugsdateien.	Das Schreiben einer Speicherabzugsdatei wird bei dieser Größe beendet.
RLIMIT_CPU	Die maximale Dauer der CPU-Zeit, die von einem Prozeß verbraucht wird.	SIGXCPU wird an den Prozeß gesendet. Wenn der Prozeß hält oder SIGXCPU ignoriert, ist das Verhalten von der Prozeßklasse abhängig.
RLIMIT_DATA	Die maximale Größe des Heaps eines Prozesses in Bytes.	brk(2) schlägt fehl, und errno enthält ENOMEM.
RLIMIT_FSIZE	Die maximale Länge einer Datei in Bytes, die von einem Prozeß erzeugt werden kann. Eine Länge von 0 verhindert die Erzeugung von Dateien.	SIGXFSZ wird an den Prozeß gesendet. Wenn der Prozeß SIGXFSZ hält oder ignoriert, schlagen weitere Versuche, die Datei zu vergrößern fehl, und errno enthält EFBIG.
RLIMIT_NOFILE	Die maximale Anzahl der geöffneten Dateideskriptoren, die ein Prozeß besitzen kann.	Funktionen, welche neue Dateideskriptoren anlegen, schlagen fehl, und errno enthält EMFILE.
RLIMIT_STACK	Die maximale Größe des Prozeß-Stapels in Bytes. Das System läßt den Stapel nicht automatisch über diesen Grenzwert hinauswachsen.	SIGSEGV wird an den Prozeß gesendet. Wenn der Prozeß SIGSEGV bearbeitet, ignoriert oder abfängt und keinen alternativen Stapel verwendet (siehe sigaltstack(2)), wird als Behandlungsmodus von SIGSEGV SIG_DFL gesetzt.
RLIMIT_VMEM	Die maximale Länge des Adreßbereichs eines Prozesses in Bytes.	Die brk(2) und mmap(2) Funktionen schlagen fehl, und errno enthält ENOMEM. Außerdem kann der Stapel nicht mehr anwachsen und die oben genannten Effekte treten auf.

Da die Grenzwertinformationen für jeden Prozeß verwaltet werden, muß die Shell-Anweisung `ulimit` direkt diesen Systemaufruf ausführen, um alle zukünftigen Prozesse zu beeinflussen, die von der Shell erzeugt werden.

Der Wert des aktuellen Grenzwerts der folgenden Betriebsmittel beeinflußt diese implementierungsabhängigen Konstanten:

Grenzwert	Implementierungsabhängige Konstante
RLIMIT_FSIZE	FCHR_MAX
RLIMIT_NOFILE	OPEN_MAX

Unter den folgenden Bedingungen schlagen die Funktionen `getrlimit` und `setrlimit` fehl und setzen `errno` auf:

`EINVAL` wenn ein ungültiges *Betriebsmittel* angegeben wurde oder während eines `setrlimit`-Aufrufs der neue `rlim_cur`-Grenzwert den neuen `rlim_max`-Grenzwert überschreitet.

`EPERM` wenn der Grenzwert aus `setrlimit` den maximalen Grenzwert erhöhen würde und der aufrufende Prozeß nicht der Systemverwalter ist.

SIEHE AUCH

`malloc`(3C), `open`(2), `sigaltstack`(2), `signal`(5).

ERGEBNIS

Nach erfolgreicher Ausführung liefert die Funktion `getrlimit` den Wert 0; ansonsten wird der Wert -1 zurückgegeben und `errno` gesetzt.

getsid - Sitzungsnummer lesen

```
#include <sys/types.h>
pid_t getsid(pid_t pid);
```

Die Funktion `getsid` liefert die Sitzungsnummer des Prozesses mit der Prozeßnummer *pid*. Wenn *pid* gleich `(pid_t)0` ist, liefert `getsid` die Sitzungsnummer des aufrufenden Prozesses zurück.

Die Funktion `getsid` schlägt unter den folgenden Bedingungen fehl und setzt `errno` auf die folgenden Werte:

`EPERM` Der Prozeß mit der Prozeßnummer *pid* ist nicht in derselben Sitzung wie der aufrufende Prozeß, und die Implementierung unterstützt den Zugriff des aufrufenden Prozesses auf die Sitzungsnummer des angegebenen Prozesses nicht.

`ESRCH` Es gibt keinen Prozeß mit der Prozeßnummer *pid*.

SIEHE AUCH

`exec`(2), `fork`(2), `getpid`(2), `setpgid`(2), `setsid`(2).

ERGEBNIS

Nach erfolgreicher Durchführung liefert die Funktion `getsid` die Sitzungsnummer des angegebenen Prozesses zurück; tritt ein Fehler auf, wird der Wert `(pid_t)`-1 zurückgegeben und `errno` gesetzt.

getuid, geteuid, getgid, getegid - Benutzer-/Gruppennummer

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid(void);
uid_t geteuid(void);
gid_t getgid(void);
gid_t getegid(void);
```

`getuid` gibt die reale Benutzernummer des aufrufenden Prozesses zurück.

`geteuid` gibt die effektive Benutzernummer des aufrufenden Prozesses zurück.

`getgid` gibt die reale Gruppennummer des aufrufenden Prozesses zurück.

`getegid` gibt die effektive Gruppennummer des aufrufenden Prozesses zurück.

SIEHE AUCH

`intro`(2), `setuid`(2).

ioctl - Geräte und STREAMS steuern

```
#include <unistd.h>
int ioctl(int fildes, int request, .../* arg */);
```

`ioctl` führt eine Vielzahl an Steuerfunktionen für Geräte und STREAMS aus. Bei Nicht-STREAMS-Dateien sind die von diesem Aufruf ausgeführten Funktionen gerätespezifische Steuerfunktionen. Die Parameter *request* und ein optionales drittes Argument mit variierendem Typ werden an die mit *fildes* bezeichnete Datei weitergereicht und vom Gerätetreiber interpretiert. Diese Steuerung wird für Nicht-STREAMS-Geräte selten verwendet, bei denen die Ein/Ausgabe-Grundfunktionen normalerweise über die Systemaufrufe `read`(2) und `write`(2) ausgeführt werden.

Für STREAMS-Dateien werden spezifische Funktionen, wie in `streamio`(7) beschrieben, durch den Aufruf `ioctl` ausgeführt.

fildes ist ein offener Dateideskriptor, der sich auf ein Gerät bezieht. *request* wählt die auszuführende Steuerfunktion aus und hängt jeweils von den adressierten Geräten ab. *arg* beinhaltet zusätzliche Informationen, die von diesen spezifischen Geräten zur Ausführung der angefragten Funktion benötigt werden. Der Datentyp von *arg* hängt von der jeweiligen Steuerfunktion ab, ist jedoch entweder eine ganze Zahl oder ein Zeiger auf eine gerätespezifische Datenstruktur.

Außer den gerätespezifischen und STREAMS-Funktionen werden generische Funktionen von mehr als einem Gerätetreiber, z.B. die allgemeine Terminalschnittstelle, bereitgestellt (siehe `termio`(7)).

`ioctl` ist bei jedem Dateityp erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EBADF` *fildes* ist kein gültiger offener Dateideskriptor.

`ENOTTY` *fildes* bezeichnet nicht einen Gerätetreiber, der Steuerfunktionen akzeptiert.

`EINTR` Ein Signal wurde während des Systemaufrufs `ioctl` abgefangen.

`ioctl` ist außerdem erfolglos, wenn der Gerätetreiber einen Fehler feststellt. In diesem Fall wird der Fehler durch `ioctl` ohne Änderung an den Aufrufer weitergeleitet. Ein bestimmter Treiber hat nicht unbedingt alle nachstehenden Fehlerfälle. Unter den folgenden Bedingungen sind Anforderungen an Gerätetreiber erfolglos, und `errno` wird auf folgende Werte gesetzt:

`EFAULT` *request* fordert eine Datenübertragung auf einen bzw. von einem Puffer, auf den *arg* zeigt, wobei jedoch ein Teil des Puffers außerhalb des dem Prozeß zugewiesenen Adreßraums liegt.

EINVAL	*request* oder *arg* sind für dieses Gerät nicht gültig.
EIO	Ein physikalischer E/A-Fehler ist aufgetreten.
ENXIO	*request* und *arg* sind für diesen Gerätetreiber gültig, jedoch kann der angeforderte Dienst nicht auf diesem Gerät ausgeführt werden.
ENOLINK	*fildes* befindet sich auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

STREAMS-Fehler sind in `streamio`(7) beschrieben.

SIEHE AUCH

`streamio`(7) in "Leitfaden für Programmierer: STREAMS".
`termio`(7) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Nach erfolgreicher Beendigung hängt der zurückgegebene Wert jeweils von der Geräte-Steuerfunktion ab, muß jedoch eine nicht negative ganze Zahl sein. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

kill - Signal an Prozeß oder Prozeßgruppe senden

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

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

`kill` sendet ein Signal an einen Prozeß oder an eine Gruppe von Prozessen. Der Prozeß oder die Gruppe von Prozessen, an die das Signal gesendet werden soll, wird durch *pid* angegeben. Das zu sendende Signal wird durch *sig* angegeben und ist entweder ein Signal von der in `signal` (siehe `signal`(5)) angeführten Liste oder 0. Wenn *sig* 0 (das Nullsignal) ist, erfolgt zwar die Fehlerprüfung, jedoch wird tatsächlich kein Signal geschickt. Dies kann zur Prüfung der Gültigkeit von *pid* verwendet werden.

Die reale oder effektive Benutzernummer des sendenden Prozesses muß mit der realen oder (von `exec`(2)) gesicherten Benutzernummer des empfangenden Prozesses übereinstimmen, wenn die effektive Benutzernummer des sendenden Prozesses nicht diejenige des Systemverwalters ist (siehe `intro`(2)), oder *sig* ist `SIGCONT`, und der sendende Prozeß hat die gleiche Sitzungskennung wie der empfangende Prozeß.

Die Prozesse mit Prozeßnummer 0 und Prozeßnummer 1 sind spezielle Systemprozesse (siehe `intro`(2)) und werden nachstehend als proc0 bzw. proc1 bezeichnet.

Wenn *pid* größer als Null ist, wird *sig* an den Prozeß geschickt, dessen Prozeßnummer *pid* ist. *pid* kann gleich 1 sein.

Wenn *pid* negativ, aber nicht (`pid_t`) -1 ist, wird *sig* an alle Prozesse gesendet, deren Prozeßgruppennummer gleich dem absoluten Wert von *pid* ist, und für die der Prozeß die Erlaubnis hat, ein Signal zu schicken.

Wenn *pid* (`pid_t`) 0 ist, wird *sig* an alle Prozesse mit Ausnahme von proc0 bzw. proc1 geschickt, deren Prozeßgruppennummer gleich der Prozeßgruppennummer des Absenders ist. Um ein Signal an eine Prozeßgruppe zu schicken, wird eine Erlaubnis benötigt.

Wenn *pid* (`pid_t`) -1 ist, und die effektive Benutzernummer des sendenden Prozesses nicht diejenige des Systemverwalters ist, wird *sig* an alle Prozesse mit Ausnahme von proc0 bzw. proc1 gesendet, deren reale Benutzernummer gleich der effektiven Benutzernummer des Absenders ist.

Wenn *pid* (`pid_t`) -1, ist und die effektive Benutzernummer des Absenders diejenige des Systemverwalters ist, wird *sig* an alle Prozesse mit Ausnahme von proc0 bzw. proc1 gesendet.

`kill` ist erfolglos, und kein Signal wird gesendet, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL` *sig* ist keine gültige Signalnummer.

`EINVAL` *sig* ist `SIGKILL` und *pid* ist `(pid_t)1` (d.h. *pid* spezifiziert proc1).

`ESRCH` Kein Prozeß (oder Prozeßgruppe) kann gefunden werden, der dem von *pid* angegebenen Wert entspricht.

`EPERM` Die Benutzernummer des sendenden Prozesses ist nicht diejenige des Systemverwalters, und seine reale bzw. effektive Benutzernummer entspricht nicht der realen oder gesicherten Nummer des empfangenden Prozesses, und der aufrufende Prozeß sendet nicht gerade `SIGCONT` an einen Prozeß, der dieselbe Sitzungsnummer hat.

SIEHE AUCH

`getpid`(2), `intro`(2), `setpgrp`(2), `signal`(2), `getsid`(2), `sigsend`(2), `sigaction`(2).
`kill`(1) in "SINIX V5.41 Kommandos".

HINWEIS

`sigsend` ist ein vielseitigerer Weg, um Signale an Prozesse zu senden. Dem Benutzer wird empfohlen, `sigsend` anstelle von `kill` zu benutzen.

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

link - Verweis auf eine Datei einrichten

```
#include <unistd.h>
int link(const char *pfad1, const char *pfad2);
```

pfad1 weist auf einen Pfadnamen, der eine vorhandene Datei benennt. *pfad2* weist auf einen Pfadnamen, der den neu zu erstellenden Dateiverzeichnis-Eintrag benennt. `link` erstellt einen neuen Verweis (Dateiverzeichnis-Eintrag) auf die vorhandene Datei und erhöht den Verweiszähler um 1.

Bei erfolgreicher Beendigung markiert `link` zum Aktualisieren das `st_ctime`-Feld der Datei. Auch die `st_ctime`- und `st_mtime`-Felder des Dateiverzeichnisses, welche die neuen Einträge enthalten, werden zur Aktualisierung markiert.

`link` ist erfolglos, und es wird kein Verweis erstellt, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Für eine Komponente eines Pfades existiert keine Suchberechtigung.

`EACCES` Der gewünschte Verweis verlangt das Schreiben in ein Dateiverzeichnis, für das keine Schreiberlaubnis existiert.

`EEXIST` Die Verbindung, benannt durch *pfad2*, ist vorhanden.

`EFAULT` *pfad1* oder *pfad2* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während des Systemaufrufs `link` abgefangen.

`ELOOP` Zu viele symbolische Verweise wurden beim Übersetzen von *pfad1* oder *pfad2* angetroffen.

`EMLINK` Die Höchstzahl der Verweise auf eine Datei würde überschritten.

`EMULTIHOP` Die Komponenten von *pfad1* oder *pfad2* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

`ENAMETOOLONG`
Die Länge des *pfad1*- oder *pfad2*-Arguments übersteigt `PATH_MAX`, oder die Länge einer *pfad1*- oder *pfad2*-Komponenten übersteigt `NAME_MAX`, während `_POSIX_NO_TRUNC` wirksam ist.

`ENOTDIR` Eine Komponente der Pfade ist kein Dateiverzeichnis.

`ENOENT` *pfad1* oder *pfad2* ist ein leerer Pfadname.

`ENOENT` Eine Komponente der Pfade ist nicht vorhanden.

`ENOENT` Die durch *pfad1* angegebene Datei ist nicht vorhanden.

ENOLINK	*pfad1* oder *pfad2* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.
ENOSPC	Das Verzeichnis, das den Verweis enthalten soll, kann nicht erweitert werden.
EPERM	Die durch *pfad1* angegebene Datei ist ein Dateiverzeichnis, und die effektive Benutzernummer ist nicht diejenige des Systemverwalters.
EROFS	Der gewünschte Verweis erfordert Schreiberlaubnis in einem Dateiverzeichnis, dessen Dateisystem schreibgeschützt ist.
EXDEV	Der durch *pfad2* angegebene Verweis und die durch *pfad1* angegebene Datei befinden sich auf verschiedenen logischen Geräten (Dateisystemen).

SIEHE AUCH

`unlink`(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

lseek - Schreib-/Lesezeiger positionieren

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

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

fildes ist ein Dateideskriptor, der von einem `creat`-, `open`-, `dup`- oder `fcntl`-Systemaufruf zurückgegeben wird. `lseek` setzt den zu *fildes* gehörenden Schreib-/Lesezeiger wie folgt:

- Wenn *whence* `SEEK_SET` ist, wird der Zeiger auf *offset* Bytes gesetzt.
- Wenn *whence* `SEEK_CUR` ist, wird der Zeiger auf die aktuelle Position plus *offset* gesetzt.
- Wenn *whence* `SEEK_END` ist, wird der Zeiger auf die Größe der Datei plus *offset* gesetzt.

Bei Erfolg wird der sich ergebende Zeigerwert, gemessen in Bytes ab Dateibeginn, durch `lseek` zurückgegeben. Es ist zu beachten, daß in den Fällen, in denen *fildes* der Dateideskriptor einer fernen Datei und *offset* negativ ist, *lseek* auch den Schreib-/Lesezeiger zurückgibt, wenn dieser negativ ist.

`lseek` erlaubt, daß der Schreib-/Lesezeiger hinter die existierenden Daten der Datei gesetzt werden kann. Wenn später Daten an dieser Stelle geschrieben werden, geben nachfolgende Lese-Operationen in der Lücke zwischen dem vorherigen Ende der Daten und den neu eingegebenen Daten solange Bytes vom Wert 0 zurück, bis Daten in die Lücke geschrieben worden sind.

`lseek` ist erfolglos, und der Schreib-/Lesezeiger bleibt unverändert, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EBADF` *fildes* ist kein offener Dateideskriptor.

`ESPIPE` *fildes* bezeichnet eine Pipe oder eine FIFO-Datei.

`EINVAL` *whence* ist nicht `SEEK_SET`, `SEEK_CUR`, oder `SEEK_END`. Der Prozeß bekommt auch ein `SIGSYS`-Signal.

`EINVAL` *fildes* ist kein Dateideskriptor einer fernen Datei, und der sich ergebende Dateizeiger würde negativ sein.

Einige Geräte können nicht positionieren. Der Wert des Dateizeigers für ein solches Gerät ist nicht definiert.

SIEHE AUCH

creat(2), dup(2), fcntl(2), open(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nicht negative ganze Zahl zurückgegeben, die den Dateizeiger-Wert anzeigt. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

memcntl - Speicherverwaltung kontrollieren

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

int memcntl(caddr_t addr, size_t len, int cmd, caddr_t arg, int attr, int mask);
```

Die Funktion `memcntl` erlaubt dem aufrufenden Prozeß die Anwendung einer Vielzahl von Kontrolloperationen auf den Adreßbereich, der durch [*addr*, *addr* + *len*) festgelegt ist.

addr muß ein Vielfaches der Seitenlänge sein, die von `sysconf`(3C) zurückgegeben wird. Der Geltungsbereich der Kontrolloperationen kann weiterhin mit weiteren Auswahlkriterien (in der Form von Attributen) durch Verknüpfung von Bitmustern in *attr* belegt werden.

Die folgenden Attribute geben die Auswahlkriterien für das Mapping (die Abbildung) der Seiten an:

`SHARED`	Seite ist als gemeinsam benutzbar abgebildet
`PRIVATE`	Seite ist als privat abgebildet

Die folgenden Attribute geben die Auswahlkriterien für den Seitenschutz an:

`PROT_READ`	Seite kann gelesen werden.
`PROT_WRITE`	Seite kann geschrieben werden.
`PROT_EXEC`	Seite kann ausgeführt werden.

Die Auswahlkriterien werden durch ODER-Verknüpfung der Attributmasken zusammengesetzt.

Außerdem können folgende Kriterien angegeben werden:

`PROC_TEXT`	Prozeßtext
`PROC_DATA`	Prozeßdaten

wobei `PROC_TEXT` alle privat abgebildeten Segmente mit Lese- und Ausführungsrechten angibt, und `PROC_DATA` alle privat abgebildeten Segmente mit Schreibrechten angibt.

Auswahlkriterien können benutzt werden, um verschiedene abstrakte Speicherobjekte innerhalb des verfügbaren Adreßbereichs zu beschreiben. Wenn eine Operation nicht durch Auswahlkriterien beschränkt sein soll, dann muß *attr* den Wert `0` enthalten.

Die auszuführende Operation wird durch das Argument *cmd* identifiziert. Die symbolischen Namen für die Operationen werden in der Datei `sys/mman.h` wie folgt definiert:

`MC_LOCK` sperrt alle Seiten im Bereich mit den Attributen *attr* im Speicher. Eine gegebene Seite kann mehrmals über unterschiedliche Mappings gesperrt werden; innerhalb eines gegebenen Mappings verschachteln sich die Seitensperren nicht. Mehrfache Sperroperationen mit derselben Adresse im selben Prozeß können durch einen Befehl freigegeben werden. Ist eine Seite in einem Prozeß gesperrt und in einem anderen abgebildet (oder sichtbar über ein anderes Mapping im sperrenden Prozeß), so wird sie im Speicher gesperrt, bis der sperrende Prozeß eine implizite oder explizite Freigabeoperation ausführt. Wenn ein gesperrtes Mapping entfernt wird oder eine Seite durch das Löschen einer Datei oder Abschneiden einer Datei gelöscht wird, wird eine implizite Freigabeoperation ausgeführt. Wenn eine schreibbare `MAP_PRIVATE`-Seite im Adreßbereich verändert wird, wird die Sperre auf die private Seite übertragen.

Momentan ist *arg* nicht in Verwendung, jedoch muß dieser Parameter `0` enthalten, um Kompatibilität mit zukünftigen Veränderungen zu gewährleisten.

`MC_LOCKAS` sperrt im Speicher alle Seiten, die vom Adreßbereich mit den Attributen *attr* abgebildet sind. Die Parameter *addr* und *len* werden gegenwärtig nicht verwendet, müssen jedoch `NULL` bzw. `0` sein, um Kompatibilität mit zukünftigen Veränderungen zu gewährleisten. *arg* ist ein Bitmuster, das die folgenden Werte enthalten kann:

`MCL_CURRENT`	aktuelle Abbildungen sperren
`MCL_FUTURE`	zukünftige Abbildungen sperren

Der Wert von *arg* bestimmt, ob die zu sperrenden Seiten momentan im Adreßbereich abgebildet sind, zukünftig abgebildet sind, oder aber beides. Wenn `MCL_FUTURE` angegeben wird, werden alle nachfolgenden Abbildungen, die dem Adreßbereich hinzugefügt werden, ebenfalls gesperrt, vorausgesetzt, daß genügend Speicher verfügbar ist.

MC_SYNC speichert alle veränderten Seiten im Bereich mit den Attributen *attr*. Wahlweise werden die Kopien im Cache ungültig gemacht. Der Speicher für eine veränderte MAP_SHARED-Abbildung ist die Datei, auf die die Seite abgebildet wird; der Speicher für eine veränderte MAP_PRIVATE-Abbildung ist der Swap-Bereich. *arg* enthält ein Bitmuster, daß sich aus den Werten zusammensetzt, welche das Verhalten der Operation bestimmen:

MS_ASYNC	asynchrone Schreibzugriffe ausführen
MS_SYNC	synchrone Schreibzugriffe ausführen
MS_INVALIDATE	Abbildungen ungültig machen

MS_ASYNC kehrt zurück, sobald alle Schreiboperationen vom Scheduler gesteuert werden; der Systemaufruf kehrt bei MS_SYNC nicht zurück, solange nicht alle Schreiboperationen vollständig ausgeführt wurden.

MS_INVALIDATE macht alle Datenkopien im Cache des Speichers ungültig, so daß bei weiteren Referenzen auf die Seiten diese vom System vom Sekundärspeicher gelesen werden müssen. Diese Operation sollte von Applikationen verwendet werden, welche ein Speicherobjekt in einem bekannten Zustand voraussetzen.

MC_UNLOCK gibt alle Seiten im Bereich mit den Attributen *attr* frei. Gegenwärtig wird *arg* nicht verwendet; dieser Parameter muß jedoch 0 enthalten, um Kompatibilität mit zukünftigen Veränderungen zu gewährleisten.

MC_UNLOCKAS entfernt die Speichersperren im Adreßbereich und die Sperren auf allen Seiten im Adreßbereich mit den Attributen *attr*. Momentan werden *addr*, *len*, und *arg* zwar nicht benutzt, aber sie müssen NULL bzw. 0 sein, um Kompatiblität mit zukünftigen Veränderungen zu gewährleisten.

Das Argument *mask* muß Null sein; es ist für zukünftige Benutzung reserviert.

Sperren, die mit Sperroperationen errichtet wurden, werden von einem Sohnprozeß nach einem fork nicht geerbt. memcntl schlägt fehl, wenn versucht wird, mehr Speicher zu sperren, als der Grenzwert des Systems erlaubt.

Aufgrund der möglichen Auswirkungen auf die Systemressourcen sind alle Operationen mit Ausnahme von MC_SYNC auf Prozesse beschränkt, welche die effektive Benutzernummer des Systemverwalters haben. Die Funktion memcntl subsummiert die Operationen von plock und mctl.

Unter den folgenden Bedingungen schlägt die Funktion `memcntl` fehl und setzt `errno` auf :

`EAGAIN` wenn der Speicher nicht ganz oder teilweise von der Operation gesperrt werden kann, wenn `MC_LOCK` oder `MC_LOCKAS` angegeben ist.

`EBUSY` wenn die Adressen im Bereich [*addr, addr + len*) ganz oder teilweise gesperrt sind und die Option `MC_SYNC` zusammen mit `MS_INVALIDATE` angegeben wurde.

`EINVAL` wenn *addr* nicht ein Vielfaches der Seitenlänge (die von `sysconf` zurückgeliefert wird) ist.

`EINVAL` wenn *addr* und/oder *len* nicht den Wert `0` enthalten, wenn `MC_LOCKAS` oder `MC_UNLOCKAS` angegeben wird.

`EINVAL` wenn *arg* für die angegebene Funktion nicht gültig ist.

`EINVAL` wenn ungültige Auswahlkriterien in *attr* angegeben wurden.

`ENOMEM` wenn die Adressen im Bereich [*addr, addr + len*) ganz oder teilweise für den Adreßbereich des Prozesses ungültig sind oder nicht verzeichnete Seiten angegeben wurden.

`EPERM` wenn die effektive Benutzernummer des Prozesses nicht die des Systemverwalters ist und eine der Optionen `MC_LOCK`, `MC_LOCKAS`, `MC_UNLOCK` oder `MC_UNLOCKAS` angewählt wurde.

SIEHE AUCH

`mmap`(2), `mprotect`(2), `plock`(2), `sysconf`(2), `mlock`(3C), `mlockall`(3C), `msync`(3C).

ERGEBNIS

Nach erfolgreicher Beendigung liefert `memcntl` den Wert 0; ansonsten wird -1 zurückgegeben und `errno` gesetzt.

mincore - Residenz von Speicherseiten bestimmen

```
#include <unistd.h>
int mincore(caddr_t addr, size_t len, char *vec);
```

`mincore` liefert den primären Speicherresidenzstatus von Seiten im Adreßbereich, der von Abbildungen im Bereich [*addr*, *addr* + *len*) abgedeckt wird. Der Status wird als Zeichen pro Seite im Feld **vec* abgelegt, welches groß genug sein muß, um die Statusinformationen aller Seiten im Adreßbereich aufzunehmen. Das niederwertige Bit jedes Zeichens wird auf 1 gesetzt, um anzuzeigen, daß die referierte Seite sich im Primärspeicher befindet, und auf 0, wenn dies nicht der Fall ist. Die Werte der anderen Bits in jedem Zeichen sind undefiniert und können bei zukünftigen Implementierungen weitere Informationen darstellen.

`mincore` liefert Residenzinformation, die für den jeweiligen Zeitpunkt korrekt ist. Da das System die Seiten im Speicher häufig umorganisieren kann, können die Informationen schnell veralten. Nur gesperrte Seiten bleiben garantiert im Speicher (siehe `memcntl`(2)).

`mincore` schlägt fehl, wenn

`EFAULT` **vec* eine Adresse enthält, auf die nicht zugegriffen werden kann.

`EINVAL` *addr* nicht ein Vielfaches der Seitengröße ist, wie sie von `sysconf`(3C) zurückgeliefert wird.

`EINVAL` das Argument *len* einen Wert kleiner oder gleich 0 enthält.

`ENOMEM` Adressen im Bereich [*addr*, *addr* + *len*) ungültig für den Adreßbereich eines Prozesses sind, oder eine oder mehrere Seiten angegeben werden, welche nicht verzeichnet sind.

SIEHE AUCH

`mlock`(3C), `mmap`(2), `sysconf`(3C).

ERGEBNIS

`mincore` liefert bei Erfolg 0 und -1 bei Fehler.

mkdir - Dateiverzeichnis erstellen

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

int mkdir(const char *Pfad, mode_t Modus);
```

`mkdir` erstellt ein neues Dateiverzeichnis mit dem Namen *Pfad*. Der Modus des neuen Dateiverzeichnisses wird mit *Modus* (siehe `chmod`(2) für mögliche Werte für Modus) initialisiert. Der Schutzbitteil des Arguments *Modus* wird durch die Dateimaske des Prozesses verändert (siehe `umask`(2)).

Die Eigentümernummer des Verzeichnisses wird auf die effektive Benutzernummer des Prozesses gesetzt. Die Gruppennummer des Verzeichnisses wird auf die effektive Gruppennummer des Prozesses gesetzt, oder die Gruppennummer dieses Verzeichnisses wird geerbt, wenn das Bit `S_ISGID` im übergeordneten Verzeichnis gesetzt ist. Das Bit `S_ISGID` des neuen Verzeichnisses wird vom übergeordneten Verzeichnis übernommen.

Wenn *Pfad* ein symbolischer Verweis ist, wird er nicht verwendet.

Das neu erzeugte Verzeichnis ist mit Ausnahme der Einträge für sich selbst und sein übergeordnetes Verzeichnis leer.

Nach erfolgreicher Beendigung kennzeichnet `mkdir` die Felder `st_atime`, `st_ctime` und `st_mtime` des Verzeichnisses zur Aktualisierung. Auch die Felder `st_ctime` und `st_mtime` des Verzeichnisses, das den neuen Eintrag enthält, werden zur Aktualisierung gekennzeichnet.

`mkdir` scheitert und erzeugt kein Verzeichnis, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

`EACCES`	Entweder darf eine Komponente des Pfades nicht durchsucht werden, oder die Schreiberlaubnis für das übergeordnete Dateiverzeichnis des neu zu erstellenden Dateiverzeichnisses wird verweigert.
`EEXIST`	Die angegebene Datei ist bereits vorhanden.
`EFAULT`	*Pfad* weist über den zugewiesenen Adreßraum des Prozesses hinaus.
`EIO`	Während des Zugriffs auf das Dateisystem ist ein E/A-Fehler aufgetreten.
`ELOOP`	Bei der Übersetzung von *Pfad* wurden zuviele symbolische Verweise angetroffen.
`EMLINK`	Die Höchstzahl von Verweisen auf das übergeordnete Dateiverzeichnis wurde überschritten.
`EMULTIHOP`	Die Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

ENAMETOOLONG
: Die Länge des Arguments *Pfad* überschreitet PATH_MAX, oder die Länge einer *Pfad*-Komponente überschreitet NAME_MAX, während _POSIX_NO_TRUNC aktiv ist.

ENOENT
: Eine Komponente des Pfades ist nicht vorhanden oder ist ein Null-Pfadname.

ENOLINK
: *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOSPC
: Auf dem Gerät, das das Verzeichnis enthält, ist kein freier Platz verfügbar.

ENOTDIR
: Eine Pfadkomponente ist kein Verzeichnis.

EROFS
: Die angegebene Datei steht in einem schreibgeschützten Dateisystem.

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

SIEHE AUCH

chmod(2), mknod(2), umask(2), stat(5).

mknod - Dateiverzeichnis, Gerätedatei oder normale Datei erstellen

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

int mknod(const char *path, mode_t mode, dev_t dev);
```

`mknod` erstellt eine neue Datei, die mit dem Pfadnamen angegeben wird, auf den *path* zeigt. Der Dateityp und die Zugriffsrechte der neuen Datei werden von *mode* bestimmt.

Der in *mode* angegebene Dateityp wird durch die Bits `S_IFMT` angegeben, die einen der folgenden Werte haben müssen:

`S_IFIFO`	FIFO-Datei
`S_IFCHR`	zeichenorientierte Datei
`S_IFDIR`	Verzeichnis
`S_IFBLK`	blockorientierte Datei
`S_IFREG`	normale Datei

Die Zugriffsrechte der Datei werden in *mode* durch die Bits 0007777 angegeben und können durch bitweises ODER der folgenden Werte erzeugt werden:

```
S_ISUID   04000   Setzen der Benutzernummer bei Ausführung
S_ISGID   020#0   Setzen der Gruppennummer, wenn # 7, 5, 3, oder 1 ist
                  Ermöglichen der Datei-/Satzsperre, wenn # 6,4,2, oder 0 ist
S_ISVTX   01000   Aufbewahren des Textsegments nach Ausführung
S_IRWXU   00700   Lesen, Schreiben, Ausführen durch Eigentümer
S_IRUSR   00400   Lesen durch Eigentümer
S_IWUSR   00200   Schreiben durch Eigentümer
S_IXUSR   00100   Ausführen bzw. Suchen durch Eigentümer
S_IRWXG   00070   Lesen, Schreiben, Ausführen durch Gruppe
S_IRGRP   00040   Lesen durch Gruppe
S_IWGRP   00020   Schreiben durch Gruppe
S_IXGRP   00010   Ausführen durch Gruppe
S_IRWXO   00007   Lesen, Schreiben, Ausführen bzw. Suchen durch Alle
S_IROTH   00004   Lesen durch Alle
S_IWOTH   00002   Schreiben durch Alle
S_IXOTH   00001   Ausführen durch Alle
```

Die Benutzernummer der Datei wird auf die effektive Benutzernummer des Prozesses gesetzt. Die Gruppennummer wird auf die effektive Gruppennummer des Prozesses gesetzt. Sollte aber das `S_ISGID`-Bit im übergeordneten Verzeichnis gesetzt sein, wird die Gruppennummer des übergeordneten Verzeichnisses übernommen. Wenn die Gruppennummer nicht mit der effektiven Gruppennummer oder einer unterstützten Gruppennummer übereinstimmt, wird das `S_ISGID`-Bit gelöscht.

Die Bits der Zugriffrechte in *mode* werden durch die Dateierzeugungsmaske des Prozesses geändert: alle Bits, die in der Dateierzeugungsmaske gesetzt sind, werden gelöscht (siehe `umask`(2)). Falls *mode* eine zeichen- oder blockorientierte Datei angibt, ist *dev* die konfigurationsabhängige Angabe dieser Datei. Falls *mode* keine zeichen- oder blockorientierte Datei angibt, wird *dev* ignoriert. Siehe `mkdev`(3C).

Alle Dateitypen, außer einer FIFO-Datei, können mit mknod nur durch den Systemverwalter aufgerufen werden.

Wenn *path* ein symbolischer Verweis ist, wird er nicht verfolgt.

mknod scheitert und erzeugt keine neue Datei, wenn einer der folgenden Punkte wahr ist:

EEXIST Die angegebene Datei existiert.

EINVAL *dev* ist ungültig.

EFAULT *path* zeigt außerhalb des Adreßraums des Prozesses.

ELOOP Bei der Übersetzung von *path* wurden zuviele symbolische Verweise angetroffen.

EMULTIHOP Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

ENAMETOOLONG
Die Länge des *path*-Arguments überschreitet PATH_MAX, oder die Länge einer Komponente von *path* überschreitet NAME_MAX, während _POSIX_NO_TRUNC aktiv ist.

ENOTDIR Eine Komponente des Pfadpräfixes ist kein Verzeichnis.

ENOENT Eine Komponente des Pfadpräfixes existiert nicht oder ist ein leerer Pfadname.

EPERM Die effektive Benutzernummer ist nicht die des Systemverwalters.

EROFS Das Verzeichnis, in dem die Datei erstellt werden soll, ist in einem Dateisystem, das nur gelesen werden kann.

ENOSPC Es ist kein Speicherplatz vorhanden.

EINTR Während des Systemaufrufs mknod wurden ein Signal empfangen.

ENOLINK *path* verweist auf einen fernen Rechner und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

chmod(2), exec(2), umask(2), mkdev(3C), mkfifo(3C), fs(4), stat(5).
mkdir(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Bei erfolgreicher Ausführung wird 0 zurückgegeben. Ansonsten wird -1 zurückgegeben und `errno` zur Anzeige des Fehlers gesetzt.

HINWEIS

Wenn `mknod` mit RFS (remote file sharing) in einem fernen Verzeichnis eine Gerätedatei erzeugt, werden Geräteklasse und Gerätenummer vom Server interpretiert.

mmap - Speicherseiten abbilden

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

caddr_t mmap(caddr_t addr, size_t len, int prot, int flags, int fd, off_t off);
```

mmap stellt eine Abbildung zwischen dem Adreßbereich eines Prozesses an der Adresse *pa* für *len* Bytes und dem Speicherobjekt, das durch den Dateideskriptor *fd* dargestellt wird, an dem Offset *off* für *len*-Bytes her. Der Wert von *pa* ist eine implementierungsabhängige Funktion von *addr* und dem Wert *flags*. Ein erfolgreicher Aufruf von mmap liefert *pa* als Ergebnis zurück. Die Adreßbereiche, die durch [*pa, pa + len*) und [*off, off + len*) abgedeckt werden, müssen für den möglichen (nicht notwendigerweise den aktuellen) Adreßbereich eines Prozesses und des in Frage kommenden Objekts zulässig sein. mmap kann eine Datei nicht vergrößern.

Die Abbildung, die durch mmap hergestellt wird, ersetzt alle vorhergehenden Abbildungen für die Seiten des Prozesses im Bereich [*pa, pa + len*).

Der Parameter *prot* bestimmt, ob gelesen, geschrieben, ausgeführt oder Kombinationen dieser Zugriffe auf die abgebildeten Seiten erlaubt werden sollen. Die Schutzoptionen werden in sys/mman.h wie folgt definiert:

PROT_READ	Seite kann gelesen werden.
PROT_WRITE	Seite kann geschrieben werden.
PROT_EXEC	Seite kann ausgeführt werden.
PROT_NONE	auf Seite kann nicht zugegriffen werden.

Nicht alle Implementierungen bieten alle möglichen Kombinationen. PROT_WRITE wird oft als PROT_READ|PROT_WRITE implementiert und PROT_EXEC als PROT_READ|PROT_EXEC. Trotzdem erlaubt keine Implementierung einen Schreibzugriff, wenn PROT_WRITE nicht gesetzt worden ist. Das Verhalten von PROT_WRITE kann durch die Option MAP_PRIVATE in dem Parameter *flags* beeinflußt werden, wie weiter unten noch näher beschrieben wird.

Der Parameter *flags* bietet andere Informationen über die Behandlung von abgebildeten Seiten. Die Optionen werden in sys/mman.h wie folgt definiert:

MAP_SHARED	Änderungen gemeinsam benutzbar machen
MAP_PRIVATE	Änderungen privat halten
MAP_FIXED	addr exakt interpretieren

MAP_SHARED und MAP_PRIVATE beschreiben die Disposition von Schreibzugriffen auf das Speicherobjekt. Wenn MAP_SHARED angegeben wird, ändern Schreibzugriffe das Speicherobjekt. Wenn MAP_PRIVATE angegeben wird, erzeugt der erste Schreibzugriff eine privat gehaltene Kopie der Seite des Speicherobjekts und leitet die Abbildung an die Kopie um. Es muß entweder MAP_SHARED oder MAP_PRIVATE angegeben werden, jedoch nicht beide Optionen zusammen. Der Abbildungstyp wird nach einem fork(2) beibehalten.

Beachten Sie, daß die privat gehaltene Kopie nicht erzeugt wird, bis der erste Schreibzugriff auftritt; bis dahin können andere Benutzer, die das Objekt als MAP_SHARED abgebildet haben, das Objekt ändern.

MAP_FIXED informiert das System darüber, daß der Wert von *pa* genau *addr* entsprechen muß. Die Benutzung von MAP_FIXED wird nicht empfohlen, da sie eine Implementierung davon abhalten kann, die Systemressourcen effektiv zu benutzen.

Wenn MAP_FIXED nicht gesetzt ist, verwendet das System *addr* in einer implementierungsabhängigen Weise, um *pa* zu erhalten. *pa* wird so gewählt, daß ein Bereich aus dem Adreßbereich ausgewählt wird, den das System zur Abbildung von *len*-Bytes auf das angegebene Objekt für passend hält. Alle Implementierungen interpretieren einen *addr*-Wert von Null dahingehend, daß *pa* frei gewählt werden kann (siehe unten). Enthält *addr* einen Wert ungleich Null, so wird dies als Vorschlag gewertet, die Abbildung nahe an einer Prozeßadresse zu wählen. Wenn das System einen Wert für *pa* auswählt, so wird dies niemals die Adresse 0 sein, noch werden Abbildungen ersetzt oder gar Abbildungen in mögliche Daten- oder Stack-Bereiche plaziert.

Der Parameter *off* wird entsprechend des Rückgabewerts von sysconf beschränkt. Wenn MAP_FIXED angegeben wird, muß der Parameter *addr* ebenfalls diesen Beschränkungen entsprechen. Das System führt Abbildungsoperationen über ganze Seiten aus. Da der Parameter *len* nicht an bestimmte Größen oder Ausrichtungen gebunden ist, wird das System jede Restseite, die bei der Abbildungsoperation des Bereiches [*pa, pa + len*) anfällt, mit in die Operation einbeziehen.

Das System füllt Teilseiten am Ende eines Objekts mit Nullen. Referenzen auf ganze Seiten hinter dem Objekt erzeugen ein SIGBUS-Signal. SIGBUS-Signale können außerdem bei verschiedenen Fehlerbedingungen des Dateisystems gesendet werden, einschließlich Überschreitung des Quotas.

Unter den folgenden Bedingungen schlägt mmap fehl und setzt errno auf folgende Werte:

EAGAIN	Die Abbildung kann im Speicher nicht gesperrt werden.
EBADF	*fd* ist nicht geöffnet.
EACCES	*fd* ist nicht zum Lesen geöffnet, unabhängig von dem angegebenen Schutzmechanismus, oder *fd* ist nicht zum Schreiben geöffnet und PROT_WRITE wurde als eine Abbildung vom Typ MAP_SHARED angefordert.
ENXIO	Adressen im Bereich [*off, off + len*) sind für *fd* ungültig.
EINVAL	Die Argumente *addr* (wenn MAP_FIXED angegeben wurde) oder *off* enthalten nicht Vielfache der Seitenlänge, wie sie von sysconf zurückgeliefert wird.
EINVAL	Das Feld in *flags* ist ungültig (entweder MAP_PRIVATE oder MAP_SHARED).
EINVAL	Das Argument *len* hat einen Wert kleiner oder gleich 0.

ENODEV *fd* bezieht sich auf ein Objekt, für das `mmap` bedeutungslos ist, wie zum Beispiel ein Terminal.

ENOMEM `MAP_FIXED` wurde angegeben, und der Bereich [*addr, addr + len*) überschreitet den erlaubten Adreßbereich eines Prozesses, oder `MAP_FIXED` wurde nicht angegeben, und es gibt nicht genügend Speicherplatz im Adreßbereich, um die Abbildung wirksam werden zu lassen.

HINWEIS

`mmap` erlaubt Zugriff auf Ressourcen über Adreßbereichsmanipulationen anstelle der `read/write`-Schnittstelle. Wird eine Datei abgebildet, muß ein Prozeß lediglich auf die Adresse zugreifen, an die das Dateiobjekt abgebildet wird. Man betrachte den folgenden (unvollständigen) Code:

```
fd = open(...)
lseek(fd, offset)
read(fd, buf, len)
/* Daten in buf verwenden */
```

Unter Verwendung von `mmap` kann der Code folgendermaßen umgeschrieben werden:

```
fd = open(...)
address = mmap((caddr_t) 0, len, (PROT_READ | PROT_WRITE),
                MAP_PRIVATE, fd, offset)
/* Daten über address verwenden */
```

SIEHE AUCH

`fcntl`(2), `fork`(2), `lockf`(3C), `mlockall`(3C), `mprotect`(2), `munmap`(2), `plock`(2), `sysconf`(2).

ERGEBNIS

Bei Erfolg liefert `mmap` die Adresse zurück, an der die Abbildung plaziert wurde. Bei Fehler wird (`caddr_t`)-1 zurückgegeben und `errno` gesetzt.

mount - Dateisystem einhängen

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

int mount(const char *spec, const char *dir, int mflag,
          .../* int fstyp, const char *dataptr, size_t datalen */);
```

mount hängt ein aushängbares Dateisystem, das sich in der durch *spec* gekennzeichneten blockorientierten Gerätedatei befindet, in das Dateiverzeichnis *dir* ein. *spec* und *dir* sind Zeiger auf Pfadnamen. *fstyp* ist die Nummer des Dateisystemtyps. Der Systemaufruf sysfs(2) kann zur Bestimmung der Nummer des Dateisystemtyps verwendet werden. Es ist zu beachten, daß der Dateisystemtyp standardmäßig auf den Root-Dateisystemtyp eingestellt ist, wenn weder MS_FSS noch MS_DATA in *mflag* gesetzt sind. *fstyp* wird nur zur Feststellung des Dateisystemtyps verwendet, wenn mindestens einer der beiden Werte gesetzt ist.

Wenn MS_DATA in *mflag* gesetzt ist, erwartet das System die Argumente *dataptr* und *datalen*. Diese beschreiben zusammen einen Block dateisystemspezifischer Daten ab Adresse *dataptr* und mit der Länge *datalen*. Diese Daten werden von dateisystemspezifischem Code im Betriebssystem interpretiert; ihr Format hängt von Dateisystemtyp ab. Ein Dateisystemtyp benötigt diese Daten möglicherweise nicht; in diesem Fall sollten sowohl *dataptr* als auch *datalen* auf 0 gesetzt werden. Bitte beachten Sie, daß MS_FSS unnötig ist und ignoriert wird, wenn MS_DATA ebenfalls gesetzt ist; wenn jedoch MS_FSS gesetzt ist und MS_DATA nicht, wird angenommen, daß sowohl *dataptr* als auch *datalen* 0 sind.

Nach erfolgreicher Beendigung von mount verweist der Name in *dir* auf das Root-Dateiverzeichnis des neu eingehängten Dateisystems.

Das niederwertige Bit von *mflag* wird zur Steuerung der Schreiberlaubnis im eingehängten Dateisystem verwendet; bei 1 ist das Schreiben unzulässig, andernfalls ist das Schreiben je nach der individuellen Zugriffserlaubnis auf die Datei zulässig.

mount kann nur vom Systemverwalter aufgerufen werden. Das Kommando ist nur für das Dienstprogramm mount(1M) vorgesehen.

mount ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EBUSY — *dir* ist zum gegebenen Zeitpunkt bereits eingehängt; eine andere Person hat *dir* als aktuelles Dateiverzeichnis, oder *dir* ist auf andere Weise belegt.

EBUSY — Die zu *spec* gehörende Gerätedatei ist gegenwärtig eingehängt.

EBUSY — Es stehen keine weiteren Einträge in der Einhängetabelle zur Verfügung.

EFAULT *spec*, *dir* oder *datalen* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

EINVAL Der Superblock hat eine ungültige Magic Number, oder *fstyp* ist ungültig.

ELOOP Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

ENAMETOOLONG
Die Länge des Arguments *path* ist größer als PATH_MAX oder NAME_MAX, während POSIX_NO_TRUNC aktiv ist.

ENOENT Eine der angegebenen Dateien ist nicht vorhanden.

ENOTDIR Eine Komponente des Pfades ist kein Dateiverzeichnis.

EPERM Die effektive Benutzernummer ist nicht diejenige des Systemverwalters.

EREMOTE *spec* ist nicht lokal und kann nicht eingehängt werden.

ENOLINK *path* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EMULTIHOP Die Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner.

ENOTBLK *spec* ist keine blockorientierte Gerätedatei.

ENXIO Die zu *spec* gehörende Gerätedatei ist nicht vorhanden.

ENOTDIR *dir* ist kein Dateiverzeichnis.

EROFS *spec* ist schreibgeschützt, und *mflag* fordert die Schreiberlaubnis an.

ENOSPC Der Dateisystemstatus im Superblock ist nicht FsOKAY, und *mflag* fordert die Schreiberlaubnis an.

SIEHE AUCH

sysfs(2), umount(2).
mount(1M), fs(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

mprotect - Zugriffsschutz für Speicherabbildung ändern

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

int mprotect(caddr_t addr, size_t len, int prot);
```

Die Funktion `mprotect` ändert den Zugriffsschutz für die Abbildungen im Bereich [*addr, addr + len*) auf *prot*. Die gültigen Werte für *prot* sind dieselben wie für `mmap` und werden in der Datei `sys/mman.h` wie folgt definiert:

```
PROT_READ     /* Seite kann gelesen werden */
PROT_WRITE    /* Seite kann geschrieben werden */
PROT_EXEC     /* Seite kann ausgeführt werden */
PROT_NONE     /* Seite kann nicht referiert werden */
```

Unter den folgenden Bedingungen schlägt die Funktion `mprotect` fehl und setzt `errno` auf die folgenden Werte:

`EACCES` wenn *prot* einen Schutz angibt, der die Zugriffsrechte des Prozesses auf das zugrundeliegende Speicherobjekt verletzt.

`EAGAIN` wenn *prot* `PROT_WRITE` für eine Abbildung vom Typ `MAP_PRIVATE` angibt und unzureichende Speicherressourcen zum Sperren der privaten Seite reservierbar sind.

`EINVAL` wenn *addr* kein Vielfaches der Seitenlänge ist, welche von `sysconf` zurückgegeben wird.

`EINVAL` wenn das Argument *len* einen Wert kleiner oder gleich 0 enthält.

`ENOMEM` wenn Adressen im Bereich [*addr, addr + len*) für den Adreßbereich eines Prozesses ungültig sind, oder eine oder mehrere Seiten angegeben werden, welche nicht abgebildet werden.

Wenn `mprotect` aus anderen Gründen als `EINVAL` fehlschlägt, können die Schutzmechanismen für einige Seiten im Bereich [*addr, addr + len*) bereits geändert worden sein. Wenn der Fehler bei einigen Seiten an der Adresse *addr2* auftritt, dann werden die Schutzmechanismen aller ganzen Seiten im Bereich [*addr, addr2*] verändert.

SIEHE AUCH

`memcntl`(2), `mmap`(2), `plock`(2), `mlock`(3C), `mlockall`(3C), `sysconf`(3C).

ERGEBNIS

Nach erfolgreicher Ausführung liefert die Funktion `mprotect` den Wert 0; ansonsten wird der Wert -1 zurückgegeben und `errno` gesetzt.

msgctl - Steuerfunktionen für Nachrichten liefern

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, .../* struct msqid_ds *buf */);
```

`msgctl` liefert Steuerfunktionen für Nachrichten, die durch *cmd* angegeben werden. Folgende *cmd*-Steuerfunktionen stehen zur Verfügung:

`IPC_STAT` schreibt den aktuellen Wert jedes Elements der zu *msqid* gehörenden Datenstruktur in die Struktur, auf die *buf* zeigt. Der Inhalt dieser Struktur ist in `intro`(2) definiert.

`IPC_SET` setzt den Wert der nachstehenden Elemente der zu *msqid* gehörenden Datenstruktur auf den entsprechenden Wert aus der Struktur, auf die *buf* zeigt:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode    /* nur niederwertige 9 Bits */
msg_qbytes
```

Diese Funktion kann nur von einem Prozeß ausgeführt werden, der die effektive Benutzernummer des Systemverwalters hat oder den Wert von `msg_perm.cuid` oder `msg_perm.uid` der zu `msqid` gehörenden Datenstruktur hat. Der Wert von `msg_qbytes` kann nur vom Systemverwalter erhöht werden.

`IPC_RMID` löscht die in *msqid* angegebene Kennung der Nachrichten-Warteschlange im System und entfernt die zugehörige Nachrichten-Warteschlange und Datenstruktur. Diese Funktion kann nur von einem Prozeß ausgeführt werden, der die effektive Benutzernummer des Systemverwalters hat oder den Wert von `msg_perm.cuid` oder `msg_perm.uid` in der zu *msqid* gehörenden Datenstruktur hat.

`msgctl` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Bei `IPC_STAT` wird dem aufrufenden Prozeß die Erlaubnis verweigert (siehe `intro`(2)).

`EFAULT` *buf* weist auf eine unzulässige Adresse.

`EINVAL` *msqid* ist keine gültige Kennung einer Nachrichten-Warteschlange.

`EINVAL` *cmd* ist keine gültige Steuerfunktion.

`EINVAL` *cmd* ist `IPC_SET` und `msg_perm.uid` oder `msg_perm.gid` ist ungültig.

EOVERFLOW *cmd* ist IPC_STAT und *uid* oder *gid* ist zu groß, um in der Struktur gespeichert zu werden, auf die *buf* weist.

EPERM ist gleich IPC_RMID oder IPC_SET. Die effektive Benutzernummer des aufrufenden Prozesses ist nicht gleich der Nummer des Systemverwalters oder dem Wert von msg_perm.cuid oder msg_perm.uid in der zu *msqid* gehörenden Datenstruktur.

EPERM *cmd* ist gleich IPC_SET, ein Versuch zur Erhöhung des Wertes von msg_qbytes wird vorgenommen, und die effektive Benutzernummer des aufrufenden Prozesses ist nicht gleich der Nummer des Systemverwalters.

SIEHE AUCH

intro(2), msgget(2), msgop(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

msgget - Kennung für Nachrichten-Warteschlangen bestimmen

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

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

`msgget` gibt die *key* (Schlüssel) zugeordnete Kennung der Nachrichten-Warteschlange zurück.

Eine Kennung der Nachrichten-Warteschlange sowie die zugehörige Nachrichten-Warteschlange und Datenstruktur (siehe `intro` (2)) werden für *key* (Schlüssel) erstellt, wenn einer der folgenden Punkte wahr ist:

- *key* ist gleich `IPC_PRIVATE` .
- *key* wurde noch keine Kennung für eine Nachrichten-Warteschlange zugeordnet und (*msgflg*&`IPC_CREAT`) ist wahr.

Bei der Erzeugung wird die zur neuen Kennung der Nachrichten-Warteschlange gehörende Datenstruktur wie folgt initialisiert:

- `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid` und `msg_perm.gid` werden so gesetzt, daß sie gleich der effektiven Benutzernummer bzw. gleich der effektiven Gruppennummer des aufrufenden Prozesses sind.
- Die niederwertigen 9 Bits von `msg_perm.mode` werden so gesetzt, daß sie den niederwertigen 9 Bits von *msgflg* entsprechen.
- `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime` und `msg_rtime` werden auf 0 gesetzt.
- `msg_ctime` wird auf die aktuellen Uhrzeit gesetzt.
- `msg_qbytes` wird auf den Systemgrenzwert gesetzt.

`msgget` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Eine Kennung der Nachrichten-Warteschlange ist für *key* vorhanden, jedoch wird entsprechend den niederwertigen 9 Bits von *msgflg* keine Zugriffserlaubnis erteilt (siehe `intro`(2)).

`ENOENT` Für *key* ist keine Kennung der Nachrichten-Warteschlange vorhanden, und (*msgflg*&`IPC_CREAT`) ist nicht wahr.

`ENOSPC` Es soll eine Kennung der Nachrichten-Warteschlange erstellt werden, jedoch wird die systembedingte Höchstzahl von insgesamt im System zulässigen Kennungen der Nachrichten-Warteschlangen überschritten.

EEXIST — Für *key* ist eine Kennung der Nachrichten-Warteschlange vorhanden, jedoch sind (*msgflg*&IPC_CREAT) und (*msgflg*&IPC_EXCL) beide wahr.

SIEHE AUCH

intro(2), msgctl(2), msgop(2), stdipc(3C).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nicht negative ganze Zahl, d.h. eine Kennung der Nachrichten-Warteschlange, zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

msgop: msgsnd, msgrcv - Nachrichten senden/lesen

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

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

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

`msgsnd` wird zum Senden einer Nachricht an die Warteschlange verwendet, die durch die Kennung *msqid* angegeben ist. *msgp* zeigt auf einen benutzerdefinierten Puffer, der als erstes ein Feld des Typs `long integer` enthalten muß, das den Typ der Meldung angibt, und dann einen Datenteil, der den Text der Meldung enthält. Folgende Elemente können Teil eines benutzerdefinierten Puffers sein:

```
long mtype;    /* Nachrichtentyp */
char mtext[];  /* Nachrichtentext */
```

`mtype` ist eine positive ganze Zahl, die vom empfangenden Prozeß für die Nachrichtenauswahl verwendet werden kann. `mtext` ist ein beliebiger Text mit der Länge von *msgsz* Byte. *msgsz* kann von `0` bis zu einem systembedingten Maximum reichen.

msgflg gibt die zu treffenden Maßnahmen an, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

- Die Anzahl der sich bereits in der Warteschlange befindenden Bytes ist gleich `msg_qbytes` (siehe `intro`(2)).
- Die Gesamtanzahl der Nachrichten in allen Warteschlangen im Gesamtsystem ist gleich dem systembedingten Grenzwert.

Diese Maßnahmen lauten wie folgt:

- Wenn (*msgflg*`&IPC_NOWAIT`) wahr ist, wird die Nachricht nicht gesendet, und der Aufruf kehrt sofort zurück.
- Wenn (*msgflg*`&IPC_NOWAIT`) nicht wahr ist, wird die Ausführung des aufrufenden Prozesses angehalten, bis einer der folgenden Punkte eintritt:
  - Besteht die für das Anhalten verantwortliche Bedingung nicht mehr, wird die Nachricht gesendet.
  - *msqid* wird vom System entfernt (siehe `msgctl`(2)). Wenn dieses eintritt, wird `errno` auf `EIDRM` gesetzt, und -1 wird zurückgegeben.
  - Der aufrufende Prozeß erhält ein Signal, das abgefangen werden soll. In diesem Fall wird die Nachricht nicht gesendet, und der aufrufende Prozeß nimmt die Ausführung in der in `signal`(2) beschriebenen Weise wieder auf.

msgsnd ist erfolglos, und eine Nachricht wird nicht gesendet, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EINVAL *msqid* ist keine gültige Kennung für die Nachrichten-Warteschlange.

EACCES Dem aufrufenden Prozeß wird die Zugriffserlaubnis verweigert (siehe intro(2)).

EINVAL *mtype* ist kleiner als 1.

EAGAIN Die Nachricht kann aus einem der oben angegebenen Gründe nicht gesendet werden, und (*msgflg*&IPC_NOWAIT) ist wahr.

EINVAL *msgsz* ist kleiner als 0 oder größer als der systembedingte Grenzwert.

EFAULT weist auf eine unzulässige Adresse.

Nach erfolgreicher Beendigung werden nachstehende Maßnahmen hinsichtlich der zu *msqid* gehörenden Datenstruktur ausgeführt (siehe intro(2)).

- msg_qnum wird um 1 erhöht.
- msg_lspid wird auf die Prozeßnummer des aufrufenden Prozesses gesetzt.
- msg_stime wird auf die aktuelle Uhrzeit gesetzt.

msgrcv liest eine Nachricht aus der Warteschlange, die der Kennung der Nachrichten-Warteschlange *msqid* gehört und schreibt sie in die benutzerdefinierte Struktur, auf die *msgp* zeigt. Die Struktur muß ein Feld für den Nachrichtentyp, gefolgt von einem Bereich für den Nachrichtentext, enthalten (siehe die Struktur mymsg oben). mtype ist der Typ der empfangenen Nachricht, wie vom sendenden Prozeß angegeben. mtext ist der Nachrichtentext. *msgsz* gibt die Größe von mtext in Bytes an. Die empfangene Nachricht wird auf *msgsz* Bytes verkürzt, wenn sie größer als *msgsz* ist und (*msgflg*&MSG_NOERROR) wahr ist. Der abgeschnittene Teil der Nachricht geht verloren; der aufrufende Prozeß wird über die Verkürzung der Nachricht nicht benachrichtigt.

msgtyp gibt den Typ der angeforderten Nachricht wie folgt an:

- Wenn *msgtyp* 0 ist, wird die erste Nachricht der Warteschlange empfangen.
- Wenn *msgtyp* größer als 0 ist, wird die erste Nachricht des Typs *msgtyp* empfangen.
- Wenn *msgtyp* kleiner als 0 ist, wird die erste Nachricht des niedrigsten Typs empfangen, der kleiner als oder gleich dem absoluten Wert von *msgtyp* ist.

msgflg gibt die zu treffenden Maßnahmen an, wenn eine Nachricht des gewünschten Typs nicht in der Warteschlange ist. Hierbei handelt es sich um folgendes:

- Wenn (*msgflg*&IPC_NOWAIT) wahr ist, kehrt der aufrufende Prozeß sofort mit dem Rückgabewert -1 zurück und setzt errno auf ENOMSG.
- Wenn (*msgflg*&IPC_NOWAIT) nicht wahr ist, wird der aufrufende Prozeß angehalten, bis eine der folgenden Situationen auftritt:

- Eine Nachricht des gewünschten Typs wird in die Warteschlange gesetzt.
- *msqid* wird vom System entfernt. Wenn dies geschieht, wird `errno` auf `EIDRM` gesetzt, und -1 wird zurückgegeben.
- Der aufrufende Prozeß empfängt ein Signal, das abgefangen werden soll. In diesem Fall wird keine Nachricht empfangen, und der aufrufende Prozeß nimmt die Ausführung in der in `signal`(2) beschriebenen Weise wieder auf.

`msgrcv` ist erfolglos und empfängt keine Nachricht, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL`	*msqid* ist keine gültige Kennung für die Nachrichten-Warteschlange.
`EACCES`	Dem aufrufenden Prozeß wird die Zugriffserlaubnis verweigert.
`EINVAL`	*msgsz* ist kleiner als 0.
`E2BIG`	Die Länge von *mtext* ist größer als *msgsz* und (*msgflg*&`MSG_NOERROR`) ist nicht wahr.
`ENOMSG`	Die Warteschlange enthält keine Nachricht des gewünschten Typs, und (*msgflg*&`IPC_NOWAIT`) ist wahr.
`EFAULT`	*msgp* weist auf eine unzulässige Adresse.

Nach erfolgreicher Beendigung werden folgende Maßnahmen in Hinblick auf die zu *msqid* gehörende Datenstruktur ausgeführt (siehe intro (2)):

- `msg_qnum` wird um 1 verringert.
- `msg_lrpid` wird auf die Prozeßnummer des aufrufenden Prozesses gesetzt.
- `msg_rtime` wird auf die aktuelle Zeit gesetzt.

SIEHE AUCH

`intro`(2), `msgctl`(2), `msgget`(2), `signal`(2).

ERGEBNIS

Wenn `msgsnd` oder `msgrcv` aufgrund des Empfangs eines Signals zurückkehren, wird -1 an den aufrufenden Prozeß zurückgegeben, und `errno` wird auf `EINTR` gesetzt. Kehren sie zurück, weil die Kennung *msqid* entfernt wurde, wird -1 zurückgegeben, und `errno` wird auf `EIDRM` gesetzt.

Nach erfolgreicher Beendigung ist der Rückgabewert wie folgt:

- `msgsnd` gibt 0 zurück.
- `msgrcv` gibt die Anzahl der Bytes zurück, die tatsächlich nach *mtext* geschrieben wurden.

Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

munmap - Abbildung von Speicherseiten aufheben

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

int munmap(caddr_t addr, size_t len);
```

Die Funktion `munmap` entfernt die Abbildung von Seiten im Bereich [*addr, addr + len*). Weitere Referenzen auf diese Seiten resultieren in einem `SIGSEGV`-Signal an den Prozeß.

Die Funktion `mmap` führt oft ein implizites `munmap` durch.

Unter den folgenden Bedingungen schlägt die Funktion `munmap` fehl und setzt `errno` auf die folgenden Werte:

`EINVAL` wenn *addr* nicht ein Vielfaches der Seitenlänge enthält, welche von `sysconf` zurückgegeben wird.

`EINVAL` wenn sich Adressen im Bereich [*addr, addr + len*) außerhalb des gültigen Adreßbereichs des Prozesses befinden.

`EINVAL` wenn das Argument *len* einen Wert kleiner oder gleich 0 enthält.

SIEHE AUCH

`mmap`(2), `sysconf`(3C).

ERGEBNIS

Nach erfolgreicher Ausführung liefert `munmap` den Wert 0; ansonsten wird der Wert -1 zurückgegeben und `errno` gesetzt.

nice - Priorität eines Prozesses ändern

```
#include <unistd.h>
int nice(int incr);
```

`nice` gibt einem Prozeß in der Klasse der Timesharing-Prozesse die Möglichkeit, seine Priorität zu ändern. Der `priocntl`-Aufruf ist eine allgemeinere Schnittstelle für Prozeßprioritäten.

`nice` addiert den Wert von *incr* zum *nice*-Wert des aufrufenden Prozesses. Der *nice*-Wert eines Prozesses ist eine nichtnegative Zahl, bei der ein höherer Wert eine niedrigere CPU-Priorität bedeutet.

Der maximale *nice*-Wert ist 39, der minimale 0 (der Standard ist 20). Aufrufe, die über oder unter den angegebenen Höchstwerten liegen, resultieren in dem Höchstwert.

`EPERM` `nice` scheitert und ändert den *nice*-Wert nicht, wenn *incr* negativ oder größer als 39 ist und der aufrufende Prozeß nicht die effektive Benutzernummer des Systemverwalters hat.

`EINVAL` `nice` scheitert, falls der aufrufende Prozeß nicht in der Timesharing-Klasse ist.

SIEHE AUCH

`exec`(2), `priocntl`(2).
`nice`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Bei Erfolg gibt `nice` den neuen Wert minus 20 zurück. Ansonsten wird -1 zurückgegeben **und** `errno` entsprechend gesetzt.

open - Datei zum Schreiben oder Lesen öffnen

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

int open (const char *Pfad, int oflag, ... /* mode_t Modus */);
```

Pfad weist auf einen Pfadnamen, der eine Datei benennt. `open` öffnet einen Dateideskriptor für die angegebene Datei und setzt die Dateistatus-Bits entsprechend *oflag*. *oflag*-Werte werden durch ODER-Verknüpfungen der folgenden Schalter aufgebaut (nur einer der ersten drei Schalter darf verwendet werden):

`O_RDONLY` nur für Lesen öffnen

`O_WRONLY` nur für Schreiben öffnen

`O_RDWR` für Lesen und Schreiben öffnen

`O_NDELAY` oder `O_NONBLOCK`
folgendes Lesen und Schreiben beeinflussen. (siehe `read`(2) und `write`(2)). Wenn beide, `O_NDELAY` und `O_NONBLOCK`, gesetzt sind, hat `O_NONBLOCK` Vorrang.

Wenn eine FIFO-Datei mit gesetztem `O_RDONLY` oder `O_WRONLY` geöffnet wird, hat das folgende Auswirkung:

- Bei gesetztem `O_NDELAY` oder `O_NONBLOCK`: Ein `open` nur zum Lesen kehrt sofort zurück; ein `open` nur zum Schreiben gibt einen Fehler zurück, wenn zum gegebenen Zeitpunkt kein Prozeß die Datei zum Lesen geöffnet hat.
- Bei gelöschtem `O_NDELAY` und `O_NONBLOCK`: Ein `open` nur zum Lesen wirkt als Blockierung, bis ein Prozeß die Datei zum Schreiben öffnet. Ein `open` nur zum Schreiben wirkt als Blockierung, bis ein Prozeß die Datei zum Lesen öffnet.

Wird eine mit einer Übertragungsleitung im Zusammenhang stehende Datei geöffnet, hat das folgende Auswirkung:

- Bei gesetztem `O_NDELAY` oder `O_NONBLOCK`: `open` wartet nicht darauf, daß das Gerät fertig oder verfügbar wird, sondern kehrt sofort zurück; das weitere Verhalten des Geräts ist gerätespezifisch.
- Bei gelöschten `O_NDELAY` und `O_NONBLOCK`: `open` ist blockiert, bis das Gerät fertig oder verfügbar ist.

`O_APPEND` Der Schreib-/Lesezeiger wird vor jedem Schreiben an das Ende der Datei positioniert.

O_SYNC Beim Öffnen einer normalen Datei ist nachfolgendes Schreiben beeinträchtigt. Jedes write(2) wartet auf die physikalische Aktualisierung von Dateidaten und Dateistatus.

O_NOCTTY Ist die Datei einem Terminal zugeordnet, wird das Terminal nicht als das steuernde Terminal des aufrufenden Prozesses allokiert.

O_CREAT Ist die Datei vorhanden, bleibt dieser Schalter wirkungslos, die Ausnahme ist unter O_EXCL angegebenen. Andernfalls wird die Datei erzeugt und die Eigentümernummer der Datei auf die effektive Benutzernummer des Prozesses, die Gruppennummer der Datei auf die effektive Gruppennummer des Prozesses gesetzt, oder, wenn das S_ISGID-Bit in dem Verzeichnis, in dem die Datei erzeugt wird, gesetzt ist, wird die Gruppennummer der Datei auf die Gruppennummer ihres übergeordneten Verzeichnisses gesetzt. Wenn die Gruppennummer der neuen Datei nicht zu der effektiven Gruppennummer oder einer der zusätzlichen Gruppennummern paßt, wird das S_ISGID-Bit gelöscht. Die Bits für die Zugriffsberechtigungen des Dateimodus werden auf den Wert von *Modus* gesetzt, der wie folgt geändert wird (siehe creat(2)):

- Alle in der Dateimodus-Erstellungsmaske des Prozesses gesetzten Bits werden gelöscht (siehe umask(2)).
- S_ISVTX wird gelöscht (siehe chmod(2)).

O_TRUNC Ist die Datei vorhanden, wird ihre Länge auf 0 gekürzt, und Modus sowie Eigentümer bleiben unverändert. O_TRUNC hat keinen Einfluß auf FIFO-Gerätedateien oder Verzeichnisse.

O_EXCL Bei gesetztem O_EXCL und O_CREAT ist open erfolglos, wenn die Datei vorhanden ist. Der Test auf die Existenz der Datei und die Erzeugung der Datei, wenn sie nicht existiert, ist atomar in bezug auf andere Prozesse, die open ausführen und denselben Dateinamen und dasselbe Verzeichnis verwenden und O_EXCL und O_CREAT gesetzt haben.

Beim Öffnen einer STREAMS-Datei kann *oflag* den Wert O_NDELAY oder O_NONBLOCK haben. Zu diesem Wert kann noch O_RDONLY, O_WRONLY oder O_RDWR durch bitweises ODER hinzugefügt werden. Andere Werte kann man bei STREAMS-Geräten nicht anwenden, und sie haben auf solche Geräte keine Wirkung. Die Werte von O_NDELAY und O_NONBLOCK beeinflussen die Ausführung von STREAMS-Treibern und bestimmten Systemaufrufen (siehe read(2), getmsg(2), putmsg(2) und write(2)). Bei Treibern ist die Implementierung von O_NDELAY und O_NONBLOCK gerätespezifisch. Jeder STREAMS-Gerätetreiber kann diese Optionen anders behandeln.

Wenn open aufgerufen wird, um einen Stream mit einem Namen zu erzeugen, auf den das connld-Modul (siehe connld(7)) gesetzt wurde, so blockiert der open-Aufruf, bis der Bedienerprozeß einen I_RECVFD ioctl-Aufruf (siehe streamio(7)) durchführt, um den Dateideskriptor zu empfangen.

Wenn *Pfad* ein symbolischer Verweis ist und O_CREAT und O_EXCL gesetzt sind, wird der Verweis nicht verfolgt.

Der zur Kennzeichnung der aktuellen Position innerhalb der Datei verwendete Schreib-/Lesezeiger wird auf den Anfang der Datei gesetzt.

Der neue Dateideskriptor ist der kleinste verfügbare Dateideskriptor, und er wird so gesetzt, daß er bei exec-Systemaufrufen offen bleibt (siehe fcntl(2)).

Einige Schalter können nach einem open-Aufruf gesetzt werden. Siehe dazu fcntl(2).

Wenn O_CREAT gesetzt ist und die Datei vorher noch nicht existierte, kennzeichnet open nach erfolgreicher Beendigung die Felder st_atime, st_ctime und st_mtime der Datei und die Felder st_ctime und st_mtime des übergeordneten Verzeichnisses für die Aktualisierung.

Wenn O_TRUNC gesetzt ist und die Datei vorher schon existierte, kennzeichnet open nach erfolgreicher Beendigung die Felder st_ctime und st_mtime zur Aktualisierung.

Die angegebene Datei wird nicht geöffnet, wenn eine der folgenden Bedingungen erfüllt ist:

EACCES — Die Datei existiert nicht und im übergeordneten Verzeichnis der zu erzeugenden Datei gibt es keine Schreiberlaubnis.

EACCES — O_TRUNC ist angegeben, und es gibt keine Schreiberlaubnis.

EACCES — Eine Komponente des Pfades darf nicht durchsucht werden.

EACCES — Die *oflag*-Erlaubnis für eine existierende Datei wird verweigert.

EAGAIN — Die Datei ist vorhanden, obligatorisches Sperren von Dateien und Dateisätzen ist gesetzt, und Datensatzsperren sind noch in der Datei vorhanden (siehe chmod(2)).

EEXIST — O_CREAT und O_EXCL sind gesetzt, und die angegebene Datei ist vorhanden.

EFAULT — *Pfad* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EINTR — Ein Signal wurde während des Systemaufrufs open abgefangen.

EIO — Während des Öffnens eines STREAMS-orientierten Gerätes ist ein Verbindungsabbau oder ein Fehler aufgetreten.

EISDIR — Die angegebene Datei ist ein Dateiverzeichnis, und *oflag* ist Schreiben oder Schreiben/Lesen.

ELOOP — Bei der Übersetzung von *Pfad* wurden zuviele symbolische Verweise angetroffen.

EMFILE — Der Prozeß hat zuviele geöffnete Dateien (siehe getrlimit(2)).

EMULTIHOP Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

ENAMETOOLONG
Das Argument *Pfad* ist länger als MAX_PATH, oder eine Pfadkomponente 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 ist nicht vorhanden.

ENOENT O_CREAT ist gesetzt, und eine Pfadkomponente existiert nicht oder ist der Null-Pfadname.

ENOLINK *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOMEM Das System ist nicht in der Lage, eine Senderkennung zuzuweisen.

ENOSPC O_CREAT und O_EXCL sind gesetzt, und das Dateisystem hat keine Indexeinträge mehr.

ENOSPC O_CREAT ist gesetzt, und das Verzeichnis für die Datei kann nicht erweitert werden.

ENOSR Ein Stream kann nicht zugewiesen werden.

ENOTDIR Eine Komponente des Pfades ist kein Dateiverzeichnis.

ENXIO Die angegebene Datei ist eine zeichen- oder blockorientierte Gerätedatei, und das zu dieser Gerätedatei gehörende Gerät ist nicht vorhanden.

ENXIO O_NDELAY oder O_NONBLOCK ist gesetzt, die angegebene Datei ist eine FIFO-Datei, O_WRONLY ist gesetzt, und kein Prozeß hat die Datei zum Lesen geöffnet.

ENXIO Eine Eröffnungsroutine für STREAMS-Modul oder -Treiber war erfolglos.

EROFS Die angegebene Datei steht in einem schreibgeschützten Dateisystem, und entweder O_WRONLY, O_RDWR, O_CREAT oder O_TRUNC ist in *oflag* gesetzt (wenn die Datei nicht existiert).

ETXTBSY Die Datei ist eine reine Programmdatei, die gerade ausgeführt wird, und oflag ist Lesen oder Lesen/Schreiben.

SIEHE AUCH

intro(2), chmod(2), close(2), creat(2), dup(2), exec(2), fcntl(2), getrlimit(2), lseek(2), read(2), getmsg(2), putmsg(2), stat(2), umask(2), write(2), stat(5).

ERGEBNIS

Nach erfolgreicher Beendigung wird der Dateideskriptor zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

pause - Prozeß bis Signal anhalten

```
#include <unistd.h>
int pause(void);
```

pause hält den aufrufenden Prozeß an, bis ein Signal empfangen wird. Dieses Signal darf zum gegebenen Zeitpunkt nicht so gesetzt sein, daß es vom aufrufenden Prozeß ignoriert wird.

Wenn das Signal die Beendigung des aufrufenden Prozesses verursacht, kehrt pause nicht zurück.

Wenn das Signal vom aufrufenden Prozeß abgefangen und die Steuerung von der Signalbehandlung (siehe signal(2)) zurückgegeben wird, beginnt der aufrufende Prozeß wieder mit der Ausführung an dem Punkt, an dem der Prozeß angehalten wurde; hierbei ist der Rückgabewert von pause -1, und errno wird auf EINTR gesetzt.

SIEHE AUCH

alarm(2), kill(2), signal(2), sigpause(2), wait(2).

pipe - Interprozeß-Kommunikationskanal einrichten

```
#include <unistd.h>
int pipe(int dk[2]);
```

`pipe` dient zum Einrichten eines als Pipe bezeichneten E/A-Mechanismus und gibt zwei Dateideskriptoren zurück, *dk*`[0]` und *dk*`[1]`. Die mit *dk*`[0]` und *dk*`[1]` verbundenen Dateien sind Streams und werden beide zum Lesen und Schreiben geöffnet. Die Schalter `O_NDELAY` und `O_NONBLOCK` werden gelöscht.

Das Einlesen von *dk*`[0]` greift auf die Daten zu, die gemäß dem FIFO-Prinzip an *dk*`[1]` geschrieben wurden, während das Einlesen von *dk*`[1]` auf jene Daten zugreift, die ebenfalls gemäß dem FIFO-Prinzip an *dk*`[0]` geschrieben wurden.

Der Schalter `FD_CLOEXEC` wird auf beiden Dateideskriptoren gelöscht.

Nach erfolgreicher Ausführung markiert `pipe` die Felder `st_atime`, `st_ctime` und `st_mtime` auf der Pipe zur Aktualisierung.

`pipe` scheitert, wenn:

`EMFILE` falls `OPEN_MAX-1` oder mehr Dateideskriptoren zum gegebenen Zeitpunkt für diesen Prozeß geöffnet sind.

`ENFILE` der Datei-Tabelleneintrag nicht zugewiesen werden konnte.

SIEHE AUCH

`fcntl`(2), `gtmsg`(2), `poll`(2), `putmsg`(2), `read`(2), `write`(2), `streamio`(7).
`sh`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Da eine Pipe bidirektional ist, gibt es zwei verschiedene Datenflüsse. Deshalb ist die Größe (`st_size`), die durch einen Aufruf an `fstat`(2) mit dem Argument *dk*`[0]` oder *dk*`[1]` beantwortet wurde, die Anzahl der Bytes, die zum Lesen von *dk*`[0]` oder *dk*`[1]` zur Verfügung standen. Früher war die Größe (`st_size`), die durch einen Aufruf an `fstat()` mit dem Argument *dk*`[1]` (dem Schreib-Ende) beantwortet wurde, jene Anzahl der Bytes, die zum Lesen von *dk*`[0]` (dem Lese-Ende) zur Verfügung standen.

plock - Prozeß, Text oder Daten sperren oder entsperren

```
#include <sys/lock.h>
int plock(int op);
```

`plock` ermöglicht dem aufrufenden Prozeß das Sperren oder Entsperren seines Textsegments (Textsperre), seines Datensegments (Datensperre) oder seines Text- und Datensegments (Prozeßsperre) im Speicher. Gesperrte Segmente sind von standardmäßigem Auslagern (swapping) ausgenommen. Die effektive Benutzernummer des aufrufenden Prozesses muß diejenige des Systemverwalters sein, damit dieser Aufruf benutzt werden kann. `plock` führt die von *op* angegebene Funktion aus:

`PROCLOCK`	Text- und Datensegmente im Speicher sperren (Prozeßsperre)
`TXTLOCK`	Textsegment im Speicher sperren (Textsperre)
`DATLOCK`	Datensegment im Speicher sperren (Datensperre)
`UNLOCK`	Sperren aufheben

`plock` ist erfolglos und wird die angeforderte Funktion nicht ausführen, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EPERM`	Die effektive Benutzernummer des aufrufenden Prozesses ist nicht diejenige des Systemverwalters.
`EINVAL`	*op* ist gleich `PROCLOCK`, und im aufrufenden Prozeß ist bereits eine Prozeßsperre, eine Textsperre oder eine Datensperre vorhanden.
`EINVAL`	*op* ist gleich `TXTLOCK`, und eine Textsperre oder eine Prozeßsperre ist bereits im aufrufenden Prozeß vorhanden.
`EINVAL`	*op* ist gleich `DATLOCK` und eine Datensperre oder eine Prozeßsperre ist bereits im aufrufenden Prozeß vorhanden.
`EINVAL`	*op* ist gleich `UNLOCK` und im aufrufenden Prozeß ist keine Sperre.
`EAGAIN`	Speicherplatz nicht ausreichend.

SIEHE AUCH

`exec`(2), `exit`(2), `fork`(2), `memcntl`(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zum aufrufenden Prozeß zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

HINWEIS

`memcntl` ist beim Prozeßsperren die bevorzugte Schnittstelle.

poll - STREAMS-Ein-/Ausgabe multiplexen

```
#include <stropts.h>
#include <poll.h>

int poll(struct poll *fds, size_t nfds, int timeout);
```

`poll` bietet dem Benutzer einen Mechanismus für das Multiplexen der Ein-/Ausgabe über einen Satz von Dateideskriptoren, die auf offene Dateien verweisen. `poll` kennzeichnet die Dateien, auf denen der Benutzer Nachrichten empfangen oder senden kann, oder auf denen bestimmte Ereignisse eingetreten sind.

fds gibt die zu prüfenden Dateideskriptoren und die Ereignisse an, die für jeden Dateideskriptor von Interesse sind. Es handelt sich um einen Zeiger auf ein Feld mit jeweils einem Element für jeden offenen Dateideskriptor von Interesse. Die Elemente des Feldes sind `pollfd`-Strukturen, die folgendes enthalten:

```
int fd;          /* Dateideskriptor */
short events;    /* angeforderte Ereignisse */
short revents;   /* gemeldete Ereignisse */
```

`fd` gibt einen offenen Dateideskriptor an, und `events` (Ereignisse) und `revents` (zurückgegebene Ereignisse) sind Bitmasken, die durch ODER-Verknüpfung beliebiger Kombinationen nachstehender Ereignisanzeiger aufgebaut werden:

`POLLIN` Daten von nicht höchster Priorität können nichtblockierend gelesen werden. Für STREAMS wird diese Option auch gesetzt, wenn die Nachricht die Länge 0 hat.

`POLLRDNORM` Gewöhnliche Daten (Priorität = 0) können nichtblockierend gelesen werden. Für STREAMS wird diese Option auch gesetzt, wenn die Nachricht die Länge 0 hat.

`POLLRDBAND` Daten mit einer Priorität ungleich 0 können nichtblockierend gelesen werden. Für STREAMS wird diese Option auch gesetzt, wenn die Nachricht die Länge 0 hat.

`POLLPRI` Daten mit höchster Priorität können nichtblockierend empfangen werden. Für STREAMS wird diese Option auch gesetzt, wenn die Nachricht die Länge 0 hat.

`POLLOUT` Normale Daten können nichtblockierend geschrieben werden.

`POLLWRNORM` wie `POLLOUT`

`POLLWRBAND` Daten mit einer Prioritätsklasse ungleich 0 können geschrieben werden. Dieses Ereignis untersucht nur Klassen, die mindestens einmal geschrieben wurden.

POLLMSG — Eine M_SIG- oder M_PCSIG-Nachricht, die ein ASIGPOLL-Signal enthält, hat den Anfang der Stream-Kopf-Warteschlange erreicht.

POLLERR — Eine Fehlermeldung liegt am Stream oder Gerät an. Dieser Schalter ist nur in der revents-Bitmaske gültig; er wird nicht im events-Feld verwendet.

POLLHUP — Ein Hangup ist im stream aufgetreten. Dieses Ereignis und POLLOUT schließen sich gegenseitig aus; auf einen Stream kann niemals geschrieben werden, wenn ein Hangup aufgetreten ist. Jedoch schließen sich dieses Ereignis und POLLIN bzw. POLLPRI nicht gegenseitig aus. Dieser Schalter ist nur in der revents-Bitmaske gültig; er wird nicht in einem events-Feld verwendet.

POLLNVAL — Der angegebene fd-Wert gehört nicht zu einer offenen Datei. Dieser Schalter ist nur im revents-Feld gültig; er wird nicht im events-Feld verwendet.

Bei jedem Element des Feldes, auf das *fds* zeigt, prüft poll den angegebenen Dateideskriptor auf die in events angegebenen Ereignisse. Die Anzahl der zu prüfenden Dateideskriptoren wird von *nfds* angegeben.

Wenn der Wert fd kleiner als Null ist, wird events ignoriert, und revents wird bei der Rückkehr von poll in diesem Eintrag auf 0 gesetzt.

Die Ergebnisse der poll-Anfrage werden im revents-Feld in der pollfd-Struktur gespeichert. Zur Anzeige, welche der angeforderten Ereignisse wahr sind, werden Bits in der revents-Bitmaske gesetzt. Wenn keine Ereignisse wahr sind, wird keines der angegebenen Bits bei der Rückkehr des poll-Aufrufs in revents gesetzt. Die Ereignisanzeiger POLLHUP, POLLERR und POLLNVAL werden stets in revents gesetzt, wenn die von ihnen angezeigten Bedingungen wahr sind; dies geschieht auch, wenn sie nicht in events vorhanden waren.

Wenn keines der definierten Ereignisse bei einem der jeweils ausgewählten Dateideskriptoren auftritt, wartet poll wenigstens *timeout* Millisekunden auf das Auftreten eines Ereignisses bei einem der gewählten Dateideskriptoren. Bei einem Rechner, bei dem die Genauigkeit auf Millisekunden nicht zur Verfügung steht, wird *timeout* auf den nächsten zulässigen Wert aufgerundet, der in diesem System zur Verfügung steht. Wenn der Wert von *timeout* 0 ist, kehrt poll sofort zurück. Ist der Wert von *timeout* gleich INFTIM (oder -1) bewirkt poll eine Blockierung, bis ein abgefragtes Ereignis auftritt, oder bis der Aufruf unterbrochen wird. poll wird von den Schaltern O_NDELAY und O_NONBLOCK nicht betroffen.

`poll` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EAGAIN` Zuweisung der internen Datenstrukturen war erfolglos, Anforderung sollte jedoch erneut versucht werden.

`EFAULT` Ein Argument zeigt auf einen Speicherplatz außerhalb des zugewiesenen Adreßraums.

`EINTR` Ein Signal wurde während des Systemaufrufs `poll` abgefangen.

`EINVAL` Das Argument `nfds` ist kleiner als Null oder größer als `OPEN_MAX`.

SIEHE AUCH

`intro`(2), `getmsg`(2), `getrlimit`(2), `putmsg`(2), `read`(2), `write`(2).
"Leitfaden für Programmierer: STREAMS".

ERGEBNIS

Nach erfolgreicher Beendigung wird ein nicht negativer Wert zurückgegeben. Ein positiver Wert zeigt die Gesamtanzahl der jeweils ausgewählten Dateideskriptoren an (d.h. Dateideskriptoren, für die das Feld `revents` ungleich Null ist). 0 zeigt an, daß die Zeit für den Aufruf abgelaufen ist und keine Dateideskriptoren gewählt wurden. Ist der Vorgang erfolglos, wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

priocntl - Prozesse verwalten

```
#include <sys/types.h>
#include <sys/priocntl.h>
#include <sys/rtpriocntl.h>
#include <sys/tspriocntl.h>

long priocntl(idtype_t idtype, id_t id, int cmd, ... /* arg */);
```

`priocntl` erlaubt die Kontrolle aktiver Prozesse.

Es gibt Prozesse verschiedener Klassen, wobei auf jede Prozeßklasse bestimmte Verwaltungsarten anwendbar sind. Die beiden gegenwärtig unterstützten Klassen sind die Echtzeit-Klasse und die Timesharing-Klasse. Die Eigenschaften dieser Klassen werden unter den entsprechend benannten Abschnitten behandelt. Das Klassenattribut eines Prozesses bleibt bei einem `fork-` oder `exec`(2)-Systemaufruf erhalten. `priocntl` kann verwendet werden, um die Prozeßklasse dynamisch zu ändern und andere Parameter eines ablaufenden Prozesses oder einer Menge von Prozessen zu beeinflussen.

In der voreingestellten Konfiguration hat ein ausführbarer Echtzeit-Prozeß Vorrang vor jedem anderen Prozeß. Aus diesem Grund kann die falsche Verwendung von Echtzeit-Prozessen einen ausgesprochen negativen Einfluß auf die Systemleistung haben.

`priocntl` erlaubt die Angabe eines einzelnen Prozesses oder einer ganzen Prozeßmenge, auf die der Systemaufruf angewendet werden soll. Der Systemaufruf `priocntlset` bietet dieselben Funktionen wie `priocntl`, stellt jedoch eine allgemeinere Schnittstelle zur Angabe von Prozeßmengen dar, auf die der Systemaufruf angewendet werden soll.

Bei `priocntl` werden die Argumente *idtype* und *id* dazu verwendet, um die Prozeßmenge anzugeben. Die Interpretierung von *id* hängt vom Wert *idtype* ab. Die möglichen Werte für *idtype* und entsprechende Interpretierungsmöglichkeiten von *id* sind:

`P_PID` *id* ist eine Prozeßnummer, welche einen einzelnen Prozeß angibt, auf den der `priocntl`-Systemaufruf angewendet werden soll.

`P_PPID` *id* ist die Prozeßnummer eines Vaterprozesses. Der `priocntl`-Systemaufruf wird auf alle Prozesse angewendet, die über eine entsprechende Vaterprozeßnummer verfügen.

`P_PGID` *id* ist eine Prozeßgruppennummer. Der `priocntl`-Systemaufruf wird auf alle Prozesse der angegebenen Prozeßgruppe angewendet.

`P_SID` *id* ist eine Sitzungsnummer. Der `priocntl`-Systemaufruf wird auf alle Prozesse der angegebenen Sitzung angewendet.

`P_CID` *id* ist eine Klassennummer (zurückgegeben von `priocntl PC_GETCID`, wie weiter unten erläutert). Der `priocntl`-Systemaufruf wird auf alle Prozesse der entsprechenden Klasse angewendet.

P_UID — *id* ist eine Benutzernummer. Der priocntl-Systemaufruf wird auf alle Prozesse mit dieser effektiven Benutzernummer angewendet.

P_GID — *id* ist eine Gruppennummer. Der priocntl-Systemaufruf wird auf alle Prozesse mit dieser effektiven Gruppennummer angewendet.

P_ALL — Der priocntl-Systemaufruf wird auf alle existierenden Prozesse angewendet. Der Wert von *id* wird ignoriert. Die weiter unten beschriebenen Zugriffsbeschränkungen bleiben dabei gültig.

Der Wert P_MYID für *id* kann in Verbindung mit *idtype* verwendet werden, um die Prozeßnummer, die Vaterprozeßnummer, die Prozeßgruppennummer, die Sitzungsnummer, die Klassennummer oder die Gruppennummer des aufrufenden Prozesses anzugeben.

Um die Parameter eines Prozesses (unter Verwendung des unten beschriebenen Kommandos PC_SETPARMS) zu ändern, muß die wirkliche oder effektive Benutzernummer des aufrufenden Prozesses der wirklichen oder effektiven Benutzernummer des empfangenden Prozesses entsprechen, oder aber die effektive Benutzernummer des aufrufenden Prozesses muß die des Systemverwalters sein. Dies sind die Mindestanforderungen, die an alle Klassen gestellt werden. Eine einzelne Klasse kann weitere Zugriffsbeschränkungen einsetzen, wenn Prozesse auf diese Klasse eingestellt werden, und/oder wenn klassenabhängige Prozeßparameter gesetzt werden.

Eine spezielle sys-Verwaltungsklasse existiert zum Verwalten der Ausführung bestimmter Systemprozesse (wie zum Beispiel der Swap-Prozeß). Es ist nicht möglich, die Klasse eines Prozesses auf sys zu setzen. Außerdem werden alle Prozesse der sys-Klasse, die sich in einer angegebenen Prozeßmenge befinden, von priocntl ignoriert. Wird beispielsweise *idtype* mit P_UID aufgerufen und enthält *id* den Wert Null, so werden alle Prozesse mit der Benutzernummer Null angegeben, außer Prozesse der Klasse sys und (wenn die Parameter mit PC_SETPARMS verändert werden) der Prozeß init.

Der Prozeß init ist dabei ein Sonderfall. Will ein priocntl-Aufruf die Klasse oder andere Verwaltungsparameter des Prozesses init ändern (Prozeßnummer 1), so muß es sich dabei um den einzigen Prozeß handeln, der durch *idtype* und *id* angegeben wird. Der init-Prozeß kann jeder Klasse zugewiesen werden, die im System konfiguriert ist, jedoch ist die Timesharing-Klasse meist die richtige.

Der Datentyp und der Wert von *arg* hängen vom Typ des Kommandos *cmd* ab.

Die folgende Struktur wird von den Kommandos PC_GETCID und PC_GETCLINFO verwendet.

```
typedef struct {

id_t  pc_cid;                  /* Klassennummer */
char  pc_clname[PC_CLNMSZ];    /* Klassenname */
long  pc_clinfo[PC_CLINFOSZ]; /* Klasseninformationen */

} pcinfo_t;
```

pc_cid ist eine Klassennummer, die von priocntl PC_GETCID zurückgegeben wird.

pc_clname ist ein Puffer der Länge PC_CLNMSZ (definiert in sys/priocntl.h), in dem sich der Klassenname befindet (RT für die Echtzeit-Klasse oder TS für die Timesharing-Klasse).

pc_clinfo ist ein Puffer der Länge PC_CLINFOSZ (definiert in sys/priocntl.h), in dem Informationen über die Attribute einer bestimmten Klasse abgelegt werden. Das Format dieser Daten ist klassenabhängig und wird unter dem entsprechenden Abschnitt weiter unten beschrieben.

Die folgende Struktur wird von den Kommandos PC_SETPARMS und PC_GETPARMS verwendet.

```
typedef struct {

id_t  pc_cid;                          /* Prozeßklasse */
long  pc_clparms[PC_CLPARMSZ];         /* Klassenabhängige Parameter */

} pcparms_t;
```

pc_cid ist eine Klassennummer (zurückgegeben von priocntl PC_GETCID). Die spezielle Klassennummer-ID PC_CLNULL kann auch pc_cid zugewiesen werden, wenn das PC_GETPARMS-Kommando so verwendet wird, wie es weiter unten beschrieben wird.

Der Puffer pc_clparms dient zur Aufnahme klassenabhängiger Verwaltungsparameter. Das Format dieses Parameters für die jeweilige Klasse wird unter dem entsprechenden Abschnitt weiter unten erklärt. PC_CLPARMSZ ist die Länge des pc_clparms-Puffers und wird in sys/priocntl.h definiert.

Kommandos

PC_GETCID Klassennummer und Klassenattribute für eine bestimmte Klasse mit gegebenen Klassennamen holen. Die Argumente *idtype* und *id* werden ignoriert. Wenn *arg* ungleich NULL ist, wird angenommen, daß auf eine Struktur des Typs pcinfo_t verwiesen wird. Der Puffer pc_clname enthält den Namen der Klasse, deren Attribute gelesen werden.

Bei erfolgreicher Ausführung wird die Klassennummer in pc_cid zurückgegeben, die Klassenattribute werden im Puffer pc_clinfo zurückgeliefert, und der priocntl-Aufruf liefert die Gesamtanzahl der im System konfigurierten Klassen zurück (einschließlich der Klasse sys). Wenn die Klasse aus pc_clname ungültig ist oder momentan nicht konfiguriert ist, liefert der priocntl-Aufruf -1 zurück und setzt errno auf EINVAL. Das Format der zurückgelieferten Attributdaten für eine gegebene Klasse ist in sys/rtpriocntl.h oder sys/tspriocntl.h definiert und wird unter dem entsprechenden Abschnitt weiter unten beschrieben.

Wenn *arg* ein Nullzeiger ist, werden keine Attributdaten zurückgegeben; der priocntl-Aufruf liefert dann lediglich die Anzahl der konfigurierten Klassen zurück.

PC_GETCLINFO

liest den Klassennamen und die Klassenattribute für eine gegebene Klassennummer. Die Argumente *idtype* und *id* werden ignoriert. Wenn *arg* ungleich Null ist, wird angenommen, daß auf eine Struktur des Typs pcinfo_t verwiesen wird. pc_cid ist die Klassennummer derjenigen Klasse, deren Attribute gelesen werden sollen.

Bei erfolgreicher Ausführung wird der Klassenname im Puffer pc_clname und die Klassenattribute im Puffer pc_clinfo zurückgegeben; der priocntl-Aufruf liefert die Gesamtanzahl der im System konfigurierten Klassen zurück (einschließlich der Klasse sys). Das Format der zurückgelieferten Attributdaten wird in den Include-Dateien sys/rtpriocntl.h oder sys/tspriocntl.h definiert und ist unter den entsprechenden Abschnitten weiter unten beschrieben.

Wenn *arg* ein Nullzeiger ist, werden keine Attributdaten zurückgegeben; der priocntl-Aufruf liefert dann lediglich die Anzahl der konfigurierten Klassen zurück.

PC_SETPARMS

setzt die Klasse und die klassenabhängigen Verwaltungsparameter des angegebenen Prozesses bzw. der angegebenen Prozesse. *arg* zeigt auf eine Struktur vom Typ pcparms_t. pc_cid gibt die Klasse an, deren Parameter geändert werden sollen; der Puffer pc_clparms enthält die klassenabhängigen Parameter, die gesetzt werden sollen. Das Format der klassenabhängigen Parameter wird in den Include-Dateien sys/rtpriocntl.h oder sys/tspriocntl.h definiert und wird unter den entsprechenden Abschnitten weiter unten erklärt.

Beim Einstellen der Parameter für eine Prozeßmenge agiert priocntl mit den Prozessen der Prozeßmenge in einer implementierungsabhängigen Weise. Wenn priocntl einen Fehler bei einem oder mehreren der Zielprozesse erkennt, wird die Operation abhängig von der Art des Fehler weitergeführt oder abgebrochen. Beruht der Fehler auf den Berechtigungen (EPERM), so wird die Operation mit den weiteren Prozessen der Prozeßmenge ausgeführt, wobei die Parameter für alle Zielprozesse entsprechend geändert werden, für die der aufrufende Prozeß entsprechende Berechtigung besitzt. priocntl liefert dann -1 zurück und setzt errno auf EPERM; dies zeigt an, daß die Operation für einen oder mehrere der Zielprozesse fehlgeschlagen ist. Wenn priocntl einen Fehler erkennt, der nicht auf Berechtigungen basiert, wird dieser Fehler sofort zurückgeliefert; die Operation wird abgebrochen.

PC_GETPARMS

Die Klasse und/oder die klassenabhängigen Verwaltungsparameter eines Prozesses werden gelesen. *arg* zeigt auf eine Struktur vom Typ `pcparms_t`.

Wenn `pc_cid` eine konfigurierte Klasse angibt und von den Werten *idtype* und *id* der `procset`-Struktur ein Prozeß dieser Klasse angegeben wird, dann werden die Verwaltungsparameter des Prozesses in dem Puffer `pc_clparms` zurückgegeben. Wenn der angegebene Prozeß nicht existiert oder nicht zu der angegebenen Klasse gehört, liefert der `priocntl`-Aufruf -1 zurück und setzt `errno` auf `ESRCH`.

Gibt `pc_cid` eine konfigurierte Klasse und eine Prozeßmenge an, so werden die Verwaltungsparameter eines Prozesses, der zu der angegebenen Prozeßmenge und Klasse gehört, in `pc_clparms` zurückgegeben, und `priocntl` liefert die Prozeßnummer des ausgewählten Prozesses zurück. Die Kriterien bei der Auswahl eines Prozesses für diesen Fall ist klassenabhängig. Wenn keiner der angegebenen Prozesse existiert oder keiner der Prozesse zu der angegebenen Klasse gehört, liefert `priocntl` -1 zurück und setzt `errno` auf `ESRCH`.

Wenn `pc_cid` gleich `PC_CLNULL` ist und ein einziger Prozeß angegeben wird, so wird die Klasse dieses Prozesses in `pc_cid` und seine Verwaltungsparameter in `pc_clparms` zurückgegeben.

PC_ADMIN — Dieses Kommando bietet die Funktionalität für die Implementierung des Kommandos `dispadmin`(1M). Es ist nicht zum allgemeinen Gebrauch durch andere Anwendungen gedacht.

Echtzeit-Klasse

Die Echtzeit-Klasse bietet eine festgelegte "preemptive scheduling policy" für diejenigen Prozesse, die schnelle und deterministische Reaktionen und absolute Benutzer-/Anwendungskontrolle mit den "scheduling"-Prioritäten benötigen. Wenn die Echtzeit-Klasse im System konfiguriert ist, sollte sie die exklusive Kontrolle über die höchsten Verwaltungsprioritäten des Systems haben. Dies garantiert, daß ein laufender Echtzeit-Prozeß durch die CPU bedient wird, bevor ein Prozeß einer anderen Klasse bedient wird.

Die Echtzeit-Klasse hat einen Bereich von Echtzeit-Prioritätswerten (`rt_pri`), die den Prozessen dieser Klasse zugewiesen werden können. Echtzeit-Prioritäten befinden sich im Bereich von 0 bis x, wobei der Wert von x konfigurierbar und für eine bestimmte Installation über das Kommando `priocntl PC_GETCID` oder `PC_GETCLINFO` einstellbar ist.

Die Echtzeit-Verwaltungsmöglichkeit ist eine festgelegte Prioritätsverwaltung. Die Verwaltungspriorität eines Echtzeit-Prozesses ändert sich nicht, außer der Benutzer oder die Anwendung möchte den Wert `rt_pri` des Prozesses ausdrücklich ändern.

Bei Prozessen der Echtzeit-Klasse ist der Wert `rt_pri` für alle praktischen Zwecke äquivalent mit der Verwaltungspriorität des Prozesses. Der Wert `rt_pri` bestimmt vollständig die Verwaltungspriorität eines Echtzeit-Prozesses relativ zu anderen Prozessen innerhalb seiner Klasse. Numerisch höhere Werte für `rt_pri` stellen höhere Prioritäten dar. Da die Echtzeit-Klasse den höchsten Bereich der Verwaltungsprioritäten im System kontrolliert, wird garantiert, daß der ausführbare Echtzeit-Prozeß mit dem höchsten `rt_pri`-Wert vor jedem anderen Prozeß im System ausgeführt wird.

Neben der Prioritätskontrolle erlaubt `priocntl` die Kontrolle des Zeitquantums, das den Prozessen der Echtzeit-Klasse zugewiesen wird. Der Wert des Zeitquantums gibt den maximalen Zeitwert an, den ein Prozeß in Anspruch nehmen kann, vorausgesetzt, der Prozeß wird nicht beendet, oder ein Ressource-Status oder Event-Wartezustand (`sleep`) tritt ein. Wird ein anderer Prozeß gestartet, der höhere Priorität als der momentan ablaufende Prozeß besitzt, so kann der momentan ablaufende Prozeß vorzeitig angehalten werden, bevor er sein Zeitquantum erhalten hat.

Der Prozeßverwalter des Systems hält die ablaufbaren Echtzeit-Prozesse in einer Menge von Verwaltungsschlangen. Es gibt eine eigene Schlange für jede konfigurierte Echtzeit-Priorität, und alle Echtzeit-Prozesse mit einem gegebenen `rt_pri`-Wert werden zusammen in der entsprechenden Schlange gehalten. Die Prozesse einer gegebenen Schlange werden in einer FIFO-Reihe angeordnet (dies bedeutet, daß der Prozeß am Anfang der Schlange am längsten auf Bedienung gewartet hat und von der CPU als nächstes bedient wird). Echtzeit-Prozesse, die nach 'sleep' wieder aktiv werden, Prozesse aus einer anderen Klasse, die in die Echtzeit-Klasse gelangen, Prozesse, die ihr volles Zeitquantum verbraucht haben, und lauffähige Prozesse, deren Priorität durch `priocntl` neu gesetzt wird, werden an das Ende der entsprechenden Prioritätsschlange geschrieben. Ein Prozeß, der durch einen privilegierteren Prozeß vorzeitig angehalten wird, bleibt am Anfang der Schlange (unabhängig davon, wieviel Zeit für sein Zeitquantum noch verbleibt) und läuft vor jedem anderen Prozeß gleicher Priorität ab. Wird ein `fork`(2)-Systemaufruf von einem Echtzeit-Prozeß ausgeführt, läuft der Vaterprozeß weiter, während der Sohnprozeß (welcher den `rt_pri`-Wert des Vaterprozesses erbt) an das Ende der Schlange gesetzt wird.

Die folgende Struktur (definiert in `sys/rtpriocntl.h`) definiert das Format der Attributsdaten für die Echtzeit-Klasse.

```
typedef struct {
short rt_maxpri;      /* Maximale Echtzeit-Priorität */
} rtinfo_t;
```

Die priocntl-Kommandos PC_GETCID und PC_GETCLINFO liefern die Attribute der Echtzeit-Klasse im Puffer pc_clinfo mit diesem Format zurück.

rt_maxpri gibt den konfigurierten Maximalwert rt_pri für die Echtzeit-Klasse zurück (wenn rt_maxpri gleich x ist, reichen die gültigen Echtzeit-Prioritäten von 0 bis x).

Die folgende Struktur (definiert in sys/rtpriocntl.h) definiert das Format zur Angabe von Verwaltungsparametern für Prozesse der Echtzeit-Klasse.

```
typedef struct {

short rt_pri;          /* Echtzeit-Priorität */
ulong rt_tqsecs;       /* Sekunden des Zeitquantums */
long  rt_tqnsecs;      /* Zusätzliche Nanosekunden des Zeitquantums */

} rtparms_t;
```

Bei Verwendung der Kommandos priocntl PC_SETPARMS oder PC_GETPARMS werden die Daten im Puffer pc_clparms in diesem Format zurückgegeben, falls pc_cid die Echtzeit-Klasse angibt.

Die oben erwähnten Kommandos können verwendet werden, um die Echtzeit-Priorität auf den entsprechenden Wert zu setzen oder den aktuellen rt_pri-Wert zu lesen. Das Einstellen des rt_pri-Werten für einen momentan ablaufenden oder ablaufbaren (nicht im sleep-Zustand befindlichen) Prozeß verursacht, daß der Prozeß an das Ende der Verwaltungsschlange für die entsprechende Priorität gesetzt wird. Der Prozeß wird an das Ende der entsprechenden Schlange gesetzt, unabhängig davon, ob die gesetzte Priorität vom vorherigen rt_pri-Wert des Prozesses abweicht. Zu beachten ist, daß ein ablaufender Prozeß die CPU freigeben kann und sich an das Ende der Verwaltungsschlange gleicher Priorität setzen kann, indem sein vorheriger rt_pri-Wert aktualisiert wird. Um das Zeitquantum eines Prozesses zu ändern, ohne daß die Priorität oder die Position des Prozesses in der Schlange geändert wird, sollte das Feld rt_pri auf Wert RT_NOCHANGE (definiert in sys/rtpriocntl.h) gesetzt werden. Die Angabe von RT_NOCHANGE beim Ändern der Prozeßklasse von Echtzeit auf eine andere Klasse resultiert darin, daß die Echtzeit-Priorität auf Null gesetzt wird.

Beim Kommando priocntl PC_GETPARMS werden die Verwaltungsparameter des Echtzeit-Prozesses mit dem höchsten rt_pri-Wert zurückgegeben, wenn pc_cid die Echtzeit-Klasse angibt und mehr als ein Echtzeit-Prozeß angegeben wird; die Prozeßnummer des Prozesses wird von priocntl zurückgegeben. Gibt es mehr als einen Prozeß mit der höchsten Priorität, so ist die zurückgegebene Prozeßnummer implementierungsabhängig.

Die Felder rt_tqsecs und rt_tqnsecs werden verwendet, um das Zeitquantum eines Prozesses oder einer Prozeßgruppe zu setzen oder zu lesen. rt_tqsecs ist die Anzahl der Sekunden und rt_tqnsecs ist die Anzahl der zuzüglichen Nanosekunden des Zeitquantums. Enthält rt_tqsecs 2 und rt_tqnsecs den Wert 500,000,000 (dezimal), so würde das Zeitquantum zweieinhalb Sekunden betragen. Der Wert 1,000,000,000 oder höher für rt_tqnsecs erzeugt einen Fehler; dabei wird errno auf EINVAL gesetzt. Ob-

wohl die Auflösung des tq_nsecs-Feldes sehr fein ist, wird das angegebene Zeitquantum vom System auf das nächste ganzzahlige Vielfache der Auflösung der Systemuhr gerundet. Beispielsweise beträgt die genaueste, momentan verfügbare Auflösung der 3B2 Anlage 10 Millisekunden (1 'Zeittakt'). Die Einstellung von rt_tqsecs gleich 0 und rt_tqnsecs gleich 34,000,000 würde ein Zeitquantum von 34 Millisekunden angeben, welches auf vier Zeittakte (40 Millisekunden) aufgerundet wird. Der Maximalwert für das Zeitquantum ist implementierungsabhängig und gleich LONG_MAX Zeittakten (definiert in limits.h). Die Angabe eines Zeitquantums, das größer als dieser Maximalwert ist, hat einen Fehler zur Folge; dabei wird errno auf ERANGE gesetzt (obwohl unendliche Zeitquanten durch einen besonderen Wert angefordert werden können; siehe Beschreibung weiter unten). Beträgt das Zeitquantum Null (wobei rt_tqsecs und rt_tqnsecs 0 enthalten), wird ein Fehler zurückgegeben und errno enthält EINVAL.

Das Feld rt_tqnsecs kann auf einen der folgenden Sonderwerte gesetzt werden (definiert in sys/rtpriocntl.h), bei denen der Wert von rt_tqsecs ignoriert wird.

RT_TQINF stellt ein unendliches Zeitquantum ein.

RT_TQDEF setzt das Zeitquantum auf den voreingestellten Wert für diese Priorität (siehe rt_dptbl(4)).

RT_NOCHANGE
Das Zeitquantum wird nicht eingestellt. Dieser Wert ist nützlich, wenn die Echtzeit-Priorität eines Prozesses geändert werden und das Zeitquantum unverändert bleiben soll. Die Angabe dieses Wertes beim Transformieren eines Prozesses in einen Echtzeit-Prozeß ist äquivalent mit der Angabe von RT_TQDEF.

Um die Klasse eines Prozesses auf Echtzeit zu ändern (von irgendeiner anderen Klasse) muß der aufrufende Prozeß über Systemverwalterrechte verfügen. Um die Priorität oder das Zeitquantum eines Echtzeit-Prozesses zu ändern, muß der Prozeß, der priocntl aufruft, über Systemverwalterrechte verfügen oder muß selbst ein Echtzeit-Prozeß sein, dessen wirkliche oder effektive Benutzernummer der wirklichen oder effektiven Benutzernummer des Zielprozesses entspricht.

Die Echtzeit-Priorität und die Zeitquanten werden über fork(2)- und exec(2)-Systemaufrufe weitervererbt.

Timesharing-Klasse

Die Verwaltungsmöglichkeiten der Timesharing-Klasse sorgt für eine gerechte und effektive Verteilung der CPU-Ressourcen zwischen Prozessen mit variierenden CPU-Anforderungen. Die Ziele der Verwaltung von Timesharing-Prozessen bestehen darin, daß gute Reaktionszeiten für interaktive Prozesse und ein guter Durchsatz für CPU-gebundene Jobs garantiert werden, wobei ein gewisser Teil der Verwaltung vom Benutzer bzw. der Anwendung kontrolliert werden kann.

Die Timesharing-Klasse verfügt über einen Bereich von Timesharing-Benutzerprioritäten (siehe `ts_upri` weiter unten), welche den Prozessen dieser Klasse zugewiesen werden können. Der Wert 0 für `ts_upri` ist als Basispriorität für die Timesharing-Klasse voreingestellt. Benutzerprioritäten reichen von $-x$ bis $+x$, wobei der Wert x konfigurierbar und für eine bestimmte Installation über das Kommando `priocntl PC_GETCID` oder `PC_GETCLINFO` einstellbar ist.

Der Zweck der Benutzerpriorität besteht darin, daß der Benutzer bzw. die Anwendung die Prozesse der Timesharing-Klasse bis zu einem gewissen Grad kontrollieren kann. Das Erhöhen oder Verringern des Wertes `ts_upri` für einen Prozeß der Timesharing-Klasse erhöht oder verringert seine Priorität. Es ist nicht garantiert, daß ein Prozeß mit einem höheren `ts_upri`-Wert vor einem Prozeß mit einem niedrigeren `ts_upri`-Wert ausgeführt wird. Dies liegt daran, daß der `ts_upri`-Wert nur ein Faktor zur Bestimmung der Priorität eines Timesharing-Prozesses ist. Das System kann die interne Verwaltungspriorität eines Timesharing-Prozesses aufgrund von anderen Faktoren, wie zum Beispiel die bisherige CPU-Benutzung, dynamisch ändern.

Neben den systemweiten Grenzen für die Benutzerpriorität (zurückgegeben von den `PC_GETCID`- und `PC_GETCLINFO`-Kommandos) gibt es ein Prioritätslimit für einen Prozeß (siehe `ts_uprilim` weiter unten), welches den maximalen `ts_upri`-Wert angibt, der für einen gegebenen Prozeß gesetzt werden kann; die Voreinstellung für `ts_uprilim` ist Null.

Die folgende Struktur (definiert in `sys/tspriocntl.h`) definiert das Format für die Attribute der Timesharing-Klasse.

```
typedef struct {
short ts_maxupri;    /* Grenze für den Bereich der Benutzerpriorität */
} tsinfo_t;
```

Die Kommandos `priocntl PC_GETCID` und `PC_GETCLINFO` liefern die Attribute für Prozesse der Timesharing-Klasse im Puffer `pc_clinfo` in diesem Format zurück.

`ts_maxupri` gibt den konfigurierten Maximalwert für die Benutzerpriorität der Timesharing-Klasse zurück. Wenn `ts_maxupri` gleich x ist, so ist der gültige Bereich für die Benutzerprioritäten und die Grenzen der Benutzerpriorität $-x$ bis $+x$.

Die folgende Struktur (definiert in `sys/tspriocntl.h`) definiert das Format zur Angabe der klassenabhängigen Parameter für einen Prozeß.

```
typedef struct {
short ts_uprilim;      /* Grenze für Timesharing-Benutzerpriorität */
short ts_upri;         /* Timesharing-Benutzerpriorität */
} tsparms_t;
```

Bei der Benutzung der Kommandos priocntl PC_SETPARMS oder PC_GETPARMS werden die Daten im Puffer pc_clparms in diesem Format zurückgegeben, wenn pc_cid die Timesharing-Klasse angibt.

Das Kommando priocntl PC_GETPARMS liefert die Verwaltungsparameter des Timesharing-Prozesses mit dem höchsten ts_upri-Wert und die Prozeßnummer des Prozesses zurück, sofern pc_cid die Timesharing-Klasse angibt und mehr als ein Timesharing-Prozeß angegeben ist. Gibt es mehrere Prozesse, welche die gleichen Benutzerprioritäten besitzen, so ist der Rückgabewert implementierungsabhängig.

Jeder Timesharing-Prozeß kann seinen eigenen ts_uprilim-Wert verkleinern (oder den eines anderen Prozesses mit derselben Benutzernummer). Nur ein Timesharing-Prozeß mit Systemverwalterrechte kann ts_uprilim erhöhen. Wird die Klasse eines Prozesses von einer anderen Klasse auf die Timesharing-Klasse geändert, so sind Systemverwalterrechte erforderlich, um ts_uprilim auf einen Wert größer als Null zu setzen. Versucht ein Prozeß, der nicht über Systemverwalterrechte verfügt, ts_uprilim zu erhöhen oder ts_uprilim größer Null zu setzen, wird -1 zurückgegeben und errno auf EPERM gesetzt.

Jeder Timesharing-Prozeß kann seinen eigenen ts_upri-Wert (oder den eines anderen Prozesses mit derselben Benutzernummer) auf einen Wert kleiner oder gleich dem Wert ts_uprilim des Prozesses setzen. Wird versucht, ts_upri größer als ts_uprilim zu setzen (und/oder ts_uprilim unter ts_upri zu setzen), so wird ts_upri gleich ts_uprilim gesetzt.

Entweder ts_uprilim oder ts_upri kann auf den besonderen Wert TS_NOCHANGE gesetzt werden (definiert in sys/tspriocntl.h), um den Wert nicht zu beeinflussen. Die Angabe von TS_NOCHANGE für ts_upri beim Einstellen von ts_uprilim auf einen Wert unter dem aktuellen ts_upri verursacht, daß ts_upri auf ts_uprilim gesetzt wird. Die Angabe von TS_NOCHANGE für einen Parameter beim Ändern der Klasse auf die Timesharing-Klasse (von einer anderen Klasse) verursacht, daß der Parameter auf den voreingestellten Wert gesetzt wird. Der voreingestellte Wert für ts_uprilim ist 0 und der voreingestellte Wert für ts_upri wird auf den ts_uprilim-Wert gesetzt, der eingestellt wird.

Die Timesharing-Benutzerpriorität und die Benutzerprioritätsgrenze werden über die Systemaufrufe fork und exec vererbt.

priocntl schlägt fehl, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

EPERM — Der aufrufende Prozeß verfügt nicht über die erforderlichen Rechte.

EINVAL — Das Argument *cmd* ist ungültig, eine ungültige oder nicht konfigurierte Klasse wurde angegeben, oder einer der angegebenen Parameter war ungültig.

EDOM — Das angegebene Zeitquantum befand sich außerhalb des erlaubten Bereichs.

ESRCH — Keiner der angegebenen Prozesse existiert.

EFAULT — Alle oder Teile der Datenbereiche, auf die die Datenzeiger verweisen, befinden sich außerhalb des Adreßbereichs des Prozesses.

ENOMEM — Das Ändern der Klasse eines Prozesses schlug fehl, weil zu wenig Speicher verfügbar war.

EAGAIN — Das Ändern der Klasse eines Prozesses schlug fehl, weil unzureichende Ressourcen (außer Hauptspeicher) verfügbar waren (beispielsweise klassenspezifische Datenstrukturen im Kernel).

SIEHE AUCH

fork(2), exec(2), nice(2), priocntlset(2).
priocntl(1) in "SINIX V5.41 Kommandos".
dispadmin(1M), rt_dptbl(4), ts_dptbl(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Falls nichts anderes gesagt wurde, liefert priocntl den Wert 0 zurück, um einen Erfolg anzuzeigen. Bei Fehlern liefert priocntl -1 zurück und setzt errno, um den Fehler anzuzeigen.

priocntlset - Prozesse kontrollieren

```
#include <sys/types.h>
#include <sys/procset.h>
#include <sys/priocntl.h>
#include <sys/rtpriocntl.h>
#include <sys/tspriocntl.h>

long priocntlset(procset_t *psp, int cmd, ... /* arg */);
```

`priocntlset` ändert die Prioritätsparameter von laufenden Prozessen. `priocntlset` hat dieselben Funktionen wie der Systemaufruf `priocntl`, jedoch können Prozesse, deren Prioritätsparameter geändert werden sollen, allgemein angegeben werden.

cmd gibt die Funktion an, die durchgeführt werden soll. *arg* ist ein Zeiger auf eine Struktur, deren Typ von *cmd* abhängt. Siehe `priocntl`(2) für die gültigen Werte von *cmd* und die entsprechenden *arg*-Strukturen.

psp ist ein Zeiger auf eine `procset`-Struktur, die `priocntlset` zur Bestimmung der Prozesse verwendet, deren Prioritätsparameter geändert werden sollen.

```
typedef struct procset {
  idop_t    p_op;       /* Operator der linken/rechten Satz verbindet */
  idtype_t  p_lidtype;  /* ID-Typ des linken Satzes */
  id_t      p_lid;      /* ID des linken Satzes */
  idtype_t  p_ridtype;  /* ID-Typ des rechten Satzes */
  id_t      p_rid;      /* ID des rechten Satzes */
} procset_t;
```

`p_lidtype` und `p_lid` bestimmen den ID-Typ und die ID von einer ('linken') Prozeßmenge; `p_ridtype` und `p_rid` bestimmen den ID-Typ und die ID einer zweiten ('rechten') Prozeßmenge. ID-Typen und IDs werden genau wie beim Systemaufruf `priocntl` angegeben. `p_op` gibt die Operation an, die für beide Prozeßmengen ausgeführt werden soll, um die resultierende Prozeßmenge zu erhalten, auf den der Systemaufruf anzuwenden ist. Gültige Werte für `p_op` und die Prozesse sind:

`POP_DIFF` Differenz bilden: Prozesse aus der linken Menge, die sich nicht in der rechten Menge befinden

`POP_AND` Schnittmenge: Prozesse sowohl in der linken als auch in der rechten Menge

`POP_OR` Vereinigungsmenge: Prozesse aus der linken oder rechten Menge oder aus beiden

`POP_XOR` Exklusiv-Oder: Prozesse aus der linken oder rechten Menge, aber nicht aus beiden.

Das folgende Makro, das in `procset.h` definiert wird, bietet eine gängige Methode zum Initialisieren einer `procset`-Struktur:

```
#define       setprocset(psp, op, ltype, lid, rtype, rid) \
(psp)->p_op       = (op), \
(psp)->p_lidtype  = (ltype), \
(psp)->p_lid      = (lid), \
(psp)->p_ridtype  = (rtype), \
(psp)->p_rid      = (rid),
```

ERGEBNIS

`priocntlset` hat dieselben Rückgabewerte und Fehler wie `priocntl`.

SIEHE AUCH

`priocntl`(2).
`priocntl`(1) in "SINIX V5.41 Kommandos".

processflags - Prozeßoptionen einstellen/abfragen

```
#include <sys/processflags.h>
processflags(int funk, int opt, int *ergeb, int id);
```

Mit `processflags` ist es möglich, bestimmte Optimierungen für einen bestimmten Rechner oder ein Teilsystem einzustellen. Sie können Options-Einstellungen für einen Prozeß setzen, hinzufügen, löschen oder abfragen.

funk kann die folgenden Werte haben:

`GET_PFLAGS` **ergeb* erhält den aktuellen Wert von *processflags*.

`SET_PFLAGS` *processflags* erhält den Wert von *opt*.

`BIS_PFLAGS` Ein Bit in *opt* wird in *processflags* eingefügt.

`BIC_PFLAGS` Ein Bit in *opt* wird in *processflags* gelöscht.

`SET_PFPRIV` Der Systemverwalter kann Rechte an *id* vergeben, so daß *id* Optionen ändern kann, die normalerweise dem Systemverwalter vorbehalten sind. Maßgeblich sind die in *opt* angegebenen Rechte.

`GET_PFPRIV` Abfragen der aktuellen Rechte für *id*.

Es ist möglich, Operationen (außer `GET_PFPRIV`) zusammen mit `GET_PFLAGS` abzusetzten; zum Beispiel (`SET_PFLAGS/GET_PFLAGS`). Für alle Operationen, außer `GET_PFLAGS` und `GET_PFPRIV`, gibt es eine implizite Operation `GET_PFLAGS`. Der Zeiger *ergeb* muß für `GET_PFLAGS` bzw. `GET_PFPRIV` einen Wert ungleich Null haben, damit die Adresse, auf die *ergeb* zeigt, gesetzt werden kann. Die Adresse, auf die *ergeb* zeigt, wird auf den endgültigen Wert von `processflags` gesetzt. Bei `GET_PFPRIV` wird die Adresse auf die aktuellen Rechte von `id` gesetzt. Siehe auch die Beschreibung zu `SET_PFPRIV`.

opt kann das Ergebnis einer logischen ODER-Operation der folgenden Werte sein:

`FORK_UNLIMITED`
Der Prozeß kann unbegrenzt neue Prozesse erzeugen.

`USR_SIGMASK`
Beim Prüfen der blockierten Signale ist das globale Register 9 zu verwenden.

`UMASK_ANY` Dieser Prozeß darf `umask` unter den systemweiten Mindestwert von `umask` setzen.

`AUID_ANY` Dieser Root-Prozeß darf seine AUID setzen oder zurücksetzen, auch wenn dieser Wert schon ungleich Null ist.

NFS_SOFT Alle NFS-Dateisysteme werden als "soft-mounted" betrachtet, wenn der Zugriff für diesen Prozeß erfolgt.

FIXEDPRI Der Prozeß läuft mit einer festgelegten Priorität ab.

FIXEDCPU Dieser Prozeß läuft nur auf einer bestimmten CPU.

FIXEDCPU_EXCL
Dieser Prozeß läuft ausschließlich auf einer bestimmten CPU. Prozesse ohne Bindung an diese CPU dürfen nicht auf dieser CPU ablaufen.

BINDCPU(Bit)
Makro für die Angabe der CPU-Nummer für FIXEDCPU-Optionen.

USR_PREMPTMASK
Dieser Prozeß kann ein Vorrecht deaktivieren, indem Bit 0 des globalen Registers 11 auf einen Wert ungleich Null gesetzt wird.

AFFIN_OFF Standard-Prozeßsteuerung für diesen Prozeß deaktivieren.

ANOTHERPROCESS
Dieser Aufruf wird an einen anderen Prozeß geleitet (die Prozeßnummer wird mit dem Argument *id* angegeben).

Weitere Informationen zu *opt*:

FORK_UNLIMITED
Beschränkungen bezüglich der maximalen Anzahl an Prozessen, die von einer Benutzernummer (nicht dem Systemverwalter) erzeugt werden können, werden aufgehoben. Wenn diese Option gesetzt ist, dringt das System nicht auf Einhaltung der normalen Beschränkung MAXUPRC bei den Aufrufen fork(2) bzw. vfork(2). Diese Option ist für bestimmte System-Dämonprozesse nützlich, die nicht unter Root laufen sollen, aber manchmal mehr als die zulässigen Prozesse erzeugen müssen (zum Beispiel Zeilendrucker-Dämon). Diese Option kann nur vom Systemverwalter gesetzt werden. Sie wird mit den Aufrufen fork(2) und exec(2) vererbt.

USR_SIGMASK
Ein Prozeß blockiert Signale aufgrund der Werte der Signalmaske und des globalen Registers 9 (GR9) des aufrufenden Prozesses. So können Signale blockiert werden, ohne daß der Systemaufruf sigsetmask(2) verwendet werden muß. GR9 wird als zusätzlicher Wert der Signalmaske benutzt und wird mit einer ODER-Operation mit der Standard-Signalmaske verbunden. USR_SIGMASK ist besonders dann sehr nützlich, wenn ein Prozeß Signale nur kurz blockieren will. Dies ist zum Beispiel dann der Fall, wenn eine Operation ausgeführt werden soll, die kurze Zeit isoliert bearbeitet werden muß. Die Bit-Angaben in GR9 entsprechen denen für sigsetmask(2). Die Bit-Angaben für SIGKILL, SIGSTOP und SIGCONT werden

stillschweigend ignoriert. Bei SIGTSTP, SIGTTIN und SIGTTOU wird das Signal nicht blockiert, wenn es sich um SIG_DFL handelt. Diese Option wird bei dem Aufruf fork(2) vererbt. Bei exec(2) wird sie nicht vererbt.

UMASK_ANY Der Prozeß kann dem aktuellen umask einen Wert zuweisen, der unterhalb des systemweiten Mindestwert von umask liegt. Diese Option kann von Prozessen benutzt werden, die keine Root-Prozesse sind. Der Wert wird bei den Aufrufen fork und exec vererbt.

AUID_ANY Ein Root-Prozeß kann seine AUID setzen oder zurücksetzen, auch wenn der Wert ungleich Null ist. Nur ein Root-Prozeß kann diese Option verwenden. Der Wert wird mit den Aufrufen fork und exec vererbt.

NFS_SOFT NFS-Dateisysteme werden als "soft-mounted" betrachtet, wenn der Zugriff über diesen Prozeß erfolgt. Wenn der Server eines NFS-Dateisystems nicht aktiv ist und ein Prozeß mit dem Attribut NFS_SOFT versucht, darauf zuzugreifen, wird ein Fehler gemeldet. Es wird nicht gewartet, bis der Server aktiviert wird. Diese Option wird bei einem Aufruf von fork(2) vererbt. Bei einem Aufruf von exec(2) wird sie nicht vererbt.

FIXEDPRI Die Standard-Mechanismen für Prioritäten werden deaktiviert und der Prozeß läuft mit einer festgelegten Priorität ab. Die Priorität wird auf PUSER gesetzt (PUSER enthält den Standardwert für benutzereigene Prozesse vor der Veränderung durch die Prioritäten-Mechanismen). Dieser Wert kann durch Setzen eines NICE-Wertes ungleich Null geändert werden (siehe nice(2)).

Diese Option ist zum Beispiel dann nützlich, wenn bestimmten "Server"-Prozessen, die für mehrere Benutzer ablaufen, aufgrund ihrer CPU-Zeit eine ungerechtfertigte Priorität zugewiesen wird.

Mit dieser Option können alle anderen Prozesse von der Bearbeitung ausgeschlossen werden! Diese Option sollte nur von erfahrenen Benutzern und mit Vorsicht eingesetzt werden, um zu verhindern, daß eine ungünstige Kombination der Priorität mit dieser Option den Systemablauf beeinträchtigt.

FIXEDCPU und FIXEDCPU_EXCL

Bei einem Multiprozessor-System wird diese Option verwendet, um einen Prozeß an eine bestimmte CPU zu binden. Die Angabe erfolgt durch eine ODER-Operation der CPU-Nummer unter Verwendung des Makros BINDCPU. Dieser Prozeß läuft dann nur auf der angegebenen CPU ab. Wenn keine exklusive Bindung hergestellt wurde, konkurriert der gebundene Prozeß mit anderen Prozessen um diese CPU, wobei die Prozeß-Priorität maßgeblich ist. Mit der Option FIXEDCPU_EXCL kann eine exklusive Bindung eingegangen werden. Solange es exklusiv gebundene Prozesse für eine CPU gibt, laufen keine Prozesse auf dieser CPU, die nicht ebenfalls an diese gebunden sind. So kann eine CPU für einen Prozeß oder eine bestimmte Menge von Prozessen reserviert werden.

Nach der Beendigung aller exklusiv gebundenen Prozesse kehrt die CPU zum normalen Betriebsmodus zurück. Eine exklusive Bindung kann zu jeder CPU im System hergestellt werden. Allerdings können nicht alle CPUs im System exklusiv gebunden werden. Es muß also mindestens eine CPU geben, die nicht exlusiv gebunden ist, und die dann die ungebundenen Prozesse übernehmen kann.

Eine Bindung an eine CPU kann nur vom Systemverwalter vorgenommen werden. Eine Änderung (eine Bindung an eine andere CPU) durch gebundene Prozesse ist jedoch möglich, auch wenn keine Systemverwalter-Rechte vorliegen.

Alle Bindungen von Prozessen werden mit den Systemaufrufen fork(2) und exec(2) vererbt.

USR_PREMPTMASK

Ein Prozeß kann direkt steuern, ob er einem anderen Prozeß mit höherer Priorität ein Vorrecht einräumt. Zur Vereinfachung der Vorrecht-Steuerung wird Bit 0 des globalen Registers 11 (GR11) verwendet. Diese Option ermöglicht in bestimmten Multiprozessor-Anwendungen mit Verwendung gemeinsamer Speicherbereiche und Sperren den Zugriffsschutz auf gemeinsame Datenstrukturen. Sie sollte nur für relativ kurze "kritische Sequenzen" verwendet werden, bei denen es im Fall einer Vorrecht-Behandlung zu Problemen kommen kann. Diese Option kann nur vom Systemverwalter verwendet werden. Sie wird beim Aufruf von fork(2) und exec(2) vererbt.

AFFIN_OFF Das Standardverhalten für die Prozeßverarbeitung wird für diesen Prozeß deaktiviert. Dadurch kann der Prozeß zwischen den CPUs wechseln, und ist nicht auf eine bestimmte CPU beschränkt. Eine Anwendung kommt vor allem für Server-Dämons hoher Priorität in Frage, da hier eine kurze Wartezeit wichtiger ist als eine möglichst geringe CPU-Zeit (ein Beispiel wäre ein Protokoll-Dämon in einem Datenbank-System). Diese Option kann nur vom Systemverwalter verwendet werden. Sie wird mit dem Aufruf von exec(2) vererbt, nicht jedoch mit dem Aufruf von fork(2).

Abhängig vom Prozeßverhalten kann diese Option die CPU-Zeit des Prozesses um 10-100% erhöhen!

ANOTHERPROCESS
Dieser Aufruf wird an den Prozeß *id* geleitet. Wenn diese Option nicht gesetzt ist, wird der Aufruf auf den Prozeß angewendet, der den Aufruf abgesetzt hat (dann wird das Argument *id* ignoriert). Diese Option kann nur vom Systemverwalter verwendet werden.

Wenn funk den Wert SET_PFPRIV hat, kann der Systemverwalter einem bestimmten Benutzer *id* die Berechtigung erteilen, Optionen zu verändern, die normalerweise dem Systemverwalter vorbehalten sind (abhängig von den Bitwerten in *opt*). Dadurch können auch andere Benutzer kritische Prozeßoptionen, wie die Vorrecht-Steuerung oder Prioritätenfestlegung, einstellen. Diese Rechte bleiben erhalten, bis das System neu gestartet wird oder der Systemverwalter diese Rechte zurücknimmt.

Wenn Benutzer Optionen ändern können, die normalerweise dem Systemverwalter vorbehalten sind, ermöglicht dies Einstellungen, die das Systemverhalten wesentlich beeinträchtigen können. Es wird deshalb empfohlen, nur sehr zurückhaltend von dieser Möglichkeit Gebrauch zu machen.

ERGEBNIS

processflags gibt 0 zurück, wenn keine Fehler aufgetreten sind. Andernfalls wird -1 zurückgegeben und errno gesetzt.

SIEHE AUCH

sigsetmask(2), mpcntl(3)

profil - Ausführungsprofil erstellen

```
#include <unistd.h>
void profil(unsigned short *buff, size_t bufsiz, int offset, unsigned scale);
```

`profil` stellt Statistiken über den CPU-Verbrauch eines Programmes bereit, indem es die aufgewendete Zeit mitprotokolliert. `profil` erzeugt Statistiken, indem es ein Ausführungshistogramm für einen laufenden Prozeß erstellt. Das Histogramm ist für einen spezifischen Programmbereich definiert, für den das Profil erstellt werden soll. Der definierte Bereich wird in mehrere Teile gleicher Größe aufgegeliedert, von denen jeder einzelne einer Zählung im Histogramm entspricht. Mit jedem Prozeßwechsel wird der aktive Teil identifiziert und seine entsprechende Histogrammzählung erhöht. Diese Zählungen geben einen relativen Wert, wieviel Zeit in jedem Code-Teil verbraucht wird. Die für einen protokollierten Bereich resultierenden Histogrammzählungen können zur Identifizierung jener Funktionen benutzt werden, die einen unverhältnismäßig großen prozentualen Anteil an CPU-Zeit verbrauchen.

buff ist ein Puffer mit *bufsiz*-Bytes, in welchem die Histogrammzählungen als eine Reihe von `unsigned short int` gespeichert sind.

offset, *scale* und *bufsiz* geben den zu protokollierenden Bereich an.

offset ist die Startadresse des zu profilierenden Bereichs.

scale ist ein Kontraktionsfaktor, der anzeigt, um wieviel kleiner der Histogrammpuffer im Gegensatz zu dem zu profilierenden Bereich ist. *scale* wird also als eine 16-Bit-Festpunktzahl ohne Vorzeichen interpretiert, deren Dezimalpunkt links vor dem ersten Bit steht. Sein Wert ist das Reziproke der Anzahl an Bytes in einer Histogrammunterteilung. Da es zwei Bytes pro Histogrammzähler gibt, ist die tatsächliche Anzahl Bytes pro Histogrammunterteilung das Doppelte der angegebenen Skala.

- Der maximale Wert von *scale*, `0xffff` (etwa 1), erzeugt Abbildungsunterteilungen mit einer Länge von 2 Bytes für jeden Zähler.
- Der minimale Wert von *scale* (dafür wird protokolliert), `0x0002` (1/32,768), erzeugt Abbildungsunterteilungen mit einer Länge von 65,536 Bytes für jeden Zähler.
- Der Standardwert von *scale* (momentan benutzt von `cc -qp`), `0x4000`, erzeugt Abbildungsunterteilungen mit einer Länge von 8 Bytes für jeden Zähler.

Die Werte werden innerhalb des Betriebssystemkerns folgenderweise benutzt: Sobald der Prozeß für einen Prozeßwechsel unterbrochen ist, wird der Wert von *offset* vom aktuellen Wert des Programmzählers subtrahiert und mit dem Rest von *scale* multipliziert, um ein Ergebnis abzuleiten. Dieses Ergebnis wird in der Histogrammreihe als Index benutzt, womit die zu inkrementierende Speicherzelle lokalisiert wird. Daher repräsentiert die Speicherzellenzählung die Anzahl der Zeitpunkte, in denen der Prozeß den

Code in der mit dieser Speicherzelle verbundenen Unterteilung ausgeführt hat, als der Prozeß unterbrochen wurde.

scale kann als (*RATIO* * 0200000) berechnet werden, wobei *RATIO* das erwünschte Zahlenverhältnis von *bufsiz* zur profilierten Bereichsgröße ist und einen Wert zwischen 0 und 1 hat. Je näher *RATIO* also 1 kommt, umso höher ist die Auflösung der Profil-Information.

bufsiz kann als (*size_of_region_to_be_profiled* * *RATIO*) berechnet werden.

SIEHE AUCH

prof(1), tims(2), monitor(3C).

HINWEIS

Die Profilierung wird abgeschaltet, wenn eine *scale* von 0 oder 1 gegeben wird. Sie wird wirkungslos, wenn eine Größe *bufsiz* von 0 eingegeben wird. Die Profilierung wird abgeschaltet, wenn ein exec(2) ausgeführt wird. Sie bleibt jedoch in einem Sohn- und Vaterprozeß nach einem fork(2) eingeschaltet. Die Profilierung wird abgeschaltet, wenn eine Änderung in *buff* einen Speicherfehler verursachen würde.

ptrace - Prozeß-Ablauf verfolgen

```
#include <unistd.h>
#include <sys/types.h>
int ptrace(int request, pid_t pid, int addr, int data);
```

Mit `ptrace` kann ein Vaterprozeß die Ausführung eines Sohnprozesses steuern. Das Kommando wird hauptsächlich zur Fehlersuche mit Hilfe von Haltepunkten verwendet (siehe `sdb`(1)). Normalerweise läuft der Sohnprozeß wunschgemäß ab, bis ein Signal eintritt (siehe Liste in `signal`(5)), bei dem er einen Stopp-Zustand erreicht, woraufhin der Vaterprozeß über `wait`(2) entsprechend benachrichtigt wird. Befindet sich der Sohnprozeß im Stopp-Zustand, kann der Vaterprozeß den 'Speicherabzug' mit `ptrace` prüfen und ändern. Auch kann der Vaterprozeß veranlassen, daß der Sohnprozeß die Ausführung entweder beendet oder weiter ausführt, wobei die Möglichkeit gegeben ist, das Signal, welches das Stoppen verursacht, zu ignorieren.

Mit dem Argument *request* wird die `ptrace` Aktion festgelegt. *request* kann folgende Werte haben:

`0` Der Sohnprozeß fordert, daß sein Ablauf vom Vaterprozeß verfolgt werden soll. Die Ablaufverfolgungsoption des Sohnprozesses wird eingeschaltet, welche bestimmt, daß der Sohnprozeß bei Empfang des Signals in einem Stopp-Zustand und nicht in dem mit *disp* angegebenen Zustand gelassen wird (siehe `signal`(2)). Die Argumente *pid*, *addr*, und *data* werden ignoriert, und ein Rückgabewert für diese Anforderung ist nicht definiert. Wenn der Vaterprozeß keine Ablaufverfolgung des Sohnprozesses durchführt, entstehen eigenartige Resultate.

Alle weiteren Anforderungen können nur vom Vaterprozeß eingesetzt werden. Für beide ist *pid* die Prozeßnummer des Sohnprozesses. Der Sohnprozeß muß sich in einem Stopp-Zustand befinden, bevor diese Anforderungen gestellt werden können.

`1, 2` Das Wort an der Stelle *addr* im Adreßraum des Sohnprozesses wird an den Vaterprozeß zurückgegeben. Sind Anweisungs- und Datenbereich voneinander getrennt, gibt Anforderung `1` ein Wort aus dem Anweisungsbereich und Anforderung `2` aus dem Datenbereich zurück. Sind die Anweisungs- und Datenbereiche nicht voneinander getrennt, führen die Anforderungen `1` und `2` zum gleichen Ergebnis. Das Argument *data* wird ignoriert. Diese beiden Anforderungen bleiben erfolglos, wenn *addr* nicht die Anfangsadresse eines Worts ist. In diesem Fall wird ein Wert von -1 an den Vaterprozeß zurückgegeben, und `errno` des Vaterprozesses wird auf `EIO` gesetzt.

3 Das Wort an der Stelle *addr* im Benutzerbereich des Sohnes im System-Prozeßverwaltungsbereich (siehe `sys/user.h`) wird an den Vaterprozeß zurückgegeben. Das Argument *data* wird ignoriert. Diese Anforderung ist erfolglos, wenn *addr* nicht die Anfangsadresse eines Worts ist oder außerhalb des Benutzerbereichs liegt. In diesem Fall wird ein Wert von -1 an den Vaterprozeß zurückgegeben, und `errno` des Vaterprozesses wird auf `EIO` gesetzt.

4, 5 Der für das Argument *data* gegebene Wert wird in den Adreßraum des Sohnprozesses an die Stelle *addr* geschrieben. Wenn Anweisungs- und Datenbereich voneinander getrennt sind, schreibt die Anforderung `4` ein Wort in den Anweisungsbereich und die Anforderung `5` ein Wort in den Datenbereich. Sind die Anweisungs- und Datenbereiche nicht voneinander getrennt, führen die Anforderungen `4` und `5` zu dem gleichen Ergebnis. Nach erfolgreicher Beendigung wird der in den Adreßraum des Sohnprozesses geschriebene Wert an den Vaterprozeß zurückgegeben. Diese beiden Anforderungen sind erfolglos, wenn *addr* nicht die Anfangsadresse eines Worts war. In diesem Fall wird ein Wert von -1 an den Vaterprozeß zurückgegeben, und `errno` des Vaterprozesses wird auf `EIO` gesetzt.

6 Mit dieser Anforderung können einige Einträge in den Benutzerbereich des Sohnprozesses geschrieben werden. *data* liefert den Wert, der geschrieben werden soll, und *addr* ist die Stelle für den Eintrag. Die Einträge können Register und Bedingungscodes vom Statuswort des Prozessors sein.

7 Der Sohnprozeß nimmt die Ausführung wieder auf. Wenn das Argument *data* 0 ist, werden alle anstehenden Signale einschließlich des Signals, das den Stopp des Sohnprozesses verursacht hat, vor der Wiederaufnahme der Ausführung aufgehoben. Wenn das Argument *data* eine gültige Signalnummer ist, nimmt der Sohnprozeß die Ausführung wieder auf, als ob er dieses Signal angetroffen hätte, und alle anderen anstehenden Signale werden aufgehoben. Das Argument *addr* muß für diese Anforderung gleich 1 sein. Nach erfolgreicher Beendigung wird der Wert von *data* wieder an den Vaterprozeß zurückgegeben. Diese Anforderung ist erfolglos, wenn *data* nicht 0 oder keine gültige Signalnummer ist. In diesem Fall wird ein Wert von -1 an den Vaterprozeß zurückgegeben, und `errno` des Vaterprozesses wird auf `EIO` gesetzt.

8 Der Sohnprozeß wird mit den gleichen Folgen beendet wie `exit`(2).

9 Das Ablauf-Verfolgungsbit im Prozessor-Statuswort des Sohnprozesses wird gesetzt und führt dann dieselben Schritte aus wie oben für Anforderung `7` aufgelistet. Das Ablauf-Verfolgungsbit bewirkt eine Unterbrechung nach Ausführung einer Maschinenanweisung. Dies ermöglicht Einzelschritt-Ablaufverfolgungen des Sohnes.

Zur Verhinderung unbefugter Eingriffe sperrt `ptrace` das Setzen des s-Bits bei den nachfolgenden `exec`(2)-Aufrufen. Wenn ein ablaufverfolgter Prozeß `exec`(2) aufruft, stoppt der Prozeß vor Ausführung der ersten Anweisung des neuen Programms mit der Anzeige von Signal `SIGTRAP`. `ptrace` ist im allgemeinen erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EIO` *request* ist eine unzulässige Nummer.

`ESRCH` *pid* kennzeichnet einen Sohnprozeß, der nicht vorhanden oder kein `ptrace` mit der Anforderung `0` ist.

`EPERM` Das aufrufende Programm hat nicht die notwendigen MAC-Privilegien.

SIEHE AUCH

`sdb`(1), `exec`(2), `signal`(2), `wait`(2).

putmsg, putpmsg - Nachricht auf einen Stream senden

```
#include <stropts.h>
int putmsg(int fd, const struct strbuf *ctlptr, const struct strbuf *dataptr, int flags);
int putpmsg(int fd, const struct strbuf *ctlptr, const struct strbuf *dataptr,
            int band, int flags);
```

putmsg erstellt aus dem vom Benutzer angegebenen Puffer eine Nachricht und sendet diese an eine STREAMS-Datei. Die Nachricht kann entweder einen Datenteil, einen Steuerteil oder auch beide Teile enthalten. Die zu sendenden Daten- und Steuerteile werden voneinander unterschieden, indem sie in verschiedene Puffer geschrieben werden. Die Semantik jedes Teils ist durch das STREAMS-Modul definiert, das die Nachricht empfängt.

Die Funktion putpmsg macht dasselbe wie putmsg, aber sie gibt dem Benutzer die Möglichkeit, Nachrichten in verschiedenen Prioritätsfolgen zu senden. Außer wenn es explizit angegeben wird, gelten alle Informationen für putmsg auch für putpmsg.

fd kennzeichnet einen Dateideskriptor, der auf einen offenen Stream verweist. *ctlptr* und *dataptr* weisen jeweils auf eine strbuf-Struktur, die folgende Elemente enthält:

```
int maxlen;     /* nicht verwendet */
int len;        /* Länge der Daten */
void *buf;      /* Zeiger auf Puffer für Daten */
```

ctlptr weist auf die Struktur, die den Steuerteil beschreibt, der in die Nachricht aufgenommen werden soll. Das Feld buf in der strbuf-Struktur weist auf den Puffer, in dem die Steuerinformationen stehen, und das Feld len weist auf die Anzahl der zu sendenden Bytes. Das Feld maxlen wird in putmsg nicht verwendet (siehe getmsg(2)). Auf gleiche Weise gibt *dataptr* die Daten an, die in die Nachricht aufgenommen werden sollen. *flags* gibt an, was für ein Nachrichtentyp gesendet werden soll.

Zum Senden des Datenteils einer Nachricht muß *dataptr* ungleich NULL sein, und das Feld len von *dataptr* muß einen Wert von 0 oder größer aufweisen. Zum Senden des Steuerteils einer Nachricht müssen die entsprechenden Werte für *ctlptr* gesetzt sein. Ein Daten-(Steuer-)Teil wird nicht gesendet, wenn entweder *dataptr* (*ctlptr*) gleich NULL ist oder das Feld len von *dataptr* (*ctlptr*) auf -1 gesetzt ist.

Wird bei putmsg ein Steuerteil angegeben, und ist *flags* auf RS_HIPRI gesetzt, wird eine Nachricht mit hoher Priorität geschickt. Ist kein Steuerteil angegeben und *flags* auf RS_HIPRI gesetzt, ist putmsg erfolglos und setzt errno auf EINVAL. Wenn *flags* auf 0 gesetzt ist, wird eine normale (keine Priorität) Nachricht geschickt. Sind kein Steuerteil und kein Datenteil angegeben und ist *flags* auf 0 gesetzt, wird keine Nachricht gesendet, und 0 wird zurückgegeben.

Der Stream-Kopf garantiert, daß der Steuerteil einer von putmsg erzeugten Nachricht höchstens 64 Bytes lang ist.

Für putpmsg sind die Optionen anders: *flags* ist eine Bitmaske, mit den sich gegenseitig ausschließenden Optionen MSG_HIPRI und MSG_BAND. Wenn *flags* auf 0 gesetzt ist, scheitert putpmsg und setzt errno auf EINVAL. Wenn ein Steuerteil angegeben ist und *flags* auf MSG_HIPRI und *band* auf 0 gesetzt sind, wird eine Nachricht mit hoher Priorität gesendet. Wenn *flags* auf MSG_HIPRI gesetzt ist, und entweder kein Steuerteil angegeben ist oder *band* auf einen Wert ungleich Null gesetzt ist, scheitert putpmsg und setzt errno auf EINVAL. Wenn *flags* auf MSG_BAND gesetzt ist, wird eine Nachricht in der durch *band* angegebenen Prioritätsklasse gesendet. Wenn kein Steuer- und kein Datenteil angegeben und *flags* auf MSG_BAND gesetzt ist, wird keine Nachricht gesendet und 0 zurückgegeben.

Normalerweise blockiert putmsg, wenn die Schreib-Warteschlange des Streams aufgrund von internen Kontrollfluß-Bedingungen voll ist. Bei Nachrichten mit hoher Priorität blockiert putmsg bei dieser Bedingung nicht. Bei anderen Nachrichten blockiert putmsg nicht, wenn die Schreib-Warteschlange voll und O_NDELAY oder O_NONBLOCK gesetzt ist. Statt dessen bleibt der Aufruf erfolglos, und errno wird auf EAGAIN gesetzt.

putmsg oder putpmsg blockieren auch, wenn sie auf die Verfügbarkeit von von Nachrichtenblöcken im Stream warten, unabhängig von der Priorität und O_NDELAY oder O_NONBLOCK. Eine Teilnachricht wird nicht gesendet.

putmsg ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EAGAIN — Eine Nachricht ohne Priorität wurde angegeben, der Anzeiger O_NDELAY oder O_NONBLOCK ist gesetzt, und die Schreib-Warteschlange des Streams ist aufgrund von internen Kontrollfluß-Bedingungen blockiert.

EBADF — *fd* ist kein gültiger zum Schreiben offener Dateideskriptor.

EFAULT — *ctlptr* oder *dataptr* weisen über den zugewiesenen Adreßraum hinaus.

EINTR — Ein Signal wurde während des Systemaufrufs putmsg abgefangen.

EINVAL — Ein undefinierter Wert wurde in *flags* angegeben, oder *flags* ist auf RS_HIPRI gesetzt, und kein Steuerteil wurde bereitgestellt.

EINVAL — Der Stream, auf den *fd* zeigt, ist über einen Multiplexer angeschlossen.

EINVAL — Bei putpmsg: *flags* ist auf MSG_HIPRI gesetzt, und *band* ist ungleich Null.

ENOSR — Für die zu erstellende Nachricht konnte wegen zu geringem STREAMS-Speicherplatz kein Puffer zugewiesen werden.

ENOSTR — Zu *fd* gehört kein Stream.

ENXIO — Ein Hangup wurde streamabwärts für den angegebenen Stream generiert, oder das andere Ende der Pipe ist geschlossen.

ERANGE Die Größe des Datenteils der Nachricht liegt nicht in dem Bereich, der durch die maximale und minimale Paketgröße des obersten Stream-Moduls angegeben wurde. Dieser Wert wird auch zurückgegeben, wenn der Steuerteil der Nachricht größer als die maximale konfigurierte Größe des Steuerteils einer Nachricht ist, oder wenn der Datenteil einer Nachricht größer als die maximal konfigurierte Größe des Datenteils einer Nachricht ist.

putmsg ist auch erfolglos, wenn eine STREAMS-Fehlermeldung den Stream-Kopf vor dem Aufruf von putmsg erreicht hat. Bei dem zurückgegebenen Fehler handelt es sich um den Wert, der in der STREAMS-Fehlermeldung enthalten ist.

SIEHE AUCH

getmsg(2), intro(2), poll(2), putmsg(2), read(2), write(2).
"Leitfaden für Programmierer: STREAMS".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Wenn zwei Prozesse eine FIFO-Datei eröffnen, wobei der eine mit putmsg eine Nachricht hoher Priorität schreibt und der andere mit getmsg eine Nachricht hoher Priorität liest, können Nachrichten verlorengehen. Dieser Verlust kann vermieden werden, wenn der Sendeprozeß durch sleep zwischen den einzelnen putmsg verlangsamt wird.

read, readv - aus einer Datei lesen

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

int read(int fildes, void *buf, unsigned nbyte);

int readv(int fildes, struct iovec *iov, int iovcnt);
```

`read` versucht, *nbyte* Bytes von der zu *fildes* gehörenden Datei in den Puffer zu lesen, auf den *buf* zeigt. Wenn *nbyte* Null ist, gibt `read` Null zurück und hat keine anderen Auswirkungen. *fildes* ist ein Dateideskriptor, der von einem `creat`-, `open`-, `dup`-, `fcntl`- oder `pipe`-Systemaufruf geliefert wird.

Bei Geräten, die positionieren können, beginnt `read` an der Stelle in der Datei, die von dem zu *fildes* gehörenden Schreib-/Lesezeiger angegeben wird. Nach der Rückkehr von `read` wird der Schreib-/Lesezeiger um die Anzahl der tatsächlich gelesenen Bytes erhöht.

Geräte, die nicht positionieren können, lesen immer von der aktuellen Position an. Der Wert eines zu einer solchen Datei gehörenden Schreib-/Lesezeigers ist nicht definiert.

`readv` macht dasselbe wie `read`, aber hier werden die Eingabedaten in die *iovcnt*-Puffer gebracht, die durch die Elemente des Feldes *iov* spezifiziert sind: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1].

Für `readv` enthält die Struktur `iovec` folgende Elemente:

```
addr_t    iov_base;
size_t    iov_len;
```

Jeder `iovec`-Eintrag gibt die Basisadresse und Länge eines Speicherbereichs an, in den Daten gebracht werden sollen. `readv` füllt einen Puffer immer vollständig, bevor es mit dem nächsten weitermacht.

Bei Erfolg geben `read` und `readv` die Anzahl der tatsächlich gelesenen und in den Puffer geschriebenen Bytes zurück; diese Zahl kann kleiner als *nbyte* sein, wenn die Datei zu einer Datenübertragungsleitung gehört (siehe `ioctl`(2) und `termio`(7)), oder wenn die Anzahl der in der Datei vorhandenen Bytes kleiner als *nbyte* Bytes ist, oder wenn die Datei eine Pipe oder Gerätedatei ist. Bei Erreichen des Dateiendes wird 0 zurückgegeben.

`read` liest Daten, die vorher in eine Datei geschrieben worden sind. Wenn ein Teil einer normalen Datei vor dem Dateiende nicht geschrieben worden ist, gibt `read` die Anzahl der als `0` gelesenen Bytes zurück. Zum Beispiel erlaubt die Routine `lseek`, daß der Schreib-/Lesezeiger hinter das Ende der vorhandenen Daten in der Datei gesetzt wird. Wenn an diesen Punkt zusätzliche Daten geschrieben werden, geben aufeinanderfol-

gende read-Aufrufe in der Lücke zwischen dem vorherigen Datenende und den neu geschriebenen Daten Bytes mit einem Wert von 0 zurück, bis Daten in die Lücke geschrieben worden sind.

Ein read oder readv von einem STREAM (siehe intro(2)) kann in drei verschiedenen Modi arbeiten: byte-stream (byteweise), message-nondiscard (Nachricht nicht löschen) und message-discard (Nachricht löschen). Der byte-stream ist der Standard. Dieser kann mit der Anforderung I_SRDOPT ioctl(2) (siehe streamio(7)) geändert und mit I_GRDOPT ioctl(2) getestet werden. Im byte-stream-Modus rufen read und readv normalerweise Daten vom Stream ab, bis sie *nbyte* Bytes geholt haben, oder bis keine weiteren Daten mehr zum Abrufen vorhanden sind. Nachrichten-Grenzen werden im allgemeinen ignoriert.

Im Modus message-nondiscard rufen read und readv Daten ab, bis *nbyte* Bytes gelesen sind oder bis eine Nachrichten-Grenze erreicht wird. Wenn read oder readv nicht alle Daten in einer Nachricht abrufen, werden die übrigen Daten in den Stream zurückgestellt und können vom nächsten read- oder readv-Aufruf abgerufen werden. Im Modus message-discard werden ebenfalls Daten abgerufen, bis *nbyte* Bytes abgerufen sind oder eine Nachrichten-Grenze erreicht wird. Jedoch werden nicht gelesene Daten, die in einer Nachricht zurückbleiben, gelöscht und stehen für ein anschließendes read, readv oder getmsg (siehe getmsg(2)) nicht mehr zur Verfügung.

Wenn von einer normalen Datei, bei der Dateisperren gesetzt sind (siehe chmod(2)) und eine von einem anderen Prozeß gesetzte Schreibsperre auf dem Dateisegment liegt, gelesen wird, hat das folgende Auswirkung:

- Wenn O_NDELAY oder O_NONBLOCK gesetzt ist, gibt read -1 zurück und setzt errno auf EAGAIN.
- Wenn O_NDELAY und O_NONBLOCK nicht gesetzt sind, schläft read bis die blockierende Dateisatzsperre aufgehoben wird.

Wenn versucht wird, von einer leeren Pipe oder einer FIFO-Datei zu lesen, hat das folgende Auswirkung:

- Wenn kein Prozeß die Pipe zum Schreiben geöffnet hat, gibt read 0 zurück, um das Dateiende anzuzeigen.
- Wenn ein Prozeß die Pipe zum Schreiben geöffnet hat und O_NDELAY gesetzt ist, gibt read 0 zurück.
- Wenn ein Prozeß die Pipe zum Schreiben geöffnet hat und O_NONBLOCK gesetzt ist, gibt read -1 zurück und setzt errno auf EAGAIN.
- Wenn O_NDELAY und O_NONBLOCK nicht gesetzt sind, blockiert read, bis Daten in die Pipe geschrieben werden, oder bis die Pipe von allen Prozessen, die sie zum Schreiben geöffnet hatten, geschlossen worden ist.

Wenn versucht wird, eine Datei zu lesen, die einem Terminal zugeordnet ist, das zur Zeit keine Daten verfügbar hat, hat das folgende Auswirkung:

- Wenn `O_NDELAY` gesetzt ist, gibt `read` 0 zurück.
- Wenn `O_NONBLOCK` gesetzt ist, gibt `read` -1 zurück und setzt `errno` auf `EAGAIN`.
- Wenn `O_NDELAY` und `O_NONBLOCK` nicht gesetzt sind, blockiert `read`, bis Daten verfügbar werden.

Wenn versucht wird, eine Datei zu lesen, die zu einem Stream gehört, der weder eine Pipe noch eine FIFO-Datei oder ein Terminal ist und der zum gegebenen Zeitpunkt keine Daten verfügbar hat, hat das folgende Auswirkung:

- Wenn `O_NDELAY` oder `O_NONBLOCK` gesetzt ist, gibt `read` -1 zurück und setzt `errno` auf `EAGAIN`.
- Wenn `O_NDELAY` und `O_NONBLOCK` frei sind, blockiert `read` solange, bis Daten verfügbar werden.

Beim Lesen von einer STREAMS-Datei wird die Bearbeitung der leeren Nachrichten durch die aktuelle Lesemodus-Einstellung bestimmt. Im `byte-stream`-Modus akzeptiert `read` Daten, bis *nbyte* Bytes gelesen worden sind, bis keine weiteren Daten mehr zu lesen sind oder bis eine leere Nachricht angetroffen wird. `read` gibt dann die Anzahl der gelesenen Bytes zurück und setzt die leere Nachricht in dem Stream zurück, wo sie vom nächsten `read` oder `getmsg` (siehe `getmsg`(2)) abgerufen werden kann. In den beiden anderen Modi gibt eine leere Nachricht einen Wert von 0 zurück, und die Nachricht wird aus dem Stream entfernt. Wenn eine leere Nachricht als erste Nachricht in einem Stream gelesen wird, erfolgt unabhängig vom jeweiligen Lese-Modus die Rückgabe des Wertes 0.

Ein `read` oder `readv` einer STREAMS-Datei gibt die Daten in der ersten Nachricht aus der Leseschlange des Stream-Kopfes zurück, unabhängig von der Priorität der Nachricht.

Normalerweise kann ein `read` von einer STREAMS-Datei nur Datennachrichten ohne Steuerinformationen verarbeiten. `read` scheitert, wenn eine Nachricht mit Steuerinformationen im Stream-Kopf angetroffen wird. Dieses standardmäßige Verhalten kann verändert werden, indem man den Stream mit Hilfe von `I_SRDOPT` `ioctl`(2) in den Kontrolldaten-Modus oder in einen Modus versetzt, bei dem alle Kontrollinformationen gelöscht werden. Im Kontrolldaten-Modus werden die Kontrolldaten in normale Daten konvertiert und können mit `read` gelesen werden. Im anderen Fall werden die Kontrolldaten von `read` gelöscht. Mit diesen Nachrichten zusammenhängende normale Daten können aber trotzdem gelesen werden.

read und readv sind erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EAGAIN Obligatorisches Sperren von Dateien und Dateisätzen war gesetzt, O_NDELAY oder O_NONBLOCK war gesetzt, und eine blockierende Dateisatzsperre war vorhanden.

EAGAIN Der Systemspeicher, der für 'raw'-Ein/Ausgabe zur Verfügung steht, ist vorübergehend nicht ausreichend.

EAGAIN In einer mit einem TTY-Gerät verbundenen Datei warten keine Daten darauf, gelesen zu werden, und O_NONBLOCK ist gesetzt.

EAGAIN In einem Stream wartet keine Nachricht darauf, gelesen zu werden, und der Anzeiger O_NDELAY oder O_NONBLOCK ist gesetzt.

EBADF *fildes* ist kein gültiger Dateideskriptor, der zum Lesen geöffnet ist.

EBADMSG Die Nachricht, die in einem Stream darauf wartet, gelesen zu werden, ist keine Datennachricht.

EDEADLK read würde in einen schlafenden Zustand gehen und dadurch einen Deadlock verursachen.

EFAULT *buf* weist über den zugewiesenen Adreßraum hinaus.

EINTR Ein Signal wurde während des Systemaufrufs read oder readv abgefangen.

EINVAL Es wurde versucht, von einem Stream zu lesen, der mit einem Multiplexer verbunden ist.

EIO Ein physikalischer Ein-/Ausgabefehler ist aufgetreten. Oder der Prozeß ist in einer Prozeßgruppe im Hintergrund und versucht von seinem steuernden Terminal zu lesen, und entweder ignoriert oder blockiert der Prozeß das Signal SIGTTIN, oder die Prozeßgruppe des Prozesses hat keinen Vaterprozeß.

ENOLCK Die Tabelle der System-Datensatzsperren war voll, daher konnte read oder readv erst schlafen, nachdem die blockierende Datensatzsperre aufgehoben wurde.

ENOLINK *fildes* liegt auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENXIO Das mit *fildes* verbundene Gerät ist eine block- oder zeichenorientierte Gerätedatei, und der Schreib-/Lesezeiger ist außerhalb des Gültigkeitsbereichs.

Zusätzlich kann `readv` einen der folgenden Fehler zurückgeben:

`EFAULT` *iov* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINVAL` *iovcnt* war kleiner oder gleich 0 oder größer als 16.

`EINVAL` Die Summe der `iov_len`-Werte im *iov*-Feld brachte eine 32-Bit Ganzzahl zum Überlauf.

Ein `read` von einer STREAMS-Datei ist auch dann erfolglos, wenn am Stream-Kopf eine Fehlermeldung empfangen wird. In diesem Fall wird `errno` auf den Wert gesetzt, der in der Fehlermeldung zurückgegeben wird. Bei Auftreten eines Hangups im Stream, der gerade gelesen wird, läuft `read` normal weiter, bis die Lesewarteschlange des Stream-Kopfes entleert ist. Danach wird 0 zurückgegeben.

SIEHE AUCH

`intro`(2), `creat`(2), `dup`(2), `fcntl`(2), `getmsg`(2), `ioctl`(2), `open`(2), `pipe`(2). `streamio`(7), `termio`(7) in "Referenzhandbuch für Sytemverwalter".

ERGEBNIS

Bei Erfolg wird eine nicht negative ganze Zahl zurückgegeben, mit der die Anzahl der tatsächlich gelesenen Bytes angezeigt wird. Andernfalls wird -1 zurückgegeben und `errno` zur Anzeige des Fehlers gesetzt.

readlink - Wert eines symbolischen Verweises lesen

```
#include <unistd.h>
int readlink(const char *path, void *buf, size_t bufsiz);
```

`readlink` schreibt den Inhalt des symbolischen Verweises, auf den *path* weist, in den Puffer *buf*, welcher die Länge *bufsiz* hat. Der Inhalt des Verweises ist bei Rückgabe nicht mit einem Null-Byte abgeschlossen.

`readlink` schlägt fehl, und der Pufferinhalt wird nicht verändert, wenn

`EACCES` die Sucherlaubnis für eine Komponente des Pfadpräfixes aus *path* nicht vorhanden ist.

`EFAULT` *path* oder *buf* sich außerhalb des allokierten Adreßbereichs des Prozesses befindet.

`EINVAL` die genannte Datei kein symbolischer Verweis ist.

`EIO` ein E/A-Fehler beim Lesen oder Schreiben des Dateisystems auftritt.

`ELOOP` zu viele symbolische Verweise bei der Übersetzung von *path* auftreten.

`ENAMETOOLONG`
die Länge des *path*-Arguments `PATH_MAX` überschreitet, oder die Länge einer *path*-Komponente `NAME_MAX` überschreitet, während `_POSIX_NO_TRUNC` aktiviert ist.

`ENOENT` die genannte Datei nicht existiert.

`ENOSYS` das Dateisystem keine symbolischen Verweise unterstützt.

ERGEBNIS

Nach erfolgreicher Ausführung liefert `readlink` die Anzahl von Zeichen zurück, die in den Puffer geschrieben wurden; bei Fehler wird -1 zurückgegeben und der Fehlercode in `errno` gesetzt.

SIEHE AUCH

`lstat`(2), `stat`(2), `symlink`(2).

rename - Dateinamen ändern

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

`rename` benennt eine Datei um. *old* ist ein Zeiger auf den Pfadnamen der Datei oder des Verzeichnisses, das umbenannt werden soll. *new* ist ein Zeiger auf den neuen Pfadnamen der Datei oder des Verzeichnisses. Sowohl *old* als auch *new* müssen vom selben Typ sein (entweder beides Dateien oder beides Verzeichnisse) und müssen sich im selben Dateisystem befinden.

Wenn *new* bereits existiert, wird die Datei entfernt. Wenn *new* ein existierendes Verzeichnis benennt, darf das Verzeichnis keine Einträge außer '.' und '..' haben. Beim Umbenennen von Verzeichnissen darf der Pfadname *new* keine Datei benennen, die von *old* abstammt. Die Implementierung von `rename` garantiert, daß nach erfolgreicher Ausführung ein Verweis mit dem Namen *new* immer existiert.

Die endgültige Komponente von *old* ist ein symbolischer Verweis, welcher umbenannt wird, und nicht etwa die Datei oder das Verzeichnis, auf das verwiesen wird.

Für die Verzeichnisse, die *old* und *new* enthalten, wird Schreiberlaubnis benötigt.

`rename` schlägt fehl, *old* wird nicht geändert und die Datei *new* nicht erzeugt, wenn eine oder mehrere der folgenden Bedingungen gegeben sind:

`EACCES` Für eine Komponente eines Pfadpräfixes besteht keine Sucherlaubnis; für eins der Verzeichnisse, die die Dateien *old* und *new* enthalten, besteht keine Schreiberlaubnis, oder für eins der Verzeichnisse, auf die *old* und *new* zeigen, existiert keine Schreiberlaubnis.

`EBUSY` Das Verzeichnis *new* wird als Einhängepunkt für ein eingehängtes Dateisystem verwendet.

`EDQUOT` Das Verzeichnis, in dem sich der Eintrag für den neuen Namen befindet, kann nicht erweitert werden, da der Benutzer die Anzahl der zulässigen Blöcke in dem Dateisystem, in dem sich das Verzeichnis befindet, überschritten hat.

`EEXIST` Der Verweis von *new* ist ein Verzeichnis, welches andere Einträge als '.' und '..' enthält.

`EFAULT` *old* oder *new* zeigen außerhalb des allokierten Adreßbereichs des Prozesses.

`EINVAL` *old* ist ein übergeordnetes Verzeichnis von *new*, oder es wurde versucht, '.' oder '..' umzubenennen.

EINTR — Während der Ausführung des Systemaufrufs rename wurde ein Signal empfangen.

EIO — Beim Anlegen und Aktualisieren eines Verzeichniseintrags tauchte ein E/A-Fehler auf.

EISDIR — *new* zeigt auf ein Verzeichnis, und *old* zeigt auf eine Datei, die kein Verzeichnis ist.

ELOOP — Zu viele symbolische Verweise traten bei der Übersetzung von *old* oder *new* auf.

EMULTIHOP — Komponenten der Pfadnamen erfordern den Sprung auf mehrere ferne Rechner, und die Dateisystemtypen erlauben dies nicht.

ENAMETOOLONG — Die Länge des Arguments *old* oder *new* überschreitet PATH_MAX, oder die Länge einer *old*- oder *new*-Komponente überschreitet NAME_MAX, während _POSIX_NO_TRUNC aktiviert ist.

ENOENT — Eine Komponente von *old* oder *new* existiert nicht, oder die Datei, auf die verwiesen wird, existiert nicht.

ENOLINK — Pfadnamen zeigen auf einen fernen Rechner, und der Verweis auf diesen Rechner ist nicht länger aktiv.

ENOSPC — Im Verzeichnis, welches *new* enthalten soll, ist kein Speicher mehr.

ENOTDIR — Eine Komponente eines Pfadpräfixes ist kein Verzeichnis; oder der Parameter *old* benennt ein Verzeichnis und der Parameter *new* benennt eine Datei.

EROFS — Die angeforderte Operation erfordert das Schreiben in ein Verzeichnis eines nur lesbaren Dateisystems.

EXDEV — Die Verweise von *old* und *new* sind auf verschiedenen Dateisystemen.

ERGEBNIS

Nach erfolgreicher Ausführung wird 0 zurückgegeben, Ansonsten wird -1 zurückgeliefert und errno gesetzt.

HINWEIS

Das System kann einen Deadlock erzeugen, wenn es eine Schleife im Dateisystemgraphen gibt. Solch eine Schleife tritt auf, wenn ein Eintrag im Verzeichnis *a*, z.B. *a/foo*, ein 'hard link' auf das Verzeichnis *b* ist, und ein Eintrag im Verzeichnis *b*, z.B. *b/bar*, ein 'hard link' auf das Verzeichnis *a* ist. Wenn solch eine Schleife existiert und zwei getrennte Prozesse versuchen, ein `rename` *a/foo b/bar* und `rename` *b/bar a/foo* auszuführen, kann das System einen Deadlock erzeugen, weil versucht wird, beide Verzeichnisse zum Ändern zu sperren. Der Systemverwalter sollte konstante Verweise auf Verzeichnisse durch symbolische Verweise ersetzen.

SIEHE AUCH

`link`(2), `unlink`(2).

rmdir - Dateiverzeichnis entfernen

```
#include <unistd.h>
int rmdir(const char *Pfad);
```

`rmdir` entfernt das Dateiverzeichnis, das der Pfadname bezeichnet, auf den *Pfad* zeigt. Das Dateiverzeichnis darf nur die Einträge '.' und '..' enthalten.

Wenn der Verweiszähler des Verzeichnisses 0 wird und kein Prozeß das Verzeichnis geöffnet hat, wird der vom Verzeichnis belegte Speicher freigegeben. Auf das Verzeichnis kann nicht länger zugegriffen werden. Wenn einer oder mehrere Prozesse das Verzeichnis geöffnet haben, während der letzte Verweis entfernt wird, werden die Einträge '.' und '..' entfernt, bevor `rmdir` zurückkehrt. Es können keine neuen Einträge in dem Verzeichnis erzeugt werden, aber das Verzeichnis wird solange nicht entfernt, bis alle Verweise darauf geschlossen worden sind.

Wenn *Pfad* ein symbolischer Verweis ist, wird ihm nicht gefolgt.

Nach erfolgreicher Beendigung kennzeichnet `rmdir` die Felder `st_ctime` und `st_mtime` des übergeordneten Verzeichnisses zur Aktualisierung.

Das angegebene Dateiverzeichnis wird entfernt, sofern nicht einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES`	Eine Komponente des Pfades darf nicht durchsucht werden.
`EACCES`	Die Schreiberlaubnis im Dateiverzeichnis, in dem das zu entfernende Dateiverzeichnis liegt, wird verweigert.
`EACCES`	Im übergeordneten Verzeichnis ist das Sticky-Bit gesetzt, und das übergeordnete Verzeichnis gehört nicht dem Benutzer; das Verzeichnis gehört nicht dem Benutzer und kann vom Benutzer nicht beschrieben werden; der Benutzer ist nicht Systemverwalter.
`EBUSY`	Das zu entfernende Dateiverzeichnis ist der Einhängepunkt für ein eingehängtes Dateisystem.
`EEXIST`	Das Dateiverzeichnis enthält außer '.' und '..' weitere Einträge.
`EFAULT`	*Pfad* weist über den zugewiesenen Adreßraum des Prozesses hinaus.
`EINVAL`	Das Verzeichnis, das entfernt werden soll, ist das aktuelle Dateiverzeichnis.
`EINVAL`	Das Verzeichnis, das entfernt werden soll, ist der '.'-Eintrag eines Dateiverzeichnisses.
`EIO`	Ein E/A-Fehler ist während des Zugriffs auf das Dateisystem aufgetreten.

ELOOP Bei der Übersetzung von *Pfad* wurden zuviele symbolische Verweise angetroffen.

EMULTIHOP Die Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner, und das Dateisystem erlaubt das nicht.

ENAMETOOLONG
Das Argument *Pfad* ist länger als PATH_MAX oder eine Pfadkomponente ist länger als NAME_MAX, während _POSIX_NO_TRUNC aktiv ist.

ENOTDIR Eine Komponente des Pfades ist kein Verzeichnis.

ENOENT Das angegebene Dateiverzeichnis ist nicht vorhanden oder ist der Null-Pfadname.

EROFS Der zu entfernende Dateiverzeichnis-Eintrag ist Teil eines schreibgeschützten Dateisystems.

ENOLINK *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

mkdir(2).
rmdir(1), rm(1) und mkdir(1) in "SINIX V5.41 Kommandos".

semctl - Semaphor-Steuerfunktionen

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

union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
};
int semctl(int semid, int semnum, int cmd, ... /* union semun arg */);
```

`semctl` stellt eine Vielzahl von Semaphor-Steuerfunktionen zur Verfügung, die durch *cmd* angegeben werden.

Nachstehende Kommandos können als *cmd* übergeben werden. Die Kommandos beziehen sich auf das durch *semid* und *semnum* angegebene Semaphor:

`GETVAL` den Wert von `semval` zurückgeben (siehe `intro`(2))

`SETVAL` den Wert von `semval` auf *arg.val* setzen. Wenn dieses Kommando erfolgreich ausgeführt wurde, wird der `semadj`-Wert, der zu dem angegebenen Semaphor gehört, in allen Prozessen gelöscht.

`GETPID` Rückgabe des Wertes von `(int) sempid`

`GETNCNT` Rückgabe des Wertes von `semncnt`

`GETZCNT` Rückgabe des Wertes von `semzcnt`

Mit den folgenden Kommandos wird jeder `semval` in der Semaphor-Menge zurückgegeben bzw. gesetzt:

`GETALL` `semval`s in das Feld schreiben, auf das *arg.array* zeigt

`SETALL` alle `semval`s auf die Werte des Feldes setzen, auf das *arg.array* zeigt. Nach erfolgreicher Ausführung dieses Kommandos werden die `semadj`-Werte, die zu jedem angegebenen Semaphor gehören, in allen Prozessen gelöscht.

Folgende Kommandos sind ebenfalls verfügbar:

`IPC_STAT` den aktuellen Wert jedes Elements der Datenstruktur, die zu *semid* gehört, in die Struktur schreiben, auf die *arg.buf* zeigt. Der Inhalt dieser Struktur wird in `intro`(2) definiert.

IPC_SET — den Wert der folgenden Elemente der Datenstruktur, die zu *semid* gehört, auf den entsprechenden Wert setzen, der in der Struktur gefunden wurde, auf die *arg.buf* zeigt:

```
sem_perm.uid
sem_perm.gid
sem_perm.mode    /* nur Zugangsberechtigungsbits */
```

Dieses Kommando kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer aufweist, die entweder gleich der des Systemverwalters oder dem Wert von `sem_perm.cuid` oder `sem_perm.uid` in der zu *semid* gehörenden Datenstruktur ist.

IPC_RMID — Die durch *semid* angegebene Semaphor-Kennung und die zugehörige Semaphor-Menge und Datenstruktur werden vom System entfernt. Dieses Kommando kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer aufweist, die entweder gleich der des Systemverwalters oder dem Wert von `sem_perm.cuid` oder `sem_perm.uid` in der zu *semid* gehörenden Datenstruktur ist.

`semctl` scheitert, wenn einer oder mehrere der folgenden Punkte zutreffen:

EACCES — Dem aufrufenden Prozeß wird die Ausführungserlaubnis verweigert (siehe `intro`(2)).

EINVAL — *semid* ist keine gültige Semaphor-Kennung.

EINVAL — *semnum* ist kleiner als 0 oder größer als `sem_nsems`.

EINVAL — *cmd* ist kein gültiges Kommando.

EINVAL — *cmd* ist `IPC_SET`, und `sem_perm.uid` oder `sem_perm.gid` ist nicht gültig.

EOVERFLOW — *cmd* ist `IPC_STAT`, und *uid* oder *gid* ist zu groß, um in der Struktur gespeichert zu werden, auf die *arg.buf* zeigt.

ERANGE — *cmd* ist `SETVAL` oder `SETALL`, und der Wert, auf den `semval` gesetzt werden soll, ist größer als der systembedingte zulässige Maximalwert.

EPERM — *cmd* ist gleich `IPC_RMID` oder `IPC_SET`, und die effektive Benutzer-ID des aufrufenden Prozesses ist nicht gleich der des Systemverwalters oder dem Wert von `sem_perm.cuid` oder `sem_perm.uid` in der Datenstruktur, die zu *semid* gehört.

EFAULT — *arg.buf* zeigt auf eine ungültige Adresse.

SIEHE AUCH

intro(2), semget(2), semop(2).

ERGEBNIS

Bei erfolgreicher Beendigung hängt der Rückgabewert folgendermaßen von *cmd* ab:

GETVAL	Wert von semval
GETPID	Wert von (int) sempid
GETNCNT	Wert von semncnt
GETZCNT	Wert von semzcnt
alle anderen	Wert 0

Sonst wird -1 zurückgegeben, und errno wird gesetzt, um den Fehler anzuzeigen.

semget - Semaphor-Kennung bestimmen

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

int semget(key_t key, int nsems, int semflg);
```

semget gibt die zu *key* gehörende Semaphor-Kennung zurück.

Eine Semaphor-Kennung und die zugehörige Datenstruktur sowie die Menge mit *nsems* Semaphoren (siehe intro(2)) werden für *key* erstellt, wenn einer der nachstehenden Punkte wahr ist:

- *key* ist gleich IPC_PRIVATE.
- *key* wurde nicht bereits eine Semaphor-Kennung zugeordnet, und (*semflg*&IPC_CREAT) ist wahr.

Bei Erstellung wird die zur neuen Semaphor-Kennung gehörende Datenstruktur wie folgt initialisiert:

- sem_perm.cui, sem_perm.uid, sem_perm.cgid und sem_perm.gid werden auf die effektive Benutzernummer bzw. die effektive Gruppennummer des aufrufenden Prozesses gesetzt.
- Die Zugriffsberechtigungsbits von sem_perm.mode werden auf Zugriffsberechtigungsbits von *semflg* gesetzt.
- sem_nsems wird auf den Wert von *nsems* gesetzt.
- sem_otime wird auf 0 und sem_ctime auf die aktuelle Zeit gesetzt.

semget ist erfolglos, wenn einer oder mehrere der folgenden Punkte zutreffen:

EINVAL — *nsems* ist entweder kleiner oder gleich Null oder größer als der systembedingte Grenzwert.

EACCES — Eine Semaphor-Kennung ist für *key* vorhanden, jedoch würde die Zugriffserlaubnis (siehe intro(2)), wie sie von den niederwertigen 9 Bits von *semflg* angegeben ist, nicht erteilt werden.

EINVAL — Eine Semaphor-Kennung ist für *key* vorhanden, jedoch ist die Anzahl der Semaphoren in der zugeordneten Semaphor-Menge kleiner als *nsems*, und *nsems* ist nicht gleich Null.

ENOENT — Eine Semaphor-Kennung ist für *key* nicht vorhanden, und (*semflg*&IPC_CREAT) ist unwahr.

ENOSPC Eine Semaphor-Kennung soll erstellt werden, jedoch würde der systembedingte Grenzwert für die Höchstzahl zulässiger Semaphoren im Gesamtsystem überschritten werden.

EEXIST Eine Semaphor-Kennung ist für *key* vorhanden, jedoch sind (*semflg*&IPC_CREAT) und (*semflg*&IPC_EXCL) wahr.

SIEHE AUCH

intro(2), semctl(2), semop(2), stdipc(3C).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nichtnegative ganze Zahl, eine Semaphor-Kennung, zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

semop - Semaphor-Operationen

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

int semop(int semid, struct sembuf *sops, size_t nsops);
```

semop wird zur automatischen Ausführung einer Reihe von Semaphor-Operationen in der zur Semaphor-Kennung gehörenden Semaphor-Menge verwendet, die durch *semid* angegeben wird. *sops* ist ein Zeiger auf das Feld der Strukturen für die Semaphor-Operationen. *nsops* ist die Anzahl dieser Strukturen in dem Feld. Jede Struktur hat folgende Elemente:

```
short sem_num;        /* Semaphor-Nr. */
short sem_op;         /* Semaphor-Operation */
short sem_flg;        /* Schalter für die Operation */
```

Jede durch *sem_op* angegebene Semaphor-Operation wird auf dem entsprechenden Semaphor ausgeführt, welches durch *semid* und *sem_num* angegeben ist.

sem_op gibt eine von drei Semaphor-Operationen wie folgt an (abhängig davon, ob sein Wert negativ, positiv oder Null ist):

Wenn *sem_op* eine negative ganze Zahl ist, läuft einer der nachstehenden Vorgänge ab:

- Wenn der Wert des Semaphors semval (siehe intro(2)) größer oder gleich dem absoluten Wert von *sem_op* ist, wird der absolute Wert von *sem_op* von semval subtrahiert. Wenn (*sem_flg*&SEM_UNDO) wahr ist, wird außerdem der absolute Wert von *sem_op* zum semadj-Wert des aufrufenden Prozesses (siehe exit (2)) für das angegebene Semaphor addiert.
- Wenn semval kleiner als der absolute Wert von *sem_op* und (*sem_flg*&IPC_NOWAIT) wahr ist, kehrt semop sofort zurück.
- Wenn semval kleiner als der absolute Wert von *sem_op* und (*sem_flg*&IPC_NOWAIT) unwahr ist, erhöht semop das zum angegebenen Semaphor gehörende semncnt und hält die Ausführung des aufrufenden Prozesses an, bis eine der nachstehenden Bedingungen eintrifft:
  - semval wird größer oder gleich dem absoluten Wert von *sem_op*. Wenn dies eintritt, wird der Wert des zum angegebenen Semaphor gehörenden semncnt verringert, der absolute Wert von *sem_op* wird von semval subtrahiert und - wenn (*sem_flg*&SEM_UNDO) wahr ist - wird der absolute Wert von *sem_op* zum semad-Wert des aufrufenden Prozesses für das angegebene Semaphor addiert.

- Das *semid*, für das der aufrufende Prozeß Maßnahmen erwartet, wird aus dem System entfernt (siehe semctl(2)). Wenn dies eintritt, wird errno auf EIDRM gesetzt und -1 zurückgegeben.
- Der aufrufende Prozeß erhält ein Signal, das abgefangen werden soll. Wenn dies geschieht, wird der Wert des zum angegebenen Semaphor gehörenden semncnt verringert, und der aufrufende Prozeß nimmt die Ausführung in der in signal(2) beschriebenen Weise wieder auf.

Wenn *sem_op* eine positive ganze Zahl ist, wird der Wert von *sem_op* zu semval addiert, und wenn (*sem_flg*&SEM_UNDO) wahr ist, wird der Wert von *sem_op* vom semadj-Wert des aufrufenden Prozesses für das angegebene Semaphor subtrahiert.

Wenn *sem_op* Null ist, läuft einer der folgenden Vorgänge ab:

- Wenn semval Null ist, kehrt semop sofort zurück.
- Wenn semval ungleich Null ist und (*sem_flg*&IPC_NOWAIT) wahr ist, kehrt semop sofort zurück.
- Wenn semval ungleich Null ist und (*sem_flg*&IPC_NOWAIT) unwahr ist, erhöht semop das zum angegebenen Semaphor gehörige semzcnt und hält die Ausführung des aufrufenden Prozesses an, bis einer der folgenden Fälle eintrifft:
  - semval wird Null. Zu diesem Zeitpunkt wird der Wert des zum angegebenen Semaphor gehörenden semzcnt verringert.
  - Das *semid*, für das der aufrufende Prozeß Maßnahmen erwartet, wird aus dem System entfernt. Wenn dies geschieht, wird errno auf EIDRM gesetzt und -1 zurückgegeben.
  - Der aufrufende Prozeß empfängt ein Signal, das abgefangen werden soll. Wenn dies geschieht, wird der Wert des zum angegebenen Semaphor gehörenden semzcnt verringert, und der aufrufende Prozeß nimmt die Ausführung in der in signal(2) beschriebenen Weise wieder auf.

semop ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte für eine der durch *sops* angegebenen Semaphor-Operationen zutreffen:

EINVAL — *semid* ist keine gültige Semaphor-Kennung.

EFBIG — *sem_num* ist kleiner als Null oder größer als bzw. gleich der Anzahl der Semaphoren in der zu *semid* gehörenden Semaphor-Menge.

E2BIG — *nsops* ist größer als das systembedingte Maximum.

EACCES — Die Zugriffserlaubnis wird dem aufrufenden Prozeß verweigert (siehe *intro*(2)).

EAGAIN — Der aufrufende Prozeß wird angehalten wird, obwohl (*sem_flg*&IPC_NOWAIT) wahr ist.

ENOSPC	Der zulässige Grenzwert für die Anzahl einzelner Prozesse, die ein SEM_UNDO anfordern, wird überschritten.
EINVAL	Die Anzahl individueller Semaphoren, für die der aufrufende Prozeß ein SEM_UNDO anfordert, wird den Grenzwert überschreiten.
ERANGE	semval überschreitet den systembedingten Grenzwert.
ERANGE	Der semadj-Wert überschreitet den systembedingten Grenzwert.
EFAULT	*sops* weist auf eine ungültige Adresse.

Nach erfolgreicher Beendigung wird der Wert von sempid für jedes Semaphor in dem Feld, auf das *sops* zeigt, auf die Prozeßnummer des aufrufenden Prozesses gesetzt.

SIEHE AUCH

intro(2), exec(2), exit(2), fork(2), semctl(2), semget(2).

ERGEBNIS

Wenn *semop* infolge des Empfangs eines Signals zurückkehrt, wird -1 an den aufrufenden Prozeß zurückgegeben, und errno wird auf EINTR gesetzt. Kehrt es infolge der Entfernung eines *semid* zurück, wird -1 zurückgegeben und errno auf EIDRM gesetzt.

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

setpgid - Prozeßgruppennummer einstellen

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

int setpgid(pid_t pid, pid_t pgid);
```

`setpgid` stellt die Prozeßgruppennummer des Prozesses mit der Prozeßnummer *pid* auf *pgid*. Wenn *pgid* gleich *pid* ist, wird der Prozeß zu einem Prozeßgruppenleiter. Wenn *pgid* ungleich *pid* ist, dann wird der Prozeß Mitglied einer existierenden Prozeßgruppe.

Wenn *pid* gleich 0 ist, wird die Prozeßnummer des aufrufenden Prozesses benutzt. Wenn *pgid* gleich 0 ist, wird der Prozeß mit Prozeßnummer *pid* zum Prozeßgruppenleiter.

`setpgid` liefert einen Fehler, wenn eine der folgenden Bedingungen erfüllt ist:

`EACCES` *pid* entspricht der Prozeßnummer eines Sohnprozesses des aufrufenden Prozesses, und der Sohnprozeß hat erfolgreich einen `exec`(2)-Aufruf ausgeführt.

`EINVAL` *pgid* ist kleiner als `(pid_t) 0`, oder größer oder gleich `PID_MAX`.

`EINVAL` Der aufrufende Prozeß verfügt über ein steuerndes Terminal, welches die Job-Kontrolle nicht unterstützt.

`EPERM` Der Prozeß mit der Prozeßnummer *pid* ist ein Sitzungsleiter.

`EPERM` *pid* entspricht der Prozeßnummer eines Sohnprozesses des aufrufenden Prozesses, und der Sohnprozeß ist nicht in derselben Sitzung wie der aufrufende Prozeß.

`EPERM` *pgid* paßt nicht auf die Prozeßnummer *pid*, und es gibt keinen Prozeß mit der Prozeßgruppennummer *pgid* in derselben Sitzung wie der aufrufende Prozeß.

`ESRCH` *pid* entspricht nicht der Prozeßnummer des aufrufenden Prozesses oder eines Sohnprozesses des aufrufenden Prozesses.

SIEHE AUCH

`exec`(2), `exit`(2), `fork`(2), `getpid`(2), `getpgid`(2), `setsid`(2).

ERGEBNIS

Nach erfolgreicher Ausführung liefert `setpgid` den Wert 0 zurück. Ansonsten wird der Wert -1 zurückgegeben, und `errno` wird auf die Fehlernummer gesetzt.

setpgrp - Prozeßgruppennummer einstellen

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

pid_t setpgrp (void);
```

Wenn der aufrufende Prozeß nicht schon ein Sitzungsleiter (session leader) ist, so stellt `setpgrp` die Prozeßgruppennummer und die Sitzungsnummer des aufrufenden Prozesses auf die Prozeßnummer des aufrufenden Prozesses und gibt das steuernde Terminal des aufrufenden Prozesses frei.

SIEHE AUCH

`intro`(2), `exec`(2), `fork`(2), `getpid`(2), `kill`(2), `setsid`(2), `signal`(2).

ERGEBNIS

`setpgrp` gibt den Wert der neuen Prozeßgruppennummer zurück.

HINWEIS

`setpgrp` wird zugunsten der `setsid`(2)-Funktion zurückgestellt.

setsid - Sitzungsnummer einstellen

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

pid_t setsid(void);
```

Wenn der aufrufende Prozeß nicht bereits ein Prozeßgruppenleiter ist, stellt `setsid` die Prozeßgruppennummer und die Sitzungsnummer des aufrufenden Prozesses auf die Prozeßnummer des aufrufenden Prozesses und gibt das steuernde Terminal des Prozesses frei.

`setsid` schlägt fehl und liefert einen Fehler, wenn folgende Bedingung erfüllt ist:

`EPERM` Der aufrufende Prozeß ist bereits Prozeßgruppenleiter, oder es gibt andere Prozesse, deren Prozeßgruppennummer gleich der Prozeßnummer des aufrufenden Prozesses ist.

SIEHE AUCH

`intro`(2), `exec`(2), `exit`(2), `fork`(2), `getpid`(2), `getpgid`(2), `getsid`(2), `setpgid`(2), `setpgrp`, `signal`(2), `sigsend`(2).

HINWEIS

Wenn der aufrufende Prozeß die letzte Komponente einer Pipe ist, die von einer Job-Kontroll-Shell gestartet worden ist, kann die Shell den aufrufenden Prozeß zum Prozeßgruppenleiter machen. Die anderen Prozesse der Pipeline werden Mitglieder der Prozeßgruppe. In diesem Fall schlägt der Aufruf von `setsid` fehl. Aus diesem Grund sollte ein Prozeß, der `setsid` aufruft und davon ausgeht, Teil einer Pipe zu sein, vorher immer ein `fork`(2) ausführen; der Vaterprozeß sollte beendet werden, und der Sohnprozeß sollte `setsid` aufrufen und dadurch versichern, daß der Prozeß zuverlässig funktioniert, ob er nun von Job-Kontroll-Shells aufgerufen wird oder nicht.

ERGEBNIS

Nach erfolgreicher Ausführung liefert `setsid` die Sitzungsnummer des aufrufenden Prozesses zurück. Andernfalls wird -1 zurückgegeben und der Fehlertyp durch `errno` angezeigt.

setuid, setgid - Benutzer- und Gruppennummer einstellen

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

int setuid(uid_t uid);

int setgid(gid_t gid);
```

Der Systemaufruf `setuid` setzt die reale Benutzernummer, die effektive Benutzernummer und die gesicherte Benutzernummer des aufrufenden Prozesses. Der Systemaufruf `setgid` setzt die reale Gruppennummer, die effektive Gruppennummer und die gesicherte Gruppennummer des aufrufenden Prozesses.

Beim Login werden die reale Benutzernummer, die effektive Benutzernummer und die gesicherte Benutzernummer des Login-Prozesses auf die Login-Nummer des Benutzers gesetzt, der für die Erzeugung des Prozesses verantwortlich ist. Dasselbe gilt für die reale, effektive und gesicherte Gruppennummer; sie werden auf die Gruppennummer des Benutzers gesetzt, der für die Erzeugung des Prozesses verantwortlich ist.

Wenn ein Prozeß `exec`(2) aufruft, um eine Datei auszuführen, können sich die Benutzer- und/oder Gruppennummern, die mit dem Prozeß verbunden sind, ändern. Wenn die ausgeführte Datei eine 'set-user-ID'-Datei ist, werden die effektive und gesicherte Benutzernummer des Prozesses auf den Benutzer der ausgeführten Datei gesetzt. Wenn die ausgeführte Datei eine 'set-group-ID'-Datei ist, werden die effektive und gesicherte Gruppennummer des Prozesses auf die Gruppe der ausgeführten Datei gesetzt. Wenn die Datei keine 'set-user-ID'- oder 'set-group-ID'-Datei ist, werden die effektive Benutzernummer, die gesicherte Benutzernummer, die effektive Gruppennummer und die gesicherte Gruppennummer nicht verändert.

`setuid`

Wenn die effektive Benutzernummer des Prozesses, der `setuid` aufruft, der des Systemverwalters entspricht, werden die reale, effektive und gesicherte Benutzernummer auf *uid* gesetzt.

Wenn die effektive Benutzernummer des aufrufenden Prozesses nicht die des Systemverwalters ist, aber *uid* entweder die reale Benutzernummer oder die gesicherte Benutzernummer des aufrufenden Prozesses ist, wird die effektive Benutzernummer auf *uid* gesetzt.

`setgid`

Wenn die effektive Gruppennummer des Prozesses, der `setgid` aufruft, der des Systemverwalters entspricht, werden die reale, effektive und gesicherte Gruppennummer auf *gid* gesetzt.

Wenn die effektive Benutzernummer des aufrufenden Prozesses nicht die des Systemverwalters ist, aber *gid* entweder die reale Gruppennummer oder die gesicherte Gruppennummer des aufrufenden Prozesses ist, wird die effektive Gruppennummer auf *gid* gesetzt.

`setuid` und `setgid` scheitern, wenn eine oder mehrere der folgenden Bedingungen zutreffen:

`EPERM` Bei `setuid`: Wenn die effektive Benutzernummer nicht die des Systemverwalters ist und der Parameter *uid* nicht auf die reale oder gesicherte Benutzernummer paßt.
Bei `setgid`: Wenn die effektive Benutzernummer nicht die des Systemverwalters ist und der Parameter *gid* nicht auf die reale oder gesicherte Gruppennummer paßt.

`EINVAL` *uid* oder *gid* liegen außerhalb des Gültigkeitsbereichs.

ERGEBNIS

Bei erfolgreicher Beendigung wird 0 zurückgegeben. Sonst wird -1 zurückgegeben und `errno` wird gesetzt, um den Fehler anzuzeigen.

SIEHE AUCH

`intro`(2), `exec`(2), `getgroups`(2), `getuid`(2), `stat`(5).

shmctl - Steuerfunktionen für Shared Memory

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

int shmctl (int shmid, int cmd, struct shmid_ds *buf);
```

`shmctl` stellt Steuerfunktionen für gemeinsam benutzte Speichersegmente (Shared Memory) zur Verfügung. Folgende mit *cmd* angegebene Steuerfunktionen stehen zur Verfügung:

`IPC_STAT` Der aktuelle Wert jedes Elements der zu *shmid* gehörenden Datenstruktur wird in die Struktur kopiert, auf die *buf* zeigt. Der Inhalt dieser Struktur ist in `intro`(2) definiert.

`IPC_SET` Die nachstehenden Elemente der zu *shmid* gehörenden Datenstruktur werden auf die Werte der Struktur gesetzt, auf die *buf* zeigt:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode    /* nur Zugangsberechtigungsbits */
```

Diese Funktion kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer hat, die gleich der Nummer des Systemverwalters oder gleich dem Wert von `shm_perm.cuid` oder `shm_perm.uid` in der zu *shmid* gehörenden Datenstruktur ist.

`IPC_RMID` Die durch *shmid* angegebene Shared-Memory-Kennung, das Shared-Memory-Segment, sowie die zugehörige Datenstruktur werden aus dem System entfernt. Diese Funktion kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer hat, die gleich der Nummer des Systemverwalters oder dem Wert von `shm_perm.cuid` oder `shm_perm.uid` in der zu *shmid* gehörenden Datenstruktur ist.

`SHM_LOCK` Das durch *shmid* angegebene Shared-Memory-Segment im Speicher wird gesperrt. Diese Funktion kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer hat, die gleich der Nummer des Systemverwalters ist.

`SHM_UNLOCK` Das durch *shmid* angegebene Shared-Memory-Segment wird freigegeben. Diese Funktion kann nur von einem Prozeß ausgeführt werden, der eine effektive Benutzernummer hat, die gleich der Nummer des Systemverwalters ist.

`shmctl` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EACCES	*cmd* ist gleich `IPC_STAT`, und der aufrufende Prozeß hat keine Lese-Zugriffsberechtigung (siehe `intro`(2)).
EINVAL	*shmid* ist keine gültige Shared-Memory-Kennung.
EINVAL	*cmd* ist keine gültige Steuerfunktion.
EINVAL	*cmd* ist `IPC_SET`, und `shm_perm.uid` oder `shm_perm.gid` sind nicht gültig.
EOVERFLOW	*cmd* ist `IPC_STAT`, wenn *uid* oder *gid* zu groß sind, um in der Struktur gespeichert zu werden, auf die *buf* zeigt.
EPERM	Bei `IPC_RMID` oder `IPC_SET` entspricht die effektive Benutzernummer des aufrufenden Prozesses nicht derjenigen des Systemverwalters, oder sie entspricht nicht dem Wert von `shm_perm.cuid` oder `shm_perm.uid` in der zu *shmid* gehörenden Datenstruktur.
EPERM	Bei `SHM_LOCK` oder `SHM_UNLOCK` entspricht die effektive Benutzernummer des aufrufenden Prozesses nicht derjenigen des Systemverwalters.
EFAULT	*buf* verweist auf eine unzulässige Adresse.
ENOMEM	*cmd* ist `SHM_LOCK`, und es steht nicht ausreichend Speicher zur Verfügung.

SIEHE AUCH

`shmget`(2), `shmop`(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Der Benutzer muß Shared-Memory-Segmente ausdrücklich entfernen, nachdem der letzte Verweis auf diese entfernt worden ist.

shmget - Kennung für Shared Memory bestimmen

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

int shmget(key_t key, int size, int shmflg);
```

`shmget` gibt die Shared-Memory-Kennung zurück, die *key* zugeordnet ist.

Eine Shared-Memory-Kennung und die zugehörige Datenstruktur sowie das Shared-Memory-Segment der Größe von wenigstens *size* Bytes (siehe `intro`(2)) werden für *key* erstellt, wenn einer der folgenden Punkte wahr ist:

- *key* entspricht `IPC_PRIVATE`.
- *key* ist noch keine Shared-Memory-Kennung zugeordnet, und (*shmflg*`&IPC_CREAT`) ist wahr.

Bei der Erstellung wird die Datenstruktur, die der Shared-Memory-Kennung zugeordnet wird, wie folgt initialisiert:

- `shm_perm.cuid`, `shm_perm.uid`, `shm_perm.cgid`, und `shm_perm.gid` werden auf die effektive Benutzernummer bzw. die effektive Gruppennummer des aufrufenden Prozesses gesetzt.
- Die Zugriffsberechtigungsbits von `shm_perm.mode` werden auf die Zugriffsberechtigungsbits von *shmflg* gesetzt. `shm_segsz` wird auf die Größe *size* gesetzt.
- `shm_lpid`, `shm_nattch`, `shm_atime` und `shm_dtime` werden auf 0 gesetzt.
- `shm_ctime` wird auf die aktuellen Zeit gesetzt.

`shmget` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL` Die Größe *size* liegt unter dem systembedingten Minimum oder über dem systembedingten Maximum.

`EACCES` Eine Shared-Memory-Kennung ist für *key* vorhanden, jedoch wird die Zugriffserlaubnis (siehe `intro`(2)), wie sie von den niederwertigen 9 Bits von *shmflg* angegeben wird, nicht gewährt.

`EINVAL` Eine Shared-Memory-Kennung für *key* ist vorhanden, jedoch ist die Größe des zugehörigen Segments kleiner als *size* und *size* ist ungleich Null.

`ENOENT` Eine Shared-Memory-Kennung für *key* ist nicht vorhanden, und (*shmflg*`&IPC_CREAT`) ist unwahr.

ENOSPC Eine Shared-Memory-Kennung soll erstellt werden, jedoch würde die systembedingte maximale Anzahl zugelassener Shared-Memory-Kennungen im gesamten System überschritten werden.

ENOMEM Eine Shared-Memory-Kennung und das zugehörige Speichersegment sollen erstellt werden, jedoch steht zur Erfüllung der Anforderung kein ausreichender Speicherplatz zur Verfügung.

EEXIST Eine Shared-Memory-Kennung ist für *key* vorhanden, jedoch sind (*shmflg*&IPC_CREAT) und (*shmflg*&IPC_EXCL) wahr.

SIEHE AUCH

intro(2), shmctl(2), shmop(2), stdipc(3C).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nichtnegative ganze Zahl, eine Shared-Memory-Kennung, zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Der Benutzer muß Shared-Memory-Segmente ausdrücklich entfernen, nachdem der letzte Verweis auf diese entfernt worden ist.

shmop: shmat, shmdt - Operationen auf Shared Memory

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

void *shmat(int shmid, void *shmaddr, int shmflg);

int shmdt (void *shmaddr);
```

`shmat` fügt das Shared-Memory-Segment (gemeinsam benutzte Speichersegment), das zur Kennung *shmid* gehört, an das Datensegment des aufrufenden Prozesses an. Das Segment wird mit der Adresse verbunden, die durch eine der folgenden Möglichkeiten angegeben wird:

- Wenn *shmaddr* gleich `(void *) 0` ist, wird das Segment mit der ersten verfügbaren Adresse verbunden, die vom System ausgewählt wird.
- Wenn *shmaddr* ungleich `(void *) 0` und (*shmflg*&`SHM_RND`) wahr ist, wird das Segment mit der von (*shmaddr* - (*shmaddr* `modulo SHMLBA`)) angegebenen Adresse verbunden.
- Wenn *shmaddr* ungleich `(void *) 0` und (*shmflg*&`SHM_RND`) unwahr ist, wird das Segment mit der von *shmaddr* angegebenen Adresse verbunden.

`shmdt` entfernt das durch die Adresse *shmaddr* angegebene Shared-Memory-Segment vom Datensegment des aufrufenden Prozesses.

Das Segment wird zum Lesen angefügt, wenn (*shmflg*&`SHM_RDONLY`) wahr ist andernfalls wird es zum Lesen und Schreiben angefügt

`shmat` ist erfolglos und macht das Shared-Memory-Segment nicht verfügbar, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL` *shmid* ist keine gültige Shared-Memory-Kennung.

`EACCES` Die Zugriffserlaubnis wird dem aufrufenden Prozeß verweigert (siehe `intro`(2)).

`ENOMEM` Der verfügbare Platz im Datensegment reicht für die Aufnahme des Shared-Memory-Segments nicht aus.

`EINVAL` *shmaddr* ist ungleich Null, und der Wert von (*sshmaddr* - (*shmaddr* `modulo SHMLBA`)) ist eine unzulässige Adresse.

`EINVAL` *shmaddr* ist ungleich Null, (*shmflg*&`SHM_RND`) ist unwahr und der Wert von *shmaddr* ist eine unzulässige Adresse.

`EMFILE` Die Anzahl der Shared-Memory-Segmente, die dem aufrufenden Prozeß zugeteilt sind, würde den systembedingten Grenzwert überschreiten.

EINVAL — shmdt ist erfolglos und entfernt das Shared-Memory-Segment nicht, wenn *shmaddr* nicht die Datensegment-Startadresse eines Shared-Memory-Segments ist.

SIEHE AUCH

intro(2), exec(2), exit(2), fork(2), shmctl(2), shmget(2).

ERGEBNIS

Nach erfolgreicher Beendigung ist der Rückgabewert wie folgt:

shmat — gibt die Datensegment-Startadresse des angefügten Shared-Memory-Segmentes zurück.

shmdt — gibt 0 zurück.

Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige auf den Fehlerwert gesetzt.

HINWEIS

Der Benutzer muß Shared-Memory-Segmente ausdrücklich entfernen, nach dem der letzte Verweis auf diese entfernt worden ist.

sigaction - Signal-Aktionen untersuchen/ändern

```
#include <signal.h>
int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);
```

Mit `sigaction` kann geprüft und bestimmt werden, welche Aktion nach der Auslösung eines bestimmten Signals ausgeführt werden soll. (Siehe `signal`(5) zur Erklärung der allgemeinen Signalkonzepte)

sig gibt das Signal an und kann irgendein Signal aus `signal`(5), mit Ausnahme von `SIGKILL` und `SIGSTOP`, enthalten.

Wenn das Argument *act* nicht `NULL` ist, dann handelt es sich um einen Zeiger auf eine Struktur, welche die neue Aktion angibt, die nach dem Empfang des Signals *sig* ausgeführt werden soll. Wenn das Argument *oact* nicht `NULL` ist, zeigt es auf eine Struktur, in der die Aktion, die zuvor mit *sig* verknüpft wurde, nach Rückkehr von `sigaction` gespeichert wird.

Die Struktur `sigaction` enthält die folgenden Komponenten:

```
void        (*sa_handler)();
sigset_t    sa_mask;
int         sa_flags;
```

`sa_handler` gibt die Disposition des Signals an und kann einen der Werte aus dem Abschnitt `signal`(5) enthalten.

`sa_mask` gibt eine Menge von Signalen an, die blockiert werden sollen, während die Signalbehandlungsroutine aktiv ist. Bei Einsprung in die Signalbehandlungsroutine wird die Signalmenge der Menge der bereits blockierten Signale hinzugefügt. Außerdem wird das Signal, welches die Signalbehandlungsroutine aktiviert hat, ebenfalls blockiert, es sei denn, es wurde die Option `SA_NODEFER` angegeben. `SIGSTOP` und `SIGKILL` können nicht blockiert werden (das System erzwingt diese Beschränkung stillschweigend).

`sa_flags` gibt eine Menge von Optionen an, welche die Übertragung des Signals beeinflußt. Die folgenden Werte können durch bitweises ODER verknüpft werden:

`SA_ONSTACK` Wenn diese Option gesetzt ist und das Signal behandelt wird und ein alternativer Signalstapel mit `sigaltstack`(2) deklariert wurde, wird das Signal über den Stapel an den aufrufenden Prozeß übertragen. Ansonsten wird das Signal auf dem Stapel des Hauptprogramms übertragen.

SA_RESETHAND
: Wenn diese Option gesetzt ist und das Signal behandelt wird, wird die Disposition des Signals auf SIG_DFL zurückgesetzt und das Signal bei Einsprung in die Signalbehandlungsroutine blockiert (SIGILL, SIGTRAP, und SIGPWR können nicht automatisch zurückgesetzt werden, wenn sie empfangen werden; das System erzwingt diese Beschränkung stillschweigend).

SA_NODEFER
: Das Signal wird vom Kernel nicht automatisch blockiert, während es von der Signalbehandlungsroutine bearbeitet wird.

SA_RESTART
: Wenn diese Option gesetzt ist und das Signal behandelt wird, wird ein Systemaufruf, der durch die Ausführung der Signalbehandlungsroutine unterbrochen wird, vom System neu gestartet. Dies geschieht transparent. Ansonsten liefert der Systemaufruf den Fehler EINTR zurück.

SA_SIGINFO
: Wenn diese Option nicht gesetzt ist und das Signal behandelt wird, wird *sig* als einziges Argument an die Funktion gesendet, welche die Signale abfängt. Wenn die Option gesetzt ist und das Signal behandelt wird, werden blockierte Signale vom Typ *sig* zuverlässig für den aufrufenden Prozeß in die Warteschlange aufgenommen und zwei zusätzliche Argumente an die Funktion übergeben, die das Signal bearbeitet. Wenn das zweite Argument nicht gleich NULL ist, zeigt es auf eine Struktur vom Typ siginfo_t, welche den Grund für das Signal enthält (siehe siginfo(5)); das dritte Argument zeigt auf eine Struktur vom Typ ucontext_t, welche den Kontext des empfangenden Prozesses zur Zeit des Signalempfangs enthält (siehe ucontext(5)).

SA_NOCLDWAIT
: Ist diese Option gesetzt und *sig* ist gleich SIGCHLD, dann erzeugt das System keine Zombie-Prozesse, wenn Sohnprozesse des aufrufenden Prozesses beendet werden. Führt der aufrufende Prozeß aufeinanderfolgende wait(2)-Aufrufe aus, wird blockiert, bis alle Sohnprozesse des aufrufenden Prozesses beendet sind; danach wird der Wert -1 zurückgegeben und errno enthält ECHILD.

SA_NOCLDSTOP
: Ist diese Option gesetzt und ist *sig* gleich SIGCHLD, wird *sig* nicht an den aufrufenden Prozeß gesendet, wenn Sohnprozesse beendet werden oder weiterlaufen.

sigaction schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

EINVAL
: Der Wert des Arguments *sig* ist keine gültige Signalnummer oder ist gleich SIGKILL oder SIGSTOP.

EFAULT
: *act* oder *oact* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

ERGEBNIS

Bei Erfolg wird von `sigaction` Null zurückgeliefert. Bei Fehler wird -1 zurückgegeben und `errno` zeigt den Fehler an.

SIEHE AUCH

`intro`(2), `exit`(2), `kill`(2), `pause`(2), `sigaltstack`(2), `signal`(2), `sigprocmask`(2), `sigsend`(2), `sigsuspend`(2), `wait`(2), `sigsetops`(3C), `siginfo`(5), `signal`(5), ucontext(5). `kill`(1) in "SINIX V5.41 Kommandos".

HINWEIS

Wenn der Systemaufruf von einem Terminal liest oder schreibt und das `NOFLSH`-Bit des Terminals gelöscht ist, können Datenpuffer geleert werden (siehe `termio`(7)).

sigaltstack - alternativen Stapelkontext eines Signals setzen/lesen

```
#include <signal.h>
int sigaltstack(const stack_t *ss, stack_t *oss);
```

Mit `sigaltstack` wird ein alternativer Stapelbereich definiert, in dem Signale bearbeitet werden können. Wenn *ss* ungleich Null ist, wird ein Zeiger auf einen Stapelbereich erwartet, auf dem die Signale bearbeitet werden können; dem System wird mitgeteilt, ob der Prozeß gerade mit dem Stapel arbeitet. Wenn die Aktion des Signals anzeigt, daß mit dem alternativen Stapel gearbeitet werden soll (angegeben durch `sigaction`(2)), prüft das System, ob der Prozeß momentan auf dem alternativen Stapel abläuft. Operiert der Prozeß nicht auf dem Signalstapel, schaltet das System für die Dauer der Ausführung der Signalbehandlungsroutine auf den alternativen Signalstapel um.

Die Struktur `sigaltstack` enthält die folgenden Komponenten:

```
int     *ss_sp
long    ss_size
int     ss_flags
```

Ist *ss* nicht `NULL`, wird ein Zeiger auf den alternativen Signalstapel erwartet, welcher nach Rückkehr von `sigaltstack` wirksam wird. Die Komponenten `ss_sp` und `ss_size` bestimmen die Basis und die Größe des Stapels, welche automatisch an die Wachstumsrichtung und Wertigkeit angepaßt wird. Die Komponente `ss_flags` gibt den Zustand des neuen Stapels an und kann die folgenden Optionen aufweisen:

`SS_DISABLE` Der Stapel wird deaktiviert und `ss_sp` und `ss_size` werden ignoriert. Wenn `SS_DISABLE` nicht gesetzt ist, wird der Stapel aktiviert.

Ist *oss* nicht `NULL`, wird ein Zeiger auf eine Struktur erwartet, welche den alternativen Signalstapel bezeichnet, der vor dem Aufruf von `sigaltstack` aktiv war. `ss_sp` und `ss_size` geben die Basis und die Größe des Stapels an. Die `ss_flags`-Komponente gibt den Zustand des Stapels an. Dieser Zustand kann die folgenden Werte annehmen:

`SS_ONSTACK` Der Prozeß wird momentan mit dem alternativen Signalstapel ausgeführt. Versuche, den alternativen Signalstapel während der Ausführung des Prozesses zu ändern, schlagen fehl.

`SS_DISABLE` Der alternative Signalstapel ist momentan deaktiviert.

`sigaltstack` schlägt fehl, wenn eine der folgenden Bedingungen gegeben ist:

`EFAULT` *ss* oder *oss* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

EINVAL Es wurde versucht, einen aktiven Stapel zu deaktivieren, oder die Komponente ss_flags enthält ungültige Optionen.

ENOMEM Die Größe des alternativen Stapelbereichs ist kleiner als MINSIGSTKSZ.

HINWEIS

Der Wert SIGSTKSZ stellt die Anzahl der Bytes dar, welche im allgemeinen für einen alternativen Stapelbereich notwendig sind. Der Wert MINSIGSTKSZ definiert dabei die minimale Stapelgröße für eine Signalbehandlungsroutine. Bei der Berechnung der Stapelgröße sollte das Programm noch diesen Minimalwert zusätzlich anlegen, um den Eigenbedarf des Betriebssystems zu berücksichtigen.

Der folgende Programmauszug wird dazu verwendet, um einen alternativen Stapelbereich zu allokieren:

```
if ((sigstk.ss_sp = (char *)malloc(SIGSTKSZ)) == NULL)
      /* Fehlerbehandlung */;

sigstk.ss_size = SIGSTKSZ;
sigstk.ss_flags = 0;
if (sigaltstack(&sigstk, (stack_t *)0) < 0)
      perror("sigaltstack");
```

SIEHE AUCH

getcontext(2), sigaction(2), sigsetjmp(3C), ucontext(5).

ERGEBNIS

Bei Erfolg liefert sigaltstack 0. Bei Fehler wird -1 zurückgegeben, und errno enthält die Fehlernummer.

signal - vereinfachte Signalverwaltung

```
#Include <signal.h>
void (*signal(int sig, void (*disp)(int)))(int);
void (*sigset(int sig, void (*disp)(int)))(int);
int sighold(int sig);
int sigrelse(int sig);
int sigignore(int sig);
int sigpause(int sig);
```

Diese Funktionen erlauben Applikationsprozessen das vereinfachte Verwalten von Signalen. Unter Abschnitt `signal`(5) finden Sie eine generelle Beschreibung der Signalkonzeption.

`signal` und `sigset` werden verwendet, um Signalbehandlungen zu verändern. *sig* gibt dabei das Signal an, welches jedes außer `SIGKILL` und `SIGSTOP` sein darf. *disp* definiert die Behandlung des Signals, welche `SIG_DFL`, `SIG_IGN` oder die Adresse einer Signalbehandlungsroutine sein darf. Wird `signal` verwendet, ist *disp* die Adresse einer Signalbehandlungsroutine, und *sig* muß ungleich `SIGILL`, `SIGTRAP` und `SIGPWR` sein. Das System stellt zuerst die Signalbehandlung auf `SIG_DFL`, bevor die Signalbehandlungsroutine aufgerufen wird. Wird `sigset` verwendet und ist *disp* die Adresse einer Signalbehandlungsroutine, fügt das System das Signal *sig* der Signalmaske des aufrufenden Prozesses hinzu, bevor die Signalbehandlungsroutine ausgeführt wird. Ist die Ausführung der Signalbehandlungsroutine beendet, stellt das System die Signalmaske des aufrufenden Prozesses wieder auf den Zustand, der vor dem Empfang des Signals herrschte. Wird `sigset` benutzt und ist *disp* gleich `SIG_HOLD`, so wird *sig* zur Signalmaske des aufrufenden Prozesses hinzugefügt, und die Signalbehandlung bleibt unverändert.

`sighold` fügt *sig* der Signalmaske des aufrufenden Prozesses hinzu.

`sigrelse` entfernt *sig* von der Signalmaske des aufrufenden Prozesses.

`sigignore` stellt die Behandlung von *sig* auf `SIG_IGN`.

`sigpause` entfernt *sig* von der Signalmaske des aufrufenden Prozesses und deaktiviert den aufrufenden Prozeß, bis ein Signal empfangen wird.

Diese Funktionen schlagen fehl, wenn eine der folgenden Bedingungen erfüllt ist:

`EINVAL` Der Wert des Arguments *sig* ist kein gültiges Signal oder ist gleich `SIGKILL` oder `SIGSTOP`.

`EINTR` Während des Systemaufrufs `sigpause` wurde ein Signal empfangen.

HINWEIS

`sighold` in Verbindung mit `sigrelse` oder `sigpause` kann dazu verwendet werden, um kritische Programmbereiche zu erstellen, in denen der Empfang eines Signals zeitweilig abgeschaltet wird.

Wenn `signal` oder `sigset` verwendet wird, um für die Behandlung `SIGCHLD` eine Signalbehandlungsroutine einzustellen, wird `SIGCHLD` nicht gesendet, wenn die Sohnprozesse des aufrufenden Prozesses gestoppt oder weiterbearbeitet werden.

Wird eine der obigen Funktionen verwendet, um die Behandlung von `SIGCHLD` auf `SIG_IGN` zu setzen, so erzeugen die Sohnprozesse des aufrufenden Prozesses keine Zombie-Prozesse, wenn sie beendet werden (siehe `exit`(2)). Wenn der aufrufende Prozeß nacheinander auf seine Sohnprozesse wartet, blockiert er, bis alle seine Sohnprozesse terminiert sind; dann wird ein Wert von -1 zurückgeliefert, und `errno` enthält die Fehlernummer `ECHILD` (siehe `wait`(2), `waitid`(2)).

ERGEBNIS

Bei Erfolg liefert `signal` die vorherige Behandlung des Signals zurück. Bei Fehler wird `SIG_ERR` geliefert und die Fehlernummer in `errno` abgelegt.

Bei Erfolg liefert `sigset` `SIG_HOLD`, wenn das Signal blockiert wurde oder die vorherige Behandlung, wenn es nicht blockiert wurde. Bei Fehler wird `SIG_ERR` zurückgegeben und `errno` enthält die entsprechende Fehlernummer.

Alle anderen Funktionen liefern bei Erfolg Null zurück. Bei Fehler liefern sie -1 und setzen `errno`.

SIEHE AUCH

`kill`(2), `pause`(2), `sigaction`(2), `sigsend`(2), `wait`(2), `waitid`(2), `signal`(5).

sigpending - blockierte und wartende Signale prüfen

```
#include <signal.h>
int sigpending(sigset_t *set);
```

Die Funktion `sigpending` überprüft Signale, welche an den aufrufenden Prozeß gesendet wurden, jedoch durch die Signalmaske des aufrufenden Prozesses ignoriert werden. Die Signale werden in dem Speicher abgelegt, auf den das Argument *set* zeigt.

`sigpending` schlägt fehl, wenn folgende Bedingung erfüllt ist:

EFAULT *set* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

SIEHE AUCH

`sigaction`(2), `sigprocmask`(2), `sigsetops`(3C).

ERGEBNIS

Bei Erfolg liefert `sigpending` 0 zurück. Bei Fehler wird -1 zurückgegeben und `errno` enthält die Fehlernummer.

sigprocmask - Signalmaske ändern oder testen

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

Die Funktion `sigprocmask` wird verwendet, um die Signalmaske des aufrufenden Prozesses zu überprüfen und/oder zu ändern. Wenn der Wert `SIG_BLOCK` ist, wird die Menge, auf die das Argument *set* zeigt, der aktuellen Signalmaske hinzugefügt. Wenn der Wert `SIG_UNBLOCK` ist, wird die Menge, auf die das Argument *set* zeigt, aus der aktuellen Signalmaske entfernt. Ist der Wert `SIG_SETMASK`, wird die aktuelle Signalmaske durch die Menge ersetzt, auf die das Argument *set* zeigt. Ist das Argument *oset* nicht `NULL`, wird die vorherige Maske an der Adresse gespeichert, auf die *oset* zeigt. Enthält *set* `NULL`, ist der Wert *how* nicht bedeutsam und die Signalmaske des Prozesses bleibt unverändert; hiermit kann der Aufruf verwendet werden, um die momentan blockierten Signale in Erfahrung zu bringen.

Gibt es auf Verarbeitung wartende, unblockierte Signale nach dem Aufruf von `sigprocmask`, so wird mindestens eines dieser Signale empfangen, bevor der Aufruf von `sigprocmask` beendet wird.

Es ist nicht möglich, Signale zu blockieren, die nicht ignoriert werden können (siehe `sigaction`(2)); diese Beschränkung wird vom System stillschweigend erzwungen.

Wenn der Aufruf von `sigprocmask` fehlschlägt, wird die Signalmaske des Prozesses nicht verändert.

`sigprocmask` schlägt fehl, wenn eine der folgenden Bedingungen wahr ist:

`EINVAL` Der Wert des Arguments *how* entspricht nicht den gültigen definierten Werten.

`EFAULT` *set* oder *oset* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

SIEHE AUCH

`sigaction`(2), `signal`(2), `sigsetopts`(3C), `signal`(5).

ERGEBNIS

Bei Erfolg wird von `sigprocmask` 0 zurückgegeben. Bei Fehler wird -1 zurückgegeben und `errno` auf die Fehlernummer gesetzt.

sigsend, sigsendset - Signal an Prozeß/Prozeßgruppe senden

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

int sigsend(idtype_t idtype, id_t id, int sig);

int sigsendset(procset_t *psp, int sig);
```

`sigsend` sendet ein Signal an einen Prozeß oder an eine Prozeßgruppe, die durch *id* und *idtype* angegeben wird. Das zu sendende Signal wird durch *sig* angegeben und ist entweder Null oder einer der unter `signal`(5) definierten Werte. Ist *sig* gleich Null (das Nullsignal), geschieht eine Fehlerüberprüfung, aber es wird kein Signal gesendet. Dies dient dazu, um die Gültigkeit von *id* und *idtype* zu überprüfen.
Die reale oder effektive Benutzernummer des sendenden Prozesses muß mit der realen oder effektiven Benutzernummer des empfangenden Prozesses übereinstimmen, es sei denn, die effektive Benutzernummer des sendenden Prozesses ist die des Systemverwalters, oder *sig* ist `SIGCONT` und der sendende Prozeß hat dieselbe Sitzungsnummer wie der empfangende Prozeß.

Die möglichen Werte für *idtype* haben folgende Auswirkung:

`P_PID` *sig* wird an den Prozeß mit der Nummer *id* gesendet.

`P_PGID` *sig* wird an jeden Prozeß mit der Prozeßgruppennummer *id* gesendet.

`P_SID` *sig* wird an jeden Prozeß mit der Sitzungsnummer *id* gesendet.

`P_UID` *sig* wird an jeden Prozeß mit der effektiven Benutzernummer *id* gesendet.

`P_GID` *sig* wird an jeden Prozeß mit der effektiven Gruppennummer *id* gesendet.

`P_CID` *sig* wird an jeden Prozeß mit der Verwaltungsklassennummer *id* gesendet (siehe `priocntl`(2)).

`P_ALL` *sig* wird an alle Prozesse gesendet, und *id* wird ignoriert.

`P_MYID` der Wert *id* wird vom aufrufenden Prozeß übernommen.

Der Prozeß mit der Prozeßnummer 0 ist immer ausgeschlossen. Der Prozeß mit der Prozeßnummer 1 wird ausgeschlossen, es sei denn *idtype* ist gleich `P_PID`.

`sigsendset` bietet eine alternative Schnittstelle zum Senden von Signalen an Prozeßmengen. Diese Funktion sendet Signale an Prozeßmengen, die durch *psp* angegeben werden. *psp* ist dabei ein Zeiger auf eine Struktur vom Typ `procset_t`, die in der Datei `sys/procset.h>` definiert wird und die folgenden Komponenten enthält:

```
idop_t      p_op;
idtype_t    p_lidtype;
id_t        p_lid;
idtype_t    p_ridtype;
id_t        p_rid;
```

`p_lidtype` und `p_lid` geben den ID-Typ und die ID einer ('linken') Prozeßmenge an; `p_ridtype` und `p_rid` geben den ID-Typ und die ID einer ('rechten') Prozeßmenge an. ID-Typen und IDs werden genau wie bei den Argumenten *idtype* und *id* im Systemaufruf `sigsend` angegeben. `p_op` gibt die Operation an, welche mit den beiden Prozeßmengen ausgeführt werden soll, um eine Prozeßmenge zu erhalten, auf die schließlich der Systemaufruf angewendet wird. Die gültigen Werte für `p_op` und die angegebenen Prozesse sind:

`POP_DIFF` Differenz: Prozesse in der linken und nicht in der rechten Menge

`POP_AND` Schnittmenge: Prozesse in der linken und rechten Menge

`POP_OR` Vereinigungsmenge: Prozesse in der linken oder rechten oder in beiden Mengen

`POP_XOR` Exklusiv-ODER: Prozesse in der linken oder der rechten Menge, aber nicht in beiden.

`sigsend` und `sigsendset` schlagen fehl, wenn wenigstens eine der folgenden Bedingungen gegeben ist:

`EINVAL` *sig* ist keine gültige Signalnummer.

`EINVAL` *idtype* enthält keinen gültigen Feldwert.

`EINVAL` *sig* ist `SIGKILL`, *idtype* ist gleich `P_PID` und *id* ist 1 (proc1).

`ESRCH` Es konnten keine Prozesse gefunden werden, welche *id* und *idtype* entsprechen.

`EPERM` Die Benutzernummer des sendenden Prozesses ist nicht die des Systemverwalters, und die reale oder effektive Benutzernummer entspricht nicht der realen oder effektiven Benutzernummer des empfangenden Prozesses, und der aufrufende Prozeß sendet nicht das Signal `SIGCONT` an einen Prozeß aus derselben Sitzung.

`sigsendset` schlägt außerdem fehl:

`EFAULT` *psp* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

SIEHE AUCH

`getpid`(2), `getpgrp`(2), `kill`(2), `priocntl`(2), `setpid`(2), `signal`(2), `signal`(5).
`kill`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Bei Erfolg gibt `sigsend` 0 zurück. Bei Fehler wird -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

sigsuspend - Signalmaske installieren und Prozeß deaktivieren

```
#include <signal.h>
int sigsuspend(const sigset_t *set);
```

`sigsuspend` ersetzt die Signalmaske des Prozesses durch die Signalmaske, auf die *set* zeigt; danach wird der Prozeß solange deaktiviert, bis ein Signal empfangen wird, das eine Signalbehandlungsroutine ausführt oder den Prozeß beendet.

Wenn die Aktion den Prozeß terminiert, kehrt `sigsuspend` nicht zurück. Wird eine Signalbehandlungsroutine aufgerufen, kehrt `sigsuspend` nach der Ausführung der Signalbehandlungsroutine zurück. Nach Rückkehr wird die Signalmaske auf den ursprünglichen Wert vor Ausführung von `sigsuspend` zurückgesetzt.

Es ist nicht möglich, Signale zu blockieren, die nicht ignoriert werden können (siehe `signal`(5)); diese Einschränkung wird vom System stillschweigend erzwungen.

`sigsuspend` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EINTR` Vom aufrufenden Prozeß wird ein Signal abgefangen, und die Kontrolle wird von der Signalbehandlungsroutine zurückgegeben.

`EFAULT` *set* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

ERGEBNIS

Da `sigsuspend` die Prozeßausführung bis auf weiteres anhält, gibt es keinen Rückgabewert, der die erfolgreiche Ausführung anzeigen könnte. Bei Fehler wird -1 zurückgegeben und `errno` gesetzt.

SIEHE AUCH

`sigaction`(2), `sigprocmask`(2), `sigpause`(2), `sigsetops`(3C), `signal`(5).

stat, lstat, fstat - Dateistatus abrufen

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *Pfad, struct stat *Puffer);
int lstat(const char *Pfad, struct stat *Puffer);
int fstat(int dk, struct stat *Puffer);
```

Pfad weist auf einen Pfadnamen, der eine Datei benennt. Eine Lese-, Schreib- oder Ausführungserlaubnis für die angegebene Datei ist nicht erforderlich, jedoch müssen alle im zur Datei führenden Pfadnamen aufgelisteten Dateiverzeichnisse durchsuchbar sein. `stat` erhält Informationen für die angegebene Datei.

Es ist zu beachten, daß bei gemeinsamer Dateibenutzung auf einem fernen Rechner (RFS) die von `stat` zurückgegebenen Informationen jeweils von der Einstellung der Benutzer-/Gruppenabbildung zwischen lokalem und fernem Rechner abhängen.

`lstat` liefert genau wie `stat` Dateiattribute. Nur wenn die angegebene Datei ein symbolischer Verweis ist, gibt `lstat` Informationen über den Verweis aus, während `stat` Informationen über die Datei ausgibt, auf die sich der Verweis bezieht.

`fstat` liefert Informationen über eine offene Datei mit einem Dateideskriptor *dk*, der von einem erfolgreichen Systemaufruf `open`, `creat`, `dup`, `fcntl` oder `pipe` geliefert wird.

Puffer ist ein Zeiger auf eine `stat`-Struktur, in die die jeweilige Datei betreffenden Informationen geschrieben werden.

Zum Inhalt der Struktur, auf die von *Puffer* gewiesen wird, gehören folgende Elemente:

```
mode_t    st_mode;    /* Dateimodus (siehe mknod(2)) */
ino_t     st_ino;     /* Dateikennziffer (i-Node) */
dev_t     st_dev;     /* Gerätekennung, die einen
                         Verzeichniseintrag für diese Datei enthält */
dev_t     st_rdev;    /* Gerätekennung, nur für zeichen- oder
                         blockorientierte Gerätedateien definiert */
nlink_t   st_nlink;   /* Anzahl der Verweise */
uid_t     st_uid;     /* Benutzerkennung des Dateibesitzers */
gid_t     st_gid;     /* Gruppenkennung des Dateibesitzers */
off_t     st_size;    /* Dateigröße in Bytes */
time_t    st_atime;   /* Zeit des letzten Zugriffs */
time_t    st_mtime;   /* Zeit der letzten Datenänderung */
time_t    st_ctime;   /* Zeit der letzten Änderung des Dateistatus
                         Die Zeit wird in Sekunden gemessen ab dem
                         1. Januar 1970, 00:00:00 Uhr */
long      st_blksize; /* Bevorzugte E/A-Blockgröße */
long      st_blocks;  /* Anzahl zugewiesener st_blksize-Blöcke */
```

`st_mode` Der Modus der Datei ist im Systemaufruf `mknod`(2) beschrieben. Zusätzlich zu den in `mknod`(2) beschriebenen Modi, kann der Modus einer Datei auch `S_IFLNK` sein, wenn die Datei ein symbolischer Verweis ist. (Beachten Sie, daß `S_IFLNK` nur von `lstat` zurückgegeben werden kann.)

`st_ino` kennzeichnet die Datei im gegebenen Dateisystem eindeutig. Das Paar `st_ino` und `st_dev` kennzeichnet normale Dateien eindeutig.

`st_dev` kennzeichnet das Dateisystem, in dem die Datei liegt, eindeutig. Der Wert kann als Eintrag für den Systemaufruf `ustat`(2) zur Bestimmung von weiteren Informationen über dieses Dateisystem verwendet werden.

`st_rdev` darf nur von Verwaltungskommandos benutzt werden. Es ist nur für block- oder zeichenorientierte Dateien gültig und hat nur in dem System eine Bedeutung, in dem die Datei eingereicht wurde.

`st_nlink` darf nur von Verwaltungskommandos benutzt werden.

`st_uid` Benutzernummer des Eigentümers der Datei.

`st_gid` Gruppennummer der Gruppe, der die Datei zugeordnet ist.

`st_size` Für normale Dateien ist dies die Größe der Datei in Bytes. Für block- oder zeichenorientierte Dateien ist dieses nicht definiert. Siehe auch `pipe`(2).

`st_atime` Uhrzeit, zu der zuletzt auf die Dateidaten zugegriffen wurde. Wird von folgenden Systemaufrufen geändert: `creat`, `mknod`, `pipe`, `utime` und `read`.

`st_mtime` Uhrzeit, zu der Daten zuletzt geändert wurden. Wird von folgenden Systemaufrufen geändert: `creat`, `mknod`, `pipe`, `utime` und `write`.

`st_ctime` Dies ist die Uhrzeit, zu der der Dateistatus zuletzt geändert wurde. Wird von folgenden Systemaufrufen geändert: `chmod`, `chown`, `creat`, `link`, `mknod`, `pipe`, `unlink`, `utime` und `write`.

`st_blksize` Ein Hinweis auf die 'beste' Größe einer Einheit für E/A-Operationen. Dieses Feld ist für block- oder zeichenorientierte Gerätedateien nicht definiert.

`st_blocks` Die Gesamtanzahl von physikalischen Blöcken der Größe 512 Bytes, die zur Zeit auf der Platte belegt sind. Dieses Feld ist für block- oder zeichenorientierte Gerätedateien nicht definiert.

`stat` und `lstat` sind erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Eine Komponente des Pfades darf nicht durchsucht werden.

`EFAULT` *Puffer* oder *Pfad* weisen auf eine ungültige Adresse.

`EINTR` Ein Signal wurde während des Systemaufrufs `stat` oder `lstat` abgefangen.

`ELOOP` Bei der Übersetzung von *Pfad* wurden zuviele symbolische Verweise angetroffen.

`EMULTIHOP` Die Komponenten von *Pfad* erfordern den Sprung auf mehrere ferne Rechner, und das Dateisystem erlaubt das nicht.

ENOENT Die angegebene Datei ist nicht vorhanden oder ist der Null-Pfadname.

ENOTDIR Eine Komponente des Pfades ist kein Dateiverzeichnis.

ENOLINK *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EOVERFLOW Eine Komponente ist zu groß, um in der Struktur, auf die *Puffer* zeigt, gespeichert zu werden.

fstat ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EBADF *dk* ist kein gültiger offener Dateideskriptor.

EFAULT *Puffer* weist auf eine ungültige Adresse.

EINTR Ein Signal wurde während des Systemaufrufs `fstat` abgefangen.

ENOLINK *dk* weist auf einen fernen Rechner , und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EOVERFLOW Eine Komponente ist zu groß, als daß sie in der Struktur, auf die *Puffer* zeigt, gespeichert werden könnte.

SIEHE AUCH

`chmod`(2), `chown`(2), `creat`(2), `link`(2), `mknod`(2), `pipe`(2), `read`(2), `time`(2), `unlink`(2), `utime`(2), `write`(2), `fattach`(3C), `stat`(5).

ERGEBNIS

Nach erfolgreicher Beendigung wird ein Wert von 0 zurückgegeben. Andernfalls wird ein Wert von -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

statvfs, fstatvfs - Dateisysteminformationen lesen

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

int statvfs (const char *path, struct statvfs *buf);

int fstatvfs (int fildes, struct statvfs *buf);
```

`statvfs` liefert einen 'generischen Superblock', der ein Dateisystem beschreibt; dieser kann dazu verwendet werden, um Informationen über eingehängte Dateisysteme zu bekommen. *buf* ist ein Zeiger auf eine Struktur, die weiter unten beschrieben wird. Diese Struktur wird während des Systemaufrufs beschrieben.

path sollte der Name einer Datei sein, welche sich in dem entsprechenden Dateisystem befindet. Der Dateisystemtyp ist dabei dem Betriebssystem bekannt. Lese-, Schreib- oder Ausführungsrechte für die angegebene Datei werden nicht benötigt, jedoch muß für jedes Verzeichnis aus dem Pfadnamen Sucherlaubnis vorhanden sein.

Die Struktur `statvfs`, auf die *buf* zeigt, enthält die folgenden Komponenten:

```
ulong   f_bsize;            /* bevorzugte Blockgröße des Dateisystems */
ulong   f_frsize;           /* grundlegende Blockgröße des Dateisystems
                               (falls unterstützt) */
ulong   f_blocks;           /* gesamte Anzahl der Blöcke auf dem
                               Dateisystem in Einheiten von f_frsize */
ulong   f_bfree;            /* gesamte Anzahl der freien Blöcke */
ulong   f_bavail;           /* Anzahl der verfügbaren freien Blöcke
                               für einen Nicht-Systemverwalter */
ulong   f_files;            /* gesamte Anzahl der Dateien (Inodes) */
ulong   f_ffree;            /* gesamte Anzahl der freien Knoten */
ulong   f_favail;           /* Anzahl der Inodes für einen
                               Nicht-Systemverwalter*/
fsid_t  f_fsid;             /* Dateisystemnummer (momentan dev) */
char    f_basetype[FSTYPSZ]; /* Typname des Zieldateisystems, nullterminiert */
ulong   f_flag;             /* Bitmaske der Optionen */
ulong   f_namemax;          /* maximale Länge der Dateinamen */
char    f_fstr[32];         /* Dateisystemspezifische Zeichenkette */
ulong   f_filler[16];       /* reserviert für zukünftige Erweiterungen */
```

`f_basetype` enthält einen nullterminierten Typnamen des Dateisystems (FST-Name) über das eingehängte Ziel (z.B. `s5` über `rfs` eingehängt resultiert in `s5`).

Die folgenden Werte können im `f_flag`-Feld zurückgeliefert werden:

```
ST_RDONLY     0x01    /* nur lesbares Dateisystem */
ST_NOSUID     0x02    /* setuid/setgid Semantik wird nicht unterstützt */
ST_NOTRUNC    0x04    /* schneidet Dateinamen länger als NAME_MAX nicht ab */
```

`fstatvfs` ist ähnlich `statvfs`, außer daß der Dateiname *path* aus `statvfs` über einen offenen Dateideskriptor *fildes* identifiziert wird, der aus einem erfolgreichen `open`-, `creat`-, `dup`-, `fcntl`- oder `pipe`-Systemaufruf resultiert.

`statvfs` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EACCES` Sucherlaubnis existiert für eine Komponente des Pfadpräfixes nicht.

`EFAULT` *path* oder *buf* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während der Ausführung von `statvfs` empfangen.

`EIO` Beim Lesen des Dateisystems trat ein E/A-Fehler auf.

`ELOOP` Zu viele symbolische Verweise traten bei der Übersetzung von *path* auf.

`EMULTIHOP` Komponenten von *path* erfordern den Sprung auf einen fernen Rechner, und die Dateisystemtypen erlauben dies nicht.

`ENAMETOOLONG`
Die Länge einer *path*-Komponente überschreitet `NAME_MAX`-Zeichen, oder die Länge von *path* überschreitet `PATH_MAX`-Zeichen.

`ENOENT` Entweder eine Komponente des Pfadpräfixes oder die Datei, die durch *path* bezeichnet wird, existiert nicht.

`ENOLINK` *path* zeigt auf einen fernen Rechner, und der Verweis auf diesen Rechner ist nicht länger aktiv.

`ENOTDIR` Eine Komponente des Pfadpräfixes von *path* ist kein Verzeichnis.

`fstatvfs` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EFAULT` *buf* zeigt auf eine ungültige Adresse.

`EBADF` *fildes* ist kein geöffneter Dateideskriptor.

`EINTR` Ein Signal wurde während der Ausführung von `fstatvfs` empfangen.

`EIO` Beim Lesen des Dateisystems trat ein E/A-Fehler auf.

SIEHE AUCH

`chmod`(2), `chown`(2), `creat`(2), `link`(2), `mknod`(2), `pipe`(2), `read`(2), `time`(2), `unlink`(2), `utime`(2), `write`(2).

ERGEBNIS

Nach erfolgreicher Ausführung wird 0 zurückgegeben. Ansonsten wird der Wert -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

stime - Uhrzeit einstellen

```
#include <unistd.h>
int stime(const time_t *tp);
```

Mit `stime` werden Zeit und Datum für das System gesetzt. *tp* weist auf den Wert der Zeit, gemessen in Sekunden seit 00:00:00 UTC am 1. Januar 1970.

`stime` scheitert, wenn:

`EPERM` die effektive Benutzernummer des aufrufenden Prozesses nicht der Systemverwalter ist.

SIEHE AUCH

`time`(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

swapctl - Swap-Bereich verwalten

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

int swapctl(int cmd, void *arg);
```

`swapctl` addiert, löscht oder liefert Informationen über Swap-Ressourcen. *cmd* gibt dabei eine der folgenden Optionen an, die in der Datei `sys/swap.h` definiert werden:

```
SC_ADD                  /* Ressource als Swap-Bereich einrichten */
SC_LIST                 /* Die Swap-Ressourcen auflisten */
SC_REMOVE               /* Eine Swap-Ressource entfernen */
SC_GETNSWP              /* Die Anzahl der Swap-Ressourcen
                           zurückgeben */
```

Wenn `SC_ADD` oder `SC_REMOVE` angegeben wird, ist *arg* ein Zeiger auf eine `swapres`-Struktur, welche die folgenden Komponenten enthält:

```
char    *sr_name;       /* Pfadname der Ressourcen */
off_t   sr_start;       /* Offset, bei dem der Swap-Bereich beginnt */
off_t   sr_length;      /* Länge des Swap-Bereiches */
```

`sr_start` und `sr_length` werden in 512-Byte Blöcken angegeben.

Wenn `SC_LIST` angegeben wird, ist *arg* ein Zeiger auf eine `swaptable`-Struktur, welche die folgenden Komponenten enthält:

```
int     swt_n;          /* Anzahl der Swap-Ressourcen */
struct  swapent swt_ent[]; /* Feld der swt_n Swap-Ressourcen */
```

Eine `swapent`-Struktur enthält die folgenden Komponenten:

```
char    *ste_path;      /* Name der Swap-Datei */
off_t   ste_start;      /* Anfangsblock zum Swappen */
off_t   ste_length;     /* Länge des Swap-Bereichs */
long    ste_pages;      /* Anzahl der Seiten zum Swappen */
long    ste_free;       /* Anzahl von ste_pages frei */
long    ste_flags;      /* ST_INDEL Bit gesetzt, wenn Swap-Datei */
                        /* nun gelöscht wird */
```

`SC_LIST` veranlaßt `swapctl` dazu, `swt_n` Einträge zurückzugeben. Der Rückgabewert von `swapctl` ist die Anzahl der wirklich zurückgelieferten Einträge. Das `ST_INDEL`-Bit wird in `ste_flags` angeschaltet, wenn die Swap-Datei gerade gelöscht wird.

Wenn `SC_GETNSWP` angegeben wird, liefert `swapctl` als Wert die Anzahl der Swap-Ressourcen zurück, die momentan verwendet werden. *arg* wird für diese Operation ignoriert.

Die `SC_ADD`- und `SC_REMOVE`-Funktionen schlagen fehl, wenn der aufrufende Prozeß nicht die entsprechenden Privilegien besitzt.

Unter den folgenden Bedingungen schlägt die Funktion swapctl fehl und setzt errno auf die folgenden Werte:

EEXIST — Ein Teil des durch sr_start und sr_length angegebenen Bereichs auf der angegebenen Ressource wird bereits als Swap-Bereich verwendet (SC_ADD).

EFAULT — *arg*, sr_name, oder ste_path weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EINVAL — Der angegebene Funktionswert ist nicht gültig, der angegebene Pfad ist keine Swap-Ressource (SC_REMOVE), Teile des Bereichs, der durch sr_start und sr_length angegeben wird, liegen außerhalb der angegebenen Ressource (SC_ADD), oder der angegebene Swap-Bereich ist kleiner als eine Seite (SC_ADD).

EISDIR — Der für SC_ADD angegebene Pfad ist ein Verzeichnis.

ELOOP — Zu viele symbolische Verweise traten bei der Übersetzung des Pfadnamens für SC_ADD oder SC_REMOVE auf.

ENAMETOOLONG — Die Länge einer Komponenten im Pfad für SC_ADD oder SC_REMOVE überschreitet NAME_MAX Zeichen, oder die Länge des Pfads überschreitet PATH_MAX Zeichen und _POSIX_NO_TRUNC ist wirksam.

ENOENT — Der Pfadname für SC_ADD oder SC_REMOVE existiert nicht.

ENOMEM — Eine unzureichende Anzahl von struct-swapent-Strukturen wurde für SC_LIST bereitgestellt, oder es stehen unzureichende Systemspeicherressourcen während einer SC_ADD- oder SC_REMOVE-Operation zur Verfügung, oder das System hätte nach einer SC_REMOVE-Operation einen zu kleinen Swap-Bereich.

ENOSYS — Der Pfadname für SC_ADD oder SC_REMOVE ist keine Datei oder keine blockorientierte Gerätedatei.

ENOTDIR — Der Pfadname für SC_ADD oder SC_REMOVE enthält eine Komponente, bei der es sich nicht um ein Verzeichnis handelt.

EPERM — Der Prozeß hat keine entsprechenden Privilegien.

EROFS — Der Pfadname für SC_ADD gibt ein Dateisystem an, das nur lesbar ist.

ERGEBNIS

Nach erfolgreicher Durchführung liefert die Funktion swapctl den Wert 0 für SC_ADD oder SC_REMOVE, die Anzahl der struct swapent-Einträge für SC_LIST, oder die Anzahl der Swap-Ressourcen für SC_GETNSWP zurück. Bei Fehlern wird -1 zurückgegeben und die Fehlernummer in errno abgelegt.

symlink - symbolischen Verweis auf eine Datei erstellen

```
#include <unistd.h>
int symlink(const char *name1, const char *name2);
```

symlink erzeugt den symbolischen Verweis *name2* auf die Datei *name1*. Beide Namen dürfen beliebige Pfadnamen sein, die beiden Dateien müssen sich nicht auf demselben Dateisystem befinden, und *name1* muß nicht vorhanden sein.

Die Datei, auf die der symbolische Verweis zeigt, wird verwendet, wenn mit dem Verweis eine open(2)-Operation durchgeführt wird. Ein stat(2) auf einen symbolischen Verweis liefert die verwiesene Datei, während ein lstat Informationen über den Verweis selbst liefert. Dies kann zu überraschenden Ergebnissen führen, wenn ein symbolischer Verweis auf ein Verzeichnis erzeugt wird. Um in Programmen diese ungewünschten Effekte zu reduzieren, kann der Aufruf readlink(2) verwendet werden, um den Inhalt eines symbolischen Verweises zu lesen.

Der symbolische Verweis wird nicht hergestellt, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

EACCES Die Sucherlaubnis für eine Komponente des Pfadpräfixes von *name2* existiert nicht.

EDQUOT Das Verzeichnis, in dem sich der Eintrag für den neuen symbolischen Verweis befinden soll, kann nicht erweitert werden, da die maximale Anzahl der Plattenblöcke des Benutzers für das Dateisystem überschritten wurde.

EDQUOT Der neue symbolische Verweis kann nicht erzeugt werden, da die maximale Anzahl der Plattenblöcke des Benutzers für das Dateisystem, welches den Verweis enthalten soll, überschritten wurde.

EDQUOT Die maximale Anzahl von Indexeinträgen des Benutzers auf dem Dateisystem, auf dem die Datei erzeugt werden soll, wurde überschritten.

EEXIST Die Datei, die durch *name2* angegeben wurde, existiert bereits.

EFAULT *name1* oder *name2* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

EIO Beim Lesen oder Schreiben des Dateisystems trat ein E/A-Fehler auf.

ELOOP Zu viele symbolische Verweise traten beim Übersetzen von *name2* auf.

ENAMETOOLONG
Die Länge des Arguments *name1* oder *name2* überschreitet PATH_MAX, oder die Länge einer Komponente von *name1* oder *name2* überschreitet NAME_MAX, während _POSIX_NO_TRUNC wirksam ist.

ENOENT	Eine Komponente des Pfadnamenpräfixes von *name2* existiert nicht.
ENOSPC	Das Verzeichnis, in dem der Eintrag für den neuen symbolischen Verweis erzeugt werden soll, kann nicht erweitert werden, da auf dem Dateisystem des Verzeichnisses kein Speicherplatz mehr frei ist.
ENOSPC	Der neue symbolische Verweis kann nicht erzeugt werden, da kein Speicherplatz mehr auf dem Dateisystem verfügbar ist, welches den Verweis enthalten soll.
ENOSPC	Es gibt keine freien Indexeinträge auf dem Dateisystem, auf dem die Datei erzeugt werden soll.
ENOSYS	Das Dateisystem unterstützt keine symbolischen Verweise.
ENOTDIR	Eine Komponente des Pfadpräfixes von *name2* ist kein Verzeichnis.
EROFS	Die Datei *name2* würde sich auf einem Dateisystem befinden, welches nur lesbar ist.

SIEHE AUCH

`link`(2), `readlink`(2), `unlink`(2).
`cp`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Ausführung liefert `symlink` den Wert 0 zurück; ansonsten wird -1 zurückgegeben und der Fehlercode in `errno` geschrieben.

sync - Superblock aktualisieren

```
#include <unistd.h>
void sync(void);
```

`sync` bewirkt das Herausschreiben aller Daten im Speicher, die auf Platte/Diskette geschrieben werden sollen aber noch im Hauptspeicher gehalten werden. Hierzu gehören auch geänderte Superblöcke, geänderte Indexeinträge und verzögerte blockorientierte-E/A-Dateien.

`sync` soll von Programmen benutzt werden, die ein Dateisystem prüfen, beispielsweise `fsck`(1M) oder `df`(1M). `sync` ist vor einem Neuladen des Systems obligatorisch.

Das Schreiben ist bei Rückkehr von `sync` nicht unbedingt schon beendet. Der Systemaufruf `fsync` beendet das Schreiben vor der Rückkehr.

SIEHE AUCH

`fsync`(2).

sysfs - Information über Dateisystemtyp abfragen

```
#include <sys/fstyp.h>
#include <sys/fsid.h>

int sysfs(int opcode, const char *fsname);

int sysfs(int opcode, int fs_index, char *buf);

int sysfs(int opcode);
```

`sysfs` gibt Informationen über die im System konfigurierten Dateisystemtypen zurück. Die Anzahl der von `sysfs` akzeptierten Argumente hängt vom Wert *opcode* ab. Die gegenwärtig akzeptierten *opcodes* und ihre jeweiligen Funktionen sind:

`GETFSIND` übersetzt *fsname*, einen mit dem Null-Byte abgeschlossenen Dateisystemnamen, in einen Index der Dateisystemtypen.

`GETFSTYP` übersetzt *fs_index*, einen Index der Dateisystemtypen, in einen mit dem Null-Byte abgeschlossenen Dateisystemnamen und schreibt diesen in den Puffer, auf den *buf* zeigt; dieser Puffer muß eine Größe von wenigstens `FSTYPSZ` aufweisen, wie in `sys/fstyp.h` definiert.

`GETNFSTYP` gibt die Gesamtzahl der im System konfigurierten Dateisystemtypen zurück.

`sysfs` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL` *fsname* weist auf einen ungültigen Dateisystemnamen; *fs_index* ist Null oder ungültig; *opcode* ist ungültig.

`EFAULT` *buf* oder *fsname* weisen über den zugewiesenen Adreßraum des Prozesses hinaus.

ERGEBNIS

Nach erfolgreicher Beendigung gibt `sysfs` den Index des Dateisystemtyps zurück, wenn der *opcode* `GETFSIND` ist, 0, wenn *opcode* `GETFSTYP` ist, oder die Anzahl der konfigurierten Dateisystemtypen, wenn *opcode* `GETNFSTYP` ist. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

sysi86 - rechnerspezifische Funktionen

```
#include <sys/sysi86.h>
int sysi86 (int cmd, ...);
```

`sysi86` realisiert rechnerspezifische Funktionen. *cmd* bestimmt die auszuführende Funktion. Der Typ des Arguments hängt von der Funktion ab.

Kommando RTODC

Wenn *cmd* `RTODC` ist, wird ein Zeiger auf eine `struct rtc_t` (wie in der Include-Datei `sys/rtc.h` definiert) als Argument erwartet:

```
struct        rtc_t {
     char rtc_sec, rtc_asec, rtc_min, rtc_amin,
     rtc_hr, rtc_ahr, rtc_dow, rtc_dom,
     rtc_mon, rtc_yr, rtc_statusa,
     rtc_statusb, rtc_statusc, rtc_statusd;
};
```

Diese Funktion liest die Hardware-Uhr und gibt die Daten in der Struktur zurück, auf die vom Argument verwiesen wird. Es ist zu beachten, daß dieses Kommando nur dem Systemverwalter zur Verfügung steht.

Kommando RDUBLK

Dieses Kommando liest den 'u-Block' (Die Informationen, die durch *structuser* in der Include-Datei `sys/user` für jeden Prozeß definiert sind) für einen gegebenen Prozeß. Wenn *cmd* `RDUBLK` ist, benötigt `sysi86` drei zusätzliche Argumente: die Prozeßnummer, eine Bereichsadresse und die Anzahl der zu lesenden Zeichen, also:

```
sysi86(RDUBLK, pid, buf, n)
    pid_t  pid;
    char *buf;
    int  n;
```

Kommando SI86FPHW

Dieses Kommando erwartet die Adresse auf eine ganze Zahl (Integer) als Argument. Nach der Rückkehr von diesem Systemaufruf ist die referierte Zahl auf einen Wert gesetzt, der angibt, auf welche Weise Gleitkommazahlen unterstützt werden.

Das niederwertige Byte der Zahl enthält den Wert der Variable *fpkind*, die angibt, ob ein 80287 oder ein 80387-Coprozessor vorhanden ist, von der Software emuliert oder nicht unterstützt wird. Die Werte sind in der Include-Datei `sys/fp.h` definiert.

```
FP_NO       kein Gleitkommaprozessor, kein Emulator (nicht unterstützt)
FP_SW       kein Gleitkommaprozessor, Software-Emulation
FP_HW       Gleitkommaprozessor vorhanden
FP_287      80287 vorhanden
FP_387      80387 vorhanden
```

Kommando SETNAME

Dieses Kommando kann nur vom Systemverwalter ausgeführt werden. Es erwartet ein Argument vom Typ *char* *, das auf einen mit dem Nullzeichen abgeschlossenen Text von höchstens sieben Zeichen Länge weist. Das Kommando ändert *sysname* und *nodename* (siehe `uname`(2)) des laufenden Systems auf den übergebenen Text.

Kommando STIME

Wenn *cmd* STIME ist, wird ein Argument vom Typ `long` erwartet. Mit dieser Funktion werden Uhrzeit und Datum des Systems gesetzt (nicht die Hardware-Uhr). Das Argument enthält die Zeit, in Sekunden gemessen ab 00:00:00 GMT am 1. Januar 1970. Es ist zu beachten, daß dieses Kommando nur dem Systemverwalter zur Verfügung steht.

Kommando SI86DSCR

Dieses Kommando definiert einen Segment- oder Gatedeskriptor im Systemkern. Die folgenden Arten von Deskriptoren werden akzeptiert:

- Programm- und Datensegmente in der LDT auf DPL 3
- ein call gate in der GDT auf DPL 3, der auf ein Segment in der LDT zeigt.

Das Argument ist ein Zeiger auf eine Anforderungsstruktur mit den Werten, die in den Deskriptor geschrieben werden sollen. Die Anforderungsstruktur ist in der Include-Datei `sys/sysi86.h` definiert.

Kommando SI86MEM

Dieses Kommando gibt die Größe des verfügbaren Speichers zurück.

Kommando SI86SWPI

Wenn *cmd* SI86SWPI ist, können individuelle Austauschbereiche hinzugefügt oder gelöscht bzw. die aktuellen Bereiche bestimmt werden. Die Adresse eines angemessenen vorbereiteten Swap-Bereichs wird als einziges Argument weitergereicht.
(Einzelheiten zum Laden des Bereichs sind in der Include-Datei `sys/swap.h` zu finden.)

Das Format des Swap-Bereichs ist wie folgt:

```
struct swapint {
    char si_cmd;     /*Kommando: SI_LIST, SI_ADD, SI_DEL*/
    char *si_buf;    /*swap-Datei Pfadzeiger*/
    int si_swplo;    /*erster Block*/
    int si_nblks;    /*Größe*/
 }
```

Es ist zu beachten, daß die Optionen Hinzufügen und Löschen des Kommandos nur vom Systemverwalter angewendet werden dürfen.

Normalerweise wird ein Swap-Bereich durch einen einzelnen Aufruf von `sysi86` hinzugefügt. Zunächst wird der Swap-Bereich mit entsprechenden Einträgen für die Strukturelemente vorbereitet. Dann wird `sysi86` aufgerufen.

```
#include <sys/sysi86.h>
#include <sys/swap.h>

struct swapint swapbuf; /*swap buffer- Zeiger */

sysi86(SI86SWPI, &swapbuf);
```

Das Kommando ist nicht erfolgreich, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EFAULT`	*swapbuf* weist auf eine ungültige Adresse
`EFAULT`	*swapbuf.si_buf* weist auf eine ungültige Adresse
`ENOTBLK`	Angegebener Swap-Bereich ist keine blockorientierte Gerätedatei
`EEXIST`	Angegebener Swap-Bereich wurde bereits zugefügt
`ENOSPC`	Zu viele Swap-Bereiche im Einsatz (beim Zufügen)
`ENOMEM`	Löschen des letzten Swap-Bereichs wurde versucht
`EINVAL`	Ungültige Argumente
`ENOMEM`	Kein Platz für ausgetauschte Seiten, wenn ein Swap-Bereich gelöscht wird

SIEHE AUCH

`uname`(2).
`swap`(1M) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird ein Wert von -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt. Wenn *cmd* ungültig ist, wird `errno` bei Rückkehr auf `EINVAL` gesetzt.

sysinfo - Systeminformationen lesen und schreiben

```
#include <sys/systeminfo.h>
long sysinfo (int command, char *buf, long count);
```

`sysinfo` kopiert Informationen über das SINIX-System, auf dem der Prozeß abläuft, in den Puffer *buf*; `sysinfo` kann außerdem bestimmte Systeminformationen einstellen, wenn entsprechende *command*s verfügbar sind. *count* ist dabei die Größe des Puffers.

Die verfügbaren Kommandoargumente sind:

`SI_SYSNAME` setzt den Inhalt von *buf* als Systemnamen; dieser Wert wird von `uname` (siehe `uname`(2)) im Feld `sysname` zurückgegeben. Dies ist der Name der Implementierung des Betriebssystems, z.B. `System V` oder `UTS`.

`SI_HOSTNAME`
kopiert den Host-Namen nach *buf*. Dieser Wert wird von `uname` (siehe `uname`(2)) im Feld `nodename` zurückgeliefert. Dieser Host- oder Knotenname ist oft der lokale Rechnername.

Der Hostname ist der Rechnername als Knoten in einem Netzwerk; verschiedene Netzwerke können verschiedene Namen für den Knoten besitzen; jedoch sollte als Knotennamen für das entsprechende Netzwerkverzeichnis oder an die Namen-/Adreß-Abbildung eine Adresse für einen Transportendpunkt weitergegeben werden.

Internet-Hostnamen können bis zu 256 Bytes lang sein (plus terminierendes Nullzeichen).

`SI_SET_HOSTNAME`
Der Inhalt von *buf* wird vom Kernel in ein Zeichenfeld kopiert, dessen Inhalt von nachfolgenden Aufrufen von `sysinfo` mit dem Kommando `SI_HOSTNAME` zurückgeliefert wird. Dieses Kommando setzt voraus, daß die effektive Benutzernummer die des Systemverwalters ist.

`SI_RELEASE` verwaltet den Inhalt von *buf*, so daß er von `uname` (siehe `uname`(2)) im Feld `version` zurückgegeben wird. Typische Werte sind `4.0` oder `3.2`.

`SI_VERSION` verwaltet den Inhalt von *buf*, so daß er von `uname` (siehe `uname`(2)) im Feld `release` zurückgegeben wird. Die Syntax und Semantik dieser Zeichenkette wird vom Systemlieferanten definiert.

`SI_MACHINE` kopiert den Inhalt von *buf*, so daß er von `uname` (siehe `uname`(2)) im Feld `machine` zurückgegeben wird (z.B. `3b2` oder `580`).

SI_ARCHITECTURE
: verwaltet den Inhalt von *buf*, der die Befehlssatzarchitektur des aktuellen Systems beschreibt, wie zum Beispiel `mc68030`, `m32100` oder `i80486`. Diese Namen dürfen nicht vordefinierten Namen des C-Übersetzungssystems entsprechen.

SI_HW_PROVIDER
: kopiert den Inhalt der Hardwarebeschreibung in das Feld *buf*.

SI_HW_SERIAL
: *buf* enthält die ASCII-Darstellung der hardwarespezifischen Seriennummer der physikalischen Maschine, auf der der Systemaufruf ausgeführt wird. Beachten Sie, daß diese Option über nur-lesbaren Speicher, über Programmkonstanten beim Erzeugen des Betriebssystems oder auf andere Weise implementiert werden kann und nichtnumerische Zeichen enthalten kann. Es wird vorausgesetzt, daß vom Hersteller nicht dieselbe Seriennummer an mehr als eine physikalische Maschine vergeben wird.

SI_SRPC_DOMAIN
: kopiert den Bereichsnamen des SRPC (Secure Remote Procedure Call) in das Feld *buf*.

SI_SET_SRPC_DOMAIN
: definiert die Zeichenkette, welche von `sysinfo` durch das Kommando `SI_SRPC_DOMAIN` in das Feld *buf* kopiert wird. Dieses Kommando setzt voraus, daß die effektive Benutzernummer diejenige des Systemverwalters ist.

`sysinfo` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen wahr ist:

EPERM
: Der Prozeß hat nicht die entsprechenden Privilegien für ein SET-Kommando.

EINVAL
: *buf* zeigt nicht auf eine gültige Adresse, oder die Daten für ein SET-Kommando überschreiten die Grenzen der Implementierung.

In vielen Fällen gibt es keine entsprechende Programmatik dieser Schnittstelle zum Setzen der Werte; solche Zeichenketten sind typischerweise nur durch den Systemverwalter einstellbar, der die Einträge im Verzeichnis `master.d` ändert oder den Code eines OEM's verwendet, um die Seriennummer zu lesen, oder Code aus nur-lesbarem Speicher oder festcodiertem Speicher verwendet.

Eine guter Wert für *count* ist 257; dadurch werden wahrscheinlich alle Zeichenketten verarbeitet, welche von dieser Schnittstelle auf typischen Installationen zurückgeliefert werden.

ERGEBNIS

Nach erfolgreicher Ausführung wird ein Wert zurückgegeben, der die benötigte Pufferlänge in Bytes angibt. Diese Pufferlänge gilt für die komplette, zurückgegebene Zeichenkette einschließlich des Nullzeichens. Ist dieser Wert nicht größer als der Wert *count*, wurde die komplette Zeichenkette übertragen. Ist der Wert größer als *count*, wurde die Zeichenkette, die nach *buf* kopiert wurde, auf *count-1* Bytes plus ein Nullzeichen gekürzt.

Ansonsten zeigt -1 einen Fehler an; `errno` enthält dann die Fehlernummer.

SIEHE AUCH

`uname`(2), `sysconf`(2).

termios - allgemeine Terminalschnittstelle

```
#include <termios.h>
int tcgetattr(int fildes, struct termios *termios_p);
int tcsetattr(int fildes, int optional_actions, const struct termios *termios_p);
int tcsendbreak(int fildes, int duration);
int tcdrain(int fildes);
int tcflush(int fildes, int queue_selector);
int tcflow(int fildes, int action);
speed_t cfgetospeed(struct termios *termios_p);
int cfsetospeed(const struct termios *termios_p, speed_t speed);
speed_t cfgetispeed(struct termios *termios_p);
int cfsetispeed(const struct termios *termios_p, speed_t speed);
#include <sys/types.h>
#include <termios.h>
pid_t tcgetpgrp(int fildes);
int tcsetpgrp(int fildes, pid_t pgid);
pid_t tcgetsid(int fildes);
```

Diese Funktionen beschreiben eine allgemeine Terminalschnittstelle zur Kontrolle asynchroner Kommunikationsanschlüsse. Eine detailliertere Übersicht der Terminalschnittstelle kann im Abschnitt `termio`(7) gefunden werden; dort wird außerdem eine `ioctl`(2)-Schnittstelle beschrieben, welche dieselbe Funktionalität besitzt. Trotz allem ist die hier beschriebene Schnittstelle die bevorzugte.

Viele der hier beschriebenen Funktionen verwenden ein *termios_p*-Argument, welches den Zeiger auf eine `termios`-Struktur darstellt. Diese Struktur enthält die folgenden Komponenten:

```
tcflag_t    c_iflag;        /* Eingabemodi */
tcflag_t    c_oflag;        /* Ausgabemodi */
tcflag_t    c_cflag;        /* Kontrollmodi */
tcflag_t    c_lflag;        /* Lokale Modi */
cc_t        c_cc[NCCS];     /* Kontrollzeichen */
```

Diese Strukturkomponenten werden im Abschnitt `termio`(7) genau beschrieben.

Terminalattribute lesen und schreiben

Die Funktion `tcgetattr` liest die Parameter, welche mit dem Objekt *fildes* verknüpft sind, und speichert sie in der Struktur `termios`, welche durch *termios_p* bezeichnet wird. Diese Funktion kann von einem Hintergrundprozeß aufgerufen werden; die Terminaleigenschaften können danach von einem Vordergrundprozeß geändert werden.

Die Funktion `tcsetattr` stellt die mit dem Terminal verbundenen Parameter ein (es sei denn, die zugrundeliegende Hardware bietet nicht die benötigte Unterstützung). Die Parameter werden der Struktur `termios` entnommen, welche durch *termios_p* wie folgt bezeichnet wird:

- Enthält das Feld *optional_actions* den Wert `TCSANOW`, erfolgt die Änderung sofort.
- Ist *optional_actions* gleich `TCSADRAIN`, erfolgt die Änderung, nachdem alle Ausgaben für *fildes* übertragen wurden. Diese Funktion sollte verwendet werden, wenn Parameter geändert werden, welche Ausgaben beeinträchtigen.
- Wenn *optional_actions* `TCSAFLUSH` enthält, erfolgt die Änderung, nachdem alle Ausgaben an das Objekt *fildes* übertragen wurden; alle Eingaben, die empfangen aber nicht gelesen wurden, werden vor der Änderung verworfen.

Die symbolischen Konstanten für die Werte aus *optional_actions* werden in der Datei `termios.h` definiert.

Zeilenkontrolle

Wenn das Terminal die asynchrone serielle Datenübertragung verwendet, verursacht die Funktion `tcsendbreak` die Übertragung eines ununterbrochenen Stroms von Null-Bits für eine bestimmte Dauer. Ist *duration* Null, wird die Übertragung von Null-Bits für mindestens 0.25 Sekunden und nicht mehr als 0.5 Sekunden aktiviert. Ist *duration* nicht Null, verhält es sich ähnlich wie bei `tcdrain`.

Verwendet das Terminal nicht die asynchrone serielle Datenübertragung, sendet die Funktion `tcsendbreak` Daten, um eine Break-Bedingung zu generieren, oder es wird keine Aktion ausgeführt.

Die Funktion `tcdrain` wartet, bis alle Ausgaben an das Objekt *fildes* übertragen wurden.

Die Funktion `tcflush` verwirft alle Daten, die auf das Objekt *fildes* geschrieben, aber nicht übertragen wurden, oder Daten, die empfangen, aber nicht gelesen wurden, abhängig vom Wert *queue_selector*:

- Ist *queue_selector* gleich `TCIFLUSH`, werden Daten gelöscht, welche empfangen und nicht gelesen wurden.
- Enthält *queue_selector* den Wert `TCOFLUSH`, werden Daten gelöscht, welche geschrieben, aber nicht übertragen wurden.

- Wenn *queue_selector* gleich TCIOFLUSH ist, werden sowohl empfangene, nicht gelesene Daten als auch geschriebene, nicht übertragene Daten gelöscht.

Die Funktion tcflow hält die Übertragung oder den Empfang von Daten für das Objekt *fildes* an, abhängig vom Wert *action*:

- Ist *action* gleich TCOOFF, wird die Ausgabe angehalten.
- Ist *action* gleich TCOON, wird die angehaltene Ausgabe wieder gestartet.
- Wenn *action* gleich TCIOFF ist, überträgt das System ein STOP-Zeichen, welches das Terminal dazu veranlaßt, die Datenübertragung zum System zu stoppen.
- Wenn *action* gleich TCION ist, überträgt das System ein START-Zeichen, welches das Terminal dazu veranlaßt, die Datenübertragung an das System zu beginnen.

Baud-Rate lesen und setzen

Die Funktionen zum Lesen und Setzen der Ein- und Ausgabebaudraten verarbeiten die Werte aus der Struktur termios. Die Auswirkungen auf das Terminalgerät, welche weiter unten beschrieben werden, werden nicht wirksam, bis die Funktion tcsetattr erfolgreich ausgeführt wird.

Die Eingabe- und Ausgabebaudraten werden in der Struktur termios abgelegt. Die in der Tabelle abgebildeten Übertragungsraten werden unterstützt. Die Namen aus der Tabelle werden in der Datei termios.h definiert:

Name	Beschreibung	Name	Beschreibung
B0	Verbindungsabbruch	B600	600 Baud
B50	50 Baud	B1200	1200 Baud
B75	75 Baud	B1800	1800 Baud
B110	110 Baud	B2400	2400 Baud
B134	134.5 Baud	B4800	4800 Baud
B150	150 Baud	B9600	9600 Baud
B200	200 Baud	B19200	19200 Baud
B300	300 Baud	B38400	38400 Baud

cfgetospeed liest die Ausgabebaudrate und speichert sie in der termios-Struktur, auf die *termios_p* zeigt.

cfsetospeed setzt die Ausgabebaudrate, welche sich in der termios-Struktur befindet, auf die *termios_p* zeigt, auf *speed*. Die Nullrate B0 wird verwendet, um die Verbindung zu beenden. Wird B0 angegeben, werden die Kontrolleitungen des Modems nicht länger verarbeitet. Normalerweise beendet dies die Verbindung.

cfgetispeed liest die Eingabebaudrate und speichert sie in der termios-Struktur, auf die *termios_p* zeigt.

cfsetispeed stellt die Eingabebaudrate, die in der termios-Struktur gespeichert ist, auf die *termios_p* zeigt, auf *speed*. Wenn die Eingabebaudrate auf Null gesetzt wird, wird die Eingabebaudrate auf den Wert der Ausgabebaudrate gesetzt. Sowohl cfsetispeed

als auch `cfsetospeed` liefern den Wert Null zurück, wenn die Operation erfolgreich ausgeführt wurde, und -1, wenn ein Fehler aufgetreten ist. Versuche, nicht unterstützte Baud-Raten einzustellen, werden ignoriert. Dies bezieht sich auf die Änderung von Baud-Raten, welche nicht von der Hardware unterstützt werden, und auf die Einstellung von Ein- und Ausgabebaudraten auf unterschiedliche Werte, wenn die Hardware dies nicht erlaubt.

Vordergrund-Prozeßgruppennummern eines Terminals lesen und setzen

`tcsetpgrp` stellt die Vordergrund-Prozeßgruppennummer des durch *fildes* angegebenen Terminals auf *pgid*. Die Datei *fildes* muß ein steuerndes Terminal des aufrufenden Prozesses bezeichnen, und das steuernde Terminal muß mit der Sitzung des aufrufenden Prozesses verbunden sein. *pgid* muß der Prozeßgruppennummer eines Prozesses entsprechen, der sich in derselben Sitzung wie der aufrufende Prozeß befindet.

`tcgetpgrp` liefert die Vordergrund-Prozeßgruppennummer des durch *fildes* angegebenen Terminals. `tcgetpgrp` kann durch einen Prozeß aufgerufen werden, welcher Mitglied einer Hintergrund-Prozeßgruppe ist; die Informationen können nachher durch einen Prozeß geändert werden, welcher Mitglied einer Vordergrund-Prozeßgruppe ist.

Sitzungsnummern des Terminals lesen

`tcgetsid` liefert die Sitzungsnummer des durch *fildes* angegebenen Terminals.

ERGEBNIS

Bei Erfolg wird von `tcgetpgrp` die Prozeßgruppennummer der Vordergrund-Prozeßgruppe zurückgeliefert, welche mit dem angegebenen Terminal verbunden ist. Ansonsten wird -1 zurückgegeben und `errno` gesetzt.

Bei Erfolg liefert `tcgetsid` die Sitzungsnummer, die mit dem angegebenen Terminal verbunden ist. Ansonsten wird -1 zurückgegeben und `errno` gesetzt.

Bei Erfolg liefern alle anderen Funktionen den Wert 0. Ansonsten wird -1 zurückgegeben und `errno` gesetzt.

Jede Funktion schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EBADF` Das *fildes*-Argument ist kein gültiger Dateideskriptor.

`ENOTTY` Die Datei *fildes* ist kein Terminal.

`tcsetattr` schlägt außerdem fehl, wenn folgende Bedingung wahr ist:

`EINVAL` Das Argument *optional_actions* enthält keinen angemessenen Wert, oder es wurde versucht, ein in der `termios`-Struktur dargestelltes Attribut auf einen nicht unterstützten Wert einzustellen.

`tcsendbreak` schlägt außerdem fehl, wenn folgende Bedingung wahr ist:

`EINVAL` Das Gerät unterstützt die Funktion `tcsendbreak` nicht.

`tcdrain` schlägt außerdem fehl, wenn eine der folgenden Bedingungen erfüllt ist:

`EINTR` Ein Signal unterbrach die `tcdrain`-Funktion.

`EINVAL` Das Gerät unterstützt die Funktion `tcdrain` nicht.

`tcflush` schlägt außerdem fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` Das Gerät unterstützt die `tcflush`-Funktion nicht, oder das Argument *queue_selector* enthält keinen entsprechenden Wert.

`tcflow` schlägt außerdem fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` Das Gerät unterstützt die Funktion `tcflow` nicht, oder *action* enthält keinen entsprechenden Wert.

`tcgetpgrp` schlägt außerdem fehl, wenn folgende Bedingung erfüllt ist:

`ENOTTY` Der aufrufende Prozeß hat kein steuerndes Terminal, oder *fildes* entspricht nicht dem steuernden Terminal.

`tcsetpgrp` schlägt außerdem fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` *pgid* ist keine gültige Prozeßgruppennummer.

`ENOTTY` der aufrufende Prozeß besitzt kein steuerndes Terminal, oder *fildes* entspricht nicht dem steuernden Terminal, oder das steuernde Terminal ist nicht länger mit der Sitzung des aufrufenden Prozesses verbunden.

`EPERM` *pgid* entspricht nicht der Prozeßgruppe eines existierenden Prozesses in derselben Sitzung wie der aufrufende Prozeß.

`tcgetsid` schlägt außerdem fehl, wenn folgende Bedingung erfüllt ist:

`EACCES` *fildes* ist ein Terminal, welches nicht für eine Sitzung zugewiesen ist.

SIEHE AUCH

`setsid`(2), `setpgid`(2).
`termio`(7) in "Referenzhandbuch für Systemverwalter".

time - Uhrzeit abfragen

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

time_t time(time_t *tloc);
```

`time` gibt die Zeit in Sekunden ab 00:00:00 UTC, 1. Januar 1970, zurück.

Wenn *tloc* ungleich Null ist, wird zusätzlich der Rückgabewert an die Stelle gespeichert, auf die *tloc* zeigt.

SIEHE AUCH

`stime`(2), `ctime`(3C).

HINWEIS

`time` scheitert, und seine Aktionen sind undefiniert, wenn *tloc* auf eine unzulässige Adresse zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung gibt `time` die Zeit zurück. Andernfalls wird `(time_t)-1` zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

times - Zeiten für Prozeß und Sohnprozeß abfragen

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

clock_t times(struct tms *buffer);
```

`times` füllt die `tms`-Struktur, auf die *buffer* zeigt, mit Zeitabrechnungsdaten auf. Die `tms`-Struktur ist in `sys/times.h` wie folgt definiert:

```
struct    tms {
          clock_t         tms_utime;
          clock_t         tms_stime;
          clock_t         tms_cutime;
          clock_t         tms_cstime;
};
```

Diese Daten kommen vom aufrufenden Prozeß und von jedem der beendeten Sohnprozesse, für die eine `wait`-Routine ausgeführt wurde. Alle Zeiten werden in Zeittakten pro Sekunde registriert. Zeittakt ist ein systemabhängiger Parameter. Der spezifische Wert einer Implementierung wird durch die Variable `CLK_TCK` definiert, die in der Include-Datei `limits.h` zu finden ist.

`tms_utime` ist die CPU-Zeit, die bei der Ausführung von Anweisungen im Benutzerbereich des aufrufenden Prozesses aufgewendet wurde.

`tms_stime` ist die CPU-Zeit, die vom System im Auftrag des aufrufenden Prozesses aufgewendet wurde.

`tms_cutime` ist die Summe von `tms_utime` und `tms_cutime` der Sohnprozesse.

`tms_cstime` ist die Summe von `tms_stime` und `tms_cstime` der Sohnprozesse.

`times` scheitert, wenn:

`EFAULT` *buffer* auf eine unzulässige Adresse zeigt.

SIEHE AUCH

`exec`(2), `fork`(2), `time`(2), `wait`(2), `waitid`(2), `waitpid`(3C).
`time`(1), `timex`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Bei erfolgreicher Beendigung gibt `times` die abgelaufene Echtzeit in Zeittakten/Sekunden zurück, gerechnet von einem willkürlich gewählten Punkt in der Vergangenheit (z.B. Systemstartzeit). Dieser Punkt ändert sich nicht von einem Aufruf von `times` zum nächsten. Wenn `times` erfolglos ist, wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

uadmin - Verwaltungsfunktionen

```
#include <sys/uadmin.h>
int uadmin(int cmd, int fcn, int mdep);
```

`uadmin` ermöglicht die Steuerung für die grundlegenden Verwaltungsfunktionen. Dieser Systemaufruf ist eng mit den Systemverwaltungsprozeduren verbunden und nicht für den allgemeinen Gebrauch bestimmt. Das Argument *mdep* ist für maschinenabhängigen Gebrauch vorgesehen und an dieser Stelle nicht erklärt.

Folgende Steuerfunktionen, die in *cmd* angegeben werden, stehen zur Verfügung:

`A_SHUTDOWN` Das System wird abgeschaltet. Alle Benutzerprozesse werden abgebrochen, der Pufferspeicher wird entleert und das Root-Dateisystem wird ausgehängt. Die nach Abschalten des Systems auszuführende Maßnahme wird durch *fcn* angegeben. Die Funktionen sind generisch; die jeweiligen Hardware-Möglichkeiten sind bei den verschiedenen Geräten unterschiedlich.

`AD_HALT`	Prozessor anhalten
`AD_BOOT`	System mit der Datei `/stand/unix` neuladen.
`AD_IBOOT`	Interaktives Neuladen; der Benutzer wird zur Eingabe des Systemnamens aufgefordert.

`A_REBOOT` Das System stoppt sofort ohne Ausführen einer weiteren Verarbeitung. Die als nächstes auszuführende Maßnahme wird wie oben durch *fcn* angegeben.

`A_REMOUNT` Das Root-Dateisystem wird nach erfolgter Reparatur wieder eingehängt. Dies sollte nur während des Systemstartvorgangs benutzt werden.

`uadmin` ist erfolglos, wenn der nachstehende Punkt wahr ist:

`EPERM` Die effektive Benutzernummer ist nicht die des Systemverwalters.

ERGEBNIS

Nach erfolgreicher Beendigung hängt der jeweils zurückgegebene Wert wie folgt von *cmd* ab:

```
A_SHUTDOWN      kehrt nie zurück
A_REBOOT        kehrt nie zurück
A_REMOUNT       0
```

Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

ulimit - Prozeßgrenzen abfragen und setzen

```
#include <ulimit.h>
long ulimit(int cmd, ... /* newlimit */ );
```

Diese Funktion ermöglicht die Steuerung der Prozeßgrenzen. Die möglichen Werte für *cmd* sind:

UL_GETFSIZE
: Es wird die für den Prozeß gesetzte maximale Größe einer normalen Datei abgefragt. Die Grenze wird in Einheiten von 512-Byte-Blöcken gemessen und wird von Sohnprozessen übernommen. Dateien beliebiger Länge können gelesen werden.

UL_SETFSIZE
: Es wird die für den Prozeß gesetzte maximale Größe einer normalen Datei auf den Wert von *newlimit*, als `long` genommen, gesetzt. Jeder Prozeß kann diesen Grenzwert verkleinern, jedoch darf nur ein Prozeß mit einer effektiven Benutzernummer des Systemverwalters den Grenzwert erhöhen.

UL_GMEMLIM
: Der größtmögliche Speichergrenzwert wird abgefragt (siehe `brk`(2)).

UL_GDESLIM
: Es wird der aktuelle Wert der maximalen Anzahl an offenen Dateien pro Prozeß erhalten, die im System erstellt wurde.

Der Systemaufruf `ulimit` scheitert, wenn folgendes zutrifft:

EINVAL
: Das Argument *cmd* ist nicht gültig.

EPERM
: Ein Prozeß mit einer effektiven Benutzernummer, die sich von der des Systemverwalters unterscheidet, versucht, seine Dateigrößenbegrenzungen auszuweiten. Es erfolgt die Rückgabe des konfigurierten Wertes für NOFILES, der Maximalzahl geöffneter Dateien je Prozeß.

SIEHE AUCH

`brk`(2), `getrlimit`(2), `write`(2).

HINWEIS

`ulimit` ist in der Größenbegrenzung regulärer Dateien wirksam. Pipes werden momentan durch `PIPE_MAX` begrenzt.

ERGEBNIS

Nach erfolgreicher Beendigung wird ein nicht negativer Wert zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

umask - Dateimodus-Erstellungsmaske setzen und abfragen

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

mode_t umask(mode_t cmask);
```

`umask` setzt die Dateimodus-Erstellungsmaske des Prozesses auf *cmask* und gibt den vorherigen Wert der Maske zurück. Nur die Zugriffsberechtigungsbits von *cmask* und die Dateimodus-Erstellungsmaske werden verwendet.

SIEHE AUCH

`chmod`(2), `creat`(2), `mknod`(2), `open`(2), `stat`(5).
`mkdir`(1), `sh`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Der vorherige Wert der Dateimodus-Erstellungsmaske des Prozesses wird zurückgegeben.

umount - Dateisystem aushängen

```
#include <sys/mount.h>
int umount(const char*datei);
```

Mit umount wird ein durch *datei* angegebenes Dateisystem ausgehängt. *datei* kann dabei eine blockorienterte Gerätedatei oder Dateiverzeichnis sein. *datei* ist ein Zeiger auf einen Pfadnamen. Nach dem Aushängen des Dateisystems wird das Dateiverzeichnis, in dem das Dateisystem eingehängt war, wieder normal interpretiert.
umount darf nur vom Systemverwalter aufgerufen werden.

umount ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

EPERM — Die effektive Benutzernummer des Prozesses ist nicht die des Systemverwalters.

EINVAL — *datei* ist nicht vorhanden.

ELOOP — Zu viele symbolische Verweise wurden aufgerufen, um den Pfad zu übersetzen, auf den durch *datei* verwiesen wurde.

ENAMETOOLONG — *datei* ist länger als PATH_MAX, oder die Länge einer *datei*-Komponente überschreitet NAME_MAX, während _POSIX_NO_TRUNK wirksam ist.

ENOTBLK — *datei* ist keine blockorientierte Gerätedatei.

EINVAL — *datei* ist nicht eingehängt.

EBUSY — Eine Datei in *datei* ist in Benutzung.

EFAULT — *datei* weist auf eine unzulässige Adresse.

EREMOTE — *datei* ist ein ferner Pfadname.

ENOLINK — *datei* befindet sich auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

EMULTIHOP — Die Komponenten des Pfades, auf den *datei* zeigt, erfordern den Sprung auf mehrere ferne Rechner.

SIEHE AUCH

mount(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

uname - Namen des aktuellen SINIX-Systems abfragen

```
#include <sys/utsname.h>
int uname(struct utsname *name);
```

uname speichert Informationen zur Identifizierung des aktuellen SINIX-Systems in der Struktur, auf die *name* zeigt.

uname verwendet die in sys/utsname.h definierte Struktur utsname mit folgenden Elementen:

```
char  sysname[SYS_NMLN];      (Systemname)
char  nodename[SYS_NMLN];     (Knotenname)
char  release[SYS_NMLN];      (Ausgabe)
char  version[SYS_NMLN];      (Version)
char  machine[SYS_NMLN];      (Gerät)
```

uname gibt eine mit dem Null-Byte abgeschlossene Zeichenkette in der Zeichenreihe sysname zurück, die der Name des aktuellen SINIX-Systems ist. Ähnlich enthält nodename den Namen, mit dem das System einem Kommunikationsnetzwerk bekannt ist. release und version identifizieren das Betriebssystem. machine enthält einen Standardnamen, der die Hardware bezeichnet, auf der das SINIX-System läuft.

EFAULT uname ist erfolglos, wenn name auf eine ungültige Adresse zeigt.

SIEHE AUCH

uname(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Beendigung wird ein nicht negativer Wert zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

unlink - Dateiverzeichnis-Eintrag entfernen

```
#include <unistd.h>
int unlink(const char *Pfad);
```

`unlink` löscht den Dateiverzeichnis-Eintrag, der durch den Pfadnamen angegeben wird, auf den *Pfad* zeigt, und vermindert den Verweiszähler der Datei, auf die sich der Dateiverzeichnis-Eintrag bezieht. Sobald alle Verweise auf eine Datei entfernt worden sind und kein Prozeß die Datei geöffnet hat, wird der von der Datei belegte Speicher freigegeben, und die Datei ist fortan nicht mehr vorhanden. Falls einer oder mehrere Prozesse die Datei während der Entfernung der letzten Verbindung geöffnet haben, wird der von der Datei belegte Speicher nicht freigegeben, bis alle Verweise auf die Datei geschlossen wurden. Wenn *Pfad* ein symbolischer Verweis ist, so wird er entfernt. *Pfad* sollte kein Verzeichnis benennen, sofern der Prozeß keine entsprechenden Privilegien besitzt. Anwendungen sollten zur Entfernung von Verzeichnissen `rmdir` benutzen.

Nach erfolgreicher Durchführung markiert `unlink` die Felder `st_ctime` und `st_mtime` des übergeordneten Verzeichnisses. Ebenso wird das Feld `st_ctime` der Datei zur Aktualisierung markiert, wenn der Verweiszähler der Datei ungleich Null ist.

Der Verweis auf die angegebene Datei wird gelöscht, außer wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EACCES` Eine Komponente von *Pfad* darf nicht durchsucht werden.

`EACCES` Die Schreiberlaubnis wird für das Dateiverzeichnis verweigert, das den zu entfernenden Verweis enthält.

`EACCES` Das übergeordnete Verzeichnis hat das Sticky-Bit gesetzt, und der Benutzer darf nicht auf die Datei schreiben; der Benutzer besitzt weder das übergeordnete Verzeichnis noch die Datei.

`EBUSY` Der Eintrag, der entfernt werden soll, ist der Einhängepunkt für ein eingehängtes Dateisystem.

`EFAULT` *Pfad* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während des `unlink`-Systemaufrufs aufgefangen.

`ELOOP` Bei der Übersetzung von *Pfad* wurden zu viele symbolische Verbindungen angetroffen.

`EMULTIHOP` Komponenten von *Pfad* erfordern den Sprung auf ferne Rechner, und das Dateisystem erlaubt dies nicht.

ENAMETOOLONG
: Die Länge von *Pfad* überschreitet PATH_MAX, oder die Länge einer Komponente von *Pfad* überschreitet NAME_MAX, während _POSIX_NO_TRUNC wirksam ist.

ENOENT
: Die genannte Datei ist nicht vorhanden oder ist ein Null-Pfadname. Der Benutzer ist kein Systemverwalter.

ENOTDIR
: Eine Komponente von *Pfad* ist kein Verzeichnis.

EPERM
: Die angegebene Datei ist ein Dateiverzeichnis, und die effektive Benutzernummer des Prozesses ist nicht diejenige des Systemverwalters.

ETXTBSY
: Die Datei ist der letzte Verweis für eine reine Programmdatei, die gerade ausgeführt wird.

EROFS
: Der zu entfernende Dateiverzeichnis-Eintrag ist Teil eines schreibgeschützten Dateisystems.

ENOLINK
: *Pfad* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

close(2), link(2), open(2), rmdir(2).
rm(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

ustat - Dateisystem-Statistik abfragen

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

int ustat(dev_t dev, struct ustat *puffer);
```

`ustat` gibt Informationen über ein eingehängtes Dateisystem zurück. *dev* ist eine Gerätenummer, mit der ein Gerät bezeichnet wird, das ein eingehängtes Dateisystem enthält (siehe `mkdev`(3C)). *puffer* ist ein Zeiger auf eine `ustat`-Struktur mit folgenden Elementen:

```
daddr_t     f_tfree;        /* Gesamtanzahl freier Blöcke */
ino_t       f_tinode;       /* Anzahl freier Indexeinträge */
char        f_fname[6];     /* Dateisystemname */
char        f_fpack[6];     /* Dateisystem, gepackter Name */
```

`ustat` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`EINVAL` *dev* ist nicht die Nummer eines Geräts, das ein eingehängtes Dateisystem enthält.

`EFAULT` *puffer* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

`EINTR` Ein Signal wurde während eines Systemaufrufs `ustat` abgefangen.

`ENOLINK` *dev* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

`ECOMM` *dev* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

`stat`(2), `statvfs`(2), `mkdev`(3C), `fs`(4).

HINWEIS

`ustat`(2) wird zugunsten der `statvfs`-Funktion in Zukunft nicht mehr unterstützt.

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und `errno` wird zur Anzeige des Fehlers gesetzt.

utime - Dateizugriffs- und Änderungszeiten setzen

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

int utime(const char *path, const struct utimbuf *times);
```

path weist auf einen Pfadnamen, der eine Datei benennt. utime setzt die Zugriffs- und Änderungszeiten dieser Datei.

Wenn *times* gleich NULL ist, werden die Zugriffs- und Änderungszeiten der Datei auf die aktuelle Uhrzeit gesetzt. Ein Prozeß muß der Eigentümer der Datei sein oder Schreiberlaubnis haben, um utime auf diese Weise verwenden zu können.

Wenn *times* nicht gleich NULL ist, wird *times* als Zeiger auf eine utimbuf-Struktur (definiert in utime.h) interpretiert und die Zugriffs- und Änderungszeiten werden auf die Werte gesetzt, die in der angegebenen Struktur enthalten sind. Nur der Eigentümer der Datei oder der Systemverwalter dürfen utime auf diese Weise verwenden.

Die in der nachstehenden Struktur angegebenen Zeiten werden in Sekunden ab 00:00:00 GMT 1. Jan. 1970 gemessen.

```
struct      utimbuf {
    time_t          actime;         /* Zugriffszeit */
    time_t          modtime;        /* Modifikationszeit */
};
```

utime ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte wahr sind:

EACCES — Eine Komponente des Pfades darf nicht durchsucht werden.

EACCES — Die effektive Benutzernummer ist nicht diejenige des Systemverwalters und nicht diejenige des Eigentümers der Datei, *times* ist gleich NULL, und der Schreibzugriff wird verweigert.

EFAULT — *times* ist ungleich NULL und weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EFAULT — *path* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EINTR — Ein Signal wurde während des Systemaufrufs utime abgefangen.

ELOOP — Während der Übersetzung von *path* wurden zu viele symbolische Verweise angetroffen.

EMULTIHOP — Die Komponenten von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dieses nicht.

ENAMETOOLONG — Die Länge von *path* ist größer als PATH_MAX oder NAME_MAX während POSIX_NO_TRUNC aktiv ist.

ENOENT	Die angegebene Datei ist nicht vorhanden.
ENOLINK	*path* weist auf einen fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.
NOTDIR	Eine Komponente des Pfades ist kein Dateiverzeichnis.
EPERM	Die effektive Benutzernummer ist nicht diejenige des Systemverwalters und nicht diejenige des Eigentümers der Datei, und *times* ist nicht gleich NULL.
EROFS	Das Dateisystem, das die Datei enthält, ist schreibgeschützt eingehängt.

SIEHE AUCH

stat(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

vfork - neuen Prozeß im virtuellen Speicher erzeugen

```
#include <unistd.h>
pid_t vfork (void);
```

`vfork` kann dazu verwendet werden, um neue Prozesse zu erzeugen, ohne daß der komplette Adreßbereich des alten Prozesses kopiert wird. Dies ist nützlich, wenn der Zweck von `fork` darin besteht, einen neuen Systemkontext für einen `execve`-Aufruf zu erzeugen. `vfork` unterscheidet sich von `fork` darin, daß sich der Sohnprozeß den Speicher und die Kontrollstrukturen des Vaterprozesses ausleiht, bis ein `execve`-Aufruf oder eine Terminierung (sei es durch `exit` oder auf andere Weise) auftritt. Der Vaterprozeß wird angehalten, während der Sohnprozeß dessen Ressourcen verwendet.

`vfork` liefert 0 im Kontext des Sohnprozesses zurück und (später) die Prozeßnummer (PID) des Sohnprozesses im Kontext des Vaterkontextes.

`vfork` kann normalerweise genau wie `fork` verwendet werden. Dies funktioniert nicht, wenn der Kontext des Sohnprozesses ausgeführt wird und aus der Routine zurückgesprungen werden soll, welche `vfork` aufgerufen hat, da die Rückkehr von `vfork` auf einen nicht mehr vorhandenen Stapelrahmen treffen würde. Seien Sie also vorsichtig, rufen Sie eher `_exit` anstelle von `exit` auf, wenn Sie nicht `execve` aufrufen können, weil `exit` die E/A-Kanäle verarbeitet und schließt und daher die E/A-Datenstrukturen des Vaterprozesses zerstört. Selbst bei `fork` ist der Aufruf von `exit` falsch, da gepufferte Daten doppelt verarbeitet werden.

ERGEBNIS

Nach erfolgreicher Durchführung liefert `vfork` den Wert Null an den Sohnprozeß zurück und liefert die Prozeßnummer des Sohnprozesses an den Vaterprozeß zurück. Ansonsten wird -1 an den Vaterprozeß zurückgeliefert, und es wird kein Sohnprozeß erzeugt; die globale Variable `errno` enthält die Fehlernummer.

`vfork` schlägt fehl und kein Sohnprozeß wird erzeugt, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

`EAGAIN` Die systembedingten Grenzen der maximal möglichen Prozesse würde überschritten werden. Diese Grenze wird festgelegt, wenn das System erzeugt wird.

`EAGAIN` Die systembedingten Grenzen der maximal möglichen Prozesse eines Benutzers würden überschritten werden. Dieser Grenzwert wird festgelegt, wenn das System erzeugt wird.

`ENOMEM` Der Swap-Bereich ist für den neuen Prozeß nicht groß genug.

SIEHE AUCH

exec(2), exit(2), fork(2), ioctl(2), wait(2).

HINWEIS

vfork wird in zukünftigen Versionen eliminiert werden. Änderungen der Systemimplementierungen machen die Effizienzvorteile von vfork gegenüber fork geringer. Die Nutzung gemeinsamen Speichers über vfork kann über andere Mechanismen erreicht werden.

Um einen möglichen Deadlock des Systems zu umgehen, sollten Prozesse, die Sohnprozesse in der Mitte eines vfork-Aufrufs sind, niemals SIGTTOU- oder SIGTTIN-Signale gesendet bekommen. Ausgabe oder *ioctl*-Aufrufe sind erlaubt; der Versuch, Eingaben zu verarbeiten, resultiert in der Anzeige von EOF.

Auf einigen Systemen verursacht vfork, daß der Vaterprozeß die Registerwerte vom Sohnprozeß erbt. Dies kann bei der Verwendung von optimierenden Übersetzern zu Problemen führen, wenn unistd.h nicht in den Quelltext, der den vfork-Aufruf enthält, eingefügt wird.

wait - auf Anhalten oder Beendigung eines Sohnprozesses warten

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

pid_t wait(int *stat_loc);
```

`wait` hält den aufrufenden Prozeß an, bis einer seiner unmittelbaren Sohnprozesse beendet wird, oder bis ein Sohnprozeß, bei dem gerade Ablaufverfolgung durchgeführt wird, anhält, weil er ein Signal erhalten hat. Der `wait`-Systemaufruf kehrt vorzeitig zurück, wenn ein Signal empfangen wird. Wenn alle Sohnprozesse vor dem Aufruf von `wait` angehalten oder beendet wurden, erfolgt eine sofortige Rückkehr.

Wenn `wait` zurückkehrt, weil der Status eines Sohnprozesses zur Verfügung steht, gibt es die Prozeßnummer des Sohnprozesses zurück. Wenn der antwortende Prozeß einen Wert für *stat_loc* angegeben hat, der ungleich Null ist, wird der Status des Sohnprozesses an jenem Ort gespeichert, der von *stat_loc* angegeben wird. Er kann mit den Makros ausgewertet werden, die in `wstat`(5) beschrieben werden. Im folgenden ist `Status` das Objekt, auf das *stat_loc* zeigt:

- Wenn der Sohnprozeß angehalten hat, enthalten die höherwertigen 8 Bits von `Status` die Nummer des Signals, das das Anhalten des Prozesses verursacht hat, und die niederwertigen 8 Bits werden auf `WSTOPFLG` gesetzt.
- Wenn der Sohnprozeß infolge eines `exit`-Aufrufs beendet wurde, sind die niederwertigen 8 Bits von `Status` auf 0 gesetzt, und die höherwertigen 8 Bits enthalten die niederwertigen 8 Bits des Arguments, das der Sohnprozeß an `exit` weitergereicht hat (siehe `exit`(2)).
- Wenn der Sohnprozeß infolge eines Signals abgebrochen wurde, sind die höherwertdgen 8 Bits von `Status` 0. Die niederwertigen 8 Bits enthalten die Nummer des Signals, das den Abbruch verursacht hat. Wenn außerdem `WCORFLG` gesetzt ist, wurde ein Speicherabzug erstellt (siehe `signal`(2)).

Wenn `wait` zurückkehrt, weil der Status eines Sohnprozesses zur Verfügung steht, dann kann der Status mit den von `wstat`(5) definierten Makros analysiert werden.

Wenn ein Vaterprozeß beendet wird, ohne auf die Beendigung seiner Sohnprozesse zu warten, wird die Vaterprozeßnummer jedes Sohnprozesses auf 1 gesetzt. Das bedeutet, daß der Initialisierungsprozeß die Sohnprozesse erbt (siehe `intro`(2)).

`wait` ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte zutreffen:

`ECHILD` Der aufrufende Prozeß hat keine Sohnprozesse oder keine, auf die er noch warten könnte.

`EINTR` Die Funktion wurde durch ein Signal unterbrochen.

SIEHE AUCH

exec(2), exit(2), fork(2), intro(2), pause(2), ptrace(2), signal(2), signal(5), wstat(5).

HINWEIS

Siehe HINWEIS in signal(2).

Wenn SIGCLD enthalten ist, erkennt wait den Abbruch der Sohnprozesse nicht.

ERGEBNIS

Wenn wait infolge eines angehaltenen oder beendeten Sohnprozesses zurückkehrt, wird die Nummer des Sohnprozesses an den aufrufenden Prozeß zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

waitid - auf Zustandsänderung von Sohnprozessen warten

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

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

Der aufrufende Prozeß wird durch `waitid` solange angehalten, bis einer der Sohnprozesse den Zustand ändert. Der aktuelle Zustand eines Sohnprozesses wird in der Struktur, auf die *infop* zeigt, aufgenommen. Wenn ein Sohnprozeß den Zustand vor dem Aufruf von `waitid` geändert hat, kehrt `waitid` sofort zurück.

Die Argumente *idtype* und *id* geben an, auf welche Sohnprozesse `waitid` warten soll.

- Wenn *idtype* gleich `P_PID` ist, wartet `waitid` auf den Sohnprozeß, der die Prozeßnummer `(pid_t)` *id* hat.
- Wenn *idtype* gleich `P_PGID` ist, wartet `waitid` auf irgendeinen Sohnprozeß mit der Prozeßgruppennummer `(pid_t)`*id*.
- Wenn *idtype* gleich `P_ALL` ist, wartet `waitid` auf irgendeinen Sohnprozeß, und *id* wird ignoriert.

Das Argument *options* wird verwendet, um die gewünschten Zustandsänderungen anzugeben, auf die `waitid` warten soll. Die Zustandsänderungen werden durch bitweise ODER-Verknüpfung der folgenden Schalter angegeben:

`WEXITED` wartet darauf, daß Prozesse terminieren (exit).

`WTRAPPED` wartet darauf, daß ablaufverfolgte Prozesse auf Unterbrechungen stoßen oder einen Haltepunkt erreichen (siehe `ptrace`(2)).

`WSTOPPED` wartet und liefert den Prozeßstatus eines Sohnprozesses, welcher nach dem Empfang eines Signals gestoppt hat.

`WCONTINUED` liefert den Status für einen Sohnprozeß, welcher angehalten und wieder aufgenommen wurde.

`WNOHANG` kehrt sofort zurück.

`WNOWAIT` hält den Prozeß in einem Zustand, daß der Wartezustand erhalten bleibt.

infop muß auf eine `siginfo_t`-Struktur zeigen, wie sie in `siginfo`(5) definiert wird. `siginfo_t` wird vom System mit dem Zustand des erwarteten Prozesses gefüllt.

`waitid` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EFAULT` *infop* zeigt auf eine ungültige Adresse.

`EINTR` `waitid` wurde unterbrochen, weil der aufrufende Prozeß ein Signal empfing.

`EINVAL` Ein ungültiger Wert wurde für *options* übergeben.

`EINVAL` *idtype* und *id* geben eine ungültige Prozeßmenge an.

`ECHILD` Die Prozeßmenge, die durch *idtype* und *id* angegeben wurde, enthält keine Prozesse, auf die noch niemand wartet oder Prozesse, die nicht existieren.

ERGEBNIS

Kehrt `waitid` aufgrund einer Zustandsänderung eines Sohnprozesses zurück, wird der Wert 0 zurückgegeben. Ansonsten wird -1 zurückgeliefert und `errno` gesetzt, um den Fehler anzuzeigen.

SIEHE AUCH

`intro`(2), `exec`(2), `exit`(2), `fork`(2), `pause`(2), `ptrace`(2), `signal`(2), `sigaction`(2), `wait`(2), `siginfo`(5).

waitpid - auf Zustandsänderung von Sohnprozessen warten

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

pid_t waitpid (pid_t pid, int *stat_loc, int options);
```

`waitpid` schiebt die Ausführung des aufrufenden Prozesses auf, bis einer der Sohnprozesse den Zustand ändert; wenn ein Sohnprozeß den Zustand vor dem Aufruf von `waitpid` änderte, wird sofort vom Aufruf `waitpid` zurückgesprungen. *pid* gibt eine Menge von Sohnprozessen an, deren Zustand überprüft werden soll.

- Ist *pid* gleich `(pid_t)-1`, so wird der Zustand aller Sohnprozesse überwacht.
- Ist *pid* größer als `(pid_t)0`, so wird hiermit die Prozeßnummer des Sohnprozesses angegeben, der überwacht werden soll.
- Ist *pid* gleich `(pid_t)0`, so wird der Zustand aller Sohnprozesse geprüft, deren Prozeßgruppennummer gleich der des aufrufenden Prozesses ist.
- Ist *pid* kleiner als `(pid_t)-1`, so wird der Zustand für jeden Sohnprozeß abgeprüft, dessen Prozeßgruppennummer gleich dem absoluten Wert von *pid* ist.

Kehrt `waitpid` zurück, weil der Zustand eines Sohnprozesses verfügbar ist, so wird der Zustand mit Hilfe der in `wstat`(5) definierten Makros ausgewertet. Hat der aufrufende Prozeß für *stat_loc* einen Wert ungleich Null angegeben, so wird der Zustand des Sohnprozesses an der Stelle abgelegt, an die *stat_loc* zeigt.

Das Argument *options* setzt sich aus der bitweisen ODER-Verknüpfung der folgenden Schalter zusammen, die in der Include-Datei `sys/wait.h` wie folgt definiert werden:

`WCONTINUED` Der Zustand eines wiederaufgenommenen Sohnprozesses, der durch *pid* angegeben wurde, und dessen Zustand seit der Wiederaufnahme nicht zurückgeliefert wurde, sollte ebenfalls an den aufrufenden Prozeß weitergegeben werden.

`WNOHANG` `waitpid` hält die Ausführung des aufrufenden Prozesses nicht an, wenn der Zustand für einen der durch *pid* angegebenen Sohnprozesse nicht sofort verfügbar ist.

`WNOWAIT` hält den Prozeß, dessen Zustand nach *stat_loc* geschrieben wird, in einem Zustand, damit die Wartebedingung erhalten bleibt. Auf den Prozeß kann wieder mit identischem Resultat gewartet werden.

`WUNTRACED` Der Zustand eines Sohnprozesses, der durch *pid* angegeben wurde und angehalten ist, und dessen Zustand seit dem Anhalten nicht angegeben wurde, soll ebenfalls an den aufrufenden Prozeß weitergegeben werden.

`waitpid` mit dem Argument *options* gleich `WUNTRACED` und *pid* gleich `(pid_t)-1` ist identisch mit dem Aufruf von `wait`(2).

`waitpid` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EINTR` `waitpid` wurde unterbrochen, weil der aufrufende Prozeß ein Signal empfing.

`EINVAL` Für *options* wurde ein ungültiger Wert angegeben.

`ECHILD` Der Prozeß oder die Prozeßgruppe, die durch *pid* angegeben wird, existiert nicht oder ist kein Sohnprozeß des aufrufenden Prozesses oder kann sich nie im Zustand *options* befinden.

SIEHE AUCH

`exec`(2), `exit`(2), `fork`(2), `intro`(2), `pause`(2), `ptrace`(2), `signal`(2), `sigaction`(2), `siginfo`(5), `wstat`(5).

ERGEBNIS

Wenn `waitpid` zurückkehrt, weil der Zustand eines Sohnprozesses verfügbar wird, sollte diese Funktion einen Wert zurückliefern, der der Prozeßnummer des Sohnprozesses entspricht, dessen Zustand zurückgegeben wurde. Wenn `waitpid` zurückspringt, weil der aufrufende Prozeß ein Signal empfing, wird der Wert -1 zurückgegeben und `errno` auf `EINTR` gesetzt. Wird diese Funktion mit der Option `WNOHANG` in *options* aufgerufen, und hat sie mindestens einen Sohnprozeß, der durch *pid* angegeben wird und dessen Zustand nicht verfügbar ist, und ist der Zustand für einen durch *pid* angegebenen Prozeß nicht verfügbar, so wird der Wert 0 zurückgegeben, ansonsten wird -1 zurückgegeben und `errno` gesetzt.

write, writev - in eine Datei schreiben

```
#include <unistd.h>

int write(int dk, const void *Puffer, unsigned nbyte);

#include <sys/types.h>
#include <sys/uio.h>

int writev(int dk, const struct iovec *iov, int iovcnt);
```

`write` versucht, *nbyte* Bytes vom Puffer, auf den *Puffer* zeigt, in die zu *dk* gehörende Datei zu schreiben. Wenn *nbyte* Null und die Datei eine reguläre Datei ist, gibt `write` Null zurück und hat keine anderen Ergebnisse. *dk* ist ein Dateideskriptor, der von einem `creat`-, `open`-, `dup`-, `fcntl`- oder `pipe`-Systemaufruf geliefert wird.

`writev` macht dasselbe wie `write`, sammelt aber die Ausgabedaten der *iovcnt*-Puffer, die durch die Mitglieder der *iov*-Felder (*iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]) festgelegt sind. *iovcnt* ist gültig, wenn es größer als 0 und kleiner oder gleich `IOV_MAX` ist.

Für `writev` enthält die Struktur `iovec` folgende Elemente:

```
caddr_t   iov_base;
int       iov_len;
```

Jeder `iovec`-Eintrag gibt die Basisadresse und die Länge eines Speicherbereichs an, aus dem Daten geschrieben werden sollen. `writev` schreibt immer einen vollständigen Bereich, bevor es zum nächsten übergeht.

Bei Geräten, die positionieren können, beginnt das tatsächliche Schreiben der Daten an der Stelle in der Datei, auf die der Schreib-/Lesezeiger zeigt. Nach Rückkehr von `write` wird der Schreib-/Lesezeiger um die Anzahl der tatsächlich geschriebenen Bytes erhöht. Bei einer regulären Datei wird die Länge der Datei auf den neuen Schreib-/Lesezeiger gesetzt, wenn der erhöhte Schreib-/Lesezeiger größer als die Dateilänge ist.

Bei Geräten, die nicht positionieren können, beginnt das Schreiben immer an der aktuellen Position. Der Wert eines zu einem derartigen Gerät gehörigen Schreib-/Lesezeigers ist undefiniert.

Ist der Anzeiger `O_APPEND` des Dateistatus-Bytes gesetzt, dann wird der Schreib-/Lesezeiger vor jedem `write` auf das Ende der Datei gesetzt.

Wenn der Anzeiger `O_SYNC` des Dateistatus-Bytes bei normalen Dateien gesetzt ist, kehrt `write` erst dann wieder zurück, wenn Dateidaten und Dateistatus physikalisch aktualisiert worden sind. Diese Funktion ist für besondere Anwendungen bestimmt, die eine zusätzliche Zuverlässigkeit auf Kosten der Leistung erfordern. Wenn `O_SYNC` bei block-orientierten Dateien gesetzt wird, kehrt `write` erst dann wieder zurück, wenn die Daten physikalisch aktualisiert worden sind.

Ein write in eine normale Datei wird blockiert, wenn das obligatorische Sperren von Dateien und Dateisätzen gesetzt ist (siehe chmod(2)) und eine Datensatzsperre, die einem anderen Prozeß gehört, auf das Segment der Datei gesetzt ist, in das geschrieben werden soll.

- Wenn O_NDELAY oder O_NONBLOCK gesetzt ist, gibt write -1 zurück und setzt errno auf EAGAIN.
- Wenn weder O_NDELAY noch O_NONBLOCK gesetzt ist, schläft write, bis alle blockierenden Sperren aufgehoben werden oder write durch ein Signal beendet wird.

Wenn ein write-Aufruf verlangt, daß mehr Bytes geschrieben werden als Speicher vorhanden ist - zum Beispiel, wenn das Schreiben die Obergrenze für Dateigrößen des Prozesses (siehe getrlimit(2) und ulimit(2)), die Obergrenze für Dateigrößen des Systems oder den freien Speicherplatz auf dem Gerät überschreitet - werden nur soviele Bytes geschrieben, wie Speicherplatz vorhanden ist. Nehmen wir zum Beispiel an, daß in einer Datei noch Platz für 20 Bytes ist, bevor man an eine Grenze stößt. Ein write von 512 Bytes gibt dann 20 zurück. Das nächste write mit einer Byte-Anzahl ungleich Null gibt dann einen Fehler zurück (außer bei Pipes und FIFO-Dateien).

Schreibanfragen an eine Pipe oder an eine FIFO-Datei werden wie solche an reguläre Dateien behandelt, dabei sind allerdings folgende Ausnahmen zu beachten:

- Es gibt keine Dateipositionierung bei Pipes, folglich wird jede Schreibanfrage am Ende der Pipe angehängt. Es ist garantiert, daß es bei Schreibanfragen von PIPE_BUF oder weniger Bytes nicht zu einer Überlappung mit Daten von anderen Prozessen kommt, die auf die gleiche Pipe schreiben (atomare Schreiboperation). Bei Schreibaufträgen, die größer als PIPE_BUF Bytes sind, kann es an willkürlichen Grenzen zu Überlappungen mit Schreibaufträgen anderer Prozesse kommen, egal, ob die O_NONBLOCK- oder O_NDELAY-Optionen gesetzt sind oder nicht.
- Wenn O_NONBLOCK und O_NDELAY nicht gesetzt sind, kann ein Schreibauftrag den Prozeß blockieren, aber bei normaler Beendigung wird *nbyte* zurückgegeben.
- Wenn O_NONBLOCK gesetzt ist, werden write-Anfragen folgendermaßen behandelt: Schreibanfragen für PIPE_BUF oder weniger Bytes werden entweder vollständig ausgeführt und geben *nbyte* zurück, oder sie geben -1 zurück und setzen errno auf EAGAIN. Ein write-Auftrag für mehr als PIPE_BUF Bytes überträgt entweder soviel er kann und gibt die Anzahl der geschriebenen Bytes zurück, oder er überträgt keine Daten, gibt -1 zurück und setzt errno auf EAGAIN. Auch wenn ein Auftrag größer als PIPE_BUF Bytes ist und alle vorher in die Pipe geschriebenen Daten schon gelesen worden sind, überträgt write mindestens PIPE_BUF Bytes.
- Wenn O_NDELAY gesetzt ist, werden write-Aufträge folgendermaßen behandelt: write blockiert nicht den Prozeß; Schreibaufträge für PIPE_BUF oder weniger Bytes werden entweder vollständig ausgeführt und geben *nbyte* zurück, oder sie geben 0 zurück. Ein write-Auftrag für mehr als PIPE_BUF Bytes überträgt entweder alles, was er kann, und gibt die Anzahl der geschriebenen Bytes zurück, oder er überträgt keine

Daten und gibt 0 zurück. Wenn ein Auftrag größer als PIPE_BUF Bytes ist und alle vorher in die Pipe geschriebenen Daten schon gelesen worden sind, überträgt write mindestens PIPE_BUF Bytes.

Beim Versuch, auf einen Dateideskriptor zu schreiben (keine Pipe oder FIFO-Datei), der nichtblockierendes Schreiben unterstützt und die Daten nicht sofort annehmen kann, passiert folgendes:

- Wenn O_NONBLOCK und O_NDELAY nicht gesetzt sind, blockiert write, bis die Daten angenommen werden können.
- Wenn O_NONBLOCK oder O_NDELAY gesetzt ist, blockiert write nicht den Prozeß. Wenn einige Daten geschrieben werden können, ohne den Prozeß zu blockieren, schreibt write soviel es kann und gibt dann die Anzahl der geschriebenen Bytes zurück. Sonst, wenn O_NONBLOCK gesetzt ist, gibt es -1 zurück und setzt errno auf EAGAIN oder, wenn O_NDELAY gesetzt ist, gibt es 0 zurück.

Bei STREAMS-Dateien (siehe intro(2))
wird die Wirkung von write durch die Werte der unteren und oberen Grenze für den Bereich von *nbyte* ('Paketgröße') bestimmt, die vom Stream angenommen werden. Diese Werte sind im obersten Stream-Modul enthalten. Solange der Benutzer dieses oberste Modul nicht auf den Stream setzt (siehe I_PUSH in streamio(7)), können diese Werte auf Benutzerebene nicht gesetzt oder getestet werden. Wenn *nbyte* im zulässigen Größenbereich liegt, werden *nbyte* Bytes geschrieben.

Wenn *nbyte* nicht in diesem Bereich liegt und der Wert der Mindestpaketgröße Null ist, zerlegt write den Puffer in Segmente maximaler Paketgröße, bevor die Daten streamabwärts geschickt werden (das letzte Segment kann kleiner als die maximale Paketgröße sein). Wenn *nbyte* nicht im Bereich liegt und der Mindestwert ungleich Null ist, ist write erfolglos und setzt errno auf ERANGE. Das Schreiben eines Puffers der Länge Null (*nbyte* ist Null) auf ein STREAMS-Gerät bewirkt, daß eine Nachricht der Länge Null gesendet und Null zurückgegeben wird. Wenn jedoch ein Puffer der Länge Null in eine Pipe oder FIFO-Datei geschrieben wird, wird keine Nachricht gesendet und Null zurückgegeben. Das Benutzerprogramm kann I_SWROPT ioctl(2) verwenden, um zu ermöglichen, daß leere Nachrichten über eine Pipe oder eine FIFO-Datei gesendet werden (siehe streamio(7)).

Beim Schreiben auf einen Stream werden Nachrichten mit der Prioritätsklasse Null erzeugt. Beim Schreiben auf einen Stream, der keine Pipe oder FIFO-Datei ist, geschieht folgendes:

- Wenn O_NDELAY und O_NONBLOCK nicht gesetzt sind und der Stream keine Daten annehmen kann (die Stream-Schreibschlange ist aufgrund von internen Bedingungen zur Datenflußsteuerung voll), blockiert write, bis Daten angenommen werden können.

- Wenn O_NDELAY oder O_NONBLOCK gesetzt ist und der Stream keine Daten annehmen kann, gibt write -1 zurück und setzt errno auf EAGAIN.
- Wenn O_NDELAY oder O_NONBLOCK gesetzt und ein Teil des Puffers schon geschrieben worden ist und dann eine Bedingung auftritt, in der der Stream keine zusätzlichen Daten annehmen kann, beendet write und gibt die Anzahl der geschriebenen Bytes zurück.

write und writev sind erfolglos, und der Schreib-/Lesezeiger bleibt unverändert, wenn eine oder mehrere der folgenden Bedingungen zutreffen:

EAGAIN Obligatorisches Sperren von Dateien und Dateisätzen ist gesetzt, O_NDELAY oder O_NONBLOCK ist gesetzt, und eine blockierende Datensatzsperre ist vorhanden.

EAGAIN Der Systemspeicher, der für 'raw'-Ein-/Ausgabe zur Verfügung steht, ist vorübergehend nicht ausreichend.

EAGAIN Es wurde versucht, in einen Stream zu schreiben, der bei gesetztem Anzeiger O_NDELAY oder O_NONBLOCK keine Daten akzeptieren kann.

EAGAIN Es wurde versucht, einen write-Auftrag von PIPE_BUF Bytes oder weniger auf eine Pipe oder FIFO-Datei zu geben, und es waren weniger als *nbytes* freier Speicherbereich vorhanden.

EBADF *dk* ist kein gültiger Dateideskriptor für eine zum Schreiben geöffnete Datei.

EDEADLK Die write-Funktion schläft, und löst dadurch einen Deadlock aus.

EFAULT *Puffer* weist über den zugewiesenen Adreßraum des Prozesses hinaus.

EFBIG Es wurde versucht, eine Datei zu schreiben, die über die zulässige Grenze für den Prozeß oder die maximale Dateigröße des Prozesses hinausgeht (siehe getrlimit(2) und ulimit(2)).

EINTR Ein Signal wurde während des Systemaufrufs write abgefangen.

EINVAL Es wurde versucht, in einen Stream zu schreiben, der an einen Multiplexer angeschlossen ist.

EIO Der Prozeß ist im Hintergrund und versucht, auf sein steuerndes Terminal zu schreiben, dessen TOSTOP-Option gesetzt ist; der Prozeß ignoriert weder SIGTTOU-Signale, noch blockiert er sie, und seine Prozeßgruppe ist verwaist.

ENOLCK Die System-Datensatzsperren-Tabelle war voll, und daher kann die write-Funktion erst schlafen, wenn die blockierende Datensatzsperre aufgehoben wird.

ENOLINK *dk* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

ENOSR Es wurde versucht, auf einen Stream mit ungenügend STREAMS-Speicherplatz im System zu schreiben.

ENOSPC Während eines write in eine normale Datei ist kein freier Speicher mehr auf dem Rechner.

ENXIO Ein Hangup ist aufgetreten, während auf den Stream geschrieben wird.

EPIPE und das Signal SIGPIPE
Es wurde versucht, in eine Pipe zu schreiben, die nicht für Lesen durch einen Prozeß geöffnet ist.

EPIPE Es wurde versucht, in eine FIFO-Datei zu schreiben, die nicht für Lesen durch einen Prozeß geöffnet ist.

EPIPE Es wurde versucht, in eine Pipe zu schreiben, die nur an einem Ende geöffnet ist.

ERANGE Es wurde versucht, in einen Stream mit *nbyte* außerhalb der vorgegebenen Mindest- und Höchstgrenzen zu schreiben, und der Mindestwert ist ungleich Null.

ENOLCK Erzwungenes Satzsperren war erlaubt, und die LOCK_MAX1-Bereiche sind bereits im System gesperrt.

Zusätzlich kann writev einen der folgenden Fehler zurückgeben:

EINVAL *iovcnt* war kleiner oder gleich 0 oder größer gleich 16.

EINVAL Einer der iov_len-Werte im *iov*-Feld war negativ.

EINVAL Die Summe der iov_len-Werte im *iov*-Feld erzeugt einen Überlauf bei einer 32-Bit Ganzzahl.

Ein write in eine STREAMS-Datei kann erfolglos sein, wenn am Stream-Kopf eine Fehlermeldung erhalten wurde. In diesem Fall wird errno auf den Wert gesetzt, der in der Fehlermeldung enthalten ist.

Bei erfolgreicher Beendigung kennzeichnen write und writev die Felder st_ctime und st_mtime der Datei zur Aktualisierung.

SIEHE AUCH

intro(2), creat(2), dup(2), fcntl(2), getrlimit(2), lseek(2), open(2), pipe(2), ulimit(2).

ERGEBNIS

Bei erfolgreicher Beendigung gibt write die Anzahl der tatsächlich geschriebenen Bytes zurück. Sonst wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

Bibliotheksfunktionen

intro - Einführung

Dieses Kapitel beschreibt Funktionen, die in den verschiedenen Bibliotheken vorkommen, mit Ausnahme der Funktionen, die die SINIX-Systemprozeduren direkt aufrufen. Diese sind im Kapitel "Systemaufrufe" beschrieben. Funktionsdeklarationen finden Sie in den in jedem Abschnitt angegebenen `#include`-Dateien. Es gibt verschiedene Funktionsgruppen, die mit einem Buchstaben hinter der Kapitelnummer gekennzeichnet sind:

(3C) Diese Funktionen bilden mit den Funktionen aus Kapitel 2 und den mit (3S) bezeichneten Funktionen die C-Standardbibliothek `libc`, die automatisch vom C-Übersetzungssystem geladen wird. Die Standard-C-Bibliothek ist als ein mehrfach benutzbares Objekt (shared library) implementiert, `libc.so`, und als ein Archiv `libc.a`. Standardmäßig werden C-Programme mit der mehrfach benutzbaren Version der Standard-C-Bibliothk gebunden. Um mit der Archivversion zu binden, geben Sie in der Kommandozeile des `cc`-Kommandos `-dn` an (siehe `cc`(1) für andere den Standard überschreibende Optionen).

(3S) Diese Funktionen bilden das 'Standard-Eingabe-/Standard-Ausgabe-Paket' (siehe `stdio`(3S)).

(3E) Diese Funktionen bilden die ELF-Zugriffsbibliothek `libelf`. Diese Bibliothek ist nicht als mehrfach benutzbares Objekt implementiert und wird nicht automatisch vom C-Übersetzungssystem gebunden. Geben Sie in der `cc`-Kommandozeile `-lelf` an, um diese Bibliothek zu binden.

(3G) Diese Funktionen bilden die Allzweck-Bibliothek `libgen`. Diese Bibliothek ist nicht als mehrfach benutzbares Objekt implementiert und wird nicht automatisch vom C-Übersetzungssystem gebunden. Geben Sie in der `cc`-Kommandozeile `-lgen` an, um diese Bibliothek zu binden.

(3M) Diese Funktionen bilden die mathematische Bibliothek `libm` (siehe `intro`(3M) und `math`(5)). Diese Bibliothek ist nicht als mehrfach benutzbares Objekt implementiert und wird nicht automatisch vom C-Übersetzungssystem gebunden. Geben Sie in der `cc`-Kommandozeile `-lm` an, um diese Bibliothek zu binden.

(3X) Spezialbibliotheken. Dateien, in denen diese Bibliotheken zu finden sind, werden in den entsprechenden Abschnitten angegeben.

Ein Zeichen ist ein Bitmuster, das in ein Byte im Rechner paßt. Das Nullzeichen ist ein Zeichen mit dem Wert 0, das in der C-Sprache im allgemeinen als '\0' dargestellt wird. Ein Zeichenfeld ist eine Folge von Zeichen. Ein mit Null-Byte abgeschlossenes Zeichenfeld ist eine Folge von Zeichen, bei denen das letzte das Nullzeichen ist. Zeichenkette (*string*) ist die Bezeichnung für ein mit Null-Byte abgeschlossenes Zeichenfeld. Die Nullzeichenkette ist ein Zeichenfeld, das nur das Nullzeichen enthält. Ein Nullzeiger ist der Wert, den man erhält, wenn man einen Zeiger auf `0` setzt. C garantiert, daß dieser Wert sonst nie als Wert eines Zeigers vorkommen kann. Daher geben viele Funktionen, die Zeiger zurückgeben, den Nullzeiger zur Anzeige eines Fehlers zurück. Das Makro `NULL` ist in `stdio.h` definiert. Typen der Form `size_t` sind in den entsprechenden Include-Dateien definiert.

DATEIEN

INCDIR	gewöhnlich `/usr/include`
LIBDIR	gewöhnlich `/usr/ccs/lib`
LIBDIR`/libc.so`	
LIBDIR`/libc.a`	
LIBDIR`/libgen.a`	
LIBDIR`/libm.a`	
LIBDIR`/libsfm.sa`	
`/usr/lib/libc.so.1`	

SIEHE AUCH

`ar`(1), `cc`(1), `ld`(1), `lint`(1), `nm`(1), `intro`(2), `intro`(3M), `stdio`(3S), `math`(5).
Kapitel "C-Übersetzungssystem" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ERGEBNIS

Bei Funktionen, die Gleitkommawerte zurückgeben, hängt die Fehlerbehandlung vom Übersetzungsmodus ab. Im K&R-Modus (Standard) geben diese Funktionen die konventionellen Werte `0`, `±HUGE` oder `NaN` zurück, wenn die Funktion für die gegebenen Argumente undefiniert ist, oder wenn der Wert nicht darstellbar ist. Im ANSI-Modus wird `HUGE_VAL` statt `HUGE` zurückgegeben. `HUGE_VAL` und `HUGE` sind in `math.h` als unendlich bzw. die größte Zahl mit einfacher Genauigkeit definiert.

HINWEIS

Keine der Funktionen, externen Variablen oder Makros dürfen in Benutzerprogrammen neu definiert werden. Alle anderen Namen können neu definiert werden, ohne das Verhalten von anderen Bibliotheksfunktionen zu beeinflussen. Beachten Sie aber, daß solche neuen Definitionen Konflikte mit Deklarationen in anderen Include-Dateien verursachen können.

Die Include-Dateien in *INCDIR* stellen für die meisten in diesem Handbuch aufgelisteten Funktionen Funktionsprototypen (Funktionsdeklarationen, die die Typen von Argumenten einschließen) zur Verfügung. Mit Funktionsprototypen kann der Übersetzer die korrekte Verwendung dieser Funktionen im Benutzerprogramm überprüfen. Der Programmprüfer `lint` kann ebenfalls verwendet werden. Er gibt selbst dann Unstimmigkeiten aus, wenn die Include-Dateien gar nicht mit `#include`-Anweisungen eingeschlossen sind. Definitionen für die Abschnitte 2, 3C und 3S werden automatisch überprüft. Andere Definitionen können durch die Option `-l` bei `lint` eingeschlossen werden. Zum Beispiel schließt `-lm` Definitionen für `libm` ein. Verwenden Sie daher auf jeden Fall den Programmprüfer `lint`.

Achten Sie aufmerksam auf den Unterschied zwischen STREAMS und *Stream*.
STREAMS ist eine Menge von Systemkern-Mechanismen, die die Entwicklung von Netzwerkdiensten und Datenkommunikationstreibern unterstützen. STREAMS besteht aus Dienstprogrammen, Systemkern-Einrichtungen und einer Reihe von Datenstrukturen. Ein *Stream* ist eine Datei mit dazugehörender Pufferung. Ein *Stream* ist als Zeiger auf den Typ `FILE` deklariert, der in `stdio.h` definiert ist.

a64l, l64a - Zahlen umwandeln

```
#include <stdlib.h>
long a64l (const char *s);
char *l64a (long l);
```

Diese Funktionen werden zum Verwalten von Zahlen verwendet, die in ASCII-Zeichen zur Basis 64 gespeichert sind. Diese Zeichen definieren eine Notation, mit der lange ganze Zahlen durch maximal sechs Zeichen dargestellt werden können; jedes Zeichen stellt eine 'Ziffer' in einer Schreibweise gemäß Basis 64 dar.

Die für die Darstellung von 'Ziffern' verwendeten Zeichen sind `.` für 0, `/` für 1, `0` bis einschließlich `9` für 2-11, `A` bis einschließlich `Z` für 12-37 und `a` bis einschließlich `z` für 38-63.

`a64l` erwartet einen Zeiger auf eine mit Null-Byte abgeschlossene Basis-64-Darstellung und gibt den entsprechenden `long`-Wert zurück. Wenn die Zeichenkette, auf die `s` zeigt, mehr als sechs Zeichen enthält, verwendet `a64l` die ersten sechs Zeichen.

`a64l` durchläuft die Zeichenkette von links nach rechts (mit der kleinsten signifikanten Ziffer links) und decodiert jedes Zeichen als 6-Bit Zahl zur Basis 64.

`l64a` erwartet ein `long`-Argument und gibt einen Zeiger auf die entsprechende Basis-64-Darstellung zurück. Wenn das Argument 0 ist, gibt `l64a` einen Zeiger auf eine Nullzeichenkette zurück.

HINWEIS

Der von `l64a` zurückgegebene Wert ist ein Zeiger in einen statischen Puffer, dessen Inhalt bei jedem Aufruf überschrieben wird.

abort - Erzeugung eines Signals für unnormale Beedigung

```
#include <stdlib.h>
void abort (void);
```

`abort` schließt, wenn möglich, zunächst alle offenen Dateien, `stdio`(3S)-Streams, Dateiverzeichnis-Streams und Nachrichtenverzeichnis-Deskriptoren und schickt das Signal `SIGABRT` an den aufrufenden Prozeß.

SIEHE AUCH

`sdb`(1), `exit`(2), `kill`(2), `signal`(2), `catopen`(3C), `stdio`(3S).
`sh`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Wenn `SIGABRT` weder abgefangen noch ignoriert wird und das aktuelle Dateiverzeichnis beschreibbar ist, wird ein Speicherabzug erzeugt und die Meldung
`abort - core dumped` - von der Shell geschrieben (siehe `sh`(1)).

abs, labs - ganzzahligen Absolutwert liefern

```
#include <stdlib.h>
int abs (int val);
long labs (long lval);
```

`abs` liefert den Absolutwert seines `int`-Operanden. `labs` liefert den Absolutwert seines `long`-Operanden zurück.

SIEHE AUCH:

`floor`(3M).

HINWEIS

In der Darstellung als Zweierkomplement ist der größte negative Absolutwert der ganzen Zahl undefiniert. Dieser Fehler wird bei einigen Implementierungen abgefangen und bei anderen einfach ignoriert.

addseverity - Liste mit Warnstufen erstellen

```
#include <fmtmsg.h>
int addseverity(int severity, const char *string);
```

Die Funktion `addseverity` erstellt für eine Anwendung eine Liste mit Warnstufen, welche mit dem Meldungsformatierer `fmtmsg` verwendet wird. *severity* ist eine ganze Zahl, die die Dringlichkeit der Bedingung angibt, und *string* ist ein Zeiger auf eine Zeichenkette, welche die Bedingung beschreibt. Die Zeichenkette ist dabei nicht auf eine bestimmte Länge beschränkt.

Wenn `addseverity` mit einem Zahlenwert aufgerufen wird, der vorher nicht definiert wurde, nimmt die Funktion diesen neuen Zahlenwert auf und schreibt die Zeichenkette in die bestehende Menge der Warnstufen.

Wird `addseverity` mit einem Zahlenwert aufgerufen, der vorher bereits definiert wurde, ordnet die Funktion diesem Zahlenwert die neue Zeichenkette zu. Definierte Warnstufen können durch eine `NULL`-Zeichenkette gelöscht werden. Wenn `addseverity` mit einer negativen Zahl oder den Werten 0, 1, 2, 3, oder 4 aufgerufen wird, schlägt die Funktion fehl und liefert -1 zurück. Die Werte 0-4 sind für die standardmäßigen Warnstufen reserviert und können nicht verändert werden. Die Bezeichner für die standardmäßigen Warnstufen sind:

`MM_HALT` zeigt an, daß die Anwendung auf einen schwerwiegenden Fehler gestoßen ist und die Bearbeitung anhält. Die Zeichenkette `HALT` wird ausgegeben.

`MM_ERROR` zeigt an, daß die Anwendung einen Fehler erkannt hat. Die Zeichenkette `ERROR` wird ausgegeben.

`MM_WARNING` zeigt an, daß ein außergewöhnlicher Zustand eingetreten ist, der ein Problem darstellen könnte und daher beobachtet werden sollte. Die Zeichenkette `WARNING` wird ausgegeben.

`MM_INFO` liefert Informationen über einen Zustand, der keinen Fehler darstellt. Die Zeichenkette `INFO` wird ausgegeben.

`MM_NOSEV` zeigt an, daß für die Meldung keine Warnstufe angegeben wurde.

Warnstufen können auch zur Laufzeit über die Umgebungsvariable `SEV_LEVEL` angegeben werden (siehe `fmtmsg`(3C)).

BEISPIELE

Wird die Funktion `addseverity` wie folgt verwendet:

```
addseverity(7,"FEHLER")
```

liefert der folgende Aufruf von `fmtmsg`:

```
fmtmsg(MM_PRINT, "UX:cat", 7, "falsche Syntax", "siehe Handbuch",
"UX:cat:001")
```

das folgende Ergebnis:

```
UX:cat: FEHLER: falsche Syntax
TO FIX: siehe Handbuch   UX:cat:001
```

SIEHE AUCH

`fmtmsg`(1M), `fmtmsg`(3C), `gettxt`(3C), `printf`(3S).

ERGEBNIS

`addseverity` liefert bei Erfolg `MM_OK` und `MM_NOTOK` bei Fehler.

atexit - Beendigungsroutine einem Programm hinzufügen

```
#include <stdlib.h>
int atexit (void (*func)(void));
```

`atexit` fügt die Funktion *func* einer Funktionsliste hinzu, die ohne Argumente bei normaler Beendigung des Programms aufgerufen wird. Das Programm wird entweder durch Aufruf von `exit` oder durch Rückkehr aus `main` normal beendet. Bis zu 32 Funktionen können durch `atexit` registriert werden; die Funktionen werden in der umgekehrten Reihenfolge aufgerufen, in der sie registriert werden.

`atexit` liefert 0, wenn die Registrierung erfolgreich war, und einen Wert ungleich null, wenn ein Fehler auftritt.

SIEHE AUCH

`exit`(2).

bsearch - sortierte Tabelle binär absuchen

```
#include <stdlib.h>
void *bsearch (const void *key, const void *base, size_t nel,
    size_t size, int (*compar)(const void *, const void *));
```

bsearch ist eine binäre Suchfunktion, eine verallgemeinerte Version des Algorithmus (6.2.1) B von Knuth. Sie gibt einen Zeiger zurück, der auf die Stelle in der Tabelle (im Feld) zeigt, an der ein gegebener Wert gefunden werden kann. Sie gibt einen Nullzeiger zurück, falls der Wert nicht gefunden werden kann. Die Tabelle muß vorher entsprechend einer bereitgestellten Vergleichsfunktion, auf die *compar* zeigt, in ansteigender Reihenfolge sortiert werden. *key* weist auf einen Bezugswert, der in der Tabelle zu suchen ist. *base* weist auf das Element am Anfang der Tabelle. *nel* ist die Anzahl der Elemente in der Tabelle. Die Funktion, auf die *compar* zeigt, wird mit zwei Argumenten, die auf zu vergleichende Elemente zeigen, aufgerufen. Die Funktion muß eine ganze Zahl liefern, die kleiner als, gleich, oder größer als Null ist, je nachdem, ob das erste Argument kleiner als, gleich oder größer als das zweite Argument ist.

BEISPIEL

In diesem Beispiel wird eine Tabelle durchsucht, die Zeiger auf Knoten enthält, die aus einer Zeichenkette und ihrer Länge bestehen. Die Tabelle ist alphabetisch nach der Zeichenkette im Knoten geordnet, auf die jeder Eintrag zeigt.

Dieses Programm liest Zeichenketten ein und findet entweder den entsprechenden Knoten und gibt die Zeichenkette und die Länge aus, oder es gibt eine Fehlermeldung aus.

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

struct node {                   /* Diese werden in der Tabelle */
      char *string;             /* gespeichert */
      int length;
};
static struct node table[] =  /* zu durchsuchende Tabelle */
{
     { "asparagus", 10 },
     { "beans", 6 },
     { "tomato", 7 },
     { "watermelon", 11 },
};

main()
{
      struct node *node_ptr, node;
      /* Routine um zwei Konten zu vergleichen */
      static int node_compare(const void *, const void *);
      char str_space[20];  /* Speicherplatz zum Einlesen der Zeichenkette */
```

```
        node.string = str_space;
        while (scanf("%20s", node.string) != EOF) {
                node_ptr = bsearch( &node,
                                table, sizeof(table)/sizeof(struct node),
                                sizeof(struct node), node_compare);
                if (node_ptr != NULL) {
                        (void) printf("string = %20s, length = %d\n"
                                node_ptr->string, node_ptr->length);
                } else {
                        (void)printf("not found: %20s\n", node.string)
                }
        }
        return(0);
}

/* Diese Routine vergleicht zwei Knoten auf Grundlage einer */
/* alphabetischen Ordnung des Zeichenkettenfelds */
static int
node_compare(const void *node1, const void *node2)
{
        return (strcmp(
                        ((const struct node *)node1)->string,
                        ((const struct node *)node2)->string));
}
```

SIEHE AUCH

hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C).

ERGEBNIS

Ein Nullzeiger wird zurückgegeben, wenn das gesuchte Element nicht in der Tabelle gefunden werden kann, sonst wird ein Zeiger, der auf die Stelle in der Tabelle zeigt, zurückgegeben.

HINWEIS

Die Zeiger auf *key* und das Element am Anfang der Tabelle sollten vom Typ 'Zeiger-auf-*Element*' sein.

Die Vergleichsfunktion muß nicht jedes Byte vergleichen, deshalb können die Elemente zusätzlich zu den Vergleichswerten willkürliche Daten enthalten.

Wenn die Anzahl von Elementen in der Tabelle kleiner als die für die Tabelle reservierte Größe ist, sollte *nel* die kleinere Zahl sein.

catgets - Programmeldung lesen

```
#include <nl_types.h>
char *catgets (nl_catd catd, int set_num, int msg_num, char *s);
```

`catgets` versucht, die Meldung *msg_num* aus der Menge *set_num* aus dem Meldungskatalog *catd* zu lesen. *catd* ist ein Verzeichnisdeskriptor, der von einem vorherigen Aufruf von `catopen` zurückgegeben wird. *s* zeigt auf eine voreingestellte Meldungszeichenkette, welche durch `catgets` zurückgeliefert wird, wenn der angegebene Meldungskatalog momentan nicht verfügbar ist.

SIEHE AUCH

`catopen`(3C).

ERGEBNIS

Wenn die angegebene Meldung erfolgreich gelesen werden kann, liefert `catgets` einen Zeiger auf einen internen Pufferbereich, der die nullterminierte Meldungszeichenkette enthält. Ist der Aufruf nicht erfolgreich, weil der Meldungskatalog *catd* momentan nicht verfügbar ist, wird ein Zeiger auf *s* zurückgeliefert.

catopen, catclose - Meldungskatalog öffnen und schließen

```
#include <nl_types.h>
nl_catd catopen (char *name, int oflag);
int catclose (nl_catd catd);
```

`catopen` öffnet ein Verzeichnis mit Meldungen und liefert einen Verzeichnisdeskriptor zurück. *name* gibt den Namen des zu verwendenden Meldungskataloges an, der geöffnet werden soll. Wenn *name* einen Schrägstrich `/` enthält, dann gibt *name* den Pfadnamen für einen Meldungskatalog an. Ansonsten wird die Umgebungsvariable `NLSPATH` verwendet. Wenn `NLSPATH` in der Umgebung nicht existiert oder der Meldungskatalog unter einem in `NLSPATH` definierten Pfad nicht geöffnet werden kann, wird der voreingestellte Pfad verwendet (siehe `nl_types`(5)).

Die Namen der Meldungskataloge und ihre Positionen auf dem Speichermedium sind je nach System verschieden. Einzelne Anwendungen können Meldungskataloge nach eigenen Gesichtspunkten benennen und anlegen. Daher wird ein Mechanismus benötigt, der genau angibt, wo sich der Meldungskatalog befindet.

Die Variable `NLSPATH` liefert sowohl die Position von Meldungskatalogen in der Form eines Suchpfads, als auch die Namenskonventionen, die mit den Meldungskatalogen verknüpft sind. Zum Beispiel:

```
NLSPATH=/nlslib/%L/%N.cat:/nlslib/%N/%L
```

Das Metazeichen `%` zeigt ein Substitutionsfeld an, wobei `%L` die aktuelle Einstellung der Umgebungsvariablen `LANG` (siehe folgenden Abschnitt) ersetzt und `%N` den Wert des Parameters *name* ersetzt, der an `catopen` übergeben wird. Im obigen Beispiel sucht `catopen` daher zuerst in `/nlslib/$LANG/`*name*`.cat` und dann in `/nlslib/`*name*`/$LANG` nach dem angegebenen Meldungskatalog.

`NLSPATH` wird üblicherweise systemweit eingestellt (z.B. in `/etc/profile`) und macht daher die Positionierungs- und Namenskonventionen für die Meldungskataloge sowohl für die Programme als auch die Benutzer transparent.

Der komplette Satz der Metazeichen umfaßt folgende Symbole:

`%N`	Wert des Namensparameters, der an `catopen` übergeben wird
`%L`	Wert von `LANG`
`%l`	Wert des Sprachenelements aus `LANG`
`%t`	Wert des Landelements aus `LANG`
`%c`	Wert des Zeichensatzelements aus `LANG`
`%%`	das Zeichen %

Die Umgebungsvariable LANG bietet die Möglichkeit, die Muttersprache, lokale Besonderheiten und den Zeichensatz des Benutzers in Form einer ASCII-Zeichenkette anzugeben:

LANG=*Sprache[_Territorialkennung[.Zeichensatz]]*

Ein Benutzer aus Österreich, der Deutsch spricht und ein Terminal mit dem ISO 8859/1-Zeichensatz verwendet, stellt die Variable LANG auf den folgenden Wert:

```
LANG=De_A.88591
```

Auf diese Weise sollte es einem Benutzer möglich sein, entsprechende relevante Meldungskataloge zu finden, vorausgesetzt, sie existieren.

Sollte die Variable LANG nicht gesetzt sein, dann wird der Wert von LC_MESSAGES verwendet, wie er von setlocale zurückgeliefert wird. Ist dieser Wert NULL, dann wird der voreingestellte Pfad verwendet, der in nl_types definiert wird.

oflag ist für zukünftige Erweiterungen reserviert und sollte auf 0 gesetzt werden. Wird dieser Parameter auf einen anderen Wert gesetzt, so ist das Resultat nicht definiert.

catclose schließt den Meldungskatalog *catd*.

SIEHE AUCH

catgets(3C), setlocale(3C), environ(5), nl_types(5).

ERGEBNIS

Bei Erfolg liefert catopen einen Deskriptor für den Meldungskatalog zurück, welcher beim späteren Aufruf von catgets und catclose verwendet werden kann. Ansonsten liefert catopen den Wert (nl_catd) -1 zurück.

catclose liefert bei Erfolg 0 zurück; tritt ein Fehler auf, wird -1 zurückgegeben.

clock - verbrauchte CPU-Zeit melden

```
#include <time.h>
clock_t clock(void);
```

`clock` gibt die CPU-Zeit in Mikrosekunden zurück, die seit dem ersten Aufruf von `clock` verbraucht wurde. Die gemeldete Zeitspanne ist die Summe der Benutzer- und Systemzeit des Aufrufprozesses und seiner beendeten Sohnprozesse, für die der `wait`-Systemaufruf, die `pclose`- oder `system`-Funktion ausgeführt wurde.

Wird der Wert, den `clock` zurückliefert, durch die in der Include-Datei `time.h` definierte Konstante `CLOCKS_PER_SEC` geteilt, ergibt das die Zeit in Sekunden.

Die Auflösung des Zeitakts ist hardwareabhängig.

SIEHE AUCH

`times`(2), `wait`(2), `popen`(3S), `system`(3S).

HINWEIS

Der von `clock` zurückgegebene Wert wird wegen der Kompatibilität zu Systemen, die CPU-Takte mit sehr viel höheren Auflösungen haben, in Mikrosekunden definiert. Deshalb beginnt der zurückgegebene Wert bereits nach 2147 Sekunden CPU-Zeit (ca. 36 Minuten) zyklisch wieder von vorne. Wenn die verbrauchte Prozeßzeit nicht verfügbar oder darstellbar ist, liefert `clock` den Wert `(clock_t)-1` zurück.

conv - Zeichen umwandeln

```
#include <ctype.h>
int toupper (int c);
int tolower (int c);
int _toupper (int c);
int _tolower (int c);
int toascii (int c);
```

`toupper` und `tolower` haben denselben Definitions- und Wertebereich wie `getc`(3S): die ganzen Zahlen von -1 bis 255. Wenn das Argument von `toupper` einen Kleinbuchstaben darstellt, ist der entsprechende Großbuchstabe das Ergebnis. Wenn das Argument von `tolower` einen Großbuchstaben darstellt, ist der entsprechende Kleinbuchstabe das Ergebnis. Alle anderen Argumente des Definitionsbereichs werden unverändert zurückgegeben.

Die Makros `_toupper` und `_tolower` haben die gleiche Wirkung wie `toupper` und `tolower`. Sie weisen jedoch eingeschränkte Definitionsbereiche auf und sind schneller. `_toupper` benötigt einen Kleinbuchstaben als Argument; das Ergebnis ist der entsprechende Großbuchstabe. Das Makro `_tolower` benötigt einen Großbuchstaben als Argument; das Ergebnis ist der entsprechende Kleinbuchstabe. Außerhalb des Definitionsbereichs liegende Argumente liefern undefinierte Ergebnisse.

Bei dem von `toascii` gelieferten Argument sind alle Bits ausgeschaltet, die nicht Teil eines 7-Bit ASCII-Standardzeichens sind; dadurch wird die Kompatibilität zu anderen Systemen erreicht.

SIEHE AUCH

`ctype`(3C), `getc`(3S).

crypt, setkey, encrypt - Zeichenketten verschlüsseln

```
#include <crypt.h>
char *crypt (const char *key, const char *salt);
void setkey (const char *key);
void encrypt (char *block, int edflag);
```

`crypt` ist die Paßwort-Verschlüsselungsfunktion. Sie beruht auf einem Einweg-Verschlüsselungsalgorithmus mit Variationen, die (unter anderem) die Anwendung von Hardware-Implementierungen für eine Schlüsselsuche verhindern sollen.

key ist die zu verschlüsselnde Eingabefolge, zum Beispiel das Paßwort eines Benutzers. *salt* ist eine Zeichenkette der Länge zwei aus der Menge `a-zA-Z0-9./`. Diese Zeichenkette wird zur Veränderung des Verschlüsselungsalgorithmus auf eine von 4096 verschiedenen Arten verwendet; danach wird die Eingabefolge als Schlüssel zum wiederholten Verschlüsseln einer konstanten Zeichenkette benutzt. Der jeweils zurückgegebene Wert zeigt auf die verschlüsselte Zeichenkette. Die ersten beiden Zeichen des Rückgabewerts sind *salt* selbst.

Die Funktionen `setkey` und `encrypt` ermöglichen einen (recht einfachen) Zugriff auf den aktuellen Verschlüsselungsalgorithmus. Das Argument von `setkey` ist ein Zeichenfeld mit einer Länge von 64, das nur die Zeichen mit den numerischen Werten 0 und 1 enthält. Diese Zeichenkette wird in Gruppen von je acht Bits aufgeteilt, dabei wird das niederwertige Bit in jeder Gruppe ignoriert; hieraus ergibt sich ein Schlüssel mit 56 Bits, der eingetragen wird. Dies ist dann der Schlüssel, der von dem Algorithmus zum Verschlüsseln der Zeichenkette *block* bei der Funktion `encrypt` verwendet wird.

Das Argument *block* für `encrypt` ist ein Zeichenfeld der Länge 64, das nur Zeichen mit den Werten 0 und 1 enthält. Das Argumentfeld wird in ein ähnliches Feld geändert, das die Bits des Arguments enthält, nachdem sie unter Verwendung des von `setkey` gesetzten Schlüssels durch den Verschlüsselungsalgorithmus verändert wurden. Das Argument *edflag*, das eher die Entschlüsselung als die Verschlüsselung anzeigt, wird ignoriert; verwenden Sie `encrypt` in `libcrypt` (siehe `crypt`(3X)) für die Entschlüsselung.

SIEHE AUCH

`getpass`(3C), `crypt`(3X), `passwd`(4).
`login`(1), `passwd`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Wenn *edflag* auf etwas anderes als Null gesetzt wird, wird `errno` auf `ENOSYS` gesetzt. Der Rückgabewert von `crypt` weist auf Daten der Speicherklasse *static*, die bei jedem Aufruf überschrieben werden.

ctermid - Dateinamen für Terminal erstellen

```
#include <stdio.h>
char *ctermid (char *s);
```

`ctermid` ermittelt den Pfadnamen des steuernden Terminals des aktuellen Prozesses und speichert ihn in einer Zeichenkette.

Wenn *s* ein Nullzeiger ist, wird die Zeichenkette in einem internen statischen Bereich gespeichert, dessen Inhalt beim nächsten Aufruf von `ctermid` überschrieben und dessen Adresse zurückgegeben wird. Andernfalls wird angenommen, daß *s* auf ein Zeichenfeld mit wenigstens `L_ctermid` Elementen zeigt; der Pfadname wird in dieses Feld geschrieben und der Wert von *s* zurückgegeben. Die Konstante `L_ctermid` ist in der Include-Datei `stdio.h` definiert.

SIEHE AUCH

`ttyname`(3C).

HINWEIS

Folgender Unterschied besteht zwischen `ctermid` und `ttyname`(3C): `ttyname` erwartet einen Dateideskriptor und liefert den aktuellen Namen des zu diesem Dateideskriptor gehörenden Terminals, während `ctermid` eine Zeichenkette (`/dev/tty`) zurückgibt, die sich, wenn sie als Dateiname verwendet wird, auf das Terminal bezieht. Daher ist `ttyname` nur von Nutzen, wenn der Prozeß bereits wenigstens eine Datei für ein Terminal geöffnet hat.

ctime - Datum und Zeit in Zeichenkette umwandeln

```
#include <time.h>
char *ctime (const time_t *clock);
struct tm *localtime (const time_t *clock);
struct tm *gmtime (const time_t *clock);
char *asctime (const struct tm *tm);
extern time_t timezone, altzone;
extern int daylight;
extern char *tzname[2];
void tzset (void);
```

`ctime`, `localtime` und `gmtime` akzeptieren Argumente vom Typ `time_t`, auf die *clock* zeigt, und die Zeit in Sekunden seit 00:00:00 UTC (Coordinated Universal Time), 1.Januar 1970 liefern. `ctime` liefert einen Zeiger auf eine 26 Zeichen lange Zeichenkette in folgender Form: Zeitzonen- und Sommerzeit-Korrekturen werden durchgeführt, bevor die Zeichenkette generiert wird. Alle Felder haben eine konstante Länge:

```
Fri Sep 13 00:00:00 1986\n\0
```

`localtime` (Ortszeit) und `gmtime` geben Zeiger auf `tm`-Strukturen zurück, die nachstehend erläutert werden. `localtime` berücksichtigt Zeitzonen und eventuelle Sommerzeit-Korrekturen; `gmtime` wandelt direkt in die UTC, d.h. die vom SINIX-System benutzte Zeit, um.

`asctime` wandelt eine `tm`-Struktur in eine Zeichenkette aus 26 Zeichen um, wie in obigem Beispiel gezeigt, und gibt einen Zeiger auf die Zeichenkette zurück.

In der Include-Datei `time.h` sind die Vereinbarungen aller Funktionen und externer Werte sowie der `tm`-Struktur enthalten. Die Strukturvereinbarung ist wie folgt:

```
struct      tm {
        int     tm_sec;         /* Sekunden - [0, 61] für übersprungene Sekunden */
        int     tm_min;         /* Minuten - [0, 59] */
        int     tm_hour;        /* Stunden - [0, 23] */
        int     tm_mday;        /* Tag des Monats - [1, 31] */
        int     tm_mon;         /* Monate - [0, 11] */
        int     tm_year;        /* Jahre seit 1900 */
        int     tm_wday;        /* Tage seit Sonntag - [0, 6] */
        int     tm_yday;        /* Tage seit dem 1. Januar - [0, 365] */
        int     tm_isdst;       /* Option für Sommerzeit */
};
```

`tm_isdst` ist positiv, wenn Sommerzeit eingestellt ist, null, wenn Sommerzeit nicht eingestellt ist, und negativ, wenn die Information nicht verfügbar ist.

Die externe time_t-Variable altzone beinhaltet den Unterschied in Sekunden zwischen UTC und der alternativen Zeitzone. Die externe Variable timezone enthält den Unterschied in Sekunden zwischen UTC und der örtlichen Zeit. Die externe Variable daylight zeigt an, ob die Sommerzeit angezeigt werden soll. timezone und altzone sind standardmäßig 0 (UTC). Die externe Variable daylight ist ungleich null, wenn eine alternative Zeitzone existiert. Die Zeitzonennamen sind in der externen Variable tzname, enthalten, die standardmäßig auf

```
char *tzname[2] = { "GMT", "      " };
```

gesetzt wird. Diese Funktionen wissen über die Eigenarten dieser Konvertierungen für verschiedene Zeitperioden in den Vereinigten Staaten (insbesondere in den Jahren 1974, 1975, und 1987) bescheid. Sie behandeln die neue Sommerzeit, beginnend mit dem ersten Sonntag im April 1987.

tzset benutzt den Inhalt der Umgebungsvariablen TZ, um die Werte unterschiedlicher externer Variablen zu überschreiben. Die Funktion tzset wird von asctime oder aber auch vom Benutzer aufgerufen. In der Beschreibung von environ(5) finden Sie nähere Angaben zu der Umgebungsvariablen TZ.

tzset überprüft den Inhalt der Umgebungsvariablen und weist die verschiedenen Felder den entsprechenden Variablen zu. Zum Beispiel lautet der vollständigste Eintrag für New Jersey 1986

```
EST5EDT4,116/2:00:00,298/2:00:00
```

oder einfach nur

```
EST5EDT
```

Ein Beispiel für die Südhalbkugel, zum Beispiel Cook Islands, könnte sein

```
KDT9:30KST10:00,63/5:00,302/20:00
```

In der langen Version des New Jersey-Beispiels von TZ, ist tzname[0] EST, timezone wird auf 5*60*60 gesetzt, tzname[1] ist EDT, altzone wird auf 4*60*60 gesetzt, die Sommerzeit beginnt am 117. Tag um 2 Uhr nachts und endet am 299. Tag um 2 Uhr nachts (es wird der Julianische Kalender benutzt). daylight wird auf einen positiven Wert gesetzt. Start- und Endzeit sind relativ zur Sommerzeit. Wenn Start- und Enddatum der Sommerzeit nicht geliefert werden, werden die für die Vereinigten Staaten in diesem Jahr gültigen Tage benutzt, und die Zeit wird 2 Uhr nachts sein. Wenn nur die Zeit nicht verfügbar ist, wird diese auf 2 Uhr nachts gesetzt.
Die Auswirkungen von tzset sind so, daß die Werte der externen Variablen timezone, altzone, daylight, und tzname geändert werden. ctime, localtime, mktime, und strftime werden ebenso diese externen Variablen aktualisieren, als hätten sie tzset zu der Zeit aufgerufen, die vom time_t- oder dem von ihnen konvertierten struct-tm-Wert spezifiziert wird.

Beachten Sie, daß in den meisten Installationen `TZ` standardmäßig durch die `/etc/profile`-Datei auf den korrekten Wert gesetzt wird, wenn der Benutzer sich einloggt (siehe `profile`(4) und `timezone`(4)).

DATEIEN

`/usr/lib/locale/`*language*`/LC_TIME`	enthält umgebungsspezifische Datums- und Zeitinformationen.

SIEHE AUCH

`time`(2), `getenv`(3C), `mktime`(3C), `putenv`(3C), `printf`(3S), `setlocale`(3C), `strftime`(3C), `cftime`(4), `profile`(4), `timezone`(4), `environ`(5).

HINWEIS

Die Ergebniswerte für `ctime`, `localtime`, und `gmtime` zeigen auf statische Daten, die bei jedem Aufruf überschrieben werden.

Das Ändern der Zeit während des Zeitraums der Änderung von `timezone` nach `altzone` oder umgekehrt kann unvorhersehbare Ergebnisse hervorrufen. Der Systemverwalter muß das Start- und Enddatum der Sommerzeit jährlich ändern, wenn das Format des Julianischen Kalenders verwendet wird.

ctype - Zeichen bearbeiten

```
#include <ctype.h>
int isalpha(int c);
int isupper(int c);
int islower(int c);
int isdigit(int c);
int isxdigit(int c);
int isalnum(int c);
int isspace(int c);
int ispunct(int c);
int isprint(int c);
int isgraph(int c);
int iscntrl(int c);
int isascii(int c);
```

Diese Funktionen klassifizieren als Zeichen codierte Integer-Werte. Jede von ihnen liefert einen Wert ungleich Null für wahr oder Null für falsch. Das Verhalten dieser Makros, mit Ausnahme von `isascii`, wird von der aktuellen Umgebung beeinflußt (siehe setlocale(3C)). Um das Verhalten zu verändern, ändern Sie die `LC_TYPE`-Kategorie in `setlocale` mit `setlocale(LC_TYPE, `*newlocale*`)`. In der `C`-Umgebung oder in einer Umgebung, in der Zeichentypinformationen nicht definiert sind, werden Zeichen entsprechend den Regeln des 7-Bit codierten US-ASCII-Zeichensatzes klassifiziert.

Das Makro `isascii` ist für alle Integer-Werte definiert; die anderen sind nur definiert, wenn das Argument ein `int` ist, dessen Wert als `unsigned char` darstellbar ist oder als `EOF`, welches in der Include-Datei `stdio.h` definiert ist und das Dateiende bedeutet.

`isalpha` untersucht auf Zeichen, für die `isupper` oder `islower` wahr ist, oder auf alle anderen Zeichen, die zu einem implementierungsabhängigen Zeichensatz gehören, und für die weder `iscntrl`, `isdigit`, `ispunct` noch `isspace` wahr sind. In der C-Umgebung liefert `isalpha` wahr nur für Zeichen, für die `isupper` oder `islower` wahr ist.

`isupper` untersucht auf Zeichen, die Großbuchstaben sind, oder auf alle anderen Zeichen, die zu einem implementierungsabhängigen Zeichensatz gehören, und für die weder `iscntrl`, `isdigit`, `ispunct`, `isspace` noch `islower` wahr ist. In der C-Umgebung liefert `isupper` nur für solche Zeichen wahr, die als US-ASCII-Großbuchstaben definiert sind.

`islower` untersucht auf Zeichen, die Kleinbuchstaben sind, oder auf alle anderen Zeichen, die zu einem implementierungsabhängigen Zeichensatz gehören, und für die weder `iscntrl`, `isdigit`, `ispunct`, `isspace` noch `isupper` wahr ist. In der `C`-Umgebung liefert `islower` wahr nur für Zeichen, die als US-ASCII-Kleinbuchstaben definiert sind.

`isdigit` untersucht, ob das Zeichen eine Dezimalzahl ist

`isxdigit` untersucht, ob das Zeichen eine Hexadezimalzahl ist (`[0-9]`, `[A-F]` oder `[a-f]`).

`isalnum` untersucht auf Zeichen, für die `isalpha` oder `isdigit` wahr ist (Buchstaben oder Ziffern).

`isspace` untersucht auf Leerzeichen, Tabulator, Return-Zeichen, Neue-Zeile-Zeichen, vertikalen Tabulator oder Seitenvorschub (Standard-Leerzeichen) oder alle anderen Zeichen, die zu einem implementierungsabhängigen Zeichensatz gehören und für die `isalnum` falsch ist. In der `C`-Umgebung liefert `isspace` wahr nur für die Standard-Leerzeichen.

`ispunct` untersucht auf alle darstellbaren Zeichen, die weder ein Leerzeichen noch ein Zeichen, für das `alphanum` wahr ist, sind.

`isprint` untersucht auf alle darstellbaren Zeichen einschließlich des Leerzeichens.

`isgraph` untersucht auf alle darstellbaren Zeichen ohne das Leerzeichen.

`iscntrl` untersucht auf alle 'Kontrollzeichen', die im Zeichensatz definiert sind.

`isascii` untersucht auf alle ASCII-Zeichen (`0` bis `0177` einschließlich).

Alle Klassifikationsmakros und Konvertierungsfunktionen benutzen Tabellen.

Es existieren Funktionen für alle oben beschriebenen Makros. Um einen Funktionsaufruf zu erzeugen, muß die Definition des Makronamen rückgängig gemacht werden (z.B. `#undef isdigit`).

DATEIEN

`/usr/lib/locale/`*`locale`*`/LC_CTYPE`

SIEHE AUCH

`chrtbl`(1M), `setlocale`(3C), `stdio`(3S), `ascii`(5), `environ`(5).

ERGEBNIS

Wenn das Argument für eines dieser Makros nicht in dessen Definitionsbereich liegt, ist das Ergebnis undefiniert.

cuserid - Zeichenkette mit Benutzernamen erzeugen

```
#include <stdio.h>
char *cuserid (char *s);
```

`cuserid` erzeugt eine Zeichenkettendarstellung des Login-Namens, unter dem der Eigentümer des aktuellen Prozesses angemeldet ist. Wenn *s* ein Nullzeiger ist, wird diese Darstellung in einem internen statischen Bereich erstellt, dessen Adresse zurückgegeben wird. Andernfalls wird angenommen, daß *s* auf ein Feld mit wenigstens `L_cuserid` Zeichen zeigt; die Darstellung wird in dieses Feld geschrieben. Die Konstante `L_cuserid` ist in der Include-Datei `stdio.h` definiert.

SIEHE AUCH

`getlogin`(3C), `getpwent`(3C).

ERGEBNIS

Wenn der Login-Name nicht gefunden werden kann, gibt `cuserid` einen Nullzeiger zurück; wenn *s* kein Nullzeiger ist, wird in diesem Fall bei *s* `[0]` das Null-Byte `\0` eingetragen.

difftime - Differenz zwischen zwei Kalenderdaten berechnen

```
#include <time.h>
double difftime (time_t time1, time_t time0);
```

`difftime` berechnet die Differenz zwischen zwei Kalenderdaten. `difftime` liefert die Differenz *(time1-time0)* in Sekunden vom Typ `double` zurück. Diese Funktion wird zur Verfügung gestellt, da es keine allgemeinen arithmetischen Operationen für den Typ `time_t` gibt.

SIEHE AUCH

`ctime`(3C).

directory - Dateiverzeichnis-Operationen

```
#include <sys/types.h> #include <dirent.h>
DIR *opendir (const char *filename);
struct dirent *readdir (DIR *dirp);
long telldir (DIR *dirp);
void seekdir (DIR *dirp, long loc);
void rewinddir (DIR *dirp);
int closedir (DIR *dirp);
```

`opendir` öffnet ein Dateiverzeichnis *filename* und ordnet ihm eine DIR-Struktur zu. `opendir` gibt einen Zeiger auf die DIR-Struktur zurück, der zur Identifizierung des Dateiverzeichnisses in den nachfolgenden Operationen verwendet wird. Der Nullzeiger wird zurückgegeben, wenn auf *filename* nicht zugegriffen werden kann, oder wenn *filename* kein Dateiverzeichnis ist, oder wenn nicht genügend Speicherplatz zur Aufnahme einer DIR-Struktur bzw. eines Puffers für die Dateiverzeichnis-Einträge mit `malloc`(3C) zur Verfügung gestellt werden kann.

`readdir` gibt einen Zeiger auf den nächsten aktiven Dateiverzeichnis-Eintrag zurück und positioniert den Dateiverzeichnis-Stream auf den nächsten Eintrag. Nichtaktive Einträge werden nicht zurückgegeben. Wird das Ende des Dateiverzeichnisses erreicht oder eine ungültige Position im Dateiverzeichnis festgestellt, dann wird der Nullzeiger zurückgegeben. `readdir` puffert verschiedene Dateiverzeichnis-Einträge für die jeweils aktuelle Lese-Operation; `readdir` markiert zur Aktualisierung das `st_atime`-Feld des Dateiverzeichnisses bei jedem aktuellen Lesevorgang im Dateiverzeichnis.

`telldir` gibt die aktuelle Position in dem Dateiverzeichnis des angegebenen Dateiverzeichniszeigers an.

`seekdir` setzt die Position der nächsten `readdir`-Operation. Die neue Position greift auf die Position zurück, die mit dem Dateiverzeichnis-Stream zur Ausführungszeit der `telldir`-Operation, die *loc* lieferte, verbunden war. Die von `telldir` zurückgegebenen Werte sind nur dann richtig, wenn das Dateiverzeichnis nicht infolge von Verdichtung oder Erweiterung verändert wurde. Dies ist kein Problem bei System V, kann jedoch bei einigen Dateisystemtypen problematisch sein.

`rewinddir` setzt die Position des angegebenen Dateiverzeichnisses auf den Anfang des Dateiverzeichnisses zurück. Es veranlaßt den Dateiverzeichnis-Stream, sich ebenfalls auf den augenblicklichen Status des mit ihm verbundenen Dateiverzeichnisses zu beziehen, so wie `opendir` es täte.

`closedir` schließt das angegebene Dateiverzeichnis und gibt die DIR-Struktur frei.

Folgende Fehler können von diesen Operationen verursacht werden:

`opendir` liefert `NULL` bei einem Fehler und setzt `errno` auf einen der folgenden Werte:

`ENOTDIR` Eine Komponente von *filename* ist kein Dateiverzeichnis.

`EACCES` Eine Komponente von *filename* verweigert die Sucherlaubnis.

`EACCES` Die Lese-Erlaubnis für das spezifizierte Dateiverzeichnis wird verweigert.

`EMFILE` Die Maximalanzahl von Dateideskriptoren ist bereits geöffnet.

`ENFILE` Die System-Dateitabelle ist voll.

`EFAULT` *filename* weist über den zugewiesenen Adreßraum hinaus.

`ELOOP` Während der Übersetzung von *filename* wurden zu viele symbolische Verweise angetroffen.

`ENAMETOOLONG`
Die Länge des *filename*-Arguments ist größer als `PATH_MAX`, oder die Länge einer *filename*-Komponente übertrifft `NAME_MAX`, während `_POSIX_NO_TRUNC` gültig ist.

`ENOENT` Eine Komponente von *filename* existiert nicht oder ist ein Null-Pfadname.

`readdir` liefert `NULL` bei einem Fehler und setzt `errno` auf einen der folgenden Werte:

`ENOENT` Der aktuelle Schreib-/Lesezeiger für das Dateiverzeichnis befindet sich nicht an einem gültigen Eintrag.

`EBADF` Der dem Dateiverzeichnis-Stream zugeordnete Dateideskriptor ist nicht mehr gültig. Dieser Fehler entsteht, wenn der Dateiverzeichnis-Stream geschlossen wurde.

`telldir`, `seekdir` und `closedir` liefern -1 bei einem Fehler und setzen `errno` auf den folgenden Wert:

`EBADF` Der dem Dateiverzeichnis zugeordnete Dateideskriptor ist nicht mehr gültig. Dieser Fehler entsteht, wenn das Dateiverzeichnis geschlossen wurde.

BEISPIEL

Es folgt ein Beispielprogramm, das die Dateinamen aller Dateien im aktuellen Dateiverzeichnis ausgibt.

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

main()
{
        DIR *dirp;
        struct dirent *direntp;

        dirp = opendir( "." );
        while ( (direntp = readdir( dirp )) != NULL )
                (void)printf( "%s\n", direntp->d_name );
        closedir( dirp );
        return (0);
}
```

SIEHE AUCH

`getdents`(2), `dirent`(4).

HINWEIS

Da `rewinddir` als Makro realisiert ist, gibt es keine verwendbare Funktionsadresse.

div, ldiv - Quotienten und den Rest berechnen

```
#include <stdlib.h>
div_t div (int dividend, int divisor);
ldiv_t ldiv (long int dividend, long int divisor);
```

`div` berechnet den Quotienten und den Rest der Division des Dividenden *dividend* durch den Divisor *divisor*. Diese Funktion bietet eine wohldefinierte Semantik für die vorzeichenbehaftete, ganzzahlige Division und die Restoperation. Das Vorzeichen des resultierenden Quotienten ist das des algebraischen Quotienten, und falls die Division ungenau sein sollte, ist die Größe des resultierenden Quotienten die größte ganze Zahl, die kleiner als der algebraische Quotient ist. Kann das Ergebnis nicht dargestellt werden, ist das Verhalten undefiniert; ansonsten ist *quotient* * *divisor* + *rest* gleich *dividend*.

`div` liefert eine Struktur vom Typ `div_t`, welche sowohl den Quotienten als auch den Rest enthält:

```
typedef struct div_t {
        int     quot;   /*Quotient*/
        int     rem;    /*Rest*/
} div_t;
```

`ldiv` ist ähnlich wie `div`, mit der Ausnahme, daß die Argumente und Komponenten der zurückgegebenen Struktur (vom Typ `ldiv_t`) alle den Typ `long int` haben.

drand48 - gleichmäßig verteilte Pseudo-Zufallszahlen erzeugen

```
#include <stdlib.h>
double drand48 (void);
double erand48 (unsigned short xsubi[3]);
long lrand48 (void);
long nrand48 (unsigned short xsubi[3]);
long mrand48 (void);
long jrand48 (unsigned short xsubi[3]);
void srand48 (long seedval);
unsigned short *seed48 (unsigned short seed16v[3]);
void lcong48 (unsigned short param[7]);
```

Diese Familie von Funktionen erstellt Pseudo-Zufallszahlen unter Anwendung des bekannten linearen Kongruenzalgorithmus und der 48-Bit-Ganzzahlarithmetik.

Die Funktionen `drand48` und `erand48` liefern nichtnegative Gleitkommazahlen doppelter Genauigkeit, die gleichmäßig über das Intervall `[0.0, 1.0]` verteilt sind.

`lrand48` und `nrand48` liefern nichtnegative Ganzzahlen vom Typ `long`, die gleichmäßig über das Intervall $[0, 2^{31})$ verteilt sind.

Die Funktionen `mrand48` und `jrand48` liefern ganze Zahlen vom Typ `long` mit Vorzeichen, die gleichmäßig über das Intervall $[-2^{31}, 2^{31})$ verteilt sind.

Die Funktionen `srand48`, `seed48` und `lcong48` sind Funktionen für die Initialisierung; eine dieser Funktionen ist aufzurufen, bevor entweder `drand48`, `lrand48` oder `mrand48` aufgerufen wird. Standardanfangswerte werden automatisch eingesetzt, wenn `drand48`, `lrand48` oder `mrand48` ohne vorherigen Aufruf einer Initialisierungsfunktion aufgerufen werden; Sie sollten aber explizit initialisieren. Für die Funktionen `erand48`, `nrand48` und `jrand48` ist ein vorheriger Aufruf einer Initialisierungsfunktion nicht erforderlich.

Bei allen Funktionen wird eine Folge von ganzzahligen 48-Bit-Werten X_i auf Grundlage der linearen Kongruenzformel erstellt:

$$X_{n+1} = (aX_n + c) \bmod m \qquad n \geq 0$$

Der Parameter m ist 2^{48}; folglich wird ganzzahlige 48-Bit-Arithmetik ausgeführt. Wenn `lcong48` nicht aufgerufen wurde, erhält man den Multiplikatorwert `a` und den Summand `c` durch

$a = 5DEECE66D_{16} = 273673163155_8$

$c = B_{16} = 13_8$.

Der von einer der Funktionen drand48, erand48, lrand48, nrand48, mrand48 oder jrand48 gelieferte Wert wird berechnet, indem man erst das nächste 48-Bit X_i in der Folge berechnet. Dann wird die entsprechende Bitanzahl, die jeweils vom Typ des zurückzugebenden Datenelements abhängt, von den links stehenden höherwertigen Bits X_i kopiert und in den Rückgabewert umgewandelt.

Die Funktionen drand48, lrand48 und mrand48 speichern das zuletzt erzeugte 48-Bit X_i in einem internen Puffer. X_i muß vor Aufruf initialisiert werden. Für die Funktionen erand48, nrand48 und jrand48 muß Speicherplatz vom aufrufenden Programm für die aufeinanderfolgenden X_i-Werte in dem Feld bereitgestellt werden, das beim Aufrufen der Funktionen als Argument angegeben wurde. Diese Funktionen müssen nicht initialisiert werden; das aufrufende Programm muß den gewünschten Anfangswert von X_i in das Feld schreiben und als Argument übergeben. Durch die Verwendung verschiedener Argumente ermöglichen die Funktionen erand48, nrand48 und jrand48 separaten Modulen eines großen Programms die Erzeugung mehrerer *unabhängiger* Mengen von Pseudo-Zufallszahlen, d.h. die Folge der Zahlen in einer Menge hängt *nicht* davon ab, wie oft diese Funktion aufgerufen wurde, um Zufallszahlen für die anderen Mengen zu erzeugen.

Die Initialisierungsfunktion srand48 setzt die 32 höherwertigen Bits von X_i auf die in ihrem Argument enthaltenen 32 Bits. Die niederwertigen 16 Bits von X_i werden auf den Wert $330E_{16}$ gesetzt.

Die Initialisierungsfunktion seed48 setzt den Wert von X_i auf den 48-Bit-Wert, der im Argumentfeld angegeben wurde. Außerdem wird der vorherige Wert von X_i in einen internen 48-Bit-Speicher kopiert, der nur von seed48 benutzt wird. Ein Zeiger auf diesen Puffer ist der von seed48 zurückgegebene Wert. Dieser zurückgegebene Zeiger kann einfach ignoriert werden, wenn er nicht benötigt wird. Er ist jedoch nützlich, wenn ein Programm später einmal von einem bestimmten Punkt aus neugestartet werden soll. Verwenden Sie den Zeiger, um auf den letzten X_i-Wert zu gehen und diesen Wert zu speichern. Verwenden Sie den Wert dann zum Neuinitialisieren über seed48, wenn das Programm neugestartet wird.

Die Initialisierungsfunktion lcong48 ermöglicht dem Anwender die Angabe des anfänglichen X_i, des Multiplikatorwerts a und des Summanden c. Die Argumentfeldelemente *param[0-2]* geben Xi an, *param[3-5]* geben den Multiplikator a und *param[6]* den 16-Bit-Summanden c an. Nach dem Aufruf von lcong48 werden die oben angegebenen Werte für 'Standard'-Multiplikator und Summand, d.h. a und c, mit einem anschließenden Aufruf von entweder srand48 oder seed48 wieder hergestellt.

SIEHE AUCH

rand(3C).

dup2 - offenen Dateideskriptor duplizieren

```
#include <unistd.h>
int dup2 (int fildes, int fildes2);
```

fildes ist ein Dateideskriptor für eine offene Datei, und *fildes2* ist eine nichtnegative ganze Zahl, die kleiner als OPEN_MAX (die maximale Anzahl offener Dateien) ist. *dup2* veranlaßt *fildes2*, auf dieselbe Datei wie *fildes* zu verweisen. Wenn *fildes2* bereits auf eine offene Datei verweist, außer auf *fildes*, wird diese erst geschlossen. Wenn *fildes2* auf *fildes* verweist, oder wenn *fildes* kein gültiger offener Dateideskriptor ist, wird *fildes2* nicht zuerst geschlossen.

dup2 ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte wahr sind:

EBADF	*fildes* ist kein gültiger offener Dateideskriptor .
EBADF	*fildes2* ist negativ, größer als oder gleich OPEN-MAX.
EINTR	Ein Signal wurde während des dup2-Aufrufs aufgefangen.
EMFILE	OPEN-MAX-Dateideskriptoren sind zum gegebenen Zeitpunkt geöffnet.

SIEHE AUCH

creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2), lockf(3C), limits(4).

ERGEBNIS

Nach erfolgreicher Beendigung wird eine nichtnegative ganze Zahl, der Dateideskriptor, zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

ecvt, fcvt, gcvt - Gleitkommazahl in Zeichenkette umwandeln

```
#include <stdlib.h>
char *ecvt (double value, int ndigit, int *decpt, int *sign);
char *fcvt (double value, int ndigit, int *decpt, int *sign);
char *gcvt (double value, int ndigit, char *buf);
```

`ecvt` wandelt *value* in eine mit dem Null-Byte abgeschlossene Zeichenkette von *ndigit* Ziffern um und gibt einen Zeiger auf die Folge zurück. Die hochwertigste Ziffer ist ungleich Null, sofern der Wert nicht Null ist. Die niederwertigste Ziffer wird gerundet. Die Position des Dezimalpunkts in Relation zum Anfang der Zeichenkette wird indirekt durch *decpt* gespeichert; negativ bedeutet links von den zurückgegebenen Ziffern. Der Dezimalpunkt ist nicht in der zurückgegebenen Zeichenkette enthalten. Ist das Vorzeichen des Ergebnisses negativ, dann ist das Wort, auf das *sign* zeigt, ungleich Null; andernfalls ist es Null.

`fcvt` ist identisch mit `ecvt`, wobei jedoch die richtige Ziffer für die `printf %f`-Ausgabe abgerundet wird von der Anzahl von Ziffern, die durch *ndigit* angegeben ist.

`gcvt` wandelt den Wert *value* in eine mit dem Null-Byte abgeschlossene Zeichenkette in dem Feld um, auf das *buf* zeigt, und gibt *buf* zurück. Es versucht, nach Möglichkeit *ndigit*-signifikante Ziffern im `%f`-Format zu erstellen; andernfalls wird das druckbereite `%e`-Format (wissenschaftliche Notation) erstellt. Ein Minuszeichen (sofern vorhanden) oder ein Dezimalpunkt wird als Teil der zurückgegebenen Zeichenkette eingeschlossen. Nachstehende Nullen werden unterdrückt.

SIEHE AUCH

`printf`(3S).

HINWEIS

Die von `ecvt` und `fcvt` zurückgegebenen Werte zeigen auf dasselbe statische Datenfeld, dessen Inhalt bei jedem Aufruf überschrieben wird.

end, etext, edata - End-Adressen im Programm

```
extern etext;
extern edata;
extern end;
```

Diese Namen bezeichnen weder Funktionen noch interessante Adreßinhalte. Nur die Adressen selbst sind bedeutsam.

`etext` Die Adresse `etext` ist die erste Adresse oberhalb des Programmtexts.

`edata` Die Adresse `edata` ist die erste Adresse oberhalb des initialisierten Datenbereichs.

`end` Die Adresse `end` ist die erste Adresse oberhalb des nicht initialisierten Datenbereichs.

SIEHE AUCH

`cc`(1), `brk`(2), `malloc`(3C), `stdio`(3S).

HINWEIS

Bei Beginn der Ausführung fällt der Programmgrenzwert 'break' des Programms (die erste Stelle hinter den Daten) mit `end` zusammen, wobei jedoch der Speichergrenzwert durch die Funktionen `brk`(2), `malloc`(3C), die Standard-Ein-/Ausgabebibliothek (siehe `stdio`(3S)), durch die die Profil-Option (`-p`) von `cc` usw. verändert werden kann. Daher muß der aktuelle Speichergrenzwert durch `sbrk (char *)(0)` bestimmt werden (siehe `brk`(2)).

fclose, fflush - Stream schließen oder leeren

```
#include <stdio.h>
int fclose (FILE *Stream);
int fflush (FILE *Stream);
```

`fclose` bewirkt, daß gepufferte Daten, die darauf warten, geschrieben zu werden, für den angegebenen *Stream* (siehe `intro`(3)) in die Datei geschrieben werden und der *Stream* geschlossen wird. Wenn der Schreib-/Lesezeiger der zugrundeliegenden Datei nicht schon am Dateiende ist und auf die Datei wahlfrei zugegriffen werden kann, wird der Schreib-/Lesezeiger so berichtigt, daß die nächste Operation in der offenen Datei auf das Byte zugreift, das vor dem Schließen der Datei als letztes gelesen oder geschrieben wurde.

`fclose` wird bei Aufruf von `exit` automatisch für alle offenen Dateien ausgeführt.

Wenn *Stream* auf einen Ausgabe- oder Aktualisierungsstream zeigt, auf dem die letzte Operation keine Eingabeoperation war, bewirkt `fflush`, daß gepufferte Daten für den *Stream*, in die Datei geschrieben werden. Alle ungelesenen in *Stream* gepufferten Daten werden entfernt. Der *Stream* bleibt offen. Wenn *Stream* zum Lesen geöffnet wurde, der Schreib-/Lesezeiger der unterliegenden Datei noch nicht am Ende der Datei steht, und die Datei für wahlfreien Zugriff geeignet ist, so wird der Schreib-/Lesezeiger so korrigiert, daß die nächste Operation auf dem offenen Dateizeiger auf das Byte zugreift, das dem zuletzt geschriebenen oder gelesenen Byte folgt.

Beim Aufruf von `fflush`, wenn *Stream* ein Nullzeiger ist, werden bei allen zum Schreiben geöffneten Dateien die Puffer geleert.

SIEHE AUCH

`close`(2), `exit`(2), `intro`(3), `fopen`(3S), `setbuf`(3S), `stdio`(3S).

ERGEBNIS

Bei erfolgreicher Beendigung geben diese Funktionen den Wert 0 zurück. Sonst wird `EOF` zurückgegeben.

ferror, feof, clearerr, fileno - Stream-Status abfragen

```
#include <stdio.h>
int ferror (FILE *Stream);
int feof (FILE *Stream);
void clearerr (FILE *Stream);
int fileno (FILE *Stream);
```

`ferror` gibt einen Wert ungleich Null zurück, wenn zuvor beim Lesen vom oder Schreiben in die angegebene Datei *Stream* (siehe `intro`(3)) ein Fehler aufgetreten ist; andernfalls wird Null zurückgegeben.

`feof` gibt einen Wert ungleich Null zurück, wenn `EOF` zuvor beim Lesen der angegebenen Eingabedatei *Stream* festgestellt wurde; andernfalls wird Null zurückgegeben.

`clearerr` setzt die Fehleranzeige und die `EOF`-Anzeige in *Stream* zurück auf 0.

`fileno` gibt den ganzzahligen Dateideskriptor zurück, der zu *Stream* gehört (siehe `open`(2)).

SIEHE AUCH

`open`(2), `fopen`(3S), `stdio`(3S).

ffs - erstes gesetztes Bit suchen

```
#include <string.h>
int ffs(const int i);
```

`ffs` sucht das erste gesetzte Bit im übergebenen Argument und liefert die Position dieses Bits zurück. Die Numerierung der Bits beginnt bei 1, angefangen mit dem niedrigstwertigen Bit. Ein Rückgabewert von Null zeigt an, daß der übergebene Wert Null ist.

fmtmsg - Meldung über stderr oder die Systemkonsole anzeigen

```
#include <fmtmsg.h>
int fmtmsg(long classification, const char *label, int severity,
    const char *text, const char *action, const char *tag);
```

Aufbauend auf der Klassifikationskomponente einer Meldung, schreibt `fmtmsg` eine formatierte Meldung auf `stderr`, auf die Systemkonsole oder auf beide.

`fmtmsg` kann anstelle der üblichen `printf`-Schnittstelle verwendet werden, um Meldungen über `stderr` auszugeben. `fmtmsg` bietet in Verbindung mit `gettxt` eine einfache Schnittstelle zum Erstellen von sprachunabhängigen Anwendungsprogrammen.

Eine formatierte Meldung besteht aus bis zu fünf Standardkomponenten, die weiter unten definiert werden. Die Komponente *classification* ist nicht Teil der Standardmeldung, die dem Benutzer angezeigt wird, sondern definiert die Quelle der Meldung und steuert die Anzeige der formatierten Meldung.

classification

enthält Bezeichner aus den folgenden Gruppen der Haupt- und Nebenklassifikationen. Jeder Bezeichner einer Nebenklassifikation kann durch ODER-Verknüpfung mit einem anderen Bezeichner einer anderen Nebenklassifikation verwendet werden. Zwei oder mehr Bezeichner aus derselben Nebenklassifikation sollten nicht zusammen verwendet werden, mit Ausnahme der Anzeigeklassifikation. Beide Bezeichner der Anzeigeklassifikation können so verwendet werden, daß die Meldungen sowohl auf `stderr` als auch auf der Systemkonsole erscheinen.

- 'Hauptklassifikationen' bezeichnen den Ursprung eines Zustands. Die Bezeichner sind: `MM_HARD` (Hardware), `MM_SOFT` (Software) und `MM_FIRM` (Firmware).
- 'Nebenklassifikationen des Meldungsursprungs' bezeichnen die Art der Software, in der das Problem auftrat. Die Bezeichner sind: `MM_APPL` (Anwendung), `MM_UTIL` (Hilfsprogramm) und `MM_OPSYS` (Betriebssystem).
- 'Nebenklassifikationen für die Anzeige' bezeichnen, wo die Meldung angezeigt werden soll. Die Bezeichner sind `MM_PRINT`, um die Meldung auf der Standard-Fehlerausgabe auszugeben, und `MM_CONSOLE`, um die Meldung auf der Systemkonsole auszugeben. Keiner, einer oder beide Bezeichner können verwendet werden.

- 'Nebenklassifikationen für den Status' geben an, ob sich das Anwendungsprogramm nach dem Zustand stabilisieren kann. Bezeichner sind: MM_RECOVER (stabilisierbar) und MM_NRECOV (nicht stabilisierbar).
- Ein weiterer Bezeichner, MM_NULLMC, gibt an, daß keine Klassifikationskomponente für die Meldung angegeben wird.

label gibt den Ursprung der Meldung an. Das Format dieser Komponente besteht aus zwei Feldern, die durch einen Doppelpunkt getrennt werden. Das erste Feld ist bis zu 10 Zeichen lang; das zweite ist bis zu 14 Zeichen lang. Es wird dazu geraten, mit *label* das Paket und das Programm oder den Anwendungsnamen zu bezeichnen. So zeigt beispielsweise der Inhalt UX:cat für *label* an, daß das Paket UNIX-System V und die Anwendung cat gemeint ist.

severity zeigt die Warnstufe des Zustands an. Bezeichner für die Standardwarnstufen für *severity* sind:

- MM_HALT zeigt an, daß die Anwendung auf einen schwerwiegenden Fehler gestoßen ist und die Bearbeitung anhält. Die Zeichenkette HALT wird ausgegeben.
- MM_ERROR zeigt an, daß die Anwendung einen Fehler erkannt hat. Die Zeichenkette ERROR wird ausgegeben.
- MM_WARNING zeigt an, daß ein ungewöhnlicher Zustand eingetreten ist, bei dem es sich um ein Problem handeln könnte und der beobachtet werden sollte. Die Zeichenkette WARNING wird ausgegeben.
- MM_INFO liefert Informationen über einen Zustand, der keinen Fehler darstellt. Die Zeichenkette INFO wird ausgegeben.
- MM_NOSEV zeigt an, daß für die Meldung keine Warnstufe existiert.

Weitere Warnstufen können mit der Funktion addseverity hinzugefügt werden.

text beschreibt die Ursache der Meldung. Die Zeichenkette *text* ist nicht auf eine bestimmte Länge beschränkt.

action beschreibt die erste Aktion, die im Fehlerbehebungsprozeß ausgeführt werden soll. fmtmsg schreibt vor dieser Zeichenkette das Präfix: TO FIX:. Die Zeichenkette *action* ist nicht auf eine bestimmte Länge beschränkt.

tag Ein Bezeichner, der auf die Online-Dokumentation für die Meldung verweist. Empfohlen wird, daß *tag* ein *label* und eine eindeutige Zahl enthält. Ein Beispiel für *tag* ist UX:cat:146.

Umgebungsvariablen

Es gibt zwei Umgebungsvariablen, die das Verhalten von `fmtmsg` beeinflussen: `MSGVERB` und `SEV_LEVEL`.

`MSGVERB` teilt `fmtmsg` mit, welche Meldungskomponenten beim Schreiben der Meldungen auf `stderr` ausgewählt werden sollen. Der Wert von `MSGVERB` besteht aus einer Liste optionaler Schlüsselwörter, die durch Doppelpunkte getrennt werden. `MSGVERB` kann wie folgt gesetzt werden:

```
MSGVERB=[Schlüsselwort[:Schlüsselwort[:...]]]
export MSGVERB
```

Gültige *Schlüsselwörter* sind: `label`, `severity`, `text`, `action`, und `tag`. Wenn `MSGVERB` ein Schlüsselwort für eine Komponente enthält und der Komponentenwert nicht den Wert Null hat, schreibt `fmtmsg` diese Komponente bei der Meldungsausgabe über `stderr` hinaus. Wenn `MSGVERB` kein Schlüsselwort für eine Meldungskomponente enthält, wird diese Komponente nicht mit der Meldung angezeigt. Die Schlüsselwörter können in einer beliebigen Reihenfolge angegeben werden. Ist `MSGVERB` nicht definiert, enthält dieser Bezeichner eine Nullzeichenkette, ist der Wert nicht im korrekten Format angegeben, oder sind ungültige Schlüsselwörter angegeben, so wählt `fmtmsg` alle Komponenten aus.

Beim ersten Aufruf von `fmtmsg` wird die `MSGVERB`-Umgebungsvariable abgeprüft, um die Meldungskomponenten selektieren zu können, wenn eine Meldung über die Standard-Fehlerausgabe `stderr` generiert wird. Die Werte, die beim ersten Aufruf akzeptiert werden, werden für die nachfolgenden Aufrufe gesichert.

`MSGVERB` beeinflußt nur die Selektion der Komponenten, die über die Standard-Fehlerausgabe angezeigt werden sollen. Bei Ausgabe auf die Konsole werden alle Meldungen selektiert.

`SEV_LEVEL` definiert die Warnstufen und weist die auszugebenden Zeichenketten zu, die von `fmtmsg` benutzt werden sollen. Die unten angegebenen Standardwarnstufen können nicht verändert werden. Weitere Warnstufen können definiert, verändert und entfernt werden. Dies geschieht über die Funktion `addseverity` (siehe `addseverity`(3C)). Wenn dieselbe Warnstufe durch `SEV_LEVEL` und `addseverity` definiert wird, so setzt sich die Definition von `addseverity` durch.

```
0   (keine Warnstufe verwendet)
1   HALT
2   ERROR
3   WARNING
4   INFO
```

`SEV_LEVEL` kann wie folgt eingestellt werden:

```
SEV_LEVEL=[Beschreibung[:Beschreibung[:...]]]
export SEV_LEVEL
```

Beschreibung enthält eine Liste mit drei Feldern, die durch Kommata getrennt werden:

Beschreibung=severity_keyword,level,printstring

severity_keyword ist eine Zeichenkette, die als Schlüsselwort für die Option `-s` *severity* vom Kommando `fmtmsg` verwendet wird. Dieses Feld wird nicht von der Funktion `fmtmsg` verwendet.

level ist eine Zeichenkette, die eine positive ganze Zahl enthält (nicht 0, 1, 2, 3 oder 4, denn diese Werte sind für die Standardwarnstufen reserviert). Wenn das Schlüsselwort *severity_keyword* verwendet wird, stellt *level* die Warnstufe des Wertes dar, der an die Funktion `fmtmsg` übergeben wurde.

printstring ist eine Zeichenkette, die von `fmtmsg` für das Standardmeldungsformat verwendet wird, wenn die Warnstufe *level* angegeben wird.

Stellt *Beschreibung* in der Liste keine durch Kommata getrennte Liste mit drei Feldern dar, oder ist das zweite Feld einer Liste keine ganze Zahl, so wird *Beschreibung* in der Liste ignoriert.

Wird `fmtmsg` erstmals aufgerufen, dann wird die Umgebungsvariable `SEV_LEVEL` überprüft, um festzustellen, ob neben den fünf Standardwarnstufen und den durch `addseverity` festgelegten zusätzliche Warnstufen definiert wurden. Die Werte, die beim erstmaligen Aufruf festgestellt wurden, werden für spätere Aufrufe gespeichert.

Benutzung in Anwendungsprogrammen

Eine oder mehrere Meldungskomponenten können systematisch aus der Meldung weggelassen werden, wenn der Nullwert der jeweiligen Komponente angegeben wird.

Die unten angegebene Tabelle zeigt die Nullwerte und Bezeichner für die Argumente von `fmtmsg`.

Argument	Typ	Nullwert	Bezeichner
label	char*	(char*) NULL	MM_NULLLBL
severity	int	0	MM_NULLSEV
class	long	0L	MM_NULLMC
text	char*	(char*) NULL	MM_NULLTXT
action	char*	(char*) NULL	MM_NULLACT
tag	char*	(char*) NULL	MM_NULLTAG

Ein weiteres Mittel zum systematischen Weglassen einer Komponenten besteht im Auslassen der Schlüsselwörter der Komponenten bei der Definition der `MSGVERB`-Umgebungsvariablen (siehe Abschnitt 'Umgebungsvariablen').

BEISPIELE

Beispiel 1:

```
fmtmsg(MM_PRINT, "UX:cat", MM_ERROR, "Falsche Syntax", "Siehe Handbuch",
"UX:cat:001")
```

liefert eine komplette Meldung mit dem Standardmeldungsformat:

```
UX:cat: ERROR: Falsche Syntax
      TO FIX: Siehe Handbuch   UX:cat:001
```

Beispiel 2:

Wird die Umgebungsvariable `MSGVERB` wie folgt gesetzt:

```
MSGVERB=severity:text:action
```

und das Beispiel 1 verwendet, so generiert `fmtmsg`:

```
ERROR: Falsche Syntax TO FIX: Siehe Handbuch
```

Beispiel 3:

Wird die Umgebungsvariable `SEV_LEVEL` wie folgt gesetzt:

```
SEV_LEVEL=note,5,NOTE
```

so liefert der folgende Aufruf von `fmtmsg`:

```
fmtmsg(MM_UTIL | MM_PRINT, "UX:cat", 5, "Falsche Syntax", "Siehe Handbuch",
"UX:cat:001")
```

die folgende Ausgabe:

```
UX:cat: NOTE: Falsche Syntax
          TO FIX: Siehe Handbuch    UX:cat:001
```

SIEHE AUCH

`addseverity`(3C), `gettxt`(3C), `printf`(3S), `fmtmsg`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Die Rückgabewerte von `fmtmsg` sind:

`MM_OK`	Die Funktion wurde erfolgreich ausgeführt.
`MM_NOTOK`	Die Funktion wurde nicht erfolgreich ausgeführt.
`MM_NOMSG`	Die Funktion konnte eine Meldung über die Standard-Fehlerausgabe nicht generieren, wurde aber ansonsten erfolgreich ausgeführt.
`MM_NOCON`	Die Funktion konnte eine Meldung über die Systemkonsole nicht generieren, wurde aber ansonsten erfolgreich ausgeführt.

fopen, freopen, fdopen - Stream öffnen

```
#include <stdio.h>
FILE *fopen (const char *Dateiname, const char *Typ);
FILE *freopen (const char *Dateiname, const char *Typ, FILE *Stream);
FILE *fdopen (int dk, const char *Typ);
```

`fopen` öffnet die Datei *Dateiname* und liefert einen Dateizeiger zurück, der auf die *Dateiname* zugeordnete `FILE`-Struktur zeigt. *Dateiname* muß auf eine Zeichenkette zeigen, die den Namen der zu öffnenden Datei enthält. Die gewünschte Zugriffsart geben Sie mit der Zeichenkette *Typ* an. *Typ* kann folgende Werte annehmen.

`"r"` oder `"rb"`	Datei zum Lesen öffnen
`"w"` oder `"wb"`	Datei zum Schreiben auf die Länge 0 abschneiden oder erstellen
`"a"` oder `"ab"`	anhängen; zum Schreiben am Ende der Datei öffnen oder zum Schreiben erstellen
`"r+"`, `"r+b"` oder `"rb+"`	Datei zum Aktualisieren öffnen (Lesen und Schreiben)
`"w+"`, `"w+b"` oder `"wb+"`	zum Aktualisieren abschneiden oder erstellen
`"a+"`, `"a+b"` oder `"ab+"`	anhängen; zum Aktualisieren am Dateiende öffnen oder erstellen

Das `b` wird in den obigen Typen ignoriert. Es ist dazu da, um binäre Dateien von Textdateien unterscheiden zu können. Es gibt jedoch auf einem SINIX-System keinen Unterschied zwischen diesen Dateitypen.

`freopen` ersetzt die durch *Stream* bezeichnete geöffnete Datei durch die Datei *Dateiname*. Zuvor wird versucht, noch ungeschriebene Daten zu schreiben, dann wird die ursprüngliche Datei geschlossen, unabhängig davon, ob das Öffnen schließlich erfolgreich ist. `freopen` gibt einen Zeiger auf die `FILE`-Struktur zurück, die *Stream* zugeordnet ist.

`freopen` wird normalerweise dazu verwendet, die Dateizeiger `stdin`, `stdout` und `stderr` anderen Dateien zuzuordnen als den voreingestellten, geöffneten Standarddateien. `stderr` ist standardmäßig nicht gepuffert, durch die Verwendung von `freopen` wird `stderr` jedoch gepuffert oder zeilengepuffert.

`fdopen` verknüpft einen Stream mit einem Dateideskriptor. Dateideskriptoren werden von `open`, `dup`, `creat` oder `pipe` geliefert. Diese Funktionen öffnen zwar die Dateien, es werden jedoch keine Zeiger auf eine `FILE`-Struktur zurückgegeben. Fast alle der Bibliotheksfunktionen von 3S erwarten als Eingabe einen Stream. Der Stream-Typ muß mit der

Zugriffsart der offenen Datei übereinstimmen. Der zu dem Stream gehörende Schreib-/Lesezeiger wird auf die Position gesetzt, die durch den Datei-Offset der zu *dk* gehörenden Datei angegeben ist.

Wenn eine Datei zur Aktualisierung geöffnet wird, sind sowohl Ein- als auch Ausgabe auf dem sich ergebenden Stream zulässig. Jedoch darf der Ausgabe keine Eingabe direkt ohne ein dazwischengesetztes `fflush`, `fseek`, `fsetpos` oder `rewind` folgen, und der Eingabe darf keine Ausgabe direkt folgen ohne eingefügtes `fseek`, `fsetpos` oder `rewind` oder einer Lese-Operation, die auf das Dateiende trifft.

Wenn eine Datei zum Anhängen geöffnet wird (d.h. wenn *Typ* `"a"`, `"ab"`, `"a+"` oder `"ab+"` ist), ist ein Überschreiben der sich bereits in der Datei befindenden Daten unmöglich. `fseek` kann zum Umsetzen des Schreib-/Lesezeigers auf eine beliebige Position in der Datei verwendet werden, jedoch wird der aktuelle Schreib-/Lesezeiger nicht beachtet, wenn eine Ausgabe in die Datei geschrieben wird. Alle Ausgaben werden an das Ende der Datei geschrieben und bewirken, daß der Schreib-/Lesezeiger an das Ende der Ausgabe positioniert wird. Wenn zwei separate Prozesse dieselbe Datei zum Anhängen öffnen, kann jeder Prozeß unbehindert in die Datei schreiben, ohne daß ein Zerstören der vom anderen Prozeß geschriebenen Ausgabe zu befürchten ist. Die Ausgabe von den beiden Prozessen wird gemischt, in der Reihenfolge, in der sie geschrieben wird, an die Datei angehängt.

Wenn ein Stream geöffnet ist, ist er genau dann vollständig gepuffert, wenn sichergestellt werden kann, daß er keine Verweise auf ein interaktives Gerät hat. Die Fehler- und Dateiende-Anzeiger werden beim Öffnen für den Stream zurückgesetzt.

SIEHE AUCH

`close`(2), `creat`(2), `dup`(2), `open`(2), `pipe`(2), `write`(2), `fclose`(3S), `fseek`(3S), `setbuf`(3S), `stdio`(3S).

ERGEBNIS

Die Funktionen `fopen` und `freopen` geben einen Nullzeiger zurück, wenn auf *Dateiname* nicht zugegriffen werden kann, *Typ* ungültig ist, oder die Datei nicht geöffnet werden kann.

Die Funktion `fdopen` gibt einen Nullzeiger zurück, wenn *dk* kein Dateideskriptor ist, der zu einer geöffneten Datei gehört, *Typ* ungültig ist, oder die Datei nicht geöffnet werden kann.

Die Funktionen `fopen` oder `fdopen` können scheitern, ohne `errno` zu setzen, wenn es keine freien `stdio`-Streams gibt.

Von `fdopen` verwendete Dateideskriptoren müssen kleiner als 255 sein.

fpgetround - IEEE-Gleitkomma-Umgebung steuern

```
#include <ieeefp.h>
fp_rnd fpgetround (void);
fp_rnd fpsetround (fp_rnd rnd_dir);
fp_except fpgetmask (void);
fp_except fpsetmask (fp_except mask);
fp_except fpgetsticky (void);
fp_except fpsetsticky (fp_except sticky);
```

Es gibt fünf Gleitkomma-Ausnahmen: Dividieren durch Null, Überlauf, Unterlauf ungenaues Ergebnis und ungültige Funktion. Bei Auftreten einer Gleitkomma-Ausnahmen wird das entsprechende Sticky-Bit gesetzt (1), und wenn das Mask-Bit eingeschaltet ist (1), wird eine Unterbrechung erzeugt. Mit diesen Funktionen kann der Benutzer das Verhalten bei Auftreten einer dieser Ausnahmen sowie den Ab-/Aufrundemodus für Gleitkommaoperationen ändern.

```
FP_X_INV        /* invalid operation exception */
FP_X_OFL        /* overflow exception */
FP_X_UFL        /* underflow exception */
FP_X_DZ         /* divide-by-zero exception */
FP_X_IMP        /* imprecise (loss of precision) */
FP_RN           /* round to nearest representative number */
FP_RP           /* round to plus infinity */
FP_RM           /* round to minus infinity */
FP_RZ           /* round to zero (truncate) */
```

`fpgetround` gibt den aktuellen Rundungsmodus zurück.

`fpsetround` setzt den Rundungsmodus und gibt den vorherigen Rundungsmodus zurück.

`fpgetmask` gibt die aktuellen Ausnahme-Mask-Bits zurück.

`fpsetmask` setzt die Ausnahme-Mask-Bits und gibt die vorherige Einstellung zurück.

`fpgetsticky` gibt die aktuellen Ausnahme-Sticky-Bits zurück.

`fpsetsticky` setzt (löscht) die Ausnahme-Sticky-Bits und gibt die vorherige Einstellung zurück.

Die Standardumgebung ist wie folgt: Der Rundungsmodus ist auf `round to nearest` (`FP_RN`) gesetzt, und alle Unterbrechungen sind abgeschaltet.

Einzelne Bits können durch die in `ieeefp.h` definierten Konstanten untersucht werden.

SIEHE AUCH

`isnan`(3C).

HINWEIS

`fpsetsticky` ändert alle Sticky-Bits. `fpsetmask` ändert alle Mask-Bits. `fpsetmask` löscht das Sticky-Bit entsprechend den freigegebenen Unterbrechungen.

Für C ist ein Abschneiden ("`Round to zero`") für Umwandlungen von Gleitkomma- in Ganzzahldarstellung erforderlich. Der aktuelle Rundungsmodus hat keinen Einfluß auf diese Umwandlungen.

Sie müssen nach einem Anhalten wegen Gleitkomma-Unterbrechung das Sticky-Bit löschen, um fortzufahren. Wird das Sticky-Bit nicht vor dem Auftreten des nächsten Anhaltens gelöscht, wird möglicherweise ein falscher Ausnahmetyp gemeldet.

fread, fwrite - binäre Ein-/Ausgabe

```
#include <stdio.h>
size_t fread (void *Zeiger, size_t Größe, size_t Anzahl, FILE *Stream);
size_t fwrite (const void *Zeiger, size_t Größe, size_t Anzahl, FILE *Stream);
```

`fread` liest bis zu *Anzahl* Einheiten von *Stream* in ein Feld, auf das *Zeiger* zeigt, wobei eine Dateneinheit eine Folge von Bytes der Länge *Größe* ist. Diese Folge von Bytes wird nicht notwendigerweise durch ein Null-Byte beendet. `fread` hört mit dem Lesen auf, wenn beim Lesen von *Stream* ein Dateiende-Zeichen oder eine Fehlerbedingung angetroffen wird oder *Anzahl* Einheiten gelesen worden sind. `fread` erhöht den Schreib-/Lesezeiger in *Stream*, so daß er auf das Byte hinter dem zuletzt gelesenen Byte zeigt, sofern es ein solches gibt. `fread` ändert den Inhalt von *Stream* nicht. `fread` gibt die Anzahl der gelesenen Einheiten zurück.

`fwrite` schreibt auf die angegebene Ausgabe *Stream* höchstens *Anzahl* Dateneinheiten aus dem Feld, auf das *Zeiger* zeigt, wobei eine Einheit eine Folge von Bytes der Länge *Größe* ist. Diese Folge von Bytes wird nicht notwendigerweise mit einem Null-Byte beendet. `fwrite` hört auf zu schreiben, wenn es *Anzahl* Dateneinheiten geschrieben hat oder wenn auf dem *Stream* eine Fehlerbedingung angetroffen wurde. `write` ändert nicht den Inhalt des Feldes, auf das *Zeiger* zeigt. `fwrite` erhöht den Schreib-/Lesezeiger in *Stream* um die Anzahl der geschriebenen Bytes. `fwrite` gibt die Anzahl der geschriebenen Dateneinheiten zurück.

Wenn *Größe* oder *Anzahl* null ist, geben `fread` und `fwrite` den Wert 0 zurück und haben keinen Einfluß auf den Zustand von *Stream*.

Die Routinen `ferror` oder `feof` müssen verwendet werden, um zwischen Fehlerbedingungen und Dateiende zu unterscheiden.

SIEHE AUCH

`exit`(2), `lseek`(2), `read`(2), `write`(2), `abort`(3C), `fclose`(3S), `fopen`(3S), `getc`(3S), `gets`(3S), `printf`(3S), `putc`(3S), `puts`(3S), `scanf`(3S), `stdio`(3S).

ERGEBNIS

Wenn ein Fehler auftritt, wird die Fehleranzeige für *Stream* gesetzt.

frexp - Teile von Gleitkommazahlen bearbeiten

```
#include <math.h>
double frexp (double value, int *eptr);
double ldexp (double value, int exp);
double logb (double value);
double nextafter (double value1, double value2);
double scalb (double value, double exp);
double modf (double value, double *iptr);
float modff (float value, float *iptr);
```

Jede Zahl, die ungleich Null ist, kann eindeutig als x * 2^n geschrieben werden, wobei die Mantisse x im Bereich von `0.5 <= | x | < 1.0` liegt und der Exponent n eine ganze Zahl ist. `frexp` gibt die Mantisse eines `double`-Wertes *value* zurück und speichert den Exponenten indirekt an der Stelle, auf die *eptr* zeigt. Wenn *value* Null ist, sind beide von `frexp` zurückgegebenen Werte Null.

`ldexp` und `scalb` geben die Größe *value* * 2^{exp} zurück. Der einzige Unterschied zwischen den beiden ist, daß `scalb` von einer `NaN` eine Ausnahmebehandlung für die ungültige Operation erzeugt.

`logb` liefert den Exponenten des Arguments als vorzeichenbehaftete, doppelt genaue Gleitkommazahl.

`modf` und `modff` (Version für einfach genaue Gleitkommazahlen) geben den mit Vorzeichen versehenen Bruchteil des Wertes *value* zurück und speichern den ganzzahligen Teil indirekt an der Stelle, auf die *iptr* zeigt.

`nextafter` liefert die nächste auf *value1* folgende, darstellbare Gleitkommazahl in der Richtung von *value2*. Das bedeutet, wenn *value2* kleiner als *value1* ist, wird die größte darstellbare Gleitkommazahl kleiner als *value1* geliefert.

SIEHE AUCH

`cc`(1), `intro`(3M).

ERGEBNIS

Wenn `ldexp` einen Überlauf verursacht, wird ±HUGE (in `math.h` definiert) zurückgegeben (je nach dem Vorzeichen des Werts *value*), und `errno` wird auf ERANGE gesetzt. Wenn `ldexp` einen Unterlauf verursachen würde, wird Null zurückgegeben und `errno` auf ERANGE gesetzt. Wenn der Eingabewert *value* für `ldexp` NaN oder unendlich ist, wird die Eingabe zurückgegeben, und `errno` wird auf EDOM gesetzt. Dieselben Fehlerbedingungen gelten für `scalb`, d.h. `scalb` von einer NaN erzeugt eine Ausnahmebehandlung für die ungültige Operation.

`logb` einer NaN liefert NaN, `logb` von unendlich liefert positiv unendlich und `logb` von Null liefert negativ unendlich. Bei `logb` von Null wird eine 'Division-durch-Null'-Ausnahmebehandlung erzeugt. Bei jedem dieser Fälle wird `errno` auf EDOM gesetzt.

Ist der Eingabewert *value1* für `nextafter` positiv oder negativ unendlich, wird der Eingabewert zurückgeliefert und `errno` wird auf EDOM gesetzt. Eine Überlauf- und Ungenauigkeitsausnahme wird erzeugt, wenn *value1* endlich ist, aber das Ergebnis von `nextafter(`*value1*`,` *value2*`)` nicht. Liegt `nextafter(`*value1*`,` *value2*`)` genau zwischen $\pm 2^{-1022}$, dann wird eine Unterlauf- und Ungenauigkeitsausnahme erzeugt. In beiden Fällen wird `errno` auf ERANGE gesetzt.

Wenn das Programm im ANSI-Modus (`cc -kansi`) übersetzt wurde, wird HUGE_VAL statt HUGE zurückgegeben.

fseek - Schreib-/Lesezeiger eines Streams neu positionieren

```
#include <stdio.h>
int fseek (FILE *Stream, long Offset, int Zeigername);
void rewind (FILE *Stream);
long ftell (FILE *Stream);
```

`fseek` setzt die Position der nächsten Ein- oder Ausgabe-Operation in *Stream* (siehe `intro`(3)). Die neue Position ist *Offset* Bytes (mit Vorzeichen) entfernt vom Anfang der Datei, von der momentanen Position oder vom Dateiende, je nachdem, ob *Zeigername* den Wert `SEEK_SET`, `SEEK_CUR` oder `SEEK_END` hat. Diese Werte sind in `stdio.h` wie folgt definiert:

`SEEK_SET` Setze Position auf *Offset*.

`SEEK_CUR` Setze Position auf momentane Position plus *Offset*.

`SEEK_END` Setze Position auf `EOF` plus *Offset*.

Mit `fseek` kann der Schreib-/Lesezeiger hinter das Ende der Daten in der Datei gesetzt werden. Wenn später an diesen Ort Daten geschrieben werden, liefern aufeinanderfolgende Lesevorgänge in der Lücke solange 0, bis Daten in die Lücke geschrieben werden. `fseek` selbst vergrößert die Datei nicht. `rewind (Stream)` ist identisch mit

```
(void) fseek (Stream, 0L, SEEK_SET);
```

mit der Ausnahme, daß `rewind` auch die Fehleranzeige von *Stream* löscht.

`fseek` und `rewind` löschen die `EOF`-Anzeige und machen die Wirkung von `ungetc`(3S) auf *Stream* rückgängig.

Nach `fseek` oder `rewind` kann die nächste Operation an einer geöffneten Datei, welche aktualisiert wird, entweder Ein- oder Ausgabe sein.

Wenn *Stream* beschreibbar ist und keine gepufferten Daten in die zugrundeliegende Datei geschrieben worden sind, bewirken `fseek` und `rewind`, daß die noch nicht geschriebenen Daten in die Datei geschrieben werden.

`ftell` gibt den Offset des aktuellen Bytes zurück, und zwar relativ zum Anfang der Datei, der der angegebene *Stream* zugeordnet ist.

SIEHE AUCH

`lseek`(2), `write`(2), `fopen`(3S), `popen`(3S), `stdio`(3S), `ungetc`(3S).

ERGEBNIS

`fseek` gibt bei ungültigem Positionieren -1 zurück, ansonsten Null. Ungültiges Positionieren kann beispielsweise ein `fseek` auf eine Datei sein, die nicht über `fopen` geöffnet wurde; insbesondere darf `fseek` nicht für ein Terminal oder für eine Datei verwendet werden, die über `popen`(3S) geöffnet wurde. Nachdem ein Stream geschlossen wurde, sind keine weiteren Operationen auf diesem Stream definiert.

HINWEIS

Obwohl im SINIX-System ein von `ftell` zurückgegebener Offset in Bytes gemessen wird und es zulässig ist, relativ zu diesem Offset zu positionieren, erfordert die Portabilität auf andere Systeme, daß `fseek` einen direkten Offset erhält. Arithmetische Operationen an einem nich-SINIX-Offset, der nicht unbedingt in Bytes gemessen wird, können nicht immer sinnvoll ausgeführt werden.

fsetpos, fgetpos - Dateizeiger im Datenstrom neu positionieren

```
#include <stdio.h>
int fsetpos (FILE *stream, const fpos_t *pos);
int fgetpos (FILE *stream, fpos_t *pos);
```

`fsetpos` setzt die Position der nächsten Ein- oder Ausgabeoperation mit *stream* auf den Wert des Objekts, auf das *pos* zeigt. Das Objekt, auf das *pos* zeigt, muß ein Wert sein, der aus einem vorhergehenden Aufruf von `fgetpos` mit demselben Stream resultiert.

`fsetpos` löscht die Dateiendeanzeige für den Stream und macht alle Wirkungen der Funktion `ungetc` auf den Stream rückgängig. Nach `fsetpos` kann mit einer änderbaren Datei Ein- oder Ausgabeoperationen durchgeführt werden.

`fgetpos` speichert den aktuellen Wert der Dateiposition von *stream* in dem Objekt, auf das *pos* zeigt. Der gespeicherte Wert enthält Informationen, mit denen `fsetpos` den Stream auf die Position einstellen kann, die zur Zeit des Aufrufs von `fgetpos` aktuell war.

Bei erfolgreicher Ausführung liefern `fsetpos` und `fgetpos` Null zurück. Ansonsten werden Werte ungleich Null zurückgegeben.

SIEHE AUCH

`fseek`(3S), `lseek`(2) `ungetc`(3S).

ftw, nftw - Dateibaum durchwandern

```
#include <ftw.h>

int ftw (const char *path, int (*fn) (const char *, const struct stat *, int), int depth);

int nftw (const char *path, int (*fn) (const char *, const struct
    stat *, int, struct FTW*), int depth, int flags);
```

`ftw` durchsucht die Dateiverzeichnis-Hierarchie, die mit *path* beginnt, rekursiv. Für jedes Objekt in der Hierarchie ruft `ftw` die benutzerdefinierte Funktion *fn* auf und übergibt ihr einen Zeiger auf eine mit einem Null-Byte endende Zeichenkette, die den Namen des Objekts, einen Zeiger auf eine `stat`-Struktur (siehe `stat`(2)), die Informationen über das Objekt sowie eine ganze Zahl enthält. Mögliche Werte der ganzen Zahl, die in der Include-Datei `ftw.h` definiert sind:

`FTW_F` Das Objekt ist eine Datei.

`FTW_D` Das Objekt ist ein Dateiverzeichnis.

`FTW_DNR` Das Objekt ist ein Dateiverzeichnis, das nicht gelesen werden kann. Nachfolgende Verzeichnisse werden nicht durchlaufen.

`FTW_NS` `stat` war auf diesem Objekt erfolglos, entweder aufgrund von mangelnden Zugriffsrechten, oder weil das Objekt ein symbolischer Verweis ist, der auf eine nicht vorhandene Datei zeigt. Der an *fn* übergebene `stat`-Puffer ist undefiniert.

Der Dateibaum wird von der obersten Hierarchiestufe an durchwandert, bis der Baum vollständig durchwandert ist, ein Aufruf von *fn* einen Wert ungleich Null zurückgibt oder ein Fehler innerhalb `ftw` (wie z.B. ein E/A-Fehler) festgestellt wird. Wenn der Baum vollständig durchwandert ist, gibt `ftw` Null zurück. Wenn *fn* einen Wert ungleich Null zurückgibt, stoppt `ftw` das Durchwandern des Baums und gibt den Wert zurück, der von *fn* zurückgegeben wurde. Wenn `ftw` einen anderen Fehler als `EACCES` feststellt, wird -1 zurückgegeben und der Fehlertyp in `errno` gesetzt.

Die Funktion `nftw` ist ähnlich wie `ftw`, nur daß sie ein weiteres Argument, *flags*, erwartet. Im Feld *flags* wird folgendes angegeben:

`FTW_PHYS` physikalisches Durchwandern; es folgt keinen symbolischen Verweisen. Sonst folgt `nftw` Verweisen, jedoch keinen Pfaden, die sich selbst kreuzen.

`FTW_MOUNT` Beim Durchwandern wird kein Einhängepunkt gekreuzt.

`FTW_DEPTH` Vor dem Verzeichnis selbst werden erst alle Unterverzeichnisse durchwandert.

`FTW_CHDIR` Zunächst wird jedes Verzeichnis aufgesucht, bevor es gelesen wird.

Die Funktion `nftw` ruft *fn* mit vier Argumenten zu jeder Datei und jedem Verzeichnis auf. Das erste Argument ist der Pfadname des Objekts, das zweite ist ein Zeiger auf den `stat`-Puffer, das dritte ist eine Ganzzahl, die zusätzliche Informationen liefert, und das vierte ist ein `struct FTW`, das die folgenden Elemente enthält:

```
int base;
int level;
```

`base` ist der Offset innerhalb des Pfadnamens für den Namen des Objekts. `level` gibt an, welche Hierarchieebenen bereits durchwandert wurden und welche noch durchwandert werden müssen. Dabei hat die unterste Ebene den Wert 0.

Die Werte des dritten Arguments haben folgende Bedeutung:

`FTW_F` Das Objekt ist eine Datei.

`FTW_D` Das Objekt ist ein Verzeichnis.

`FTW_DP` Das Objekt ist ein Verzeichnis, Unterverzeichnisse wurden bereits durchwandert.

`FTW_SLN` Das Objekt ist ein symbolischer Verweis, der auf eine nicht vorhandene Datei zeigt.

`FTW_DNR` Das Objekt ist ein Verzeichnis, das nicht gelesen werden kann. *fn* wird für keinen seiner Nachfolger aufgerufen.

`FTW_NS` `stat` kann das Objekt aufgrund unzureichender Zugriffsrechte nicht bearbeiten. Der an *fn* übergebene `stat`-Puffer ist undefiniert. Wenn `stat` aus anderen Gründen scheitert, wird das als ein Fehler betrachtet, und `nftw` gibt -1 zurück.

Sowohl `ftw` als auch `nftw` verwenden jeweils einen Dateideskriptor für jede Ebene im Baum. Das Argument *depth* begrenzt die Anzahl der verwendeten Dateideskriptoren. Ist *depth* null oder negativ, hat es die gleiche Wirkung wie mit dem Wert 1. *depth* darf nicht größer als die Anzahl der zum gegebenen Zeitpunkt zur Verfügung stehenden Dateideskriptoren sein. `ftw` läuft schneller, wenn *depth* wenigstens genauso groß wie die Anzahl der Ebenen im Dateibaum ist. Wenn `ftw` und `nftw` zurückkehren, schließen sie alle Dateideskriptoren, die sie geöffnet haben; sie schließen aber keine Dateideskriptoren, die von *fn* geöffnet worden sein könnten.

SIEHE AUCH

stat(2), malloc(3C).

HINWEIS

Da ftw rekursiv ist, besteht die Möglichkeit, daß es mit einem Speicherfehler abbricht, wenn es auf Dateibäume mit zu vielen Hierarchieebenen angewendet wird.

ftw verwendet malloc(3C) zur Zuweisung von dynamischem Speicher zur Ausführungszeit. Wenn ftw abgebrochen wird, wie z. B. durch Ausführen von longjmp durch *fn* oder durch eine Unterbrechungsroutine, hat ftw keine Möglichkeit, diesen Speicher freizugeben, so daß der Speicher ständig belegt bleibt. Unterbrechungen können bearbeitet werden, indem veranlasst wird, daß *fn* beim nächsten Aufruf einen Wert zurückgibt, der ungleich Null ist.

getc, getchar, fgetc, getw - Zeichen/Wort aus Stream lesen

```
#include <stdio.h>
int getc (FILE *Stream);
int getchar (void);
int fgetc (FILE *Stream);
int getw (FILE *Stream);
```

`getc` gibt das nächste Zeichen (d.h. Byte) aus der angegebenen Eingabedatei *Stream* (siehe `intro`(3)) als in `unsigned char` umgewandelt in ein `int` zurück. Es setzt auch den Schreib-/Lesezeiger sofern dieser definiert ist um ein Zeichen in *Stream* weiter. `getchar` ist als `getc(stdin)` definiert. `getc` und `getchar` sind Makros.

`fgetc` verhält sich wie `getc`, ist jedoch eine Funktion und kein Makro. `fgetc` läuft langsamer als `getc`, beansprucht jedoch weniger Speicherplatz pro Aufruf und sein Name kann als Argument für eine Funktion verwendet werden.

`getw` liefert das nächste Wort (d.h. eine Ganzzahl) von der angegebenen Eingabedatei *Stream*. `getw` erhöht den zugehörigen Schreib-/Lesezeiger (sofern definiert), so daß dieser auf das nächste Wort zeigt. Die Größe eines Wortes ist die Größe einer Zahl vom Typ `int`. Sie ist bei den einzelnen Rechnern unterschiedlich. `getw` setzt keine besondere Ausrichtung in der Datei voraus.

SIEHE AUCH

`fclose`(3S), `ferror`(3S), `fopen`(3S), `fread`(3S), `gets`(3S), `putc`(3S), `scanf`(3S), `stdio`(3S), `ungetc`(3S).

ERGEBNIS

Diese Funktionen geben am Ende der Datei oder bei einem Fehler die Konstante `EOF` zurück und setzen je nachdem `EOF` oder die Fehleranzeige von *Stream*. Da `EOF` eine gültige ganze Zahl ist, sollte zur Ermittlung von `getw`-Fehlern `ferror` verwendet werden.

HINWEIS

Wenn der von `getc`, `getchar` oder `fgetc` zurückgegebene ganzzahlige Wert in einer Variablen vom Typ `char` gespeichert und dann mit der ganzzahligen Konstanten `EOF` verglichen wird, ist dieser Vergleich möglicherweise nie erfolgreich, weil die Vorzeichenpropagierung beim Übergang von `char` zu `int` implementierungsabhängig ist.

Die Makroversion von `getc` wertet das *Stream*-Argument mehr als einmal aus und könnte Seiteneffekte falsch behandeln. So arbeitet insbesondere `getc(*f++)` nicht sinnvoll. Verwenden Sie statt dessen `fgetc`.

Wegen der möglichen Unterschiede in Wortlänge und Byte-Ordnung sind die unter Verwendung von `putw` geschriebenen Dateien implementierungsabhängig und können möglicherweise nicht mit `getw` auf einem anderen Prozessor gelesen werden.

Für alle oben definierten Makros gibt es Funktionen. Um die Funktionenform zu erhalten, muß die Definition des Makronamens rückgängig gemacht werden (d.h. `#undef getc`).

getcwd - Pfadnamen des aktuellen Dateiverzeichnisses abfragen

```
#include <unistd.h>
char *getcwd (char *buf, int size);
```

getcwd gibt einen Zeiger auf den Pfadnamen des aktuellen Dateiverzeichnisses zurück. Der Wert von *size* muß wenigstens um eins größer als die Länge des zurückzugebenden Pfadnamens sein.

Wenn *buf* nicht NULL ist, wird der Pfadname in dem Speicherplatz, auf den *buf* zeigt, gespeichert.

Wenn *buf* ein Nullzeiger ist, erhält getcwd durch Aufruf von malloc(3C) *size* Bytes Speicherplatz. In diesem Fall kann der von getcwd zurückgegebene Zeiger als Argument in einem nachfolgenden Aufruf von free verwendet werden.

getcwd ist erfolglos, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

EACCES — Ein übergeordnetes Verzeichnis kann nicht gelesen werden, um seinen Namen zu erhalten.

EINVAL — *size* ist gleich 0.

ERANGE — ist kleiner als 0 oder größer als 0 und kleiner als die Länge des Pfadnamens plus 1.

BEISPIEL

Es folgt ein Programm, das das aktuelle Dateiverzeichnis ausgibt:

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

main()
{
        char *cwd;
        if ((cwd = getcwd(NULL, 64)) == NULL)
        {
                perror("pwd");
                exit(2);
        }
        (void)printf("%s\n", cwd);
        return(0);
}
```

SIEHE AUCH

malloc(3C).

ERGEBNIS

Es wird NULL zurückgegeben und errno gesetzt, wenn *size* nicht groß genug ist oder wenn ein Fehler in einer unten liegenden Funktion auftritt.

getdate - Datums- und Zeitangaben umwandeln

```
#include <time.h>
struct tm *getdate (const char *string);
extern int getdate_err;
```

`getdate` wandelt benutzerdefinierbare Datums- und/oder Zeitangaben aus *string* in eine `tm`-Struktur um. Die Strukturdeklaration befindet sich in der Datei `time.h` (siehe auch `ctime`(3C)).

Zum Zerlegen und Interpretieren der Eingabezeichenkette werden benutzerdefinierte Schablonen verwendet. Diese Schablonen sind Textdateien, welche der Benutzer anlegt; diese Textdateien werden über die Umgebungsvariable `DATEMSK` angegeben. Jede Zeile der Schablone stellt eine akzeptierbare Datums- und/oder Zeitangabe dar; dabei werden einige der Felddeskriptoren verwendet, die auch das Kommando `date` verwendet. Die erste Zeile in der Schablone, die der Eingabespezifikation entspricht, wird zur Interpretation und Umwandlung in das interne Zeitformat verwendet. Ist die Operation erfolgreich, so liefert die Funktion `getdate` einen Zeiger auf eine Struktur vom Typ `tm` zurück; ansonsten wird `NULL` zurückgegeben und die globale Variable `getdate_err` gesetzt.

Die folgenden Felddeskriptoren werden unterstützt:

`%%`	das gleiche wie `%`
`%a`	abgekürzter Wochentagsname
`%A`	ausgeschriebener Wochentagsname
`%b`	abgekürzter Monatsname
`%B`	ausgeschriebener Monatsname
`%c`	lokale Datums- und Zeitdarstellung
`%d`	Monatstag (01 - 31; die führende 0 ist optional)
`%e`	das gleiche wie `%d`
`%D`	Datum als `%m/%d/%y`
`%h`	abgekürzter Monatsname
`%H`	Stunde (00 - 23)
`%I`	Stunde (01 - 12)
`%m`	Monatsnummer (01 - 12)
`%M`	Minute (00 - 59)
`%n`	das gleiche wie `\n`
`%p`	lokales Äquivalent zu AM oder PM
`%r`	Zeit als `%I:%M:%S %p`
`%R`	Zeit als `%H:%M`
`%S`	Sekunde (00 - 59)
`%t`	Tabulator einfügen

%T	Zeit als %H:%M:%S
%w	Wochentagsnummer (0 - 6; Sonntag = 0)
%x	lokale Datumsrepräsentation
%X	lokale Zeitrepräsentation
%y	Jahr mit Jahrhundert (00 - 99)
%Y	Jahr als ccyy (z.B. 1986)
%Z	Zeitzonenname oder keine Zeichen, wenn keine Zeitzone existiert

Die Monats- und Wochentagsnamen können aus einer beliebigen Kombination von kleinen und großen Buchstaben bestehen. Der Benutzer kann bestimmen, daß die Angabe der Eingabezeit oder des Eingabedatums sprachabhängig ist. Dies geschieht durch Setzen der Werte LC_TIME und LC_CTYPE bei setlocale.

Das folgende Beispiel zeigt den möglichen Inhalt einer Schablone:

```
%m
%A %B %d %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p,%B %dnd
%A den %d. %B %Y %H.%M Uhr
```

Einige Beispiele für gültige Eingabespezifikationen für die obigen Schablonen:

```
getdate("10/1/87 4 PM")
getdate("Friday")
getdate("Friday September 19 1987, 10:30:30")
getdate("24,9,1986 10:30")
getdate("at monday the 1st of december in 1986")
getdate("run job at 3 PM, december %2nd")
```

Wenn die Umgebungsvariable LANG auf german gesetzt wird, ist folgende Angabe gültig:

```
getdate("Freitag den 10. Oktober 1986 10.30 Uhr")
```

Lokale Zeit- und Datumsangaben werden ebenfalls unterstützt. Das folgende Beispiel zeigt, wie lokale Datums- und Zeitangaben in Schablonen definiert werden können.

Aufruf	Zeile in Schablonendatei
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

Die folgenden Regeln gelten für die Umwandlung von Eingabespezifikationen in das interne Format:

- Ist nur der Wochentag gegeben, wird der aktuelle Tag angenommen, wenn der angegebene Wochentag identisch mit dem aktuellen Tag ist. Liegt der gegebene Tag vor dem aktuellen, wird der Wochentag aus der nächsten Woche genommen.

- Ist nur der Monat angegeben, wird der aktuelle Monat angenommen, wenn der angegebene Monat gleich dem aktuellen Monat ist. Ist der angegebene Monat kleiner als der aktuelle Monat, wird das nächste Jahr angenommen, wenn ansonsten kein Jahr angegeben ist. (Der erste Tag des Monats wird angenommen, wenn kein Tag angegeben ist).
- Wird keine Stunde, Minute und Sekunde angegeben, wird die aktuelle Stunde, Minute und Sekunde übernommen.
- Wird kein Datum angegeben, wird der aktuelle Tag angenommen, wenn die angegebene Stunde größer als die aktuelle Stunde ist. Ist die angegebene Stunde kleiner als die aktuelle, so wird der nächste Tag angenommen.

Die folgenden Beispiele verdeutlichen die obigen Regeln. Es wird angenommen, daß das aktuelle Datum Montag 22. September 12:19:47 EDT 1986 ist und die Umgebungsvariable LANG nicht eingestellt ist.

Eingabe	Zeile in Schablonendatei	Datum
Mon	%a	Mon Sep 22 12:19:48 EDT 1986
Sun	%a	Sun Sep 28 12:19:49 EDT 1986
Fri	%a	Fri Sep 26 12:19:49 EDT 1986
September	%B	Mon Sep 1:19:49 EDT 1986
January	%B	Thu Jan 1:19:49 EST 1987
December	%B	Mon Dec 1:19:49 EST 1986
Sep Mon	%b %a	Mon Sep 1:19:50 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:50 EST 1987
Dec Mon	%b %a	Mon Dec 1:19:50 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:51 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

DATEIEN

/usr/lib/locale/<locale>/LC_Zeit	sprachabhängige druckbare Dateien
/usr/lib/locale/<locale>/LC_CTYPE	zeichensatzabhängige druckbare Dateien

SIEHE AUCH

setlocale(3C), ctype(3C), environ(5).

ERGEBNIS

Bei einem Fehler liefert `getdate` den Wert `NULL` und setzt die Variable `getdate_err`, um den Fehler anzuzeigen.

Die möglichen Werte von `getdate_err` haben folgende Bedeutung.

1 Die Umgebungsvariable `DATEMSK` ist undefiniert oder Null.

2 Die Schablonendatei kann nicht zum Lesen geöffnet werden.

3 Der Dateistatus konnte nicht gelesen werden.

4 Die Schablonendatei ist keine reguläre Datei.

5 Ein Fehler trat beim Lesen der Schablonendatei auf.

6 `malloc` konnte nicht erfolgreich ausgeführt werden, da zu wenig Speicherplatz verfügbar war.

7 Es gibt keine Zeile aus der Schablonendatei, die der Eingabe entspricht.

8 Das Eingabeformat ist ungültig (z.B. `February 31`).

HINWEIS

Nachfolgende Aufrufe von `getdate` ändern den Inhalt von `getdate_err`.

Daten vor 1970 und nach 2037 sind ungültig.

`getdate` macht ausdrücklichen Gebrauch von den in `ctype`(3C) definierten Makros.

getenv - Wert für Umgebungsvariable zurückgeben

```
#include <stdlib.h>
char *getenv (const char *name);
```

`getenv` durchsucht die Umgebung des Prozesses (siehe `environ`(5)) nach einer Zeichenkette der Form *name=value* und gibt einen Zeiger auf den Wert *value* in der aktuellen Umgebung zurück, wenn eine solche Zeichenkette vorhanden ist. Andernfalls gibt `getenv` den Nullzeiger zurück.

SIEHE AUCH

`exec`(2), `putenv`(3C), `environ`(5).

getgrent - Gruppendatei-Eintrag bestimmen

```
#include <grp.h>
struct group *getgrent (void);
struct group *getgrgid (gid_t gid);
struct group *getgrnam (const char *name);
void setgrent (void);
void endgrent (void);
struct group *fgetgrent (FILE *f);
```

`getgrent`, `getgrgid` und `getgrnam` geben jeweils Zeiger auf ein Objekt mit nachstehender Struktur zurück, die die einzelnen Felder einer Zeile der Datei `/etc/group` enthält. Jede Zeile enthält eine 'Gruppen-'Struktur, die in der Include-Datei `grp.h` definiert ist, mit folgenden Elementen:

```
struct        group {
      char    *gr_name;       /* Name der Gruppe */
      char    *gr_passwd;     /* verschlüsseltes GruppenpaßwortT */
      gid_t   gr_gid;         /* numerische Gruppennummer */
      char    **gr_mem;       /* Zeiger auf Namen der Gruppenmitglieder */
};
```

Beim ersten Aufruf gibt `getgrent` einen Zeiger auf die erste Gruppen-Struktur in der Datei zurück; danach gibt es einen Zeiger auf die nächste Gruppen-Struktur in der Datei zurück. Auf diese Weise können aufeinanderfolgende Aufrufe zum Absuchen der gesamten Datei verwendet werden. `getgrgid` durchsucht die Datei vom Anfang, bis eine numerische Gruppennummer gefunden wird, die *gid* entspricht, und gibt einen Zeiger auf die Struktur zurück, in der sie gefunden wurde.

`getgrnam` durchsucht die Datei vom Anfang ab, bis ein Gruppenname gefunden wird, der *name* entspricht, und gibt einen Zeiger auf die Struktur zurück, in der sie gefunden wurde. Wenn beim Lesen ein Dateiende oder Fehler gefunden wird, geben diese Funktionen einen Nullzeiger zurück.

Ein Aufruf von `setgrent` bewirkt ein Zurücksetzen des Schreib-/Lesezeigers auf den Anfang der Gruppendatei und ermöglicht damit ein wiederholtes Suchen. `endgrent` kann am Ende der Verarbeitung aufgerufen werden, um die Gruppendatei zu schließen.

`fgetgrent` gibt einen Zeiger auf die nächste Gruppen-Struktur in der Datei *f* zurück, die das Format von `/etc/group` haben muß.

DATEIEN

/etc/group

SIEHE AUCH

getlogin(3C), getpwent(3C).
group(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

getgrent, getgrgid, getgrnam und fgetgrent geben bei EOF oder einem Fehler einen Nullzeiger zurück.

HINWEIS

Alle Informationen sind in einem statischen Bereich enthalten. Dieser muß daher kopiert werden, wenn er gesichert werden soll.

getitimer, setitimer - Intervall-Timer lesen und setzen

```
#include <sys/time.h>
int getitimer(int which, struct itimerval *value);
int setitimer(int which, struct itimerval *value, struct itimerval *ovalue);
```

Das System bietet jedem Prozeß drei Intervall-Timer an, die in der Datei `sys/time.h` vereinbart werden. Der Aufruf `getitimer` speichert den aktuellen Wert des Timers *which* in der Struktur, auf die *value* zeigt. Der Aufruf `setitimer` setzt den Wert von *which* auf den Wert, der in der Struktur steht, auf die *value* zeigt; ist *ovalue* nicht gleich `NULL`, wird der vorherige Wert des Timers in der Struktur abgelegt, auf die *ovalue* zeigt.

Die Einstellung eines Timers wird durch die Struktur `itimerval` (siehe `gettimeofday`(3C) für die Definition von `timeval`) definiert, welche die folgenden Komponenten enthält:

```
struct timeval   it_interval;    /* Uhrintervall */
struct timeval   it_value;       /* aktueller Wert */
```

Wenn `it_value` ungleich Null ist, wird der Zeitpunkt des nächsten Ablaufens des Timers angegeben. Wenn `it_interval` ungleich Null ist, wird ein Wert angegeben, auf den `it_value` gesetzt wird, wenn der Timer abläuft. Wird `it_value` auf Null gesetzt, so wird der Timer deaktiviert, unabhängig des Werts von `it_interval`. Das Setzen von `it_interval` auf Null deaktiviert den Timer nach seinem nächsten Ablauf (vorausgesetzt, daß `it_value` ungleich Null ist).

Sind Zeitwerte kleiner als die Auflösung der Systemuhr, so werden diese auf die Auflösung der Systemuhr gerundet.

Die drei Timer sind:

`ITIMER_REAL`
: dekrementiert in Echtzeit. Das Signal `SIGALRM` wird gesendet, wenn dieser Timer abläuft.

`ITIMER_VIRTUAL`
: dekrementiert in der virtuellen Prozeßzeit. Dieser Timer läuft nur, wenn der Prozeß ausgeführt wird. Das Signal `SIGVTALRM` wird gesendet, wenn dieser Timer abläuft.

`ITIMER_PROF`
: dekrementiert beide in virtueller Prozeßzeit, wenn das System aufgrund des Prozesses läuft. Dieser Timer dient Interpretern zur statistischen Profilerstellung der Ausführungszeit interpretierter Programme. Jedesmal, wenn der Timer `ITIMER_PROF` abläuft, wird das Signal `SIGPROF` gesendet. Da dieses Signal Systemaufrufe des Prozesses unterbricht, müssen diejenigen Programme, die diesen Timer verwenden, darauf vorbereitet sein, die unterbrochenen Systemaufrufe zu wiederholen.

SIEHE AUCH

alarm(2), gettimeofday(3C).

ERGEBNIS

Wenn der Aufruf erfolgreich war, wird der Wert 0 zurückgegeben. Tritt ein Fehler auf, wird -1 zurückgegeben und der Fehlercode in die globale Variable errno geschrieben.

Unter den folgenden Bedingungen schlagen die Funktionen getitimer und setitimer fehl und setzen errno auf:

EINVAL Die angegebenen Sekunden sind größer als 100.000.000, die angegebenen Mikrosekunden sind größer oder gleich 1.000.000, oder der Parameter *which* wurde nicht erkannt.

HINWEIS

Das Feld mit den Mikrosekunden darf keinen Wert enthalten, der gleich oder größer als eine Sekunde ist.

setitimer ist unabhängig vom Systemaufruf alarm.

Benutzen Sie setitimer nicht zusammen mit der sleep-Routine. Ein Aufruf von sleep nach setitimer löscht die Informationen der Signalbehandlungsroutine des Benutzers.

getlogin - Login-Namen abfragen

```
#include <stdlib.h>
char *getlogin (void);
```

`getlogin` gibt einen Zeiger auf den Login-Namen zurück, der in `/var/adm/utmp` gefunden wird. Es kann in Verbindung mit `getpwnam` zur Ermittlung des korrekten Paßwortdatei-Eintrags verwendet werden, wenn dieselbe Benutzernummer von mehreren Login-Namen benutzt wird.

Wenn `getlogin` innerhalb eines Prozesses aufgerufen wird, der nicht mit einem Terminal verbunden ist, gibt es einen Nullzeiger zurück. Das korrekte Verfahren zur Bestimmung des Login-Namens besteht im Aufrufen von `cuserid` oder von `getlogin` und - falls dieses erfolglos ist - im Aufrufen von `getpwuid`.

DATEIEN

`/var/adm/utmp`

SIEHE AUCH

`cuserid`(3S), `getgrent`(3C), `getpwent`(3C), `utmp`(4).

ERGEBNIS

Gibt einen Nullzeiger zurück, wenn der Login-Name nicht gefunden wird.

HINWEIS

Der Rückgabezeiger zeigt auf statische Daten, deren Inhalt bei jedem Aufruf überschrieben wird.

getmntent, getmntany - Dateieintrag mnttab holen

```
#include <stdio.h>
#include <sys/mnttab.h>
int getmntent (FILE *fp, struct mnttab *mp);
int getmntany (FILE *fp, struct mnttab *mp, struct mnttab *mpref);
```

Die Funktionen `getmntent` und `getmntany` füllen die Struktur, auf die *mp* zeigt, mit den Feldern einer Zeile aus der Datei `/etc/mnttab`. Jede Zeile aus der Datei enthält eine Struktur vom Typ `mnttab`; diese Struktur wird in der Datei `sys/mnttab.h` wie folgt deklariert:

```
struct mnttab {
        char    *mnt_special;
        char    *mnt_mountp;
        char    *mnt_fstype;
        char    *mnt_mntopts;
        char    *mnt_time;
};
```

Die Bedeutung der einzelnen Felder wird im Abschnitt `mnttab`(4) erklärt.

`getmntent` liefert einen Zeiger auf die nächste `mnttab`-Struktur aus der Datei; aufeinanderfolgende Aufrufe können somit verwendet werden, um die gesamte Datei zu durchsuchen. `getmntany` durchsucht die Datei, die durch *fp* bezeichnet wird, solange, bis eine Zeile der Datei *mpref* entspricht. *mpref* entspricht der Zeile, wenn alle Einträge aus *mpref*, die nicht Null sind, den entsprechenden Einträgen aus der Datei gleichen. Beachten Sie, daß diese Routinen die Datei weder öffnen, schließen noch zurückpositionieren.

DATEIEN

`/etc/mnttab`

SIEHE AUCH

`mnttab`(4).

ERGEBNIS

Wenn der nächste Eintrag von `getmntent` erfolgreich gelesen werden kann oder die Funktion `getmntany` einen entsprechenden Eintrag gefunden hat, wird 0 zurückgegeben. Ist das Dateiende erreicht, wird -1 zurückgegeben. Tritt ein Fehler auf, wird ein Wert größer als 0 zurückgegeben. Die möglichen Fehlercodes sind:

`MNT_TOOLONG` Eine Zeile aus der Datei überschreitet die interne Pufferlänge `MNT_LINE_MAX`.

`MNT_TOOMANY` Eine Zeile aus der Datei enthält zu viele Felder.

`MNT_TOOFEW` Eine Zeile aus der Datei enthält zu wenig Felder.

HINWEIS

Die Komponenten der Struktur `mnttab` zeigen auf Daten, die sich in einem statischen Bereich befinden; diese Daten müssen also kopiert werden, wenn sie zur weiteren Bearbeitung gesichert werden sollen.

getopt - Optionsbuchstaben aus Argumentvektor abfragen

```
#include <stdlib.h>
int getopt (int argc, char * const *argv, const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

`getopt` gibt den nächsten Optionsbuchstaben in *argv* zurück, der einem Buchstaben in *optstring* entspricht. Es unterstützt alle Regeln der Norm für die Kommandosyntax (siehe `intro`(1)). Da alle neuen Kommandos der Kommandosyntax-Norm folgen sollen, sollten Sie `getopts`(1) `getopt`(3C) oder `getsubopts`(3C) zur Analyse der Positionsparameter und zur Prüfung auf Optionen verwenden, die für das entsprechende Kommando zulässig sind.

optstring muß die Optionsbuchstaben enthalten, die vom Kommando erkannt werden, das `getopt` verwendet; wenn hinter einem Buchstaben ein Doppelpunkt folgt, wird erwartet, daß die Option ein Argument oder eine Gruppe von Argumenten hat, die durch ein Leerzeichen von der Option getrennt sein können. *optarg* wird so gesetzt, daß es bei der Rückkehr von `getopt` auf den Anfang des Optionsarguments weist.

`getopt` setzt den *argv*-Index des nächsten zu verarbeitenden Arguments in *optind*. *optind* ist extern und wird vor dem ersten Aufruf von `getopt` auf 1 initialisiert.

Nach Verarbeitung aller Optionen (d.h. bis zum ersten Argument, das keine Option ist) gibt `getopt` `EOF` zurück. Zur Begrenzung des Endes der Optionen kann die Sonderoption -- (zwei Bindestriche) verwendet werden; wird diese angetroffen, erfolgt die Rückgabe von `EOF`, und -- wird übersprungen. Das ist sinnvoll, um Argumente zu bestimmen, die keine Optionen sind und mit - (Bindestrich) beginnen.

BEISPIEL

Der nachstehende Code-Ausschnitt zeigt, wie man die Argumente für ein Kommando verarbeiten kann, das die sich gegenseitig ausschließenden Optionen `a` und `b` sowie die Option `o` annehmen kann, die ein Optionsargument erfordert:

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

main (int argc, char **argv)
{
      int c;
      extern char *optarg;
      extern int optind;
      int aflg = 0;
      int bflg = 0;
      int errflg = 0;
      char *ofile = NULL;

      while ((c = getopt(argc, argv, "abo:")) != EOF)
            switch (c) {
```

```
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bflg++;
                        break;
                case 'o':
                        ofile = optarg;
                        (void)printf("ofile = %s\n", ofile);
                        break;
                case '?':
                        errflg++;
                }
        if (errflg) {
                (void)fprintf(stderr,
                        "usage: cmd [-a|-b] [-o<file>] files...\n");
                exit (2);
        }
        for ( ; optind < argc; optind++)
                (void)printf("%s\n", argv[optind]);
        return 0;
}
```

SIEHE AUCH

`getsubopt`(3C).
`getopts`(1), `intro`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

`getopt` gibt eine Fehlermeldung auf der Standard-Fehlerausgabe aus und gibt ein `?` (Fragezeichen) zurück, wenn es einen Optionsbuchstaben antrifft, der in *optstring* nicht enthalten ist, oder wenn es kein Argument nach einer Option findet, die ein Argument erwartet. Diese Fehlermeldung kann durch Setzen von `opterr` auf 0 abgeschaltet werden. Der Wert des Zeichens, das den Fehler verursachte, ist in `optopt`.

HINWEIS

Die Bibliotheksroutine `getopt` überprüft nicht vollständig auf notwendige Argumente. Wenn zum Beispiel eine Optionszeichenkette `a:b` und die Eingabe `-a -b` gegeben sind, nimmt `getopt` an, daß `-b` das notwendige Argument für die Option `-a` ist, und nicht, daß ein notwendiges Argument für `-a` fehlt.

Es ist eine Verletzung der Kommandosyntax-Norm (siehe `intro`(1)) für Optionen, wenn mehrere Optionen zusammengefaßt werden und die letzte davon ein Argument benötigt. Das Kommando `cmd -aboxxx Datei`, bei dem `a` und `b` normale Optionen sind und die Option `o` das Argument `xxx` benötigt, ist ein Beispiel dafür. Obwohl diese Syntax in der aktuellen Implementierung erlaubt ist, sollte sie nicht verwendet werden, da sie in zukünftigen Versionen vielleicht nicht mehr unterstützt wird. Die korrekte Syntax ist `cmd -ab -oxxx Datei`.

getpass - Paßwort lesen

```
#include <stdlib.h>
char *getpass (const char *prompt);
```

`getpass` gibt die Zeichenkette *prompt* auf der Standard-Fehlerausgabe aus, schaltet den Echo-Modus aus und liest anschließend von der Datei `/dev/tty` bis zu einem Neue-Zeile-Zeichen oder bis zu einem `EOF`. Ein Zeiger auf eine mit dem Null-Byte abgeschlossene Zeichenkette von höchstens acht Zeichen wird zurückgegeben. Kann `/dev/tty` nicht eröffnet werden, wird ein Nullzeiger zurückgegeben. Ein Interrupt beendet den Eintrag und sendet vor der Rückkehr ein Unterbrechungssignal an den Aufrufprozeß.

DATEIEN

`/dev/tty`

HINWEIS

Der Rückgabewert weist auf statische Daten, deren Inhalt bei jedem Aufruf überschrieben wird.

getpw - Benutzernummer feststellen

```
#include <stdlib.h>
int getpw (uid_t uid, char *buf);
```

`getpw` durchsucht die Paßwortdatei nach der Benutzernummer *uid*, kopiert die Zeile der Paßwortdatei, in der *uid* gefunden wurde, in das Feld, auf das *buf* zeigt, und gibt 0 zurück. `getpw` gibt ungleich 0 zurück, wenn die Benutzernummer *uid* nicht gefunden werden kann.

Diese Funktion ist hier nur im Hinblick auf die Kompatibilität mit Vorgänger-Systemen vorhanden und sollte nicht verwendet werden; siehe `getpwent`(3C), das Funktionen enthält, die statt dessen zu verwenden sind.

DATEIEN

`/etc/passwd`

SIEHE AUCH

`getpwent`(3C).
`passwd(4)` in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

`getpw` gibt ungleich Null bei einem Fehler zurück.

getpwent - Paßwortdatei-Eintrag ändern

```
#include <pwd.h>
struct passwd *getpwent (void);
struct passwd *getpwuid (uid_t uid);
struct passwd *getpwnam (const char *name);
void setpwent (void);
void endpwent (void);
struct passwd *fgetpwent (FILE *f);
```

`getpwent`, `getpwuid` und `getpwnam` geben jeweils einen Zeiger auf ein Objekt mit nachstehender Struktur zurück, die die einzelnen Felder einer Zeile der Datei `/etc/passwd` enthält. Jede Zeile in der Datei enthält eine `passwd`-Struktur (Paßwort-Struktur), die in der Include-Datei `pwd.h` deklariert ist:

```
struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        uid_t   pw_uid;
        gid_t   pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

Beim ersten Aufruf gibt `getpwent` einen Zeiger auf die erste `passwd`-Struktur in der Datei zurück. Danach gibt es einen Zeiger auf die nächste `passwd`-Struktur in der Datei zurück. Auf diese Weise können aufeinanderfolgende Aufrufe zum Durchsuchen der gesamten Datei verwendet werden. `getpwuid` sucht vom Anfang der Datei an, bis eine numerische Benutzernummer gefunden wird, die gleich *uid* ist, und gibt einen Zeiger auf die Struktur zurück, in der sie gefunden wurde. `getpwnam` sucht vom Anfang der Datei, bis ein Login-Name gefunden wird, der zu *name* paßt, und gibt einen Zeiger auf die Struktur zurück, in der er gefunden wurde. Wird beim Lesen ein Dateiende oder ein Fehler gefunden, geben diese Funktionen einen Nullzeiger zurück.

Ein Aufruf von `setpwent` bewirkt das Zurücksetzen des Schreib-/Lesezeigers der Paßwortdatei auf den Dateianfang und ermöglicht somit ein wiederholtes Suchen. `endpwent` kann zum Schließen der Paßwortdatei nach Beendigung der Verarbeitung aufgerufen werden.

`fgetpwent` gibt einen Zeiger auf die nächste `passwd`-Struktur in der Datei *f* zurück, die das Format von `/etc/passwd` haben muß.

DATEIEN

`/etc/passwd`

SIEHE AUCH

`getlogin`(3C), `getgrent`(3C).
`passwd`(4) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

`getpwent`, `getpwnid`, `getpwnam` und `fgetpwent` geben einen Nullzeiger zurück, wenn `EOF` oder ein Fehler vorliegt.

HINWEIS

Alle Daten werden in einen statischen Bereich geschrieben und müssen daher kopiert werden, wenn sie gesichert werden sollen.

gets, fgets - Zeichenkette aus einem Stream lesen

```
#include <stdio.h>
char *gets (char *s);
char *fgets (char *s, int n, FILE *Stream);
```

`gets` liest Zeichen von der Standardeingabe `stdin` (siehe `intro`(3)) in das Feld, auf das *s* zeigt, bis ein Neue-Zeile-Zeichen gelesen oder das Dateiende erreicht wird. Das Neue-Zeile-Zeichen wird gelöscht und die Zeichenkette mit dem Null-Byte beendet.

`fgets` liest Zeichen von *Stream* in das Feld, auf das *s* zeigt, bis *n* -1 Zeichen gelesen sind oder ein Neue-Zeile-Zeichen auf *s* übertragen wurde oder das Dateiende erreicht ist. Die Zeichenkette wird mit dem Null-Byte beendet.

Bei der Verwendung von `gets` kann ein unbestimmtes Verhalten auftreten, wenn die Länge einer Eingabezeile die Größe von *s* überschreitet. Verwenden Sie deshalb `fgets` anstelle von `gets`.

SIEHE AUCH

`lseek`(2), `read`(2), `ferror`(3S), `fopen`(3S), `fread`(3S), `getc`(3S), `scanf`(3S), `stdio`(3S), `ungetc`(3S).

ERGEBNIS

Wenn das Dateiende erreicht wird und keine Zeichen gelesen wurden, werden keine Zeichen in *s* abgelegt und ein Nullzeiger zurückgegeben. Wenn ein Lesefehler auftritt, wie z.B. beim Versuch, diese Funktionen auf eine Datei anzuwenden, die nicht zum Lesen geöffnet wurde, wird ein Nullzeiger zurückgegeben und die Fehleranzeige für den Stream gesetzt. Wenn das Dateiende erreicht ist, wird die `EOF`-Anzeige des Streams gesetzt. Andernfalls wird *s* zurückgegeben.

getspent - Eintrag der Shadow-Paßwortdatei ändern

```
#include <shadow.h>
struct spwd *getspent (void);
struct spwd *getspnam (const char *name);
int lckpwdf (void);
int ulckpwdf (void);
void setspent (void);
void endspent (void);
struct spwd *fgetspent (FILE *fp);
```

Die Routinen `getspent` und `getspnam` liefern beide einen Zeiger auf ein Objekt zurück, das die Felder einer Zeile aus der Datei `/etc/shadow` enthält. Jede Zeile aus der Datei enthält eine 'Shadow-Kennwort'-Struktur, die in der Datei `shadow.h` vereinbart wird:

```
struct spwd{
        char    *sp_namp;
        char    *sp_pwdp;
        long    sp_lstchg;
        long    sp_min;
        long    sp_max;
        long    sp_warn;
        long    sp_inact;
        long    sp_expire;
        unsigned long   sp_flag;
};
```

Die Routine `getspent` liefert beim ersten Aufruf einen Zeiger auf die erste Struktur vom Typ `spwd` aus der Datei zurück; danach wird der Zeiger auf die nächste `spwd`-Struktur aus der Datei zurückgegeben; die komplette Datei kann somit durch aufeinanderfolgende Funktionsaufrufe durchsucht werden. Die Routine `getspnam` sucht vom Anfang der Datei an, bis der Anmeldename *name* gefunden wird, und liefert einen Zeiger auf die entsprechende Struktur zurück, in der der Name gefunden wurde. Die Funktionen `getspent` und `getspnam` schreiben in die Felder `sp_min`, `sp_max`, `sp_lstchg`, `sp_warn`, `sp_inact`, `sp_expire`, oder `sp_flag` den Wert -1, wenn das entsprechende Feld aus der Datei `/etc/shadow` leer ist. Ist das Dateiende erreicht, tritt ein Lese- oder Formatfehler in der Datei auf, liefern diese Funktionen einen Nullzeiger und setzen `errno` auf `EINVAL`.

`/etc/.pwd.lock` ist die Sperrdatei. Sie wird verwendet, um den Zugriff zum Ändern der Kennwortdateien `/etc/passwd` und `/etc/shadow` zu erlangen. `lckpwdf` und `ulckpwdf` sind Routinen, die den Änderungszugriff für die Kennwortdatei über die Sperrdatei erhalten. Ein Prozeß ruft zuerst `lckpwdf` auf, um die Sperrdatei zu sperren; dadurch erhält er das exklusive Recht, die Dateien `/etc/passwd` und/oder `/etc/shadow` zu ändern. Nach Durchführung der Änderungen sollte ein Prozeß die Sperre der Sperrdatei über

ulckpwdf wieder aufheben. Dieser Mechanismus verhindert das gleichzeitige Ändern der Kennwortdateien.

lckpwdf versucht, die Datei /etc/.pwd.lock innerhalb von 15 Sekunden zu sperren. Gelingt dies nicht, weil /etc/.pwd.lock beispielsweise schon gesperrt ist, wird -1 zurückgeliefert. Bei erfolgreicher Durchführung wird ein Wert ungleich -1 zurückgegeben.

ulckpwdf versucht, die Datei /etc/.pwd.lock freizugeben. Gelingt dies nicht, weil /etc/.pwd.lock bereits freigegeben ist, wird -1 zurückgegeben. Bei erfolgreicher Durchführung wird 0 zurückgegeben.

Der Aufruf der Routine setspent verursacht das Zurückspulen der Shadow-Kennwortdatei; dadurch können wiederholte Suchdurchläufe ausgeführt werden. Die Routine endspent wird aufgerufen, um die Shadow-Kennwortdatei zu schließen, nachdem die Bearbeitung beendet ist.

Die Routine fgetspent liefert einen Zeiger auf die nächste spwd-Struktur im Stream *fp* zurück, die dem Format von /etc/shadow entspricht.

DATEIEN

```
/etc/shadow
/etc/passwd
/etc/.pwd.lock
```

SIEHE AUCH

getpwent(3C), putpwent(3C), putspent(3C).

ERGEBNIS

getspent, getspnam, lckpwdf, ulckpwdf, und fgetspent liefern einen Nullzeiger bei Erreichen des Dateiendes oder bei Auftreten eines Fehlers.

HINWEIS

Diese Routine ist nur zum internen Gebrauch gedacht; Kompatibilität wird nicht garantiert.

Alle Daten befinden sich in einem statischen Bereich; die Daten sollten also kopiert werden, falls sie gesichert werden sollen.

getsubopt - Unteroptionen aus einer Zeichenkette heraustrennen

```
#include <stdlib.h>
int getsubopt (char **optionp, char * const *tokens, char **valuep);
```

`getsubopt` trennt Unteroptionen aus einem Schalterargument heraus, welches zuerst durch `getopt` verarbeitet wurde. Diese Unteroptionen werden durch Kommata getrennt und dürfen entweder aus einem einzelnen Token oder einer Token-Wert-Kombination bestehen, die durch ein Gleichheitszeichen getrennt wird. Da Kommata Unteroptionen in der Optionszeichenkette begrenzen, dürfen sie nicht Teil der Unteroption oder des Wertes einer Unteroption sein. Ein Kommando, das diese Syntax verwendet, ist `mount`(1M); es erlaubt dem Benutzer die Angabe der Parameter zum Einhängen über die Option `-o`:

```
mount -o rw,hard,bg,wsize=1024 speed:/usr /usr
```

In diesem Beispiel gibt es vier Unteroptionen: `rw`, `hard`, `bg`, und `wsize`; der letzten Unteroption wird der Wert 1024 zugewiesen.

`getsubopt` erhält die Adresse eines Zeigers auf die Optionszeichenkette, die einen Vektor möglicher Tokens darstellt, und die Adresse eines Zeigers auf eine Wertzeichenkette. Der Index des Tokens, das der Unteroption aus der übergebenen Zeichenkette entspricht, wird zurückgeliefert; wird keine entsprechende Unteroption gefunden, wird -1 zurückgegeben. Wenn die Optionszeichenkette bei *optionp* nur eine Unteroption enthält, aktualisiert `getsubopt` *optionp* so, daß auf auf das Nullzeichen am Ende der Zeichenkette gezeigt wird; ansonsten wird die Suboption isoliert, indem das trennende Komma durch ein Nullzeichen ersetzt wird, und *optionp* zeigt auf den Anfang der nächsten Unteroption. Wird der Unteroption ein Wert zugewiesen, aktualisiert `getsubopt` *valuep*, so daß auf das erste Zeichen des Wertes gezeigt wird. Ansonsten wird *valuep* auf `NULL` gesetzt.

Der Token-Vektor wird als Folge von Zeigern auf durch Null abgeschlossene Zeichenketten organisiert. Das Ende des Token-Vektors wird durch einen Nullzeiger gekennzeichnet.

`getsubopt` liefert dann, wenn *valuep* nicht `NULL` ist, die Unteroption zurück, der ein Wert zugewiesen wurde. Das aufrufende Programm kann diese Information verwenden, um zu bestimmen, ob das Vorhandensein oder das Fehlen eines Wertes für diese Unteroption einen Fehler darstellt.
Findet `getsubopt` keine Unteroption im Feld *tokens*, sollte das aufrufende Programm entscheiden, ob es sich hierbei um einen Fehler handelt, oder ob die nichterkannte Option an ein anderes Programm übergeben werden sollte.

BEISPIEL

Der folgende Programmauszug zeigt, wie die Optionen des Kommandos `mount` mit `getsubopt` verarbeitet werden:

```
#include <stdlib.h>

char *myopts[] = {
#define READONLY        0
                        "ro",
#define READWRITE       1
                        "rw",
#define WRITESIZE       2
                        "wsize",
#define READSIZE        3
                        "rsize",
                        NULL};
main(argc, argv)
        int  argc;
        char **argv;
{
        int sc, c, errflag;
        char *options, *value;
        extern char *optarg;
        extern int optind;
        .
        .
        .
        while((c = getopt(argc, argv, "abf:o:")) != -1) {
                switch (c) {
                case 'a': /* Option a bearbeiten */
                        break;
                case 'b': /* Option b bearbeiten */
                        break;
                case 'f':
                        ofile = optarg;
                        break;
                case '?':
                        errflag++;
                        break;
                case 'o':
                        options = optarg;
                        while (*options != '\0')
                                switch(getsubopt(&options,myopts,&value)) {
                                case READONLY : /* Option ro bearbeiten */
                                        break;
                                case READWRITE : /* Option rw bearbeiten */
                                        break;
                                case WRITESIZE : /* Option wsize bearbeiten */
                                        if (value == NULL) {
                                                error_no_arg();
                                                errflag++;
                                        } else
                                                write_size = atoi(value);
                                        break;
                                case READSIZE : /* Option rsize bearbeiten */
                                        if (value == NULL) {
                                                error_no_arg();
                                                errflag++;
                                        } else
                                                read_size = atoi(value);
                                        break;
                                default :
                                        /* unbekanntes Token bearbeiten */
                                        error_bad_token(value);
                                        errflag++;
                                        break;
                                }
                        }
                        break;
```

```
                }
        }
        if (errflag) {
                /* Hinweise zu Benutzung ausgeben */
        }
        for (; optind<argc; optind++) {
                /* restliche Argumente bearbeiten */
        }
        .
        .
        .
}
```

SIEHE AUCH

`getopt`(3C).

ERGEBNIS

`getsubopt` liefert -1, wenn sich das angegebene Token nicht im Token-Vektor befindet. Die Variable, die durch *valuep* adressiert wird, enthält einen Zeiger auf das erste Zeichen des Tokens, welches nicht erkannt wurde.

Die durch *optionp* adressierte Variable zeigt auf die nächste Option, die verarbeitet werden soll, oder auf ein Nullzeichen, wenn es keine weiteren Optionen gibt.

HINWEIS

Während der Verarbeitung der Tokens werden Kommata aus der Optionszeichenkette in Nullzeichen geändert. Leerzeichen in Tokens oder Token-Wert-Kombinationen müssen vor der Shell durch Anführungszeichen geschützt werden.

gettimeofday, settimeofday - Datum und Zeit lesen und setzen

```
#include <sys/time.h>
int gettimeofday (struct timeval *tp);
int settimeofday (struct timeval *tp);
```

`gettimeofday` liest und `settimeofday` setzt die aktuelle Zeit für das System. Die aktuelle Zeit wird in verstrichenen Sekunden und Mikrosekunden seit dem 1. Januar 1970, 00:00 (Coordinated Universal Time) angegeben. Die Auflösung der Systemuhr ist hardwareabhängig; die Zeit kann stetig oder in Zeittakten aktualisiert werden.

tp zeigt auf eine Struktur vom Typ `timeval`, welche die folgenden Komponenten enthält:

```
long    tv_sec;     /* Sekunden seit dem 1. Januar 1970 */
long    tv_usec;    /* und Mikrosekunden */
```

Wenn *tp* ein Nullzeiger ist, wird die aktuelle Zeit weder gelesen noch gesetzt.

Die Umgebungsvariable `TZ` enthält Zeitzoneninformationen. Siehe `timezone`(4).

Nur privilegierte Benutzer können die Zeit einstellen.

SIEHE AUCH

`adjtime`(2), `ctime`(3C), `timezone`(4).

ERGEBNIS

Der Rückgabewert -1 zeigt an, daß ein Fehler aufgetreten ist; `errno` wird dann gesetzt. Die folgenden Fehlercodes sind für `errno` möglich:

`EINVAL` *tp* gibt eine ungültige Zeit an.

`EPERM` Ein nichtprivilegierter Benutzer hat versucht, die Zeit oder die Zeitzone einzustellen.

HINWEIS

Die Implementierung von `settimeofday` ignoriert das Feld `tv_usec` von `tp`. Wenn die Zeit mit größerer Genauigkeit als einer Sekunde eingestellt werden muß, sollte `settimeofday` für die Sekunden und danach `adjtime` für die Feinabstimmung verwendet werden.

gettxt - Zeichenkette aus Meldungsdatei holen

```
#include <unistd.h>
char *gettxt (const char *msgid, const char *dflt_str);
```

`gettxt` liest eine Zeichenkette aus einer Meldungsdatei. Die Argumente der Funktion bestehen aus einer Meldungsidentifikation *msgid* und einer voreingestellten Zeichenkette *dflt_str*, welche verwendet wird, wenn das Lesen fehlschlägt.

Die Zeichenketten befinden sich in Dateien, die mit dem Programmierwerkzeug `mkmsgs` (siehe `mkmsgs`(1)) erstellt wurden. Diese Dateien stehen im Verzeichnis `/usr/lib/locale/`*locale*`/LC_MESSAGES`.

Das Verzeichnis *locale* kann als die Sprache betrachtet werden, in der die Zeichenketten geschrieben wurden. Sie können bestimmen, daß Meldungen in einer bestimmten Sprache angezeigt werden, indem sie die Umgebungsvariable `LC_MESSAGES` setzen. Wenn `LC_MESSAGES` nicht gesetzt ist, wird die Umgebungsvariable `LANG` verwendet. Ist `LANG` nicht eingestellt, werden die Dateien aus `/usr/lib/locale/C/LC_MESSAGES/*` verwendet.

Sie können die Sprache der angezeigten Meldungen auch durch Aufruf der Funktion `setlocale` mit entsprechenden Argumenten ändern.

Kann `gettxt` eine Meldung in einer bestimmten Sprache nicht einlesen, so wird versucht, dieselbe Meldung in amerikanischem Englisch einzulesen. Tritt dabei ein Fehler auf, hängt das weitere Verhalten vom zweiten Argument *dflt_str* ab. Es wird ein Zeiger auf das zweite Argument zurückgeliefert, wenn das zweite Argument keine Nullzeichenkette darstellt. Wenn *dflt_str* auf eine Nullzeichenkette zeigt, wird ein Zeiger auf die englische Meldung `"Message not found!!\n"` zurückgeliefert.

Die Syntax für *msgid* beim Aufruf von `gettxt` sieht wie folgt aus:

<msgid> = <msgfilename>:<msgnumber>

Das erste Feld wird verwendet, um die Datei anzugeben, welche die Zeichenketten enthält; die Länge dieses Feldes ist auf 14 Zeichen beschränkt. Bei diesen Zeichen darf es sich nicht um das Zeichen \0 (Nullzeichen) und den ASCII-Codes für / (Schrägstrich) und : (Doppelpunkt) handeln. Die Namen der Meldungsdateien müssen dieselben sein, die durch `mkmsgs` erzeugt und im Verzeichnis `/usr/lib/locale/`*locale*`/LC_MESSAGES/*` installiert wurden. Das numerische Feld zeigt die Position der Zeichenkette in der Datei an. Die Zeichenketten sind von *1* bis *n* durchnumeriert, wobei *n* die Anzahl der Zeichenketten in der Datei darstellt.

Wird kein korrekter Wert für `msgid` oder keine gültige Meldungsnummer an `gettxt` übergeben, wird ein Zeiger auf die Zeichenkette `"Message not found!!\n"` zurückgeliefert.

BEISPIEL

```
gettxt("UX:10", "Hallo Leute\n")
gettxt("UX:10", "")
```

`UX` ist der Name der Datei, welche die Meldungen enthält. `10` ist die Meldungsnummer.

DATEIEN

`/usr/lib/locale/C/LC_MESSAGES/*`
: enthält voreingestellte Meldungsdateien, die durch `mkmsgs` erzeugt wurden.

`/usr/lib/locale/`*`locale`*`/LC_MESSAGES/*`
: enthält Meldungsdateien für verschiedene Sprachen, die durch `mkmsgs` erzeugt wurden.

SIEHE AUCH

`fmtmsg`(3C), `setlocale`(3C), `environ`(5).
`exstr`(1), `mkmsgs`(1), `srchtxt`(1) in "SINIX V5.41 Kommandos".

getut - auf utmp-Dateieintrag zugreifen

```
#include <utmp.h>
struct utmp *getutent (void);
struct utmp *getutid (const struct utmp *id);
struct utmp *getutline (const struct utmp *line);
struct utmp *pututline (const struct utmp *utmp);
void setutent (void);
void endutent (void);
int utmpname (const char *file);
```

`getutent`, `getutid`, `getutline` und `pututline` geben jeweils einen Zeiger auf eine Struktur mit den folgenden Elementen zurück:

```
char    ut_user[8];       /* Login-Name des Benutzers */
char    ut_id[4];         /* /sbin/inittab id (meist # der Zeile) */
char    ut_line[12];      /* Gerätename (Konsole, lnxx) */
short   ut_pid;           /* Prozeßnummer    */
short   ut_type;          /* Eintragstyp */
struct  exit_status {
} ut_exit;                /* Exit-Status eines Prozesses, */
                          /* der als DEAD_PROCESS markiert ist. */
time_t  ut_time;          /* Uhrzeit-Eintrag ist erfolgt */
```

Die Struktur *exit_status* umfaßt die folgenden Elemente:

```
short   e_termination;    /* Ende-Status */
short   e_exit;           /* Exit-Status */
```

`getutent` liest den nächsten Eintrag von einer `utmp`-ähnlichen Datei. Ist die Datei noch nicht geöffnet, wird sie von diesem Kommando geöffnet. Ist das Dateiende erreicht, ist `getutent` erfolglos.

`getutid` sucht vom aktuellen Punkt vorwärts in der `utmp`-Datei, bis es einen Eintrag mit einem *ut_type* findet, der mit *id->ut_type* übereinstimmt, wenn der angegebene Typ `RUN_LVL`, `BOOT_TIME`, `OLD_TIME` oder `NEW_TIME` ist. Wenn der in *id* angegebene Typ `INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS` oder `DEAD_PROCESS` ist, dann gibt `getutid` einen Zeiger auf den ersten Eintrag zurück, dessen Typ einer dieser vier Typen ist und dessen *ut_id*-Feld *id->ut_id* entspricht. Wenn das Ende der Datei ohne passenden Eintrag erreicht wird, ist das Kommando erfolglos.

`getutline` sucht vom aktuellen Punkt vorwärts in der `utmp`-Datei, bis es einen Eintrag des Typs `LOGIN_PROCESS` oder `USER_PROCESS` findet, der außerdem auch eine Zeichenkette *ut_line* aufweist, die der Zeichenkette *line->ut_line* entspricht. Wenn das Ende der Datei erreicht ist, ohne daß ein entsprechender Eintrag gefunden wurde, ist das Kommando erfolglos.

`pututline` schreibt die angegebene `utmp`-Struktur in die `utmp`-Datei. Es sucht unter Verwendung von `getutid` vorwärts nach der richtigen Stelle. Es wird erwartet, daß der Benutzer von `pututline` bereits mit einer `getut`-Routine nach dem richtigen Eintrag gesucht hat. In diesem Fall führt `pututline` keine Suche aus. Wenn `pututline` keine entsprechende Stelle für den neuen Eintrag findet, hängt es diesen an das Ende der Datei. Es gibt einen Zeiger auf die Struktur `utmp` zurück.

Wenn die gesamte Datei geprüft werden soll, muß vor der Suche nach einem Eintrag der Schreib-/Lesezeiger an den Anfang der Datei zurückgesetzt werden. Dieses Rücksetzen ist vor jeder Suche nach einem neuen Eintrag auszuführen, wenn eine Prüfung der gesamten Datei gewünscht ist.

`endutent` schließt die geöffnete Datei.

`utmpname` ermöglicht dem Benutzer, eine andere Datei als `/var/adm/utmp` zu durchsuchen. Üblicherweise handelt es sich dabei um die Datei `/var/adm/wtmp`. Wenn diese Datei nicht vorhanden ist, zeigt sich dies erst beim ersten Versuch, auf diese Datei zuzugreifen. `utmpname` eröffnet die Datei nicht. Es schließt lediglich die alte Datei, wenn sie gerade geöffnet ist, und sichert den neuen Dateinamen.

DATEIEN

```
/var/adm/utmp
/var/adm/wtmp
```

SIEHE AUCH

`ttyslot`(3C), `utmp`(4).

ERGEBNIS

Ein Nullzeiger wird bei erfolglosem Lesen zurückgegeben, wobei es sich um fehlende Erlaubnis, Erreichen des Dateiendes oder erfolgloses Schreiben handeln kann.

HINWEIS

Der aktuellste Eintrag wird in einer statischen Struktur gesichert. Soll auf eine Datei mehrfach zugegriffen werden, müssen die Einträge vor dem nächsten Zugriff kopiert werden. Bei jedem Aufruf von `getutid` oder `getutline` prüft die Funktion zunächst die statische Struktur, bevor sie weitere E/A ausführt. Wenn der Inhalt der statischen Struktur dem Inhalt der gesuchten Struktur entspricht, wird die Suche beendet. Aus diesem Grunde ist es für `getutline` zum Suchen nach mehrfachem Auftreten erforderlich, die statische Struktur nach jedem Erfolg zu löschen, weil `getutline` andernfalls immer nur dieselbe Struktur zurückgeben würde. Es gibt jedoch eine Ausnahme zu der Regel über das Löschen der Struktur vor dem Ausführen weiterer Lese-Operationen. Das implizierte Lesen, das von `pututline` ausgeführt wird (wenn es feststellt, daß es sich nicht schon am richtigen Platz in der Datei befindet), beschädigt den Inhalt der von den Funktionen `getutent`, `getutid` oder `getutline` zurückgegebenen statischen Struktur nicht, wenn der Benutzer diesen Inhalt gerade geändert und den Zeiger an `pututline` zurückgegeben hat.

Diese Funktionen verwenden gepufferte Standard-E/A für die Eingabe; `pututline` jedoch schreibt ungepuffert, um Wettlaufbedingungen zwischen den Prozessen zu vermeiden, die versuchen, die Dateien `utmp` und `wtmp` zu ändern.

getutx - auf utmpx-Eintrag zugreifen

```
#include <utmpx.h>
struct utmpx *getutxent (void);
struct utmpx *getutxid (const struct utmpx *id);
struct utmpx *getutxline (const struct utmpx *line);
struct utmpx *pututxline (const struct utmpx *utmpx);
void setutxent (void);
void endutxent (void);
int utmpxname (const char *file);
void getutmp (struct utmpx *utmpx, struct utmp *utmp);
void getutmpx (struct utmp *utmp, struct utmpx *utmpx);
void updwtmp (char *wfile, struct utmp *utmp);
void updwtmpx (char *wfilex, struct utmpx *utmpx);
```

`getutxent`, `getutxid` und `getutxline` liefern einen Zeiger auf eine Struktur des folgenden Typs:

```
struct     utmpx {
     char     ut_user[32];  /* Anmeldenamen des Benutzers */
     char     ut_id[4];     /* /sbin/inittab id (normalerweise Zeilennummer) */
     char     ut_line[32];  /* Gerätename (Konsole, lnxx) */
     pid_t    ut_pid;       /* Prozeßnummer */
     short    ut_type;      /* Art des Eintrags */
     struct     exit_status {
         short     e_termination; /* Ende-Status */
         short     e_exit;        /* Exit-Status */
     } ut_exit;    /* Exit-Status eines Prozesses markiert als DEAD_PROCESS */
     struct timeval     ut_tv;     /* Zeiteintrag gemacht */
     short ut_syslen;     /* signifikante Länge von ut_host */
                          /* einschließlich abschließender Null */
     char     ut_host[257];     /* Host-Name, falls gegeben */
};
```

`getutxent` liest den nächsten Eintrag aus einer `utmpx`-ähnlichen Datei. Wenn die Datei noch nicht geöffnet ist, wird sie geöffnet. Wird das Dateiende erreicht, scheitert die Operation.

`getutxid` sucht von der aktuellen Position in der Datei `utmpx` vorwärts, bis ein Eintrag gefunden wird, dessen *ut_type* dem *id->ut_type* entspricht, wenn der angegebene Typ `RUN_LVL`, `BOOT_TIME`, `OLD_TIME`, oder `NEW_TIME` ist. Ist der in *id* angegebene Typ `INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS` oder `DEAD_PROCESS`, dann liefert `getutxid` einen Zeiger auf den ersten Eintrag, dessen Typ einem dieser vier Typen entspricht und dessen *ut_id*-Feld *id->ut_id* entspricht. Wird das Dateiende erreicht, ohne daß ein entsprechender Eintrag gefunden wurde, scheitert die Operation.

`getutxline` sucht vorwärts von der aktuellen Position in der Datei `utmpx`, bis ein Eintrag mit dem Typ `LOGIN_PROCESS` oder `USER_PROCESS` gefunden wird, dessen *ut_line* Zeichenkette *line->ut_line* entspricht. Wenn das Dateiende erreicht wird, ohne daß ein entsprechender Eintrag gefunden wird, scheitert die Operation.

`pututxline` schreibt die angegebene `utmpx`-Struktur in die Datei `utmpx`. `getutxid` wird verwendet, um nach der korrekten Position in der Datei zu suchen, falls diese noch nicht gegeben ist. Es wird erwartet, daß der Anwender der `pututxline`-Routine den entsprechenden Eintrag mit einer der `getutx`-Routinen gesucht hat. Ist dies der Fall, führt `pututxline` keine Suche durch. Wenn `pututxline` keine entsprechende Stelle für den neuen Eintrag findet, wird der Eintrag am Dateiende angehängt. Ein Zeiger auf die Struktur `utmpx` wird zurückgegeben.

`setutxent` setzt die Position des Eingabe-Streams auf den Dateianfang. Dies sollte gemacht werden, bevor in der gesamten Datei nach einem neuen Eintrag gesucht wird.

`endutxent` schließt die geöffnete Datei.

Mit `utmpxname` läßt sich der Dateiame `/var/adm/utmpx` in einen anderen Namen ändern. In den meisten Fällen handelt es sich bei dieser anderen Datei um `/var/adm/wtmpx`. Wenn die Datei nicht existiert, fällt dies erst auf, wenn der erste Zugriff auf diese Datei ausgeführt wird. `utmpxname` öffnet die Datei nicht; die alte Datei wird lediglich geschlossen, sofern sie geöffnet war, und der neue Dateiname wird gespeichert. Der neue Dateiname muß mit dem Zeichen `x` enden, um die Namensbehandlung zu vereinfachen. Ansonsten wird der Fehlercode 1 zurückgeliefert.

`getutmp` kopiert den Inhalt der `utmpx`-Struktur in die `utmp`-Struktur. Wenn die Daten aus einer Komponente von `utmpx` nicht in das entsprechende Feld von `utmp` passen, werden Daten abgetrennt.

`getutmpx` kopiert die Daten aus den Komponenten der Struktur `utmp` in die entsprechenden Komponenten der `utmpx`-Struktur.

`updwtmp` prüft die Existenz von *wfile* und deren Paralleldatei, dessen Name durch Anhängen eines `x` an *wfile* generiert wird. Wenn nur eine der Dateien existiert, wird die zweite erzeugt und initialisiert, so daß sie den Status der existierenden Datei widerspiegelt. *utmp* wird in die Datei *wfile* geschrieben; die entsprechende `utmpx`-Struktur wird in die Paralleldatei geschrieben.

`updwtmpx` prüft die Existenz von *wfilex* und deren Paralleldatei, dessen Name durch Abtrennen des letzten `x` von *wfilex* generiert wird. Wenn nur eine der Dateien existiert, wird die zweite erzeugt und initialisiert, so daß sie den Status der existierenden Datei widerspiegelt. *utmpx* wird in die Datei *wfilex* geschrieben; die entsprechende `utmp`-Struktur wird in die Paralleldatei geschrieben.

DATEIEN

/var/adm/utmp, /var/adm/utmpx
/var/adm/wtmp, /var/adm/wtmpx

SIEHE AUCH

ttyslot(3C), utmp(4), utmpx(4).

ERGEBNIS

Tritt beim Prüfen der Zugriffsrechte oder des Dateiende-Status ein Lese- oder Schreibfehler auf, so wird ein Nullzeiger zurückgegeben.

HINWEIS

Der aktuellste Eintrag wird in einer statischen Struktur abgelegt. Bevor erneut auf die Datei zugegriffen wird, muß dieser Eintrag kopiert werden. Bei jedem Aufruf von getutxid oder getutxline überprüfen die Routinen die statische Struktur, bevor weitere E/A-Operationen ausgeführt werden. Wenn der Inhalt der statischen Struktur dem gesuchten Muster entspricht, wird nicht weitergesucht. Sollen mit getutxline mehrere identische Einträge gesucht werden, so muß nach jeder erfolgreichen Suchoperation die statische Struktur gelöscht werden; macht man dies nicht, würde getutxline dieselbe Struktur immer wieder zurückgeben. Es gibt eine Ausnahme für die obige Regel: Das implizite Lesen durch pututxline (wenn die korrekte Position in der Datei noch nicht erreicht wurde) verletzt nicht den Inhalt der statischen Struktur, die von getutxent, getutxid, oder getutxline zurückgeliefert wird, wenn der Benutzer die Inhalte verändert hat und den Zeiger an pututxline zurückgibt.

Diese Routinen verwenden die gepufferte Standard-E/A zur Eingabe; pututxline verwendet eine ungepufferte Schreiboperation, um Konflikte zwischen Prozessen zu vermeiden, welche die utmpx- und wtmpx-Dateien ändern wollen.

getvfsent - vfstab-Dateieintrag lesen

```
#include <stdio.h>
#include <sys/vfstab.h>
int getvfsent (FILE *fp, struct vfstab *vp);
int getvfsfile (FILE *fp, struct vfstab *vp, char *file);
int getvfsspec (FILE *fp, struct vfstab *vp, char *spec);
int getvfsany (FILE *fp, struct vfstab *vp, struct vfstab *vref);
```

`getvfsent`, `getvfsfile`, `getvfsspec`, und `getvfsany` füllen die Struktur, auf die *vp* zeigt, mit den Feldern einer Zeile aus der Datei `/etc/vfstab`. Jede Zeile der Datei enthält eine Struktur vom Typ `vfstab`, die in der Datei `sys/vfstab.h` vereinbart wird:

```
char    *vfs_special;
char    *vfs_fsckdev;
char    *vfs_mountp;
char    *vfs_fstype;
char    *vfs_fsckpass;
char    *vfs_automnt;
char    *vfs_mntopts;
```

Die Bedeutung der einzelnen Felder wird in `vfstab`(4) beschrieben.

`getvfsent` liefert einen Zeiger auf die nächste `vfstab`-Struktur in der Datei zurück. Aufeinanderfolgende Aufrufe können somit verwendet werden, um die komplette Datei zu durchsuchen. `getvfsfile` durchsucht die Datei, die durch *fp* bezeichnet wird, nach einem Einhängepunkt, der *file* entspricht; *vp* wird mit den Feldern aus der Zeile gefüllt. `getvfsspec` durchsucht die Datei *fp*, bis eine Gerätedatei gefunden wird, die *spec* entspricht; *vp* wird mit den Feldern aus der Zeile gefüllt. *spec* versucht, dem Gerätetyp (block- oder zeichenorientiert) und der höher- (major) und niedrigerwertigen (minor) Gerätenummer zu entsprechen. Kann keine Entsprechung gefunden werden, werden die Zeichenketten verglichen. `getvfsany` durchsucht die Datei *fp* nach einer Zeile, die *vref* entspricht. *vref* entspricht einer Zeile, wenn alle Einträge aus *vref*, die nicht Null sind, den Feldern aus der Datei gleichen.

Beachten Sie, daß diese Routinen die Datei weder öffnen, schließen noch zurückspulen.

DATEIEN

`/etc/vfstab`

ERGEBNIS

Wenn der nächste Eintrag von `getvfsent` erfolgreich gelesen werden kann oder mit `getvfsfile`, `getvfsspec` oder `getvfsany` eine Übereinstimmung gefunden wird, so wird 0 zurückgegeben. Wird das Dateiende beim Lesen erreicht, liefern diese Funktionen -1 zurück. Tritt ein Fehler auf, wird ein Wert größer als 0 zurückgegeben. Die möglichen Fehlercodes sind:

`VFS_TOOLONG` Die Länge einer Zeile der Datei überschritt die interne Pufferlänge `VFS_LINE_MAX`.

`VFS_TOOMANY` Eine Zeile der Datei enthält zu viele Felder.

`VFS_TOOFEW` Eine Zeile der Datei enthält zu wenig Felder.

HINWEIS

Die Komponenten der Struktur `vfstab` zeigen auf Daten, welche sich in einem statischen Bereich befinden; um die Daten zu sichern, müssen sie kopiert werden.

hsearch, hcreate, hdestroy - Hash-Suchtabellen verwalten

```
#include <search.h>
ENTRY *hsearch (ENTRY item, ACTION action);
int hcreate (size_t nel);
void hdestroy (void);
```

`hsearch` ist eine auf der Grundlage des Algorithmus D (6.4) von Knuth verallgemeinerte Suchfunktion für Hash-Tabellen. Sie gibt einen Zeiger in eine Hash-Tabelle zurück, der die Stelle anzeigt, an der ein Eintrag gefunden wurde. Die von `hsearch` benutzte Vergleichsfunktion ist `strcmp` (siehe `string`(3C)). *item* ist eine Struktur des in der Include-Datei `search.h` definierten Typs `ENTRY`, die zwei Zeiger enthält: *item.key* weist auf den Vergleichsschlüssel, und *item.data* weist auf alle anderen Daten in Zusammenhang mit diesem Schlüssel. Zeiger auf Typen, die nicht `void` sind, sind zu Zeigern auf `void` umzuwandeln. *action* ist ein Element des Aufzählungstyps `ACTION` (definiert in `search.h`), das die Behandlung des Eintrags angibt, wenn dieser nicht in der Tabelle gefunden werden kann. `ENTER` zeigt an, daß *item* an einem geeigneten Punkt in die Tabelle eingetragen werden soll. Ist ein Duplikat eines existierenden Eintrags vorhanden, so wird der neue Eintrag nicht eingetragen, und `hsearch` gibt den Zeiger zu dem existierenden Eintrag zurück. `FIND` zeigt an, daß kein Eintrag vorgenommen werden soll. Eine erfolglose Suche wird durch die Rückgabe eines Nullzeigers gemeldet.

`hcreate` weist ausreichend Speicher für die Tabelle zu und muß vor `hsearch` aufgerufen werden. *nel* schätzt die größtmögliche Anzahl von Einträgen, die eine Tabelle enthalten wird. Diese Zahl kann durch den Algorithmus nach oben justiert werden, damit bestimmte, mathematisch günstige Umstände erreicht werden.

`hdestroy` zerstört die Suchtabelle. Ein weiterer Aufruf von `hcreate` kann folgen.

BEISPIEL

Im folgenden Beispiel werden Zeichenketten, gefolgt von zwei Zahlen, eingelesen und in einer Hash-Tabelle gespeichert, wobei Duplikate gelöscht werden. Anschließend werden Zeichenketten eingelesen und der übereinstimmende Eintrag in der Hash-Tabelle gesucht und ausgedruckt.

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

struct info { /* dies sind die in der Tabelle gespeicherten */
      int age, room;  /* Daten außer dem Schlüssel. */
};

#define NUM_EMPL   5000    /* # der Elemente in Suchtabelle */

main( )
{
```

```
        /* Platz zum Speichern von Zeichenketten */
        char string_space[NUM_EMPL*20];
        /* Platz zum Speichern von Mitarbeiterdaten */
        struct info info_space[NUM_EMPL];
        /* nächster verfügbarer Platz im string_space */
        char *str_ptr = string_space;
        /* nächster verfügbarer Platz im info_space */
        struct info *info_ptr = info_space;
        ENTRY item, *found_item;
        /* zu suchender Name in Tabelle */
        char name_to_find[30];
        int i = 0;

        /* Tabelle erstellen */
        (void) hcreate(NUM_EMPL);
        while (scanf("%s%d%d", str_ptr, &info_ptr->age,
                &info_ptr->room) != EOF && i++ <NUM-EMPL) {
                /* Daten in Struktur und Struktur in item setzen */
                item.key = str_ptr;
                item.data = (void *)info_ptr;
                str_ptr += strlen(str_ptr) + 1;
                info_ptr++;
                /* item in Tabelle setzen */
                (void) hsearch(item, ENTER);
        }

        * auf Tabelle zugreifen */
                item.key = name_to_find;
                while (scanf("%s", item.key) != EOF) {
                   if ((found_item = hsearch(item, FIND)) != NULL) {
                        /* wenn item in Tabelle ist */
                        (void)printf("found %s, age = %d, room = %d\n"
                            found_item->key,
                             ((struct info *)found_item->data)->age,
                             ((struct info *)found_item->data)->room);
                   } else {
                        (void)printf("diesen Mitarbeiter gibt es nicht: %s\n"
                             name_to_find)
                   }
                }
        return 0;
}
```

SIEHE AUCH

bsearch(3C), lsearch(3C), malloc(3C), malloc(3X), string(3C), tsearch(3C).

ERGEBNIS

hsearch gibt einen Nullzeiger zurück, wenn die Aktion FIND (suchen) ist und der Eintrag nicht gefunden werden konnte oder wenn die Aktion ENTER (eintragen) ist und die Tabelle voll ist.

hcreate gibt Null zurück, wenn es nicht genug Speicherplatz für die Tabelle zuweisen kann.

HINWEIS

hsearch und hcreate verwenden malloc(3C) um Speicherplatz zuzuweisen.

Es kann jeweils nur eine Hash-Suchtabelle aktiv sein.

initgroups - zusätzliche Gruppenzugriffslisten initialisieren

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

int initgroups (const char *name, gid_t basegid)
```

`initgroups` liest die Gruppendatei unter Verwendung von `getgrent`, um die Gruppenmitgliedschaft für den Benutzer *name* herauszufinden und initialisiert die zusätzliche Gruppenzugriffsliste des aufrufenden Prozesses mit `setgroups`. Die Gruppennummer *basegid* ist ebenfalls in der zusätzlichen Gruppenzugriffsliste enthalten. Dies ist typischerweise die echte Gruppennummer aus der Kennwortdatei.

Wird die Anzahl der Gruppen einschließlich des Eintrags *basegid* größer als `NGROUPS_MAX`, so werden weitere Gruppeneinträge ignoriert.

`initgroups` schlägt fehl und ändert die zusätzliche Gruppenzugriffsliste nicht, wenn `EPERM` die effektive Benutzernummer nicht die des Systemverwalters ist.

SIEHE AUCH

`setgroups`(2), `getgrent`(3C).

ERGEBNIS

Nach erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. Ansonsten wird -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

insque, remque - Element aus Queue entfernen / in Queue einfügen

```
#include <search.h>
void insque(struct qelem *elem, struct qelem *pred);
void remque(struct qelem *elem);
```

`insque` und `remque` ändern Queues, die aus doppelt verketteten Elementen erzeugt werden. Jedes Element in dem Queue muß der folgenden Form entsprechen:

```
struct qelem {
        struct  qelem *q_forw;
        struct  qelem *q_back;
        char    q_data[];
};
```

`insque` fügt *elem* in einem Queue direkt hinter *pred* ein. `remque` entfernt den Eintrag *elem* aus einem Queue.

isnan - Gleitkommazahl-Typ bestimmen

```
#include <ieeefp.h>
int isnand (double dsrc);
int isnanf (float fsrc);
int finite (double dsrc);
fpclass_t fpclass (duoble dsrc);
int unordered (double dsrc1, double dsrc2);
#include <math.h>
int isnan (double dsrc);
```

`isnan`, `isnand`, und `isnanf` geben 1 zurück, wenn das Argument *dsrc* oder *fsrc* keine gültige Zahlendarstellung ist (`NaN`); andernfalls wird 0 zurückgegeben. Die Funktionalität von `isnan` ist identisch mit `isnand`.

`isnanf` ist ausgeführt als ein Makro, welches in der `ieeefp.h` Include-Datei enthalten ist. `fpclass` gibt die Klasse, zu welcher *dsrc* gehört, zurück. Die zehn möglichen Klassen sind folgende:

```
FP_SNAN       signalisierende NaN
FP_QNAN       ruhige NaN
FP_NINF       negative Unendlichkeit
FP_PINF       positive Unendlichkeit
FP_NDENORM    negative nichtnormalisierte Zahl ungleich Null
FP_PDENORM    positive nichtnormalisierte Zahl ungleich Null
FP_NZERO      negative Null
FP_PZERO      positive Null
FP_NNORM      negative normalisierte Zahl ungleich Null
FP_PNORM      positive normalisierte Zahl ungleich Null
```

`finite` gibt 1 zurück, wenn der Parameter *dsrc* weder unendlich, noch `NaN` ist; anderenfalls gibt es 0 zurück.

`unordered` gibt 1 zurück, wenn einer seiner Parameter bezüglich des anderen ungeordnet ist. Dieses ist gleichbedeutend mit der Meldung, daß einer der Parameter `NaN` ist. Falls keiner der Parameter `NaN` ist, wird 0 zurückgegeben.

SIEHE AUCH

`fpgetround`(3C), `intro`(3M).

l3tol, ltol3 - Zahlen umwandeln

```
#include <stdlib.h>
void l3tol (long *lp, const char *cp, int n);
void ltol3 (char *cp, const long *lp, int n);
```

`l3tol` wandelt eine Liste von *n* ganzen Drei-Byte-Zahlen, die in eine Zeichenkette gepackt sind und auf die *cp* zeigt, in eine Liste ganzer Zahlen vom Typ `long` um, auf die *lp* zeigt.

`ltol3` führt die umgekehrte Umwandlung von `long`-Zahlen (*lp*) in ganze Drei-Byte-Zahlen aus (*cp*).

Diese Funktionen sind für die Dateisystemverwaltung nützlich, in der die Blocknummern drei Bytes lang sind.

SIEHE AUCH

`fs`(4).

HINWEIS

Wegen möglicher Differenzen in der Reihenfolge der Bytes sind die numerischen Werte der Zahlen vom Typ `long` maschinenabhängig.

localeconv - numerische Formatierinformationen lesen

```
#include <locale.h>
struct lconv *localeconv (void);
```

`localeconv` setzt die Komponenten eines Objekts vom Typ `struct lconv` (definiert in `locale.h`) auf die Werte, die den Formatierungskonventionen der numerischen Größen (monetäre und andere Größen) der aktuellen lokalen Umgebung entsprechen (siehe `setlocale`(3C)). Die Definition von `struct lconv` ist im folgenden aufgeführt (die Werte für die Komponenten in der lokalen C-Umgebung sind innerhalb der Kommentarzeichen angegeben):

```
char *decimal_point;        /* "." */
char *thousands_sep;        /* "" (Zeichenkette der Länge 0) */
char *grouping;             /* "" */
char *int_curr_symbol;      /* "" */
char *currency_symbol;      /* "" */
char *mon_decimal_point;    /* "" */
char *mon_thousands_sep;    /* "" */
char *mon_grouping;         /* "" */
char *positive_sign;        /* "" */
char *negative_sign;        /* "" */
char int_frac_digits;       /* CHAR_MAX */
char frac_digits;           /* CHAR_MAX */
char p_cs_precedes;         /* CHAR_MAX */
char p_sep_by_space;        /* CHAR_MAX */
char n_cs_precedes;         /* CHAR_MAX */
char n_sep_by_space;        /* CHAR_MAX */
char p_sign_posn;           /* CHAR_MAX */
char n_sign_posn;           /* CHAR_MAX */
```

Die Komponenten der Struktur vom Typ `char *` sind Zeichenketten, von denen jede (mit Ausnahme von `decimal_point`) auf "" zeigen kann; dadurch wird angegeben, daß der Wert der aktuellen lokalen Umgebung nicht verfügbar ist oder die Länge Null hat. Die Komponenten mit dem Typ `char` sind nichtnegative Zahlen, von denen jede den Wert `CHAR_MAX` (definiert in der Datei `limits.h`) annehmen kann; dadurch wird angezeigt, daß der Wert der aktuellen lokalen Umgebung nicht verfügbar ist. Die Komponenten haben folgende Bedeutungen:

`char *decimal_point`
: Dezimalpunktzeichen zur Formatierung nichtmonetärer Größen

`char *thousands_sep`
: Zeichen zur Trennung von Zifferngruppen links vom Dezimalpunkt zur Formatierung nichtmonetärer Größen

`char *grouping`
: Zeichenkette, in der jedes Element als ganze Zahl behandelt wird; die Anzahl der Ziffern, die die aktuelle Gruppe in einer formatierten, nichtmonetären Größe annimmt. Die Elemente von `grouping` werden wie folgt interpretiert:

CHAR-MAX
Keine weitere Gruppierung wird durchgeführt.

0 Das vorherige Element wird für die restlichen Ziffern wiederholt verwendet.

other
Der Wert ist die Anzahl der Ziffern, welche sich in der aktuellen Gruppe befinden. Das nächste Element wird überprüft, um die Größe der nächsten Zifferngruppe links von der aktuellen Gruppe zu bestimmen.

`char *int_curr_symbol`
für den Standort gültiges internationales Währungssymbol; steht linksbündig innerhalb eines vier Zeichen langen Feldes, das mit Leerzeichen an die Länge angepaßt wird. Die Zeichenketten sollten mit denen aus der folgenden Quelle übereinstimmen: *ISO 4217 Codes for the Representation of Currency and Funds.*

`char *currency_symbol`
lokales Währungssymbol für die aktuelle lokale Umgebung

`char *mon_decimal_point`
Dezimalpunkt für die Formatierung von monetären Größen

`char *mon_thousands_sep`
Trennzeichen für Zifferngruppen links vom Dezimalpunkt in formatierten, monetären Größen

`char *mon_grouping`
Zeichenkette, in der jedes Element als ganze Zahl verarbeitet wird, welche die Anzahl der Stellen angibt, die die aktuelle Gruppierung für formatierte, monetäre Größen darstellt; die Elemente von `mon_grouping` werden nach den Regeln verarbeitet, die auch für `grouping` gelten.

`char *positive_sign`
Zeichenkette, welche eine nichtnegative, formatierte, monetäre Größe anzeigt

`char *negative_sign`
Zeichenkette, welche eine negative, formatierte, monetäre Größe anzeigt

`char int_frac_digits`
Anzahl der Dezimalstellen, die in international formatierten, monetären Größen angezeigt werden

`char frac_digits`
Anzahl der Dezimalstellen, die in einer formatierten, monetären Größe dargestellt werden

`char p_cs_precedes`
: wird auf 1 oder 0 gesetzt, wenn das Währungssymbol `currency_symbol` vor oder hinter dem Wert für eine nichtnegative, formatierte, monetäre Größe erscheinen soll.

`char p_sep_by_space`
: wird auf 1 gesetzt, wenn das Währungssymbol `currency_symbol` durch ein Leerzeichen vom Wert einer nichtnegativen, formatierten, monetären Größe getrennt wird. Sonst wird die Komponente auf 0 gesetzt.

`char n_cs_precedes`
: Enthält diese Komponente den Wert 1, wird das Währungssymbol `currency_symbol` vor den Wert einer negativen, formatierten, monetären Größe geschrieben. Sonst wird die Komponente auf 0 gesetzt.

`char n_sep_by_space`
: wird auf 1 gesetzt, wenn das Währungssymbol `currency_symbol` durch ein Leerzeichen vom Wert einer negativen, formatierten, monetären Größe getrennt wird. Sonst wird die Komponente auf 0 gesetzt.

`char p_sign_posn`
: Diese Komponente wird auf einen Wert gesetzt, der die Position des positiven Vorzeichens `positive_sign` für eine nichtnegative, formatierte, monetäre Größe angibt. Der Wert von `p_sign_posn` wird wie folgt interpretiert:

  `0` Die Größe und das Währungssymbol `currency_symbol` werden in Klammern gesetzt.

  `1` Das Vorzeichen steht vor der Größe und dem Währungssymbol `currency_symbol`.

  `2` Das Vorzeichen steht hinter der Größe und dem Währungssymbol `currency_symbol`.

  `3` Das Vorzeichen steht direkt vor dem Währungssymbol `currency_symbol`.

  `4` Das Vorzeichen steht direkt hinter dem Währungssymbol `currency_symbol`.

`char n_sign_posn`
: wird auf einen Wert gesetzt, der die Position des negativen Vorzeichens `negative_sign` für eine negative, formatierte, monetäre Größe angibt. Der Wert von `n_sign_posn` wird nach den unter `p_sign_posn` beschriebenen Regeln interpretiert.

ERGEBNIS

`localeconv` liefert einen Zeiger auf das ausgefüllte Objekt. Die Struktur, auf die der Rückgabewert zeigt, kann durch einen weiteren Aufruf `localeconv` überschrieben werden.

BEISPIELE

Die folgende Tabelle demonstriert die Regeln zur Formatierung monetärer Größen anhand von vier Ländern:

Land	Positives Format	Negatives Format	Internationales Format
Italien	L.1.234	-L.1.234	ITL.1.234
Niederlande	F\ 1.234,56	F\ -1.234,56	NLG\ 1.234,56
Norwegen	kr1.234,56	kr1.234,56-	NOK\ 1.234,56
Schweiz	SFrs.1,234.56	SFrs.1,234.56C	CHF\ 1,234.56

Für diese vier Länder werden die entsprechenden Werte für die monetären Komponenten von `localeconv` wie folgt zurückgegeben:

	Italien	Niederlande	Norwegen	Schweiz
int_curr_symbol	"ITL."	"NLG\ "	"NOK\ "	"CHF\ "
currency_symbol	"L."	"F"	"kr"	"SFrs."
mon_decimal_point	""	","	","	"."
mon_thousands_sep	"."	"."	"."	","
mon_grouping	"\3"	"\3"	"\3"	"\
positive_sign	""	""	""	""
negative_sign	"-"	"-"	"-"	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2

DATEIEN

`/usr/lib/locale/`*`locale`*`/LC_MONETARY`	`LC_MONETARY` Datenbank für *locale*
`/usr/lib/locale/`*`locale`*`/LC_NUMERIC`	`LC_NUMERIC` Datenbank für *locale*

SIEHE AUCH

`setlocale`(3C).
`chrtbl`(1M), `montbl`(1M) in "Referenzhandbuch für Systemverwalter".

lockf - Sätze in Dateien sperren

```
#include <unistd.h>
int lockf (int fildes, int function, long size);
```

Mit `lockf` können Dateiabschnitte gesperrt werden; dabei hängen empfohlene oder obligatorische Schreibsperren jeweils von den Modusbits der Datei ab (siehe `chmod`(2)). Sperraufrufe von anderen Prozessen, die versuchen, einen bereits gesperrten Dateiabschnitt zu sperren, führen entweder zur Rückgabe eines Fehlerwerts oder pausieren solange, bis das Betriebsmittel freigegeben wird. Alle Sperren für einen Prozeß werden aufgehoben, wenn der Prozeß beendet wird.

fildes ist ein offener Dateideskriptor. Der Dateideskriptor muß die `O_WRONLY`- oder `O_RDWR`-Erlaubnis haben, damit die Sperre mit diesem Funktionsaufruf eingerichtet werden kann.

function ist ein Steuerwert, der die zu treffenden Maßnahmen angibt. Die zulässigen Werte für *function* sind, wie folgt, in `unistd.h` definiert:

```
#define    F_ULOCK    0   /* gesperrten Abschnitt freigeben */
#define    F_LOCK     1   /* Abschnitt exklusiv sperren */
#define    F_TLOCK    2   /* Abschnitt testen und exklusiv sperren */
#define    F_TEST     3   /* Abschnitt auf Sperren anderer Prozesse testen */
```

Alle anderen Werte von *function* sind für zukünftige Erweiterungen reserviert und führen zu einer Fehlermeldung, wenn sie nicht implementiert sind.

`F_TEST` wird verwendet um festzustellen, ob in einem Abschnitt eine Sperre eines anderen Prozesses existiert. `F_LOCK` und `F_TLOCK` sperren jeweils einen Abschnitt einer Datei, wenn dieser Abschnitt verfügbar ist. `F_ULOCK` hebt die Sperren eines Dateiabschnitts auf.

size ist die Anzahl zusammenhängender Bytes, die gesperrt oder entsperrt werden sollen. Das zu sperrende oder entsperrende Betriebsmittel beginnt am aktuellen Offset in der Datei und erstreckt sich vorwärts für ein positives *size* und rückwärts für ein negatives *size* (die vorhergehenden Bytes bis ausschließlich des aktuellen Offsets). Wenn *size* Null ist, wird der Abschnitt vom aktuellen Offset bis zum größten Datei-Offset gesperrt, d.h. vom aktuellen Offset bis zum gegenwärtigen oder bis zu jedem zukünftigen Dateiende. Ein Bereich braucht nicht einer Datei zugewiesen sein, damit er gesperrt werden kann, weil diese Sperren auch über das Ende der Datei hinausgehen können.

Die mit `F_LOCK` oder `F_TLOCK` gesperrten Abschnitte können einen vorher von demselben Prozeß gesperrten Abschnitt ganz oder teilweise enthalten bzw. in diesem Abschnitt enthalten sein. Gesperrte Abschnitte werden ab dem Punkt des Offsets entsperrt, bis *size* Bytes entsperrt worden sind oder bis zum Dateiende, wenn *size* 0 ist. Wenn diese Situation diese Situation in diesem oder in benachbarten Abschnitten eintritt, werden die Abschnitte zu einem Abschnitt zusammengefaßt. Wenn mit der Anforderung ein neues Element zur Tabelle der aktiven Sperren hinzugefügt werden muß und

diese Tabelle bereits voll ist, erfolgt eine Fehlermeldung, und der neue Abschnitt wird nicht gesperrt.

Die Anforderungen von F_LOCK und F_TLOCK unterscheiden sich nur in der Maßnahme, die getroffen wird, wenn das Betriebsmittel nicht zur Verfügung steht. F_LOCK bewirkt, daß der aufrufende Prozeß pausiert, bis das Betriebsmittel zur Verfügung steht. F_TLOCK bewirkt, daß die Funktion -1 zurückgibt und errno auf den Fehler EACCES setzt, wenn der Abschnitt bereits von einem anderen Prozeß gesperrt ist.

F_ULOCK-Anforderungen können einen oder mehrere gesperrte, vom Prozeß gesteuerte Abschnitte teilweise oder ganz freisetzen. Wenn die Abschnitte nicht ganz entsperrt werden, bleiben die übrigen Abschnitte weiterhin vom Prozeß gesperrt. Die Freigabe des mittleren Abschnitts eines gesperrten Abschnitts erfordert einen zusätzlichen Eintrag in der Tabelle der aktiven Sperren. Wenn diese Tabelle voll ist, wird errno auf ENOLK gesetzt und der angeforderte Abschnitt nicht freigegeben.

Die Möglichkeit eines Deadlocks entsteht, wenn ein Prozeß, der ein gesperrtes Betriebsmittel kontrolliert, durch Anforderung des gesperrten Betriebsmittels eines anderen Prozesses zum Pausieren veranlaßt wird. Daher wird bei Aufruf von lockf oder fcntl zunächst auf mögliche Deadlocks geprüft, bevor der Prozeß bis zur Freigabe eines noch gesperrten Betriebsmittels angehalten wird. Wenn das Warten auf ein gesperrtes Betriebsmittel ein Deadlock verursachent, wird eine Fehlermeldung zurückgegeben.

Das Warten auf ein Betriebsmittel wird mit einem beliebigen Signal unterbrochen. Der Systemaufruf alarm kann für die Bereitstellung einer Zeitsperre bei Anwendungen verwendet werden, die eine derartige Einrichtung benötigen.

lockf ist erfolglos, wenn einer oder mehrere der nachstehenden Punkte wahr sind:

EBADF — *fildes* ist keine gültige Kennzahl einer offenen Datei.

EACCES — *function* ist F_TLOCK oder F_TEST, und der Abschnitt ist bereits von einem anderen Prozeß gesperrt.

EDEADLK — *function* ist F_LOCK und ein Deadlock würde auftreten.

ENOLK — *function* ist F_LOCK, F_TLOCK oder F_ULOCK und die Anzahl der Einträge in der Sperrtabelle würde die in dem System zulässige Anzahl überschreiten.

ECOMM — *fildes* ist auf einem fernen Rechner, und die Verbindung zu diesem Rechner ist nicht mehr aktiv.

SIEHE AUCH

intro(2), alarm(2), chmod(2), close(2), creat(2), fcntl(2), open(2), read(2), write(2).

ERGEBNIS

Nach erfolgreicher Beendigung wird 0 zurückgegeben. Andernfalls wird -1 zurückgegeben, und errno wird zur Anzeige des Fehlers gesetzt.

HINWEIS

Unerwartete Ergebnisse können in Prozessen auftreten, die im Adreßraum des Benutzers puffern. Der Prozeß kann später Daten lesen oder schreiben, die gesperrt sind bzw. waren. Das Standard-E/A-Paket ist die häufigste Ursache für unerwartete Pufferungen.

Da die Variable errno in Zukunft auf EAGAIN und nicht auf EACCES gesetzt wird, wenn ein Dateiabschnitt bereits von einem anderen Prozeß gesperrt ist, müssen portable Anwenderprogramme beide Werte erwarten und prüfen.

lsearch, lfind - linear suchen und ändern

```
#include <search.h>

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

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

`lsearch` ist eine lineare Suchfunktion, die eine Verallgemeinerung des Algorithmus (6.1) S von Knuth ist. Sie gibt einen Zeiger in eine Tabelle zurück, der die Stelle angibt, an der ein gesuchter Wert gefunden wurde. Wenn der gesuchte Wert nicht auftritt, wird er am Ende der Tabelle eingetragen. *key* zeigt auf den Wert, der in der Tabelle gesucht werden soll. *base* zeigt auf das erste Element in der Tabelle. *nelp* zeigt auf eine ganze Zahl, die die aktuelle Anzahl der Elemente in der Tabelle enthält. Die Zahl wird erhöht, wenn der Wert zur Tabelle hinzugefügt wird. *width* ist die Größe eines Elements in Bytes. *compar* ist ein Zeiger auf die Vergleichsfunktion, die der Benutzer zur Verfügung stellen muß (`strcmp` zum Beispiel). Es werden zwei Argumente erwartet, die auf die Elemente zeigen, die verglichen werden. Die Funktion gibt 0 zurück, wenn die Elemente gleich sind, sonst ungleich 0.

`lfind` wirkt wie `lsearch`, wobei jedoch der gesuchte Wert nicht zur Tabelle hinzugefügt wird, wenn er nicht gefunden wird. Statt dessen wird ein Nullzeiger zurückgegeben.

HINWEIS

Die Zeiger auf den Schlüssel und das Element an der Basis der Tabelle können Zeiger auf einen beliebigen Typ sein.

Die Vergleichsfunktion muß nicht jedes Byte vergleichen, und so können die Elemente zusätzlich zu den zu vergleichenden Werten beliebige Daten enthalten.

Der zurückgegebene Wert sollte sich in den Typ Zeiger-auf-Element umwandeln lassen.

BEISPIEL

Das folgende Programm liest weniger als TABSIZE Zeichenketten mit einer Länge kleiner als ELSIZE ein, speichert sie bei gleichzeitiger Entfernung von Duplikaten in einer Tabelle und gibt dann jeden Eintrag aus.

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

#define TABSIZE 50
#define ELSIZE 120

main()
{
      char line[ELSIZE];      /* Puffer für die Eingabe */
      char tab[TABSIZE][ELSIZE];      /* Tabelle der Zeichenketten */
      size_t nel = 0;         /* Anzahl der Einträge in tab */
      int i;

      while (fgets(line, ELSIZE, stdin) != NULL &&
              nel < TABSIZE)
              (void) lsearch(line, tab, &nel, ELSIZE, mycmp);
      for( i = 0; i < nel; i++ )
              (void)fputs(tab[i], stdout);
      return 0;
}
```

SIEHE AUCH

bsearch(3C), hsearch(3C), string(3C), tsearch(3C).

ERGEBNIS

Wird der gesuchte Wert gefunden, geben lsearch und lfind einen Zeiger auf den Wert zurück. Andernfalls gibt lfind NULL zurück und lsearch einen Zeiger auf das neu hinzugefügte Element.

Undefinierte Ergebnisse können auftreten, wenn nicht genügend Speicherplatz für ein neues Element vorhanden ist.

makecontext, swapcontext - Benutzerkontext ändern

```
#include <ucontext.h>
void makecontext (ucontext_t *ucp, (void(*)())func, int argc,...);
int swapcontext (ucontext_t *oucp, ucontext_t *ucp);
```

Diese Funktionen sind nützlich für die Implementierung eines Kontextwechsels zwischen mehreren Kontrollfüssen innerhalb eines Benutzerprozesses.

`makecontext` verändert den durch *ucp* angegebenen Kontext, der über `getcontext` initialisiert wurde; wird dieser Kontext mit `swapcontext` oder `setcontext` aktiviert (siehe `getcontext`(2)), wird die Programmausführung durch Aufruf der Funktion *func* wieder aufgenommen, wobei die Argumente, die auf *argc* folgen, an `makecontext` übergeben werden. Der ganzzahlige Wert von *argc* muß der Anzahl der Argumente entsprechen, die auf *argc* folgt. Ansonsten ist das Verhalten undefiniert.

`swapcontext` sichert den aktuellen Kontext in der Kontextstruktur, auf die *oucp* zeigt, und setzt den Kontext auf die Kontextstruktur, auf die *ucp* zeigt.

Diese Funktionen schlagen fehl, wenn eine der folgenden Bedingungen erfüllt ist:

`ENOMEM` *ucp* hat nicht mehr genügend Stapelspeicher, um die Operation durchzuführen.

`EFAULT` *ucp* oder *oucp* zeigt auf eine ungültige Adresse.

SIEHE AUCH

`exit`(2), `getcontext`(2), `sigaction`(2), `sigprocmask`(2), `ucontext`(5).

ERGEBNIS

Nach erfolgreicher Ausführung liefert `swapcontext` den Wert Null zurück. Ansonsten wird -1 zurückgegeben und `errno` gesetzt.

HINWEIS

Die Länge der Struktur `ucontext_t` kann sich in zukünftigen Versionen ändern. Um eine Binärkompatibilität zu garantieren, müssen die Benutzer dieser Funktionen immer `makecontext` oder `getcontext` verwenden, um neue Kontexte zu erzeugen.

makedev, major, minor - Gerätenummer verwalten

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

dev_t makedev(major_t maj, minor_t min);

major_t major(dev_t device);

minor_t minor(dev_t device);
```

Die `makedev`-Routine liefert bei erfolgreicher Ausführung eine formatierte Gerätenummer zurück; bei einem Fehler wird `NODEV` zurückgegeben. *maj* ist die höherwertige Gerätenummer und *min* die niederwertige Gerätenummer. `makedev` kann verwendet werden, um eine Gerätenummer für `mknod`(2) zu erzeugen.

Die `major`-Routine liefert die höherwertige Komponente für *device*.

Die `minor`-Routine liefert die niederwertige Komponente für *device*.

`makedev` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EINVAL` Eines oder beide der Argumente *maj* und *min* sind zu groß.

`EINVAL` Die Gerätenummer, die aus *maj* und *min* erzeugt wurde, ist `NODEV`.

`major` schlägt fehl, wenn eine der folgenden Bedingungen erfüllt ist:

`EINVAL` Das Argument *device* ist `NODEV`.

`EINVAL` Die höherwertige Komponente von *device* ist zu groß.

`minor` schlägt fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` Das Argument *device* ist `NODEV`.

SIEHE AUCH

`stat`(2), `mknod`(2).

ERGEBNIS

Bei einem Fehler wird `NODEV` zurückgegeben und `errno` gesetzt.

malloc, free, realloc, calloc - Hauptspeicherplatz verwalten

```
#include <stdlib.h>
void *malloc (size_t size);
void free (void *ptr);
void *realloc (void *ptr, size_t size);
void *calloc (size_t nelem, size_t elsize);
void *memalign (size_t alignment, size_t size);
void *valloc (size_t size);
```

`malloc` und `free` stellen ein einfaches Paket zur Speicherplatzverwaltung zur Verfügung. `malloc` gibt einen Zeiger auf einen Block mit einer Größe von wenigstens *size* Bytes zurück, der für jeden Gebrauch geeignet ausgerichtet ist.

Das Argument von `free` ist ein Zeiger auf einen Block, der vorher von `malloc, calloc` oder `realloc` zugewiesen wurde; nach der Ausführung von `free` wird dieser Speicherplatz für eine neue Zuordnung zur Verfügung gestellt. Wenn *ptr* ein Nullzeiger ist, geschieht nichts.

Wenn der durch `malloc` zugewiesene Speicherplatz überschritten oder ein unbestimmtes Element an `free` übergeben wird, ergeben sich undefinierte Resultate.

`realloc` ändert die Größe des Blocks, auf den *ptr* zeigt, auf *size* Bytes und gibt einen Zeiger auf den (möglicherweise verlagerten) Block zurück. Der Inhalt bleibt bis zur kleineren der neuen bzw. alten Größe unverändert. Wenn *ptr* `NULL` ist, verhält sich `realloc` für *size* wie `malloc`. Wenn *size* Null ist und *ptr* kein Nullzeiger ist, wird das Objekt, auf das gezeigt wird, freigegeben.

`calloc` weist Speicher für ein Feld von *nelem* Elementen der Größe *elsize* zu. Der Speicherplatz wird mit Nullen initialisiert.

`memalign` weist einer bestimmten Ausrichtungsgrenze *size* Bytes zu und gibt einen Zeiger auf den zugewiesenen Block zurück. Der Wert der zurückgegebenen Adresse ist ein geradzahliges Vielfaches von *alignment*. Der Wert von *alignment* muß 2^n sein und muß größer oder gleich der Länge eines Wortes sein.

`valloc(`*size*`)` ist gleichbedeutend mit `memalign(sysconf(_SC_PAGESIZE),`*size*`)`.

Jede der Zuweisungsroutinen gibt einen Zeiger auf einen für die Speicherung jedes beliebigen Objekttyps geeignet ausgerichteten Speicherplatz zurück.

`malloc, realloc, calloc, memalign` und `valloc` können nicht erfolgreich ausgeführt werden, wenn nicht ausreichend Speicherplatz vorhanden ist.

SIEHE AUCH

`malloc`(3X).

ERGEBNIS

`malloc, realloc, calloc, memalign` und `valloc` geben einen Nullzeiger zurück, wenn nicht genug Speicher zur Verfügung steht. Wenn `realloc NULL` zurückgibt, wird der Block, auf den *ptr* zeigt, nicht zerstört. Ein Zeiger auf den angeforderten Platz wird zurückgeliefert.

Wenn bei `malloc` *size* gleich 0 oder bei `calloc` *elsize* gleich 0 ist, dann wird ein Zeiger mit einem eindeutigen Wert zurückgeliefert, der an `free` weitergegeben werden kann.

Wenn bei `realloc` *size* gleich 0 ist, wird der Speicherplatz mit der Adresse *ptr* wieder zur Verfügung gestellt. Diese Adresse muß eine gültige Adresse sein, die mit `malloc` zurückgeliefert wurde.

mbchar - Mehrbyte-Zeichen umwandeln

```
#include <stdlib.h>
int mbtowc (wchar_t *pwc, const char *s, size_t n);
int mblen (const char *s, size_t n);
int wctomb (char *s, wchar_t wchar);
```

Mehrbyte-Zeichen werden verwendet, um Zeichen in einem erweiterten Zeichensatz darzustellen. Dies wird benötigt, wenn acht Bits nicht ausreichen, um alle Zeichen des Zeichensatzes darzustellen.

Mit diesen Funktionen können Mehrbyte-Zeichen in Langzeichen übersetzt werden und umgekehrt. Langzeichen sind vom Typ `wchar_t` (definiert in `stdlib.h`); dieser Typ ist ganzzahlig und umfaßt einen Bereich, der alle Komponenten des größten erweiterten Zeichensatzes einer unterstützten lokalen Umgebung darstellen kann.

Für jede lokale Umgebung werden maximal drei erweiterte Zeichensätze unterstützt. Die Anzahl der Bytes in einem erweiterten Zeichensatz wird mit der Kategorie `LC_CTYPE` des Standorts definiert (siehe `setlocale`(3C)). Die maximale Anzahl von Bytes, die zur Darstellung eines Mehrbyte-Zeichens verwendet werden kann, darf nie größer als `MB_LEN_MAX` sein; diese Konstante wird in `limits.h` definiert. Die maximale Anzahl von Zeichen für einen erweiterten Zeichensatz für die aktuelle Umgebung wird durch das Makro `MB_CUR_MAX` festgelegt, welches in `stdlib.h` definiert ist.

`mbtowc` bestimmt die Anzahl der Bytes, die das Mehrbyte-Zeichen enthält, auf das *s* zeigt. Wenn *pwc* kein Nullzeiger ist, konvertiert `mbtowc` die Mehrbyte-Zeichen in ein Langzeichen und schreibt das Ergebnis in das Objekt *pwc*. Der Wert des Langzeichens für das Nullzeichen entspricht Null. Bis zu *n* Zeichen werden überprüft, angefangen mit dem Zeichen, auf das *s* zeigt.

Wenn *s* ein Nullzeiger ist, liefert `mbtowc` 0 zurück. Wenn *s* kein Nullzeiger ist, aber auf das Nullzeichen zeigt, wird 0 zurückgegeben; wenn die nächsten *n* oder weniger Bytes ein gültiges Mehrbyte-Zeichen darstellen, liefert `mbtowc` die Anzahl der Bytes, welche das konvertierte Mehrbyte-Zeichen belegt; ansonsten zeigt *s* nicht auf ein gültiges Mehrbyte-Zeichen und `mbtowc` liefert -1.

Zeigt *s* auf ein Mehrbyte-Zeichen, so bestimmt `mblen` die Anzahl der Bytes in diesem Mehrbyte-Zeichen. Diese Funktion ist äquivalent mit:

```
mbtowc ((wchar_t *)0, s, n);
```

`wctomb` bestimmt die Anzahl der Bytes, die zur Darstellung des Mehrbyte-Zeichens mit dem Code *wchar* benötigt werden; wenn *s* kein Nullzeiger ist, wird die Mehrbyte-Zeichendarstellung im Feld *s* abgelegt. Maximal `MB_CUR_MAX`-Zeichen werden gespeichert.

Wenn *s* ein Nullzeiger ist, liefert `wctomb` 0 zurück. Wenn *s* kein Nullzeiger ist, liefert `wctomb` -1, wenn der Wert *wchar* nicht einem gültigen Mehrbyte-Zeichen entspricht; ansonsten wird die Anzahl der Bytes zurückgegeben, welche das Mehrbyte-Zeichen mit dem Code *wchar* belegt.

SIEHE AUCH

`mbstring`(3C), `setlocale`(3C), `environ`(5).
`chrtbl`(1M) in "Referenzhandbuch für Systemverwalter".

mbstring - Funktionen für Mehrbyte-Zeichen

```
#include <stdlib.h>
size_t mbstowcs (wchar_t *pwcs, const char *s, size_t n);
size_t wcstombs (char *s, const wchar_t *pwcs, size_t n);
```

`mbstowcs` konvertiert eine Folge von Mehrbyte-Zeichen aus dem Feld, auf das *s* zeigt, in eine Folge von Langzeichen und speichert diese im Feld, auf das *pwcs* zeigt; nachdem *n*-Zeichen gespeichert wurden oder der Wert Null (ein konvertiertes Nullzeichen) auftritt, stoppt die Ausführung. Wenn ein ungültiges Mehrbyte-Zeichen auftritt, liefert `mbstowcs` (`size_t`)-1 zurück. Ansonsten gibt `mbstowcs` die Anzahl der Feldelemente zurück, die modifiziert worden sind, ohne das abschließende Nullzeichen (falls vorhanden).

`wcstombs` konvertiert eine Folge von Langzeichen aus dem Feld *pwcs* in eine Folge von Mehrbyte-Zeichen und speichert diese Mehrbyte-Zeichen in dem Feld, auf das *s* zeigt; nachdem *n* Mehrbyte-Zeichen verarbeitet wurden oder ein Nullzeichen auftrat, wird die Ausführung beendet. Wenn ein Langzeichen auftritt, welches keinem gültigen Mehrbyte-Zeichen entspricht, liefert `wcstombs` (`size_t`)-1 zurück. Ansonsten liefert `wcstombs` die Anzahl der modifizierten Bytes zurück, ausschließlich des abschließenden Nullzeichens (falls vorhanden).

SIEHE AUCH

`mbchar`(3C), `setlocale`(3C), `environ`(5).
`chrtbl`(1M) in "Referenzhandbuch für Systemverwalter".

memory - Speicherfunktionen

```
#include <string.h>
void *memccpy (void *s1, const void *s2, int c, size_t n);
void *memchr (const void *s, int c, size_t n);
int memcmp (const void *s1, const void *s2, size_t n);
void *memcpy (void *s1, const void *s2, size_t n);
void *memmove (void *s1, const void *s2, size_t n);
void *memset (void *s, int c, size_t n);
```

Diese Funktionen ermöglichen effiziente Bearbeitung von Speicherbereichen. Speicherbereiche sind Byte-Felder, die durch eine Anzahl begrenzt sind und nicht durch das Null-Byte. Die angegebenen Speicherbereiche werden nicht auf Überlauf geprüft.

`memccpy` kopiert Bytes aus dem Speicherbereich *s2* nach *s1* und hält an, nachdem entweder *c* zum ersten Mal kopiert wurde (wobei es in ein `unsigned char` konvertiert wird), oder nachdem *n* Bytes kopiert wurden. `memccpy` gibt einen Zeiger auf das Byte nach der Kopie von *c* in *s1* zurück oder einen Nullzeiger, wenn *c* nicht in den ersten *n* Zeichen von *s2* gefunden wurde.

`memchr` gibt einen Zeiger auf das erste *c* in den ersten *n* Bytes (jedes als `unsigned char` interpretiert) des Speicherbereichs *s* zurück, oder einen Nullzeiger, wenn *c* nicht auftritt.

`memcmp` vergleicht die Argumente *s1* und *s2*, wobei es nur die ersten *n* Bytes (jedes als `unsigned char` interpretiert) vergleicht und gibt eine ganze Zahl zurück, die kleiner, gleich oder größer als 0 ist, je nachdem, ob *s1* lexikografisch kleiner, gleich oder größer als *s2* ist.

`memcpy` kopiert *n* Bytes aus dem Speicherbereich *s2* nach *s1*. Es gibt *s1* zurück.

`memmove` kopiert *n* Bytes aus den Speicherbereichen *s2* bis *s1*. Kopieren zwischen überlappenden Objekten wird ordungsgemäß ausgeführt. *s1* wird zurückgegeben.

`memset` setzt die ersten *n* Bytes im Speicherbereich *s* auf den Wert des Zeichens *c* (in einen `unsigned char` umgewandelt). Es gibt *s* zurück.

SIEHE AUCH

`string`(3C).

mkfifo - neue FIFO-Gerätedatei erzeugen

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

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

Die Routine `mkfifo` erzeugt eine neue FIFO-Gerätedatei unter dem Pfadnamen *path*. Der Modus der neuen FIFO-Gerätedatei wird aus *mode* übernommen. Die Zugriffsrechte aus dem Argument *mode* werden von der Dateierzeugungsmaske des Prozesses modifiziert (siehe `umask`(2)).

Die Eigentümernummer der FIFO-Gerätedatei wird auf die effektive Benutzernummer des Prozesses gesetzt. Die Gruppennummer der FIFO-Gerätedatei wird auf die effektive Gruppennummer des Prozesses gesetzt; ist das `S_ISGID`-Bit im übergeordneten Verzeichnis gesetzt, dann wird die Gruppennummer des FIFOs vom übergeordneten Verzeichnis übernommen.

`mkfifo` ruft den Systemaufruf `mknod` auf, um die Datei zu erzeugen.

SIEHE AUCH

`chmod`(2), `exec`(2), `mknod`(2), `umask`(2), `fs`(4), `stat`(5).
`mkdir`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Nach erfolgreicher Ausführung wird der Wert 0 zurückgeliefert. Ansonsten wird -1 zurückgeliefert und `errno` gesetzt, um den Fehler anzuzeigen.

HINWEIS

Enthält *mode* andere gesetzte Bits, welche nicht die Dateizugriffsrechte regeln, so werden diese ignoriert.

mktemp - eindeutigen Dateinamen erstellen

```
#include <stdlib.h>
char *mktemp(char *template);
```

`mktemp` ersetzt den Inhalt der Zeichenkette, auf die *template* zeigt, durch einen eindeutigen Namen, und gibt den verarbeiteten *template* zurück. Die Zeichenkette in *template* sollte wie ein Dateiname mit sechs nachgestellten `X` aussehen. `mktemp` ersetzt die sechs `X` durch eine Zeichenkette, die einen eindeutigen Dateinamen darstellt.

SIEHE AUCH

`tmpfile`(3S), `tmpnam`(3S).

ERGEBNIS

`mktemp` weist *template* eine leere Zeichenkette zu, wenn es keinen eindeutigen Namen erstellen kann.

HINWEIS

`mktemp` kann nur 26 eindeutige Dateinamen pro Prozeß für jedes eindeutige *template* erzeugen.

mktime - tm-Struktur in Kalenderzeit umwandeln

```
#include <time.h>
time_t mktime (struct tm *timeptr);
```

`mktime` wandelt die Zeit, die durch die `tm`-Struktur dargestellt wird und auf die *timeptr* zeigt, in die Kalenderzeit um (die Anzahl der Sekunden seit 00:00:00 UTC, 1. Januar, 1970).

Die `tm`-Struktur hat das folgende Format:

```
struct        tm {
        int     tm_sec;         /* Sekunden [0, 61]  */
        int     tm_min;         /* Minuten [0, 59] */
        int     tm_hour;        /* Stunde [0, 23] */
        int     tm_mday;        /* Monatstag [1, 31] */
        int     tm_mon;         /* Monat [0, 11] */
        int     tm_year;        /* Jahre seit 1900 */
        int     tm_wday;        /* Tage seit Sonntag [0, 6] */
        int     tm_yday;        /* Tage seit 1. Januar 1 [0, 365] */
        int     tm_isdst;       /* Schalter für Sommerzeit */
};
```

Neben der Berechnung der Kalenderzeit normalisiert `mktime` die übergebene `tm`-Struktur. Die Originalwerte der Komponenten `tm_wday` und `tm_yday` werden ignoriert; die Originalwerte der anderen Komponenten der Struktur sind nicht auf die oben angegebenen Grenzen beschränkt. Bei erfolgreicher Ausführung werden die Komponenten `tm_wday` und `tm_yday` entsprechend gesetzt; die anderen Komponenten werden so eingestellt, daß sie die angegebene Kalenderzeit darstellen, wobei die entsprechenden Wertebereiche eingehalten werden. Der endgültige Wert von `tm_mday` wird nicht gesetzt, bis `tm_mon` und `tm_year` bestimmt sind.

Die ursprünglichen Werte der Komponenten können größer oder kleiner als die angegebenen Bereiche sein. Beispielsweise zeigt der Wert -1 für `tm_hour` eine Stunde vor Mitternacht an; enthält `tm_mday` den Wert 0, so wird der Tag vor dem aktuellen Monat bezeichnet; steht `tm_mon` auf -2, so bedeutet dies zwei Monate vor Januar des Jahres `tm_year`.

Ist `tm_isdst` positiv, wird angenommen, daß sich die ursprünglichen Werte in der alternativen Zeitzone befinden. Stellt sich heraus, daß die alternative Zeitzone für die berechnete Kalenderzeit ungültig ist, werden die Komponenten an die primäre Zeitzone angepaßt. Wenn `tm_isdst` Null ist, wird angenommen, daß sich die Originalwerte in der primären Zeitzone befinden; diese Werte werden in die alternative Zeitzone übersetzt, falls die primäre Zeitzone ungültig ist. Wenn `tm_isdst` negativ ist, wird die korrekte Zeitzone bestimmt, und die Komponenten werden nicht angepaßt.

Die lokale Zeitzoneninformation wird so verwendet, als wenn `mktime tzset` aufrufen würde.

`mktime` liefert die angegebene Kalenderzeit. Wenn die Kalenderzeit nicht dargestellt werden kann, wird `time_t-1` zurückgegeben.

BEISPIEL

Was für ein Wochentag ist der 3.Oktober 2001 ?

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

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

SIEHE AUCH

`ctime`(3C), `getenv`(3C), `timezone`(4).

HINWEIS

`tm_year` in der `tm`-Struktur muß mindestens 1970 oder später sein. Kalenderzeiten vor 00:00:00 UTC, 1. Januar 1970 oder nach 03:14:07 UTC, 19. Januar 2038, können nicht dargestellt werden.

mlock, munlock - Speicherseiten sperren oder freigeben

```
#include <sys/types.h>
int mlock(caddr_t addr, size_t len);
int munlock(caddr_t addr, size_t len);
```

Die Funktion `mlock` verwendet die eingestellten Verweise für den Adreßbereich [*addr, addr + len*), um Seiten zu sperren. Der Wirkung von `mlock(`*addr*`, `*len*`)` entspricht `memcntl(`*addr*`, `*len*`, MC_LOCK, 0, 0, 0)`.

`munlock` entfernt die mit `mlock` gesperrten Seiten. Die Wirkung von `munlock(`*addr*`, `*len*`)` entspricht `memcntl(`*addr*`, `*len*`, MC_UNLOCK, 0, 0, 0)`.

Mit `mlock` eingestellte Sperren werden vom Sohnprozeß nach einem `fork`-Aufruf nicht geerbt und nicht geschachtelt.

SIEHE AUCH

`fork`(2), `memcntl`(2), `mmap`(2), `mlockall`(3C), `plock`(2), `sysconf`(3C).

ERGEBNIS

Nach erfolgreicher Ausführung liefern die Funktionen `mlock` und `munlock` den Wert 0 zurück; tritt ein Fehler auf, wird -1 zurückgegeben und `errno` gesetzt.

HINWEIS

Um `mlock` und `munlock` verwenden zu können, muß der Benutzer entsprechende Privilegien besitzen.

mlockall, munlockall - Adreßbereich sperren oder freigeben

```
#include <sys/mman.h>
int mlockall(int flags);
int munlockall(void);
```

Die Funktion `mlockall` sperrt alle Seiten, die durch einen Adreßbereich abgebildet werden, im Speicher. Die Wirkung von `mlockall(`*flags*`)` ist äquivalent zu:

```
memcntl(0, 0, MC_LOCKAS, flags, 0, 0)
```

Der Wert von *flags* bestimmt, ob die Seiten gesperrt werden sollen, die momentan durch den Adreßbereich abgebildet werden, oder diejenigen, die zu einem späteren Zeitpunkt abgebildet werden, oder beide:

`MCL_CURRENT`	aktuelle Verweise sperren
`MCL_FUTURE`	zukünftige Verweise sperren

Die Funktion `munlockall` entfernt Sperren aus einem Adreßbereich und Sperren für Verweise aus dem Adreßbereich. Die Wirkung von `munlockall` entspricht:

```
memcntl(0, 0, MC_UNLOCKAS, 0, 0, 0)
```

Sperren, die mit `mlockall` verhängt wurden, werden nicht durch einen Sohnprozeß nach einem `fork` geerbt und nicht geschachtelt.

SIEHE AUCH

`fork`(2), `memcntl`(2), `mlock`(3C), `mmap`(2), `plock`(2), `sysconf`(3C).

ERGEBNIS

Nach erfolgreicher Ausführung liefern die Funktionen `mlockall` und `munlockall` den Wert 0; tritt ein Fehler auf, wird -1 zurückgegeben und `errno` gesetzt.

HINWEIS

Um `mlockall` und `munlockall` verwenden zu können, muß der Benutzer über entsprechende Privilegien verfügen.

monitor - Ausführungsprofil vorbereiten

```
#include <mon.h>
void monitor (int (*lowpc)(), int (*highpc)(), WORD *buffer, size_t bufsize, size_t nfunc);
```

`monitor` ist eine Schnittstelle zu `profil`. `monitor` wird automatisch für jedes Programm, welches durch `cc -p` erstellt wurde, mit Standard-Parametern aufgerufen. `monitor` muß nur explizit aufgerufen werden, wenn zusätzliche Informationen zum Ausführungsprofil gewünscht werden.

`monitor` wird mindestens am Anfang und am Ende eines Programms aufgerufen. Der erste Aufruf von `monitor` startet die Aufzeichnung zweier Arten von Informationen bezüglich des Ausführungsprofils: die Verteilung der Laufzeiten und die Anzahl der Aufrufe von Funktionen. Die Daten zur Verteilung der Laufzeiten werden von `profil` erzeugt, die Zählung der Funktionsaufrufe durch den Programmcode, der durch die Option `cc -p` den Objektdateien zur Verfügung gestellt wurde. Der letzte Aufruf von `monitor` schreibt die gesammelten Daten in die Ausgabedatei `mon.out`.

lowpc und *highpc* sind die Anfangs- und Endadressen des Bereichs, der gemessen werden soll.

buffer wird von `monitor` dazu verwendet, das von `profil` erstellte Histogramm und die Zählung der Funktionsaufrufe zu speichern.

bufsize gibt die Anzahl von Feldelementen in *buffer* an.

nfunc bezeichnet die Anzahl der Feldelemente in *buffer*, die für die Zählung von Funktionsaufrufen reserviert wurden. Weitere Feldelemente werden automatisch zugewiesen, sobald sie benötigt werden.

bufsize sollte nach folgender Formel berechnet sein (C-Notation):

```
size_of_buffer =
    sizeof(struct hdr) +
    nfunc * sizeof(struct cnt) +
    ((highpc-lowpc)/BARSIZE) * sizeof(WORD) +
    sizeof(WORD) - 1 ;

bufsize = (size_of_buffer / sizeof(WORD)) ;
```

Dabei gilt:

- *lowpc*, *highpc*, *nfunc* haben dieselben Werte wie die Argumente für `monitor`;
- *BARSIZE* ist die Anzahl der Programmbytes, die jedem Balken des Histogramms oder jedem Feld des `profil`-Puffers zugeordnet sind.
- Die Stukturen `hdr` und `cnt` und der Typ `WORD` sind in der Include-Datei `mon.h` definiert.

`monitor` wird standardmäßig folgendermaßen aufgerufen (C-Notation):

```
monitor (&eprol, &etext, wbuf, wbufsz, 600);
```

Dabei ist:

- `eprol` der Anfang des Benutzerprogramms, das mit der Option `cc -p` übersetzt wurde (siehe `end`(3C));
- `etext` das Ende des Benutzerprogramms (siehe `end`(3C));
- *wbuf* ein Feld von `WORD` mit *wbufsz* Elementen;
- *wbufsz* wird mit der oben gezeigten Formel für *bufsize* berechnet, wobei für *BARSIZE* der Wert 8 eingesetzt wird;
- `600` ist die Anzahl der Felder zum Zählen der Funktionsaufrufe, die in *buffer* reserviert wurden.

Mit diesen Parametereinstellungen kann das Histogramm zur Laufzeitverteilung unter Verwendung von `profil` für das gesamte Programm berechnet werden. Zu Beginn werden 600 Einträge in *buffer* reserviert, und es werden genügend Histogrammeinträge zur Verfügung gestellt, um signifikante Ergebnisse bezüglich der Laufzeitverteilung zu erzeugen. (Weitere Informationen zu dem Einfluß von *bufsize* auf die Berechnung der Laufzeitverteilung finden Sie unter `profil`(2).)

Sie können die Messung stoppen und die Ergebnisse in eine Datei schreiben, wenn Sie die folgende Programmzeile verwenden:

```
monitor((int (*)())0, (int (*)())0, (WORD *)0, 0, 0);
```

Verwenden Sie `prof` zur Analyse der Ergebnisse.

DATEIEN

`mon.out`

SIEHE AUCH

`cc`(1), `prof`(1), `profil`(2), `end`(3C).

HINWEIS

Zusätzliche Aufrufe von `monitor` nach dem Aufruf von `main` und vor `exit` fügen weitere Felder zum Zählen der Funktionsaufrufe hinzu. Solche Aufrufe setzen jedoch auch die Berechnung des `profil`-Histogramms zurück.

Der Name der von `monitor` erzeugten Datei wird durch die Umgebungsvariable `PROFDIR` gesteuert. Wenn `PROFDIR` nicht existiert, wird die Datei `mon.out` im aktuellen Verzeichnis angelegt. Wenn `PROFDIR` existiert, aber keinen Wert trägt, dann unternimmt `monitor` keine Messungen und erzeugt keine Ergebnisdatei. Wenn `PROFDIR` auf *dirname* gesetzt ist und `monitor` automatisch aufgrund der Übersetzung mit der Option `cc -p` aufgerufen wird, so heißt die erzeugte Datei *dirname/pid.progname*. Dabei ist *progname* der Name des Programms.

msync - Speicher synchronisieren

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

int msync(caddr_t addr, size_t len, int flags);
```

Die Funktion msync schreibt alle veränderten Kopien von Seiten im Bereich [*addr, addr + len*) auf deren Speichermedien zurück. msync kann Kopien ungültig machen, so daß spätere Zugriffe auf die Seiten über das Speichermedium möglich sind. Das Speichermedium für einen veränderten MAP_SHARED-Verweis ist die Datei, auf die die Seite abgebildet wird; das Speichermedium für einen veränderten MAP_PRIVATE-Verweis ist sein Swap-Bereich.

flags ist ein Bitmuster, das sich aus den folgenden Werten zusammensetzt:

MS_ASYNC	asynchrone Schreibzugriffe durchführen
MS_SYNC	synchrone Schreibzugriffe durchführen
MS_INVALIDATE	Verweise ungültig machen

Wenn MS_ASYNC gesetzt ist, kehrt msync zurück, sobald alle Schreiboperationen veranlaßt wurden; wird MS_SYNC gesetzt, kehrt msync erst dann zurück, wenn alle Schreiboperationen durchgeführt wurden.

MS_INVALIDATE macht alle Kopien von Daten, die sich in einem Cache-Speicher befinden, ungültig; spätere Referenzen auf diese Seiten werden vom System über das zugrundeliegende Speichermedium bedient.

Die Wirkung von msync(*addr*, *len*, *flags*) ist äquivalent zu:

memcntl(*addr*, *len*, MC_SYNC, *flags*, 0, 0)

SIEHE AUCH

memcntl(2), mmap(2), sysconf(3C).

ERGEBNIS

Nach erfolgreicher Durchführung liefert die Funktion msync den Wert 0; tritt ein Fehler auf, wird -1 zurückgegeben und errno gesetzt.

HINWEIS

msync sollte von Programmen verwendet werden, welche voraussetzen, daß sich ein Speicherobjekt in einem bekannten Zustand befindet, z.B. bei Transaktionsverarbeitung.

nl_langinfo - Sprachinformationen liefern

```
#include <nl_types.h>
#include <langinfo.h>
char *nl_langinfo (nl_item item);
```

`nl_langinfo` liefert einen Zeiger auf eine nullterminierte Zeichenkette mit Informationen über eine bestimmte Sprache oder einen Kulturkreis, das durch die Umgebung im Programm definiert wird. Die Konstanten und Werte von *item* werden in der Datei `langinfo.h` definiert.

Zum Beispiel:

```
nl_langinfo (ABDAY_1);
```

liefert einen Zeiger auf die Zeichenkette "`Dim`", wenn die erkannte Sprache Französisch ist und eine entsprechende Umgebung korrekt installiert wurde; ist die erkannte Sprache Englisch, so wird ein Zeiger auf "`Sun`" zurückgeliefert.

SIEHE AUCH

`gettxt`(3C), `localeconv`(3C), `setlocale`(3C), `strftime`(3C), `langinfo`(5), `nl_types`(5).

ERGEBNIS

Wenn `setlocale` nicht erfolgreich aufgerufen wurde oder wenn die 'langinfo'-Daten für eine unterstützte Sprache entweder nicht verfügbar sind oder *item* keine definierte Variable ist, dann liefert `nl_langinfo` einen Zeiger auf die entsprechende Zeichenkette in der C-Umgebung. Wenn *item* eine ungültige Zeichenkette enthält, liefert `nl_langinfo` immer einen Zeiger auf eine leere Zeichenkette.

HINWEIS

Das Feld, auf das der Rückgabewert zeigt, sollte nicht vom Programm verändert werden. Spätere Aufrufe von `nl_langinfo` können dieses Feld überschreiben.

offsetof - Offset einer Strukturkomponente

```
#include <stddef.h>
size_t offsetof (type, member-designator);
```

`offsetof` ist ein Makro, das in `stddef.h` definiert wird und einen ganzzahligen Ausdruck darstellt. Der Typ dieses Ausdrucks ist `size_t`; der Wert stellt den Offset der Strukturkomponente (bezeichnet durch *member-designator*) von der Anfangsadresse der Struktur (bezeichnet durch *type*) in Bytes dar.

perror - Systemfehlermeldungen ausgeben

```
#include <stdio.h>
void perror (const char *s);
```

`perror` erzeugt standardmäßig eine Meldung auf der Standard-Fehlerausgabe, die den letzten Fehler beschreibt, der während eines Systemaufrufs oder während des Aufrufs einer Bibliotheksfunktion aufgetreten ist. Zuerst wird die Zeichenkette *s* ausgegeben, dann ein Doppelpunkt und eine Leerstelle und anschließend die Meldung und ein Neue-Zeile-Zeichen. Wenn jedoch *s* der Nullzeiger ist oder auf eine Nullzeichenkette weist, wird der Doppelpunkt nicht ausgegeben. Um die Argumentzeichenkette *s* sinnvoll zu nutzen, sollte sie den Namen des Programms enthalten, in dem der Fehler auftrat. Die Fehlernummer wird der externen Variablen `errno` entnommen, die beim Auftreten eines Fehlers gesetzt, jedoch nicht gelöscht wird, wenn fehlerfreie Aufrufe vorgenommen werden.

SIEHE AUCH

`intro`(2), `fmtmsg`(3C), `strerror`(3C).

popen, pclose - Pipe zu Kommando öffnen/schließen

```
#include <stdio.h>
FILE *popen (const char *Kommando, const char *Typ);
int pclose (FILE *Stream);
```

popen erzeugt eine Pipe zwischen dem aufrufenden Programm und dem auszuführenden Kommando. Die Argumente von popen sind Zeiger auf mit Null-Byte endende Zeichenketten. *Kommando* besteht aus einer Shell-Kommandozeile. *Typ* ist ein E/A-Modus, d.h. entweder r für Lesen oder w für Schreiben. Bei dem zurückgegebenen Wert handelt es sich um einen Dateizeiger, so daß man im E/A-Modus w auf die Standardeingabe des Kommandos schreiben kann, indem man in die Datei *Stream* schreibt (siehe intro(3)); und man kann von der Standardausgabe des Kommandos lesen, wenn der E/A-Modus r ist, indem man von der Datei *Stream* liest.

Eine durch popen geöffnete Pipe sollte mit pclose geschlossen werden, das auf die Beendigung des zugehörigen Prozesses wartet und den Ende-Status des Kommandos zurückgibt.

Da offene Dateien mehrfach benutzt werden, kann ein Kommando des Typs r als Eingabefilter und ein Typ w als Ausgabefilter verwendet werden.

BEISPIEL

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

main()
{
        char *cmd = "/usr/bin/ls *.c";
        char buf[BUFSIZ];
        FILE *ptr;

        if ((ptr = popen(cmd, "r")) != NULL)
                while (fgets(buf, BUFSIZ, ptr) != NULL)
                        (void) printf("%s", buf);
        return 0;
}
```

Dieses Programm schreibt alle Dateinamen im aktuellen Dateiverzeichnis, die auf .c enden, auf die Standardausgabe (siehe stdio(3S)).

SIEHE AUCH

pipe(2), wait(2), fclose(3S), fopen(3S), stdio(3S), system(3S).

ERGEBNIS

`popen` gibt einen Nullzeiger zurück, wenn keine Dateien oder Prozesse erzeugt werden können.

`pclose` gibt -1 zurück, wenn *Stream* nicht mit `popen` geöffnet wurde.

HINWEIS

Wenn der Originalprozeß und der durch `popen` erzeugte Prozeß gleichzeitig eine gemeinsame Datei lesen oder beschreiben, darf keiner der Prozesse gepufferte E/A verwenden. Probleme mit einem Ausgabefilter können durch sorgfältiges Entleeren des Puffers, z.B. mit `fflush` (siehe `fclose`(3S)) vermieden werden.

Durch die Umgebungsvariablen `IFS` und `PATH` können Sicherheitsprobleme entstehen. Verwenden Sie vollständige Pfadnamen oder setzen Sie `PATH` zurück, und setzen Sie `IFS` auf Leerzeichen und Tabulator " `\t`".

printf, fprintf, sprintf - formatierte Ausgabe

```
#include <stdio.h>
int printf(const char *Format, .../* Argumente */);
int fprintf(FILE *strm, const char *Format, .../* Argumente */);
int sprintf(char *s, const char *Format, .../* Argumente */);
```

`printf` schreibt auf die Standardausgabe `stdout`.

`fprintf` schreibt in die angegebene Ausgabedatei *strm*.

`sprintf` schreibt mit abschließendem Null-Byte in die auf den Speicherplatz *s* folgenden Bytes.

Der Benutzer ist dafür verantwortlich, daß ausreichender Speicherplatz zur Verfügung steht. Jede Funktion gibt die Anzahl der übertragenen Zeichen zurück (ausschließlich `\0` im Fall von `sprintf`) oder einen negativen Wert, wenn ein Ausgabefehler bemerkt wurde.

Diese Funktionen konvertieren, formatieren und schreiben ihre *Argumente* in der durch *Format* angegebenen Form. *Format* ist eine Zeichenkette, die drei Objekttypen enthalten kann:

1. normale Zeichen, die einfach in den Ausgangsstrom kopiert werden
2. Escape-Sequenzen, die nichtgrafische Zeichen darstellen
3. Umwandlungsanweisungen

Die folgenden Escape-Sequenzen erzeugen die entsprechende Aktion auf Geräten, die diese Aktion ausführen können:

`\a` Alarm. Ein Ton wird ausgegeben.

`\b` Rückschritt. Die Ausgabeposition wird ein Zeichen vor die aktuelle Position verschoben, außer wenn die aktuelle Position der Zeilenanfang ist.

`\f` Seitenvorschub. Die Ausgabeposition wird an die Startposition der nächsten logischen Seite geschoben.

`\n` Neue Zeile. Die Ausgabeposition wird an den Anfang der nächsten Zeile geschoben.

`\r` Return. Die Ausgabeposition wird an den Anfang der aktuellen Zeile geschoben.

`\t` Horizontaler Tabulator. Die Ausgabeposition wird auf die Position des nächsten implementierungsabhängigen horizontalen Tabulators der aktuellen Zeile verschoben.

\v Vertikaler Tabulator. Die Ausgabeposition wird auf die Position des nächsten implementierungsabhängigen vertikalen Tabulators verschoben.

Alle Formen der printf-Funktionen berücksichtigen die Eingabe eines sprachabhängigen Dezimalpunktes. Der Dezimalpunkt wird durch die Umgebung des Programms definiert (LC_NUMERIC). In der C-Umgebung oder in einer Umgebung, in der der Dezimalpunkt nicht definiert ist, ist es standardmäßig ein Punkt.

Umwandlungsanweisungen

Jede Umwandlungsanweisung wird durch das Zeichen % eingeleitet. Hinter dem Zeichen % erscheinen nacheinander:

- Ein optionales Feld, das aus einer Folge von Dezimalzahlen besteht, denen ein $ folgt. Dieses Feld gibt die nächsten Argumente an, die formatiert werden sollen. Wenn dieses Feld nicht vorhanden ist, werden die Argumente verwendet, die den letzten formatierten Argumenten folgen.
- Eine optionale Folge von Dezimalzahlen zur Angabe einer minimalen Feldgröße. Wenn der umgewandelte Wert weniger Zeichen als die Feldgröße hat, wird er links (oder rechts, wenn die unten beschriebene Option zur Links-Justierung angegeben wurde) bis zur Feldgröße aufgefüllt.
- Eine optionale Genauigkeitsangabe, die die Minimalzahl der Ziffern angibt, die bei den Formatierungen d, i, o, u, x oder X erscheinen sollen (das Feld wird mit führenden Nullen aufgefüllt), die Anzahl von Ziffern, die hinter einem Dezimalpunkt für die Formatierungen e, E und f erscheinen sollen, oder die maximale Anzahl von Zeichen, die von einer Zeichenkette bei der s-Formatierung ausgegeben werden sollen. Die Genauigkeitsangabe ist ein Punkt, gefolgt von einer Folge aus Dezimalzahlen; eine leere Ziffernfolge wird als Null behandelt. Ein durch die Genauigkeitsangabe festgelegtes Auffüllen überschreibt das Auffüllen, das durch die Feldgröße festgelegt wird.
- Ein optionales h gibt an, daß eine folgende Umwandlungsanweisung d, i, o, u, x oder X auf ein Argument short int oder unsigned short int angewendet wird. Das Argument wird entsprechend der C-Regeln zur Wertumwandlung für ganze Zahlen unterstützt, und sein Wert wird vor dem Drucken in ein short int oder unsigned short int umgewandelt. Ein optionales h gibt an, daß eine nachfolgendes Umwandlungsanweisung n ein Zeiger auf ein short int ist. Ein optionales l gibt an, daß eine nachfolgende Umwandlungsanweisung d, i, o, u, x oder X auf ein long int- oder unsigned long int-Argument angewendet wird. Ein optionales l gibt an, daß eine nachfolgende Umwandlungsanweisung n ein Zeiger auf ein long int ist. Ein optionales L gibt an, daß eine nachfolgende Umwandlungsanweisung e, E, f, g oder G auf ein long double-Argument angewendet wird. Wenn vor einer anderen Umwandlungsanweisung ein h, l oder L erscheint, ist das Verhalten undefiniert.
- Ein Zeichen, das den Typ der anzuwendenden Umwandlung angibt.

Eine Feldgröße oder Genauigkeitsangabe kann statt einer Ziffernfolge auch durch einen Stern (*) angezeigt werden. In diesem Fall gibt eine ganzzahliges Argument die Feldgröße oder Genauigkeit an. Das umzuwandelnde Argument wird solange nicht gelesen, bis das entsprechende Umwandlungszeichen gefunden ist. Argumente, die Feldgröße oder Genauigkeit angeben, müssen also vor dem umzuwandelnden Argument stehen. Wenn die Genauigkeit negativ ist, wird es auf Null geändert. Eine negative Feldgröße wird als '-'-Option betrachtet, dem eine positive Feldgröße folgt.

In Formatzeichenketten, die die Form *%Ziffer$* der Umwandlungsanweisung enthalten, kann die Feldgröße oder die Genauigkeit auch durch die Folge **Ziffer$* angegeben werden, wobei *Ziffer* die Position einer Ganzzahl in der Argumentliste angibt.

Wenn numerierte Argumentangaben verwendet werden, verlangt die Angabe des nten Arguments, daß alle vohergehenden Argumente, vom ersten zum (n-1)ten, in der Format-Zeichenkette angegeben werden.

Optionen

- Das Ergebnis der Umwandlung wird im Feld linksbündig ausgegeben. (Es wird rechtsbündig ausgegeben, wenn dieses Option nicht angegeben ist.)

+ Das Ergebnis einer vorzeichenbehafteten Umwandlung beginnt immer mit einem Vorzeichen (+ oder -).

Leerzeichen
Wenn das erste Zeichen einer vorzeichenbehafteten Umwandlung kein Vorzeichen ist, wird vor dem Ergebnis ein Leerzeichen eingefügt. Das bedeutet, daß das Leerzeichen ignoriert wird, wenn es zusammen mit dem Zeichen + angegeben wird.

Der Wert soll in eine andere Form umgewandelt werden. Bei c, d, i, s und u zeigt diese Option keine Wirkung. Bei o erhöht sie die Genauigkeit, um zu erzwingen, daß die erste Ziffer des Ergebnisses eine Null ist. Bei x (oder X) wird einem Ergebnis, das ungleich null ist, ein 0x (oder 0X) vorangestellt. Bei e, E, f, g und G enthält das Ergebnis immer einen Dezimalpunkt, selbst wenn dem Punkt keine Ziffern mehr folgen (normalerweise erscheint bei dem Ergebnis dieser Umwandlungen nur dann ein Dezimalpunkt, wenn eine Ziffer folgt). Bei g und G werden abschließende Nullen nicht vom Ergebnis entfernt, wie sonst üblich.

0 Bei d, i, o, u, x, X, e, E, f, g und G werden führende Nullen (die irgendeiner Vorzeichen- oder Basisangabe folgen) verwendet, um auf die Feldgröße aufzufüllen; Leerzeichen werden nicht verwendet. Wenn die Optionen 0 und - zusammen angegeben werden, wird die Option 0 ignoriert. Bei d, i, o, u, x und X wird diese Option ignoriert, wenn eine Genauigkeitsangabe vorhanden ist. Bei anderen Umwandlungen ist das Verhalten nicht definiert.

Jedes Umwandlungszeichen holt null oder mehr Argumente. Die Ergebnisse sind nicht definiert, wenn zu wenig Argumente vorhanden sind. Wenn das Format verbraucht ist, während noch Argumente verbleiben, werden die überschüssigen Argumente ignoriert.

Umwandlungszeichen

`d, i, o, u, x, X`

Das Argument vom Typ `int` wird in eine vorzeichenbehaftete dezimale (`d` oder `i`), vorzeichenlose oktale (`o`), vorzeichenlose dezimale (`u`) oder vorzeichenlose hexadezimale (`x` und `X`) Zahl umgewandelt. Die Umwandlung `x` verwendet die Buchstaben `abcdef`, und die Umwandlung `X` verwendet die Buchstaben `ABCDEF`. Die Genauigkeitsangabe gibt die minimale Anzahl von Ziffern an, die ausgegeben werden sollen. Wenn der umzuwandelnde Wert durch weniger Ziffern angegeben werden kann, wird er mit führenden Nullen erweitert. Die Voreinstellung für die Genauigkeit ist 1. Das Ergebnis einer Umwandlung des Wertes 0 mit der Genauigkeit Null gibt kein Zeichen.

`f`

Ein Argument vom Typ `double` wird in eine dezimale Darstellung der Form [-]*ddd.ddd* umgewandelt, wobei die Anzahl der Dezimalziffern hinter dem Dezimalpunkt (siehe `setlocale`(3C)) gleich der Genauigkeitsangabe ist. Ist die Genauigkeit nicht angegeben, wird diese mit dem Wert 6 angenommen. Ist die Genauigkeit Null und die Option # ist nicht angegeben, erscheint kein Dezimalpunkt. Wenn ein Dezimalpunkt ausgegeben wird, dann wird davor mindestens eine Ziffer ausgegeben. Der Wert wird auf die entsprechende Ziffernanzahl gerundet.

`e, E`

Ein Argument vom Typ `double` wird in eine dezimale Darstellung der Form [-]*d.ddd*e±*dd* umgewandelt, wobei eine Ziffer vor dem Dezimalpunkt steht (die ungleich null ist, falls das Argument ungleich null ist) und die Anzahl der Ziffern danach die Genauigkeit angibt. Wenn die Genauigkeitsangabe fehlt, wird der Wert 6 angenommen; wenn die Genauigkeitsangabe gleich null ist und die Option # nicht angegeben ist, wird kein Dezimalpunkt ausgegeben. Das Umwandlungszeichen `E` erzeugt eine Zahl, die `E` statt `e` am Anfang des Exponenten stehen hat. Der Wert wird auf die entsprechende Ziffernanzahl gerundet.

`g, G` Ein Argument vom Typ `double` wird im Format `f` oder `e` (oder im Format `E` im Falle der `G`-Umwandlungsanweisung) ausgegeben. Die Genauigkeit gibt die Anzahl der signifikanten Stellen an. Wenn die Genauigkeitsangabe null ist, wird der Wert 1 angenommen. Das verwendete Format hängt von dem umgewandelten Wert ab: Das Format `e` (oder `E`) wird nur dann verwendet, wenn der bei der Umwandlung entstehende Exponent kleiner als -4 oder größer oder gleich der Genauigkeitsangabe ist. Abschließende Nullen werden von dem gebrochenen Teil des Ergebnisses entfernt. Ein Dezimalpunkt erscheint nur dann, wenn ihm eine Ziffer folgt.

`c` Das Argument vom Typ `int` wird in einen `unsigned char` umgewandelt, und das entstehende Zeichen wird ausgegeben.

`s` Das Argument ist vom Typ Zeiger auf `char` und die Zeichen der Zeichenkette werden bis ausschließlich des abschließenden Nullzeichens geschrieben; wenn die Genauigkeit angegeben ist, werden nicht mehr Zeichen als angegeben geschrieben. Wenn die Genauigkeit nicht angegeben ist, wird sie als unendlich betrachtet, so werden alle Zeichen bis zum ersten Nullzeichen ausgegeben. Ein `NULL`-Wert für ein Argument führt zu nicht definierten Ergebnissen.

`p` Das Argument ist ein Zeiger auf `void`. Der Wert des Zeigers wird in eine implementierungsabhängige Menge von Folgen druckbarer Zeichen umgewandelt, die der Menge von Zeichenketten entsprechen sollte, die durch die `%p`-Formatangabe der Funktion `scanf` erkannt werden.

`n` Das Argument ist ein Zeiger auf ein `int`, in welches die Anzahl der Zeichen geschrieben wurde, die bis zu diesem Zeitpunkt von den Aufrufen `printf`, `fprintf` oder `sprintf` auf die Standardausgabe erzeugt wurden. Kein Argument wird umgewandelt.

`%` Ausgabe eines `%`; kein Argument wird umgewandelt.

Wenn das Zeichen nach `%` oder nach `%`*Ziffer*`$` kein gültiges Umwandlungszeichen ist, ist das Ergebnis der Umwandlung nicht definiert.

Wenn ein Gleitkomma-Wert der internen Darstellung für unendlich entspricht, wird [±]*inf* ausgegeben, wobei *inf* entweder `inf` oder `INF` ist, abhängig vom Umwandlungszeichen. Die Ausgabe des Vorzeichens folgt den oben beschriebenen Regeln.

Wenn ein Gleitkomma-Wert der internen Darstellung für `NaN` entspricht, wird [±]*nan*`0x`*m* ausgegeben. Abhängig vom Umwandlungszeichen ist *nan* entweder `nan` oder `NAN`. Zusätzlich stellt `0x`*m* den wichtigsten Teil der Mantisse dar. Abhängig vom Umwandlungszeichen ist *x* entweder `x` oder `X`, und *m* verwendet die Buchstaben `abcdef` oder `ABCDEF`. Bei der Ausgabe des Vorzeichens werden die oben beschriebenen Regeln befolgt.

In keinem Fall bewirkt eine nicht vorhandene oder kleine Feldgröße, daß ein Feld abgeschnitten wird; wenn das Ergebnis einer Umwandlung größer als die Feldgröße ist, wird das Feld erweitert, um das Ergebnis aufnehmen zu können. Zeichen, die durch `printf` und `fprintf` erzeugt wurden, werden so ausgegeben, als ob die Routine `putc` aufgerufen worden wäre.

BEISPIEL

Das folgende Beispiel dient dazu, das Datum und die Zeit in der Form `Sunday, July 3, 10:02` auszugeben, wobei `Wochentag` und `Monat` Zeiger auf mit Nullzeichen abgeschlossene Zeichenketten sind:

```
printf("%s, %s %i, %d:%.2d",
        Wochentag, Monat, Tag, Stunde, Minute);
```

Um π auf 5 Stellen genau auszugeben:

```
printf("pi = %.5f", 4 * atan(1.0));
```

SIEHE AUCH

`exit`(2), `lseek`(2), `write`(2), `abort`(3C), `ecvt`(3C), `putc`(3S), `scanf`(3S), `setlocale`(3C), `stdio`(3S).

ERGEBNIS

`printf`, `fprintf` und `sprintf` geben die Anzahl der übertragenen Zeichen zurück oder einen negativen Wert, wenn ein Fehler aufgetreten ist.

psignal, psiginfo - Signalmeldung des Systems ausgeben

```
#include <siginfo.h>
void psignal (int sig, const char *s);
void psiginfo (siginfo_t *pinfo, char *s);
```

`psignal` und `psiginfo` liefern über die Standard-Fehlerausgabe (stderr) eine Meldung, die das angegebene Signal beschreibt. *sig* ist ein Signal, das als erstes Argument an eine Signalbehandlungsroutine übergeben wurde. *pinfo* ist ein Zeiger auf eine `siginfo`-Struktur, die als zweites Argument an eine erweiterte Signalbehandlungsroutine übergeben wird (siehe `sigaction`(2)). Zuerst wird die Zeichenkette *s* ausgegeben, dann ein Doppelpunkt und ein Leerzeichen, danach eine Systemmeldung, die das Signal beschreibt, und ein Zeilenvorschub.

SIEHE AUCH

`sigaction`(2), `perror`(3), `siginfo`(5), `signal`(5).

putc - Zeichen/Wort auf einen Stream ausgeben

```
#include <stdio.h>
int putc (int c, FILE *stream);
int putchar (int c);
int fputc (int c, FILE *stream);
int putw (int w, FILE *stream);
```

`putc` schreibt *c* (umgeformt zu einem `unsigned char`) in die Ausgabedatei *stream* (siehe `intro`(3)) an die Stelle, auf die der Schreib-/Lesezeiger, sofern er definiert ist, zeigt und transportiert den Schreib-/Lesezeiger an die entsprechende Stelle. Wenn die Datei Positionierungsanforderungen nicht unterstützt, oder wenn *stream* mit einem Append-Modus eröffnet wurde, wird das Zeichen an das Ende der Ausgabe angehängt. `putchar(c)` ist als `putc(c, stdout)` definiert. `putc` und `putchar` sind Makros.

`fputc` verhält sich wie `putc`, ist jedoch eine Funktion und kein Makro. `fputc` läuft langsamer als `putc`, beansprucht jedoch weniger Speicherplatz pro Aufruf, und der Name kann als Argument an eine Funktion übergeben werden.

`putw` schreibt das Wort (`int`) *w* in die Ausgabedatei *stream*, auf die der Schreib-/Lesezeiger, sofern definiert, zeigt. Die Größe eines Worts ist die maschinenspezifische Größe von `int`. `putw` setzt keine besondere Ausrichtung in der Datei voraus und bewirkt diese auch nicht.

SIEHE AUCH

`exit`(2), `lseek`(2), `write`(2), `abort`(3C), `fclose`(3S), `ferror`(3S), `fopen`(3S), `fread`(3S), `printf`(3S), `puts`(3S), `setbuf`(3S), `stdio`(3S).

ERGEBNIS

Wenn diese Funktionen erfolgreich sind, gibt jede den von ihr geschriebenen Wert zurück (mit Ausnahme von `putw`). `putw` gibt `ferror (`*stream*`)` zurück. Bei Nichterfolg geben sie die Konstante `EOF` zurück. Dieses Verhalten tritt zum Beispiel auf, wenn die Datei *stream* nicht zum Schreiben geöffnet ist, oder wenn die Ausgabedatei nicht mehr vergrößert werden kann.

HINWEIS

Da `putc` als Makro implementiert ist, wird *stream* mehr als einmal ausgewertet. So verhält sich insbesondere `putc(c, *f++);` nicht wie erwartet. Statt dessen sollte man `fputc` verwenden.

Wegen der möglichen Unterschiede in Wortlänge und Byte-Ordnung sind die unter Verwendung von `putw` geschriebenen Dateien maschinenabhängig und können möglicherweise nicht mit `getw` auf einem anderen Prozessor gelesen werden.

Funktionen existieren für alle oben definierten Makros. Um die entsprechende Funktion aufzurufen, muß der Makroname undefiniert sein (z.B. durch `#undef putc`).

putenv - Umgebungsvariablen ändern/hinzufügen

```
#include <stdlib.h>
int putenv (char *string);
```

string weist auf eine Zeichenkette der Form *Name=Wert*. putenv setzt durch Ändern einer vorhandenen Variablen oder Erstellung einer neuen den Wert der Umgebungsvariablen *Name* gleich dem Wert *Wert*. In beiden Fällen wird die Zeichenkette, auf die *string* zeigt, Teil der Umgebung, so daß das Ändern der Zeichenkette zu einer Änderung der Umgebung führt. putenv legt keine Kopie von *string* an. Die Zeichenkette *string* sollte als static deklariert werden, wenn sie innerhalb einer Funktion deklariert wird. Der von *string* belegte Speicherplatz wird nicht mehr benötigt, nachdem eine neue Zeichenkette, die *Name* definiert, an putenv übergeben wurde.

SIEHE AUCH

exec(2), getenv(3C), malloc(3C), environ(5).

ERGEBNIS

putenv gibt einen Wert ungleich Null zurück, wenn es über malloc nicht ausreichend Speicherplatz für eine erweiterte Umgebung bereitstellen konnte. Andernfalls wird Null zurückgegeben.

HINWEIS

putenv bearbeitet die Umgebung, auf die *environ* zeigt, und kann in Verbindung mit getenv verwendet werden. Jedoch wird *envp* (das dritte Argument für *main*) nicht geändert.

Diese Funktion verwendet malloc(3C) zur Erweiterung der Umgebung.

Nach Aufruf von putenv sind die Umgebungsvariablen nicht in alphabetischer Reihenfolge. Eine mögliche Fehlerquelle ist der Aufruf von putenv mit einem Zeiger auf eine auto-Variable als Argument und eine anschließende Rückkehr von der aufrufenden Funktion, während *string* noch Teil der Umgebung ist.

putpwent - Eintrag in die Paßwortdatei schreiben

```
#include <pwd.h>
int putpwent (const struct passwd *p, FILE *f);
```

`putpwent` ist die Umkehrung von `getpwent`(3C). Mit einer von `getpwent` (oder `getpwuid` oder `getpwnam`) erstellten `passwd`-Struktur als Argument schreibt `putpwent` eine Zeile in die Datei *f*, die das Format von `/etc/passwd` haben muß.

SIEHE AUCH

`getpwent`(3C).

ERGEBNIS

`putpwent` gibt ungleich Null zurück, wenn ein Fehler während der Ausführung festgestellt wurde. Andernfalls wird Null zurückgegeben.

puts, fputs - Zeichenkette auf einen Stream schreiben

```
#include <stdio.h>
int puts (const char *s);
int fputs (const char *s. FILE *stream);
```

`puts` schreibt die Zeichenkette, auf die *s* zeigt, zusammen mit einem abschließenden Neue-Zeile-Zeichen auf die Standardausgabe `stdout` (siehe `intro`(3)).

`fputs` schreibt die durch ein Null-Byte abgeschlossene Zeichenkette, auf die *s* zeigt, in die angegebene Ausgabedatei *stream*.

Keine der Funktionen schreibt das abschließende Null-Byte.

SIEHE AUCH

`exit`(2), `lseek`(2), `write`(2), `abort`(3C), `fclose` (3s), `ferror`(3S), `fopen`(3S), `fread`(3S), `printf`(3S), `putc`(3S), `stdio`(3S).

ERGEBNIS

Bei erfolgreicher Ausführung geben beide Funktionen die Anzahl der geschriebenen Zeichen zurück, anderenfalls `EOF`.

HINWEIS

`puts` hängt ein Neue-Zeile-Zeichen an, `fputs` nicht.

putspent - Eintrag in Shadow-Paßwortdatei schreiben

```
#include <shadow.h>
int putspent (const struct spwd *p, FILE *fp);
```

Die Routine putspent ist die Umkehrfunktion zu getspent. *p* ist ein Zeiger auf eine *spwd*-Struktur, die von getspent oder von getspnam erzeugt wurde. putspent schreibt die Informationen dieser *spwd*-Struktur als eine Zeile in den Stream *fp*. Die Zeile entspricht dabei dem Format der Shadow-Kennwortdatei /etc/shadow.

Wenn das Feld sp_min, sp_max, sp_lstchg, sp_warn, sp_inact, oder sp_expire der spwd-Struktur -1 enthält, oder wenn sp_flag gleich 0 ist, dann wird das entsprechende Feld des in die Datei *fp* geschriebenen Eintrags gelöscht.

SIEHE AUCH

getspent(3C), getpwent(3C), putpwent(3C).

ERGEBNIS

putspent liefert einen Wert ungleich Null, wenn während der Operation ein Fehler auftrat; ansonsten wird Null zurückgegeben.

HINWEIS

putspent ist nur für den internen Gebrauch gedacht; Kompatibilität wird nicht garantiert.

qsort - mit Quicksort sortieren

```
#include <stdlib.h>

void qsort (void* base, size_t nel, size_t width, int (*compar)
        (const void *, const void *));
```

`qsort` ist eine Realisierung des Quicksort-Algorithmus. Es sortiert eine Tabelle von Daten. Die Daten der Tabelle werden in aufsteigender Reihenfolge sortiert, entsprechend der Vergleichsfunktion. *base* zeigt auf das Element am Anfang der Tabelle. *nel* ist die Anzahl der Elemente in der Tabelle. *width* spezifiziert die Größe eines jeden Elements in Bytes. *compar* ist der Name der vom Benutzer definierten Vergleichsfunktion, die von `qsort` mit zwei Argumenten aufgerufen wird, die auf die zu vergleichenden Elemente zeigen. Diese Funktion muß eine ganze Zahl kleiner, gleich oder größer als Null zurückgeben, um anzuzeigen, ob das erste Argument kleiner, gleich oder größer als das zweite ist.

SIEHE AUCH

`bsearch`(3C), `lsearch`(3C), `string`(3C).
`sort`(1) in "SINIX V5.41 Kommandos".

HINWEIS

Wenn zwei Elemente nach dieser Funktion gleich sind, dann ist deren Reihenfolge in der sortierten Tabelle unbestimmt.

raise - Signal an Programm senden

```
#include <signal.h>
int raise (int sig);
```

`raise` sendet das Signal *sig* an das ausführende Programm.

`raise` liefert Null zurück, wenn die Operation erfolgreich durchgeführt wird; ansonsten gibt `raise` -1 zurück und `errno` wird gesetzt, um den Fehler anzuzeigen. `raise` verwendet `kill`, um das Signal an das ausführende Programm zu senden:

```
kill(getpid(), sig);
```

Unter `kill`(2) finden Sie eine detaillierte Liste der Fehlerbedingungen. Unter `signal`(2) ist eine Liste der Signale aufgeführt.

SIEHE AUCH

`getpid`(2), `kill`(2), `signal`(2).

rand, srand - einfacher Zufallszahlengenerator

```
#include <stdlib.h>
int rand (void);
void srand (unsigned int seed);
```

`rand` verwendet einen multiplikativen kongruenten Zufallszahlengenerator der Periode 2^{32}, der aufeinanderfolgende Pseudo-Zufallszahlen im Bereich von 0 bis `RAND_MAX` (in `stdlib.h` definiert) zurückgibt.

Die Funktion `srand` verwendet das Argument *seed* als Startwert für eine neue Folge von Pseudo-Zufallszahlen, die von aufeinanderfolgenden Aufrufen von `rand` zurückgegeben werden. Wenn die Funktion `srand` dann mit demselben `seed`-Wert aufgerufen wird, wird die Folge von Pseudo-Zufallszahlen wiederholt. Wenn die Funktion `rand` aufgerufen wird, bevor irgendein Aufruf von `srand` erfolgte, wird dieselbe Folge erzeugt, als wenn `srand` zuerst mit einem *seed*-Wert von 1 aufgerufen worden wäre.

HINWEIS

Die Spektraleigenschaften von `rand` sind begrenzt. `drand48`(3C) stellt einen erheblich besseren, jedoch komplizierteren Zufallszahlengenerator zur Verfügung.

SIEHE AUCH

`drand48`(3C).

realpath - echten Dateinamen ausgeben

```
#include <stdlib.h>
#include <sys/param.h>

char *realpath (char * file_name, char * resolved_name);
```

`realpath` löst alle Verweise und Referenzen für '.' und '..' in *file_name* auf und speichert sie in *resolved_name*.

Es können sowohl relative als auch absolute Pfadnamen verarbeitet werden. Bei absoluten Pfadnamen und relativen Pfadnamen, deren aufgelöster Name nicht relativ ausgedrückt werden kann (z.B. `../../reldir`), wird der aufgelöste absolute Name zurückgegeben. Für die anderen relativen Pfadnamen wird der aufgelöste relative Name zurückgegeben.

resolved_name muß groß genug sein (`MAXPATHLEN`), um den aufgelösten Pfadnamen aufzunehmen.

SIEHE AUCH

`getcwd`(3C).

ERGEBNIS

Bei erfolgreicher Ausführung liefert `realpath` einen Zeiger auf *resolved_name* zurück. Ansonsten wird ein Nullzeiger zurückgegeben und der Name der Datei, die den Fehler verursacht hat, nach *resolved_name* geschrieben. Die globale Variable `errno` wird gesetzt.

HINWEIS

`realpath` behandelt nullterminierte Zeichenketten.

Sie sollten Ausführungsrechte für alle Verzeichnisse besitzen, die sich im gegebenen und aufgelösten Pfad befinden.

`realpath` kann unter Umständen nicht zum aktuellen Verzeichnis zurückkehren, falls ein Fehler auftritt.

remove - Datei löschen

```
#include <stdio.h>
int remove(const char *path);
```

`remove` hat zur Folge, daß die durch *path* angegebene Datei oder das leere Verzeichnis nicht länger unter dem Namen verfügbar ist. Ein weiterer Versuch, die Datei unter dem Namen zu öffnen, wird fehlschlagen, es sei denn, die Datei wird neu angelegt.

Für Dateien ist `remove` identisch mit `unlink`. Für Verzeichnisse ist `remove` identisch mit `rmdir`.

Siehe `rmdir`(2) und `unlink`(2); dort finden Sie eine detaillierte Liste der Fehlerbedingungen.

SIEHE AUCH

`rmdir`(2), `unlink`(2).

ERGEBNIS

Nach erfolgreicher Ausführung liefert `remove` den Wert 0; andernfalls wird -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

scanf, fscanf, sscanf - formatierte Eingabe

```
#include <stdio.h>
int scanf(const char *Format, ...);
int fscanf(FILE *strm, const char *Format, ...);
int sscanf(const char *s, const char *Format, ...);
```

`scanf` liest von der Standardeingabe `stdin`.

`fscanf` liest von der angegebenen Eingabedatei *strm*.

`sscanf` liest aus der Zeichenkette *s*.

Jede dieser Funktionen liest Zeichen, interpretiert sie nach dem angegebenen Format und speichert die Ergebnisse in den Argumenten. Jede Funktion erwartet eine Steuerzeichenkette *Format*, die unten beschrieben ist, sowie einen Satz von Zeigerargumenten, die angeben, wo die umgewandelte Eingabe gespeichert werden soll. Wenn nicht genug Argumente für das Format vorhanden sind, ist das Verhalten nicht definiert. Wenn das Format ausgeschöpft ist und noch Argumente übrig sind, werden die überzähligen Argumente ignoriert.

Umwandlungsanweisungen

Die Steuerzeichenkette enthält im allgemeinen Umwandlungsanweisungen, die zur Steuerung der Interpretation von Eingabefolgen verwendet werden. Die Steuerzeichenkette kann folgendes enthalten:

1. Zwischenraum (Leerzeichen, Tabulatoren, Neue-Zeile-Zeichen oder Seitenvorschub-Zeichen), der mit Ausnahme der zwei nachstehend beschriebenen Fälle das Lesen der Eingabe bis zum nächsten Zeichen bewirkt.
2. Ein normales Zeichen (nicht `%`), das zum nächsten Zeichen des Eingabestroms passen muß.
3. Umwandlungsanweisungen, die aus dem Zeichen `%` oder der Zeichenkette *%Ziffer*`$`, einem optionalen Zeichen zur Zuweisungsunterdrückung `*`, einer Folge aus Dezimalziffern, die eine optionale, numerische Maximalfeldgröße angibt, einem optionalen Buchstaben `l`, `L` oder `h`, der die Größe des empfangenden Objekts angibt, und einem Umwandlungszeichen bestehen. Den Umwandlungszeichen `d`, `i` und `n` sollte ein `h` vorangehen, wenn das entsprechende Argument ein Zeiger auf `short int` statt auf `int` ist, oder ein `l`, wenn es ein Zeiger auf `long int` ist. Ebenso sollte den Umwandlungszeichen `o`, `u` und `x` ein `h` vorangehen, wenn das entsprechende Argument ein Zeiger auf `unsigned short int` statt eines Zeigers auf `unsigned int` ist, oder ein `l`, wenn es ein Zeiger auf `unsigned long int` ist. Schließlich sollte den

Umwandlungszeichen `e`, `f` und `g` ein `l` vorangehen, wenn das entsprechende Argument ein Zeiger auf `double` statt auf `float` ist, oder ein `L`, wenn es ein Zeiger auf `long double` ist. Bei jedem anderen Umwandlungszeichen werden `h`, `l` oder `L` ignoriert.

Eine Umwandlungsanweisung steuert die Umwandlung des nächsten Eingabefelds; das Ergebnis wird in die Variable geschrieben, auf die das entsprechende Argument zeigt, sofern keine Zuweisungsunterdrückung durch das Zeichen * ausgegeben war. Mit der Unterdrückung der Zuweisung kann ein Eingabefeld beschrieben werden, das übersprungen werden soll. Ein Eingabefeld wird als Zeichenkette ohne Zwischenraumzeichen definiert; es erstreckt sich bis zum nächsten nicht passenden Zeichen oder bis zur Feldbreite, falls diese angegeben ist. Mit Ausnahme von `[` und `c` werden bei allen Formatangaben die Leerzeichen vor einem Eingabefeld ignoriert.

Umwandlungen können auch auf das *nte*-Argument in der Argumentenliste statt auf das nächste noch nicht verwendete Argument angewendet werden. In diesem Fall wird das Umwandlungszeichen `%` durch die Folge `%`*Ziffer*`$` ersetzt, wobei *Ziffer* eine Dezimalzahl *n* ist, die die Position des Arguments in der Argumentenliste angibt. Das erste Argument, `%1$`, folgt unmittelbar auf *Format*. Die Steuerzeichenkette kann entweder die Form einer Umwandlungsanweisung, d.h. `%`, oder die Form `%`*Ziffer*`$` enthalten, aber beide Formen können nicht innerhalb einer einzigen Steuerzeichenkette gemischt werden.

Umwandlungszeichen

Das Umwandlungszeichen zeigt die Interpretation des Eingabefelds an; das entsprechende Zeigerargument muß im Normalfall einen vorgeschriebenen Typ haben. Für ein zu unterdrückendes Feld wird kein Zeigerargument angegeben. Folgende Umwandlungszeichen sind gültig:

`%` Ein einzelnes `%` wird an dieser Stelle in der Eingabe erwartet; eine Zuweisung erfolgt nicht.

`d` liest eine Dezimalzahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie die Zeichenkette der Funktion `strtol` mit dem Wert 10 für das Argument *base*. Das entsprechende Argument soll ein Zeiger auf `int` sein.

`u` liest eine Dezimalzahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie Zeichenkette der Funktion `strtoul` mit dem Wert 10 für das Argument *base*. Das entsprechende Argument soll ein Zeiger auf `unsigned int` sein.

`o` liest eine Oktalzahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie die Zeichenkette der Funktion `strtoul` mit dem Wert 8 für das Argument *base*. Das entsprechende Argument soll ein Zeiger auf `unsigned int` sein.

`x` liest eine Hexadezimalzahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie die Zeichenkette der Funktion `strtoul` mit dem Wert 16 für das Argument *base*. Das entsprechende Argument soll ein Zeiger auf `unsigned int` sein.

`i` liest eine ganze Zahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie die Zeichenkette der Funktion `strtol` mit dem Wert 0 für das Argument *base*. Das entsprechende Argument soll ein Zeiger auf `int` sein.

`n` Es wird keine Eingabe verarbeitet. Das entsprechende Argument sollte ein Zeiger auf `int` sein, in den die Anzahl der Zeichen geschrieben werden, die bisher durch den Aufruf der Funktion von der Eingabe gelesen wurden. Die Ausführung einer `%n`-Anweisung erhöht nicht den Zuweisungszähler, der bei der Beendigung der Ausführung der Funktion zurückgegeben wird.

`e,f,g` liest eine Gleitkommazahl, wahlweise mit oder ohne Vorzeichen, die das gleiche Format hat, wie die Zeichenkette der Funktion `strtod`. Das entsprechende Argument sollte ein Zeiger auf `float` sein.

`s` liest eine Zeichenkette; das entsprechende Argument sollte ein Zeiger auf `char` sein, der auf ein Zeichenfeld zeigt, dessen Größe für die Aufnahme der Zeichenkette und ein abschließendes `\0` ausreicht, das automatisch hinzugefügt wird. Das Eingabefeld wird mit einem Leerzeichen beendet.

`c` liest eine Zeichenkette der Länge, die durch die Feldgröße angegeben ist (1, wenn in dieser Anweisung keine Feldgröße angegeben ist). Das entsprechende Argument sollte ein Zeiger auf das erste Zeichen eines Feldes sein, das groß genug ist, um die Folge aufzunehmen. Es wird kein Nullzeichen hinzugefügt. Das normale Überlesen von Leerzeichen wird unterdrückt.

`[` liest eine nichtleere Zeichenkette aus einer Menge von erwarteten Zeichen (Zeichenmenge). Das entsprechende Argument sollte ein Zeiger auf das erste Zeichen eines Feldes sein, das für die Folge groß genug ist, einschließlich eines abschließenden Nullzeichens, das automatisch hinzugefügt wird. Die Umwandlungsanweisung enthält alle aufeinanderfolgenden Zeichen in der Zeichenkette *Format*, bis einschließlich der entsprechenden rechten eckigen Klammer `]`. Die Zeichen zwischen den eckigen Klammern (Zeichenliste) umfassen die Zeichenmenge, außer wenn das Zeichen hinter der linken eckigen Klammer ein `^` ist. In diesem Fall enthält die Zeichenmenge alle Zeichen, die nicht in der Zeichenliste zwischen `^` und der rechten eckigen Klammer erscheinen. Wenn die Umwandlungsanweisung mit `[]` oder `[^]` beginnt, ist die rechte eckige Klammer in der Zeichenliste, und die nächste rechte eckige Klammer ist die rechte Klammer, die die Anweisung beendet. Sonst ist es immer die erste rechte

eckige Klammer, die die Anweisung beendet.
Ein Bereich von Zeichen in der Zeichenmenge kann in der Form *erstes* - *letztes* dargestellt werden; so kann `[0123456789]` auch durch `[0-9]` ausgedrückt werden. Bei einer solchen Konstruktion muß *erstes* lexikalisch kleiner oder gleich *letztes* sein, sonst steht - für sich selbst. Das Zeichen - steht auch immer dann für sich selbst, wenn es das erste oder letzte Zeichen in der Zeichenliste ist. Um die eckige rechte Klammer als ein Element der Zeichenmenge zu erhalten, muß sie als erstes Zeichen (möglicherweise mit ^ davor) in der Zeichenliste erscheinen. In diesem Fall wird sie syntaktisch nicht als schließende Klammer interpretiert. Es muß wenigstens ein Zeichen für diese Umwandlung passen, damit sie als erfolgreich betrachtet wird.

`p` liest eine implementierungsabhängige Menge von Folgen, die dieselbe sein sollte wie die Menge, die von der Umwandlung `%p` der Funktion `printf` erzeugt werden kann. Das entsprechende Argument sollte ein Zeiger auf `void` sein. Die Interpretation der Eingabeelements ist implementierungsabhängig. Wenn das Eingabeelement ein Wert ist, der schon früher bei derselben Programmausführung umgewandelt wurde, sollte der Ergebniszeiger genau diesem Wert entsprechen, sonst ist das Verhalten der Umwandlung `%p` nicht definiert.

Wenn auf das `%` ein ungültiges Umwandlungszeichen folgt, sind die Ergebnisse der Operation nicht vorhersehbar.

Die Umwandlungszeichen `E`, `G` und `X` sind auch gültig. Im ANSI-Modus (`cc -kansi`) verhalten sich `E`, `G` und `X` genau wie `e`, `g` bzw. `x`. Im K&R-Modus (`cc -kcc`) verhalten sich `E`, `G` und `X` genau wie `le`, `lg` bzw. `lx`.

Jede Funktion berücksichtigt die Erkennung eines sprachabhängigen Dezimalpunkts in der Eingabefolge. Der Dezimalpunkt ist durch die Umgebung des Programms definiert (`LC_NUMERIC`). In der C-Umgebung oder in einer Umgebung, in der der Dezimalpunkt nicht definiert ist, entspricht er standardmäßig dem Punkt.

Die Umwandlung durch `scanf` endet am Dateiende, am Ende der Steuerzeichenkette, oder wenn ein Eingabezeichen in Konflikt mit der Steuerzeichenkette gerät.

Wenn das Dateiende erreicht wird, bevor Zeichen, die auf die aktuelle Anweisung passen, gelesen wurden (außer führenden Leerzeichen, wo erlaubt), wird die Ausführung der aktuellen Anweisung mit einem Eingabefehler beendet. Wird das Dateiende innerhalb der Ausführung einer der folgenden Anweisungen erreicht, so wird die Umwandlung mit einem Eingabefehler beendet.

Wenn die Umwandlung bei einem Konflikt (einem nicht passenden Eingabezeichen) beendet wird, wird das fehlerhafte Eingabezeichen ungelesen in der Eingabe belassen. Abschließende Leerzeichen (einschließlich Neue-Zeile-Zeichen) bleiben ungelesen, wenn sie nicht auf eine Anweisung passen. Der Erfolg von Zeichenübereinstimmungen und unterdrückten Zuweisungen kann nur mit der Anweisung `%n` bestimmt werden.

BEISPIELE

Der Aufruf der Funktion `scanf`:

```
int i, n; float x; char name[50];
n = scanf ("%d%f%s", &i, &x, name);
```

mit der Eingabezeile:

```
25 54.32E-1 thompson
```

weist `n` den Wert `3`, `i` den Wert `25` und `x` den Wert `5.432` zu, und `name` enthält die Zeichenkette `thompson\0`.

Der Aufruf der Funktion `scanf`:

```
int i; float x; char name[50];
(void) scanf ("%2d%f%*d\ %[0-9]", &i, &x, name);
```

mit der Eingabezeile:

```
56789 0123 56a72
```

weist `i` `56` und `x` `789.0` zu, überliest `0123` und bringt die Zeichen `56\0` nach `name`. Das nächste von `stdin` gelesene Zeichen wird `a` sein.

SIEHE AUCH

`cc`(1), `printf`(3S), `strtod`(3C), `strtol`(3C), `strtoul`(3C).

ERGEBNIS

Die Funktion gibt die Anzahl erfolgreich gelesener und zugewiesener Eingabesymbole zurück. Diese Zahl kann null sein, wenn schon früh ein Konflikt zwischen einem Eingabezeichen und der Steuerzeichenkette aufgetreten ist. Wenn die Eingabe vor dem ersten Konflikt oder der ersten Umwandlung aufhört, wird `EOF` zurückgegeben.

setbuf, setvbuf - Pufferung für Stream festlegen

```
#include <stdio.h>
void setbuf (FILE *stream, char *buf);
int setvbuf (FILE *stream, char *buf, int type, size_t size);
```

`setbuf` kann verwendet werden, nachdem eine stream-Datei eröffnet wurde (siehe `intro`(3)), aber bevor sie gelesen bzw. geschrieben wird. Es bewirkt, daß das Feld, auf das *buf* zeigt, anstelle eines automatisch zugewiesenen Puffers verwendet wird. Wenn *buf* der Nullzeiger ist, sind Eingabe und Ausgabe völlig ungepuffert.

Die Puffergröße ist nicht begrenzt; die Konstante `BUFSIZ`, die in der Include-Datei `stdio.h` definiert ist, bezeichnet jedoch eine geeignete Puffergröße:

```
char buf[BUFSIZ];
```

`setvbuf` kann verwendet werden, nachdem eine gepufferte Datei eröffnet wurde, aber bevor sie gelesen bzw. geschrieben wird. `type` bestimmt, wie *stream* gepuffert wird. Die zulässigen Werte für *type* (in `stdio.h` definiert) haben folgende Bedeutung:

`_IOFBF` vollständige Pufferung der Ein-/Ausgabe

`_IOLBF` Zeilenpufferung der Ausgabe; der Puffer wird entleert, wenn das Neue-Zeile-Zeichen geschrieben wird, der Puffer voll ist oder eine Eingabe angefordert wird

`_IONBF` keine Ein-/Ausgabe-Pufferung

Wenn *buf* nicht der Nullzeiger ist, wird das Feld, auf das gezeigt wird, anstelle des automatisch zugewiesenen Puffers verwendet. *size* gibt die Größe des zu verwendenden Puffers vor. Ist die Ein-/Ausgabe nicht gepuffert, werden *buf* und *size* ignoriert.

SIEHE AUCH

`fopen`(3S), `getc`(3S), `malloc`(3C), `putc`(3S), `stdio`(3S).

ERGEBNIS

Wird ein unzulässiger Wert für *type* verwendet, dann gibt `setvbuf` einen Wert ungleich null zurück. Andernfalls wird null zurückgegeben.

HINWEIS

Eine häufige Fehlerquelle besteht darin, daß als Puffer in einem Programmblock eine Variable der Speicherklasse `auto` verwendet und die Datei in diesem Block dann nicht geschlossen wird.

Teile von *buf* werden für interne Verwaltungsinformationen des Streams benötigt, deswegen enthält *buf* weniger als *size* Bytes wenn er voll ist. Sie sollten bei der Verwendung von `setvbuf` automatisch zugewiesene Puffer verwenden.

setjmp, longjmp - nichtlokaler Sprung

```
#include <setjmp.h>
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
```

Mit diesen Funktionen können Fehler und Unterbrechungen bearbeitet werden, die in einer betriebssystemnahen Funktion eines Programms auftreten.

`setjmp` sichert die Aufrufumgebung in *env* (dessen Typ, `jmp_buf`, in der Include-Datei `setjmp.h` definiert ist) für eine spätere Verwendung durch `longjmp`. Es gibt den Wert 0 zurück.

`longjmp` stellt die Umgebung wieder her, die durch den letzten Aufruf von `setjmp` mit dem entsprechenden Argument *env* gesichert wurde. Nach Durchführung von `longjmp` läuft das Programm weiter, als ob der entsprechende Aufruf von `setjmp` (der in der Zwischenzeit nicht zurückgekehrt sein darf) gerade den Wert `val` zurückgegeben hat. `longjmp` kann `setjmp` nicht zur Rückgabe des Wertes 0 veranlassen. Wenn `longjmp` mit einem zweiten Argument 0 aufgerufen wird, gibt `setjmp` 1 zurück. Zum Zeitpunkt der zweiten Rückkehr von `setjmp` haben alle externen und statisch deklarierten Variablen die Werte vom Zeitpunkt des `longjmp` Aufrufs (siehe Beispiel). Die Werte von Variablen, die als Register oder automatisch deklariert sind, sind nicht definiert.

Register oder automatische Variablen, deren Wert zuverlässig sein muß, müssen als `volatile` deklariert sein.

BEISPIEL

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

jmp_buf env;
int i = 0;
main ()
{
      void exit();

      if(setjmp(env) != 0) {
            (void) printf("Wert von i bei 2. Rückkehr von setjmp: %d\n", i);
            exit(0);
      }
      (void) printf("Wert von i bei 1. Rückkehr von setjmp: %d\n", i);
      i = 1;
      g();
      /*NOTREACHED*/
}
g()
{
      longjmp(env, 1);
      /*NOTREACHED*/
}
```

Wird das sich aus diesem C-Code ergebende `a.out` ausgeführt, sieht die Ausgabe folgendermaßen aus:

```
Wert von i bei 1. Rückkehr von setjmp:0

Wert von i bei 2. Rückkehr von setjmp:1
```

SIEHE AUCH

`signal`(2), `sigsetjmp`(3C).

HINWEIS

Wenn `longjmp` aufgerufen wurde, obwohl *env* nie durch einen Aufruf von `setjmp` vorbereitet wurde, oder wenn der letzte derartige Aufruf in einer Funktion war, die inzwischen beendet wurde, ist ein absolutes Chaos garantiert.

setlocale - Umgebung eines Programms abfragen/einstellen

```
#include <locale.h>
char *setlocale (int category, const char *locale);
```

`setlocale` wählt den entsprechenden Teil der lokalen Umgebung eines Programms aus, wie er in *category* und *locale* angegeben wird. *category* kann die folgenden Werte enthalten: `LC_CTYPE`, `LC_NUMERIC`, `LC_TIME`, `LC_COLLATE`, `LC_MONETARY`, `LC_MESSAGES` und `LC_ALL`. Diese Namen werden in der Datei `locale.h` definiert. `LC_CTYPE` beeinflußt das Verhalten der Funktionen zur Zeichenbearbeitung (`isdigit`,`tolower`, etc.) und die Mehrbyte-Funktionen zur Zeichenbearbeitung (wie `mbtowc` und `wctomb`). `LC_NUMERIC` beeinflußt den Dezimalpunkt für die formatierte Ein/Ausgabe und die Zeichenketten-Umwandlungsfunktionen und die nichtmonetären Formatierungsinformationen, die von `localeconv` zurückgegeben werden (siehe `localeconv`(3C)). `LC_TIME` beeinflußt das Verhalten von `ascftime`, `cftime`, `getdate` und `strftime`. `LC_COLLATE` beeinflußt das Verhalten von `strcoll` und `strxfrm`. `LC_MONETARY` beeinflußt die monetären Formatierungsinformationen, die von `localeconv` zurückgegeben werden. `LC_MESSAGES` beeinflußt das Verhalten von `gettxt`, `catopen`, `catclose` und `catgets` (siehe `catopen`(3C) und `catgets`(3C)). `LC_ALL` bezeichnet die vollständige lokale Umgebung des Programms.

Jede Kategorie entspricht einer Datenbasis, die die relevanten Informationen für jede definierte lokale Umgebung enthält. Die Position einer Datenbasis wird durch den Pfad `/usr/lib/locale/`*locale*`/`*category* gegeben, wobei *locale* und *category* die Namen der lokalen Umgebung und der Kategorien sind. Beispielsweise werden die Daten für die Kategorie `LC_CTYPE` für die lokale Umgebung "german" unter `/usr/lib/locale/german/LC_CTYPE` angesprochen.

Der Wert "C" für *locale* gibt die voreingestellte lokale Umgebung an.

Der Wert "" für *locale* gibt an, daß die lokale Umgebung aus den Umgebungsvariablen übernommen werden soll. Die Reihenfolge, in der die Umgebungsvariablen auf die verschiedenen Kategorien überprüft werden, ist:

Kategorie	1. Umg. Var.	2. Umg.Var.
LC_CTYPE:	LC_CTYPE	LANG
LC_COLLATE:	LC_COLLATE	LANG
LC_TIME:	LC_TIME LANG	
LC_NUMERIC:	LC_NUMERIC	LANG
LC_MONETARY:	LC_MONETARY	LANG
LC_MESSAGES:	LC_MESSAGES	LANG

Bei Programmstart wird das Äquivalent von

```
setlocale(LC_ALL, "C")
```

ausgeführt. Dadurch wird jede Kategorie auf der lokalen Umgebung initialisiert, die durch die lokale Umgebung "C" beschrieben ist.

Wird für *locale* ein Zeiger auf eine Zeichenkette übergeben, so versucht `setlocale`, die lokale Umgebung für die gegebene Kategorie auf *locale* zu setzen. Ist `setlocale` erfolgreich, kehrt *locale* zurück. Scheitert `setlocale`, wird ein Nullzeiger zurückgegeben, und die lokale Umgebung des Programms wird nicht verändert.

Für die Kategorie `LC_ALL` ist das Verhalten etwas anders. Wird für *locale* ein Zeiger auf eine Zeichenkette übergeben und ist die Kategorie gleich `LC_ALL`, dann versucht `setlocale`, die Umgebungseinstellungen für alle Kategorien auf *locale* zu setzen. *locale* kann dabei eine einfache oder eine zusammengesetzte lokale Umgebung sein. Eine zusammengesetzte lokale Umgebung ist eine Zeichenkette, die mit einem `/` beginnt, auf den die lokale Umgebung jeder Kategorie folgt, wiederum getrennt durch ein `/`. Wenn `setlocale` für eine Kategorie scheitert, wird ein Nullzeiger zurückgegeben, und die Kategorien der Umgebungseinstellungen des Programms werden nicht geändert. Andernfalls wird die lokale Umgebung zurückgeliefert.

Ein Nullzeiger für *locale* veranlaßt `setlocale`, die aktuelle lokale Umgebung zurückzugeben, die für *category* eingestellt ist. Die Einstellung wird geändert.

DATEIEN

`/usr/lib/locale/C/LC_CTYPE - LC_CTYPE`
`/usr/lib/locale/C/LC_NUMERIC - LC_NUMERIC`
`/usr/lib/locale/C/LC_TIME - LC_TIME`
`/usr/lib/locale/C/LC_COLLATE - LC_COLLATE`
`/usr/lib/locale/C/LC_MESSAGES - LC_MESSAGES`
`/usr/lib/locale/`*locale*`/`*category*

Dateien mit umgebungsspezifischen Informationen für jede lokale Umgebung und jede Kategorie.

SIEHE AUCH

`ctime`(3C), `ctype`(3C), `getdate`(3C), `gettxt`(3G), `localeconv`(3C), `mbtowc`(3C), `printf`(3S), `strcoll`(3C), `strftime`(3C), `strtod`(3C), `strxfrm`(3C), `wctomb`(3C), `environ`(5).

sigsetjmp, siglongjmp - nichtlokaler Sprung mit Signalstatus

```
#include <setjmp.h>
int sigsetjmp (sigjmp_buf env, int savemask);
void siglongjmp (sigjmp_buf env, int val);
```

Mit diesen Funktionen könne Fehler und Unterbrechungen innerhalb von Low-Level-Routinen eines Programms bearbeitet werden.

`sigsetjmp` sichert die Register des aufrufenden Prozesses und die Umgebung (siehe `sigaltstack`(2)) in *env* (dessen Typ, `sigjmp_buf`, in der Datei `setjmp.h` definiert wird) zur späteren Verwendung durch `siglongjmp`. Wenn *savemask* ungleich Null ist, werden die Signalmaske des aufrufenden Prozesses (siehe `sigprocmask`(2)) und die Verwaltungsparameter (siehe `priocntl`(2)) ebenfalls gesichert. `sigsetjmp` liefert den Wert 0.

`siglongjmp` stellt die im Argument *env* durch den letzten Aufruf von `sigsetjmp` gesicherte Umgebung wieder her. Nachdem `siglongjmp` ausgeführt wurde, wird das Programm weiter abgearbeitet, als wenn der entsprechende Aufruf von `sigsetjmp` den Wert *val* zurückgegeben hätte. `siglongjmp` kann `sigsetjmp` nicht dazu veranlassen, den Wert 0 zurückzugeben. Wird `siglongjmp` mit dem zweiten Argument Null aufgerufen, liefert `sigsetjmp` den Wert 1. Bei dem zweiten Rücksprung von `sigsetjmp` enthalten alle externen und statischen Variablen diejenigen Werte, die sie zur Zeit des Aufrufs von `siglongjmp` besaßen. Die Werte der Register und der Variablen der Speicherklasse `auto` sind undefiniert. Register oder `auto`-Variablen, deren Werte beibehalten werden sollen, müssen mit `volatile` deklariert werden.

Wenn eine Signalbehandlungsfunktion den Aufruf `sleep` unterbricht und `siglongjmp` aufruft, um eine Umgebung wiederherzustellen, die vor dem Aufruf von `sleep` abgelegt wurde, so ist die Aktion und die Zeit, die mit `SIGALRM` verknüpft ist, nicht angegeben. Gleiches gilt, wenn das Signal `SIGALRM` blockiert wird, es sei denn, die Signalmaske des Prozesses wird als Teil der jeweiligen Umgebung wiederhergestellt.

Die Funktion `siglongjmp` stellt die gesicherte Signalmaske nur dann wieder her, wenn das Argument *env* durch den Aufruf von `sigsetjmp` mit dem Argument *savemask* initialisiert wurde, welches ungleich Null ist.

SIEHE AUCH

`getcontext`(2), `priocntl`(2), `sigaction`(2), `sigaltstack`(2), `sigprocmask`(2), `setjmp`(3C).

HINWEIS

Wenn `siglongjmp` aufgerufen wird, obwohl *env* nicht durch `sigsetjmp` initialisiert wurde, oder wenn der letzte derartige Aufruf in einer Funktion stattfand, deren Ausführung inzwischen beendet ist, wird absolutes Chaos garantiert.

sigsetops - Signalmengen bearbeiten

```
#include <signal.h>
int sigemptyset (sigset_t *set);
int sigfillset (sigset_t *set);
int sigaddset (sigset_t *set, int signo);
int sigdelset (sigset_t *set, int signo);
int sigismember (sigset_t *set, int signo);
```

Diese Funktionen bearbeiten die Datentypen *sigset_t*, welche die von der Implementierung unterstützte Signalmenge darstellt.

`sigemptyset` initialisiert die Signalmenge, auf die *set* zeigt so, daß alle vom System definierten Signale ausgeschlossen werden.

`sigfillset` initialisiert die Signalmenge, auf die *set* zeigt so, daß alle vom System definierten Signale eingeschlossen werden.

`sigaddset` fügt das Signal *signo* zu der Signalmenge *set* hinzu.

`sigdelset` löscht das Signal *signo* aus der Signalmenge *set*.

`sigismember` prüft, ob das Signal *signo* Teil der Signalmenge *set* ist.

Jedes Objekt des Typs *sigset_t* muß entweder durch `sigemptyset` oder `sigfillset` initialisiert werden, bevor eine andere Operation mit dem Objekt durchgeführt werden kann.

`sigaddset`, `sigdelset` und `sigismember` schlagen fehl, wenn folgende Bedingung erfüllt ist:

`EINVAL` Der Wert des Argumenrts *signo* ist keine gültige Signalnummer.

`sigfillset` schlägt fehl, wenn folgende Bedingung erfüllt ist:

`EFAULT` Das Argument *set* gibt eine ungültige Adresse an.

SIEHE AUCH

`sigaction`(2), `sigprocmask`(2), `sigpending`(2), `sigsuspend`(2), `signal`(5).

ERGEBNIS

Nach erfolgreicher Ausführung liefert die Funktion `sigismember` den Wert 1 zurück, wenn das angegebene Signal zu der angegebenen Signalmenge gehört; andernfalls wird 0 zurückgegeben. Nach erfolgreicher Ausführung liefern die anderen Funktionen den Wert 0 zurück; andernfalls wird -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

sleep - Ausführung unterbrechen

```
#include <unistd.h>
unsigned sleep (unsigned Sekunden);
```

Die Ausführung des laufenden Prozesses wird für *Sekunden* angehalten. Die tatsächliche Unterbrechung kann kürzer als angefordert sein, weil jedes abgefangene Signal `sleep` im Anschluß an die Ausführung beendet. Auch kann die Unterbrechung durch andere Aktivitäten im System länger als angefordert sein. Der von `sleep` zurückgegebene Wert ist der 'Nichtpausiert'-Betrag (die angeforderte Länge der Unterbrechung minus der Zeit, in der der Prozeß tatsächlich angehalten war), wenn der Aufruf eine Alarmuhr gesetzt hat, die vor dem Ende der mit `sleep` angeforderten Pause abläuft, oder wenn der Prozeß aufgrund eines anderen abgefangenen Signals vorzeitig weiterläuft.

Die Funktion ist durch Setzen eines Alarmsignals und Pausieren bis zum Auftreten dieses (oder eines anderen) Signals realisiert. Der vorherige Status des Alarmsignals wird gesichert und wiederhergestellt. Das Aufrufprogramm kann vor dem Aufruf von `sleep` ein Alarmsignal gesetzt haben. Wenn die `sleep`-Zeit die Dauer bis zu einem solchen Alarmsignal überschreitet, pausiert der Prozeß nur bis zu dem Zeitpunkt, an dem das Alarmsignal ausgelöst wird. Die Alarmbehandlung des Aufrufs wird unmittelbar vor Rückkehr der `sleep`-Funktion ausgeführt. Wenn jedoch die `sleep`-Dauer kürzer als die Zeit bis zur Auslösung eines solchen Alarms ist, wird die vorherige Alarmzeitdauer so zurückgesetzt, daß sie zum gleichen Zeitpunkt ausläuft wie ohne den dazwischenliegenden `sleep`-Aufruf.

SIEHE AUCH

`alarm`(2), `pause`(2), `signal`(2), `wait`(2).

ssignal, gsignal - Softwaresignale

```
#include <signal.h>
int (*ssignal (int sig, int (*action) (int))) (int);
int gsignal (int sig);
```

`ssignal` und `gsignal` bieten softwarespezifische Möglichkeiten, die ähnlich denen von `signal`(2) sind.

Die zur Verfügung gestellten Softwaresignale werden über ganze Zahlen von 1 bis 17 (inklusive) angesprochen. Ein Aufruf von `ssignal` ordnet dem Softwaresignal *sig* eine Prozedur *action* zu. *sig*, wird durch einen Aufruf von `gsignal` erzeugt. Es bewirkt, daß die für das betreffende Signal eingerichtete Aktion ausgeführt wird.

Das erste Argument für `ssignal` ist eine Zahl, mit der der Signaltyp bezeichnet wird, für den eine Maßnahme eingerichtet wird. Das zweite Argument definiert die Maßnahme; es ist entweder der Name einer (vom Benutzer definierten) Funktion *action* oder eine der vordefinierten Konstanten `SIG_DFL` (Standard) oder `SIG_IGN` (Ignorieren). `ssignal` gibt die Maßnahme zurück, die vorher für diesen Signaltyp festgelegt wurde; wenn keine Maßnahme eingerichtet wurde oder die Signalnummer unzulässig ist, gibt `ssignal` `SIG_DFL` zurück.

`gsignal` löst das Signal aus, das durch das Argument *sig* angegeben wird:

- Wenn eine Aktions für *sig* eingerichtet wurde, wird diese auf `SIG_DFL` zurückgesetzt und die Aktion mit dem Argument *sig* aufgerufen. `gsignal` gibt den Wert zurück, der von der Aktion zurückgegeben wurde.
- Wenn die Aktion für *sig* `SIG_IGN` ist, gibt `gsignal` den Wert 1 zurück und führt keine weiteren Maßnahmen mehr aus.
- Wenn die Aktion für *sig* `SIG_DFL` ist, gibt `gsignal` den Wert 0 zurück und führt keine weiteren Maßnahmen mehr aus.
- Wenn *sig* einen unzulässigen Wert aufweist oder keine Maßnahmen für *sig* angegeben worden sind, gibt `gsignal` den Wert 0 zurück und führt keine weiteren Maßnahmen mehr aus.

SIEHE AUCH

`signal`(2), `sigset`(2), `raise`(3C).

stdio - Standardfunktionen für gepufferte Ein-/Ausgabe

```
#include <stdio.h>
FILE *stdin, *stdout, *stderr;
```

Die in den Abschnitte 3S dieses Handbuchs beschriebenen Funktionen stellen einen leistungsfähigen E/A-Pufferungsmechanismus auf Benutzerebene dar. Die Makros `getc` und `putc` können Zeichen rasch bearbeiten. Die Makros `getchar` und `putchar` und die Routinen auf höherer Ebene `fgetc`, `fgets`, `fprintf`, `fputc`, `fputs`, `fread`, `fscanf`, `fwrite`, `gets`, `getw`, `printf`, `puts`, `putw` und `scanf` verwenden `getc` und `putc` oder funktionieren so, als ob sie diese verwenden; sie sind beliebig gemischt verwendbar.

Eine Datei mit zugeordneten E/A-Puffer wird als *Stream* (siehe `intro`(3)) bezeichnet und als Zeiger auf einen vordefinierten Typ `FILE` vereinbart. `fopen` erstellt verschiedene Daten für eine solche Datei und gibt einen Zeiger zurück, über den die Datei bei allen weiteren Vorgängen angesprochen wird. Normalerweise gibt es drei offene Dateien mit konstanten Dateizeigern, die in der Include-Datei `stdio.h` definiert und den Standarddateien zugeordnet sind:

`stdin`	Standardeingabe
`stdout`	Standardausgabe
`stderr`	Standardfehlerausgabe

Die folgenden symbolischen Werte in `unistd.h` definieren die Dateideskriptoren, die mit `stdin`, `stdout` und `stderr` der Sprache C verbunden werden, wenn das Anwendungsprogramm gestartet wird:

`STDIN_FILENO`	Standardeingabe-Wert, `stdin` (0).
`STDOUT_FILENO`	Standardausgabe-Wert, `stdout` (1).
`STDERR_FILENO`	Standardfehler-Wert, `stderr` (2).

Eine Konstante `NULL` bezeichnet einen Nullzeiger.

Eine ganzzahlige Konstante `EOF` (-1) wird von den meisten ganzzahligen Funktionen, die mit Dateizeigern arbeiten, bei einem Dateiende oder einem Fehler ausgegeben (siehe Details in den Einzelbeschreibungen).

Eine ganzzahlige Konstante `BUFSIZ` gibt die Größe der Puffer an, die für die jeweilige Implementierung definiert ist.

Eine ganzzahlige Konstante `FILENAME_MAX` gibt die Größe für ein `char`-Feld an, das den längsten Pfadnamen, der in der Implementierung geöffnet werden kann, aufnehmen kann.

Eine ganzzahlige Konstante FOPEN_MAX gibt die Mindestanzahl von Dateien an, von denen die Implementierung garantiert, daß sie gleichzeitig geöffnet werden können. Beachten Sie, daß nicht mehr als 255 Dateien mit fopen geöffnet werden können, und daß nur Dateideskriptoren zwischen 0 und 255 gültig sind.

Jedes mit diesem Paket arbeitende Programm muß die Include-Datei mit den notwendigen Makrodefinitionen wie folgt verwenden:

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

Die in den Abschnitten 3S dieses Handbuchs erwähnten Funktionen und Konstanten werden in der Include-Datei deklariert und benötigen keine weitere Deklaration. Die Konstanten und nachstehenden 'Funktionen' sind als Makros implementiert (ein Neudeklarieren dieser Namen ist gefährlich!): getc, getchar, putc, putchar, ferror, feof, clearerr und fileno. Es gibt auch Funktionsversionen von getc, getchar, putc, putchar, ferror, feof, clearerr und fileno.

Mit Ausnahme der Standard-Fehlerausgabe stderr wird die Ausgabe standardmäßig gepuffert, wenn sie sich auf eine Datei bezieht, und zeilenweise gepuffert, wenn die Ausgabe auf ein Terminal geht. Die Standard-Fehlerausgabe stderr ist standardmäßig ungepuffert, jedoch kann die Pufferung oder die zeilenweise Pufferung mit freopen (siehe fopen(3S)) eingestellt werden. Wenn die Ausgabe ungepuffert ist, werden die Daten sofort beim Schreiben in die Warteschlange für die Zieldatei oder den Bildschirm gebracht; wenn die Ausgabe gepuffert ist, werden Zeichen gespeichert und dann als Block geschrieben. Wenn die Ausgabe zeilenweise gepuffert ist, wird jede Ausgabezeile in die Warteschlange des zugeordnete Bildschirms eingetragen, und sobald die Zeile beendet ist (d.h., sobald ein Neue-Zeile-Zeichen geschrieben oder Eingabe vom Terminal angefordert wird) ausgegeben. setbuf oder setvbuf (beide in setbuf(3S) beschrieben) können die Pufferungsstrategie einer Datei ändern.

SIEHE AUCH

open(2), close(2), lseek(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S).

ERGEBNIS

Ungültige Dateizeiger verursachen meistens große Unordnung, die möglicherweise sogar zum Programmabbruch führen kann. Eine genauere Erläuterung der möglichen Fehlerbedingungen ist bei den einzelnen Funktionsbeschreibungen zu finden.

stdipc: ftok - Standardpaket für Interprozeßkommunikation

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

key_t ftok(const char *path, int id);
```

Für jede Art von Interprozeßkommunikation muß der Benutzer einen Schlüssel bereitstellen, der von den Systemaufrufen `msgget`(2), `semget`(2) und `shmget`(2) zum Erhalt der Kennung für die Interprozeßkommunikation verwendet wird. Eine Möglichkeit, einen Schlüssel zu definieren, bietet die Funktion `ftok`. Eine andere Möglichkeit für das Erstellen von Schlüsseln besteht darin, die Projektnummer im höchstwertigen Byte abzuspeichern und den übrigen Teil für eine laufende Nummer zu verwenden. Es gibt viele weitere Möglichkeiten zum Aufbauen von Schlüsseln, jedoch ist es notwendig, daß jedes System bestimmte Normen für deren Bildung festlegt. Wenn eine bestimmte Norm hierfür nicht eingehalten wird, besteht die Gefahr, daß nicht zugehörige Prozesse unbeabsichtigt den Ablauf anderer Prozesse stören. Es ist immer noch möglich, daß zwei Prozesse zufällig den gleichen Schlüssel benutzen. Achten Sie daher darauf, daß das höchstwertige Byte eines Schlüssels auf irgendeine Weise auf ein Projekt verweist, so daß die Schlüssel nicht innerhalb eines Systems in Konflikt geraten.

`ftok` gibt einen Schlüssel zurück, der auf *path* und *id* basiert und der in nachfolgenden Systemaufrufen `msgget`, `semget` und `shmget` verwendet werden kann. *path* muß der Pfadname einer bestehenden Datei sein, auf die der Prozeß zugreifen kann. *id* ist ein Zeichen, das ein Projekt eindeutig kennzeichnet. Beachten Sie, daß `ftok` für Dateien, die Verweise sind, den gleichen Schlüssel zurückgibt, wenn es mit der gleichen Kennung *id* aufgerufen wird, und daß es andere Schlüssel zurückgibt, wenn es mit demselben Dateinamen, jedoch mit anderen Kennungen *id* aufgerufen wird.

SIEHE AUCH

`intro`(2), `msgget`(2), `semget`(2), `shmget`(2).

ERGEBNIS

`ftok` gibt `(key_t) -1` zurück, wenn *path* nicht vorhanden ist, oder wenn der Prozeß nicht darauf zugreifen kann.

HINWEIS

Wenn die Datei entfernt wird, deren Pfad an `ftok` weitergereicht wurde, während die Schlüssel weiterhin auf diese Datei verweisen, erfolgt bei späteren Aufrufen von `ftok` mit gleichem *path* und *id* die Ausgabe eines Fehlers. Wenn dieselbe Datei wieder erstellt wird, gibt `ftok` wahrscheinlich einen anderen Schlüssel als bei dem ursprünglichen Aufruf zurück.

strcoll - Zeichenketten sortieren

```
#include <string.h>
int strcoll (const char *s1, const char *s2);
```

`strcoll` liefert eine ganze Zahl, die größer, kleiner oder gleich Null ist, abhängig davon, ob die Zeichenkette *s1* größer, gleich oder kleiner als die Zeichenkette *s2* ist. Der Vergleich der Zeichenketten hängt davon ab, wie die Kategorie `LC_COLLATE` der jeweiligen lokalen Umgebung eingestellt ist (siehe `setlocale`(3C)).

Mit `strcoll` und `strxfrm` können Zeichenketten umgebungsabhängig sortiert werden. `strcoll` ist für Anwendungen gedacht, bei denen die Anzahl der Vergleiche pro Zeichenkette gering ist. Wenn Zeichenketten oft verglichen werden, sollte `strxfrm` verwendet werden, da dann der Transformationsprozeß nur einmal stattfindet.

DATEIEN

`/usr/lib/locale/`*locale*`/LC_COLLATE` Datenbasis für *locale*.

SIEHE AUCH

`setlocale`(3C), `string`(3C), `strxfrm`(3C), `environ`(5).
`colltbl`(1M) in "Referenzhandbuch für Systemverwalter".

strerror - Fehlermeldung feststellen

```
#include <string.h>
char *strerror (int errnum);
```

`strerror` liefert einen Zeiger auf die Fehlermeldung zurück, die durch die Nummer *errnum* angegeben wird. `strerror` verwendet dieselbe Menge von Fehlermeldungen wie `perror`. Die zurückgelieferte Zeichenkette sollte nicht überschrieben werden.

SIEHE AUCH

`perror`(3C).

strftime - Datum und Zeit in Zeichenkette umwandeln

```
#include <time.h>
size_t *strftime (char *s, size_t maxsize, const char *format, const struct tm *timeptr);
int cftime (char *s, char *format, const time_t *clock);
int ascftime (char *s, const char *format, const struct tm *timeptr);
```

strftime, ascftime, und cftime formatieren Datum und Zeit nach den Angaben aus der Zeichenkette *format* in das Feld, auf das s zeigt. Dabei besteht die Zeichenkette *format* aus einer oder mehreren Umwandlungsanweisungen und gewöhnlichen Zeichen. Alle gewöhnlichen Zeichen, einschließlich des abschließenden Nullzeichens, werden unverändert in die Zeichenkette kopiert. Bei Verwendung von strftime werden nicht mehr als *maxsize* Zeichen in die Zeichenkette geschrieben.
Wenn *format* gleich (char *)0 ist, dann wird für strftime das voreingestellte Format "%c" und für cftime und ascftime das voreingestellte Format verwendet. "%C". cftime und ascftime versuchen zuerst, den Wert der Umgebungsvariablen CFTIME festzustellen; ist diese Umgebungsvariable nicht definiert oder leer, wird das voreingestellte Format verwendet.

Jede Umwandlungsanweisung wird durch die entsprechenden Zeichen ersetzt, die in der folgenden Liste beschrieben werden. Die entsprechenden Zeichen werden durch die Kategorie LC_TIME der lokalen Umgebung, bei strftime und ascftime durch den Inhalt von *timeptr* und bei cftime durch den Inhalt von *clock* bestimmt:

```
%%    ist das Zeichen %
%a    abgekürzter Wochentagsname der lokalen Umgebung
%A    ausgeschriebener Wochentagsname der lokalen Umgebung
%b    abgekürzter Monatsname der lokalen Umgebung
%B    ausgeschriebener Monatsname der lokalen Umgebung
%c    entsprechende Datums- und Zeitdarstellung der lokalen Umgebung
%C    Datums- und Zeitdarstellung nach date(1)
%d    Monatstag (01 - 31)
%D    Datum als %m/%d/%y
%e    Monatstag (1-31; vor einzelnen Ziffern steht ein Leerzeichen)
%h    abgekürzter Monatsname der lokalen Umgebung
%H    Stunde (00 - 23)
%I    Stunde (01 - 12)
%j    Tag des Jahres (001 - 366)
%m    Nummer des Monats (01 - 12)
%M    Minute (00 - 59)
%n    entspricht \n
%p    Bezeichnung der lokalen Umgebung entweder AM oder PM
%r    Zeit im Format %I:%M:%S [AM|PM]
%R    Zeit im Format %H:%M
%S    Sekunden (00 - 61), erlaubt Schaltsekunden
%t    fügt ein Tabulatorzeichen ein
%T    Zeit im Format %H:%M:%S
%U    Nummer der Woche im Jahr (00 - 53), Sonntag ist der erste Tag der Woche 1
%w    Wochentag als Zahl (0 - 6), Sonntag = 0
%W    Nummer der Woche im Jahr (00 - 53), Montag ist der erste Tag der Woche 1
%x    die entsprechende Datumsdarstellung der lokalen Umgebung
%X    die entsprechende Zeitdarstellung der lokalen Umgebung
%y    Jahr innerhalb des Jahrhunderts (00 - 99)
%Y    Jahr in der Form ccyy (z.B. 1986)
%Z    Zeitzonenname; enthält keine Zeichen, wenn keine Zeitzone existiert
```

Der Unterschied zwischen %U und %W besteht darin, welcher Tag der erste in einer Woche ist. Die Woche 01 ist die erste Woche im Januar, die mit einem Sonntag (bei %U) oder einem Montag (bei %W) anfängt. Die Wochennummer 00 enthält diejenigen Tage vor dem ersten Sonntag (%U) oder Montag (%W) im Januar.

Ist die Anzahl der resultierenden Zeichen einschließlich Nullzeichen nicht größer als *maxsize*, liefern strftime, cftime und ascftime die Anzahl der Zeichen zurück, die nach *s* kopiert wurden (ausschließlich des Nullzeichens). Ansonsten wird 0 zurückgegeben, und der Inhalt von *s* ist unbestimmt. cftime und ascftime liefern die Anzahl der Zeichen (ausschließlich des abschließenden Nullzeichens) zurück, die nach *s* kopiert wurden.

Auswahl der Ausgabesprache

Die voreingestellte Sprache für die Ausgaben von strftime, cftime und ascftime ist amerikanisches Englisch. Der Benutzer kann die Ausgabesprache von strftime, cftime oder ascftime durch Einstellung der Kategorie LC_TIME mit setlocale für die lokale Umgebung auswählen.

Zeitzone

Die Zeitzone wird aus der Umgebungsvariablen TZ übernommen (siehe ctime(3C)).

BEISPIELE

Das folgende Beispiel demonstriert die Benutzung von strftime. Nach der Ausführung des folgenden Funktionsaufrufs mit dem Datum Donnerstag, 28. August, 1986 um 12:44:36 in Baden-Württemberg in *tmptr*:

```
strftime (str, strsize, "%A %b %d %j", tmptr)
```

erhält man in str das Resultat "Donnerstag Aug 28 240".

DATEIEN

/usr/lib/locale/*locale*/LC_TIME	standortabhängige Datums- und Zeitinformationen

SIEHE AUCH

ctime(3C), getenv(3C), setlocale(3C), strftime(4), Zeitzone(4), environ(5).

HINWEIS

cftime und ascftime sind veraltet. Statt dessen sollte strftime verwendet werden.

string - Zeichenketten bearbeiten

```
#include <string.h>
char *strcat (char *s1, const char *s2);
char *strdup (const char *s1);
char *strncat (char *s1, const char *s2, size_t n);
int strcmp (const char *s1, const char *s2);
int strncmp (const char *s1, const char *s2, size_t n);
char *strcpy (char *s1, const char *s2);
char *strncpy (char *s1, const char *s2, size_t n);
size_t strlen (const char *s);
char *strchr (const char *s, int c);
char *strrchr (const char *s, int c);
char *strpbrk (const char *s1, const char *s2);
size_t strspn (const char *s1, const char *s2);
size_t strcspn (const char *s1, const char *s2);
char *strtok (char *s1, const char *s2);
char *strstr (const char *s1, const char *s2);
```

Die Argumente *s*, *s1*, und *s2* zeigen auf Zeichenketten. Die Funktionen `strcat`, `strncat`, `strcpy`, `strncpy` und `strtok` verändern die Zeichenkette *s1*. Diese Funktionen prüfen nicht, ob das Feld, auf das *s1* zeigt, über seine Grenzen hinaus beschrieben wird.

`strcat` hängt eine Kopie der Zeichenkette *s2*, einschließlich des abschließenden Nullzeichens, am Ende der Zeichenkette *s1* an. `strncat` hängt maximal *n* Zeichen an. Die Funktionen liefern einen Zeiger auf das mit einem Nullzeichen abgeschlossene Ergebnis. Das erste Zeichen von *s2* überschreibt das Nullzeichen am Ende von *s1*.

`strcmp` vergleicht die Argumente und liefert eine ganze Zahl zurück, die kleiner, gleich oder größer als 0 ist, abhängig davon, ob *s1* lexikografisch kleiner, gleich oder größer als *s2* ist. `strncmp` führt denselben Vergleich aus, beschränkt sich dabei jedoch auf maximal *n* Zeichen. Zeichen, die nach einem Nullzeichen stehen, werden nicht verglichen.

`strcpy` kopiert die Zeichenkette *s2* nach *s1*, einschließlich des abschließenden Nullzeichens; nachdem das Nullzeichen kopiert wurde, stoppt der Kopiervorgang. `strncpy` kopiert genau *n* Zeichen, wobei Teile von *s2* abgeschnitten werden, oder, falls notwendig, Nullzeichen zu *s1* hinzugefügt werden. Das Ergebnis ist nicht mit einem Nullzeichen abgeschlossen, wenn die Länge von *s2* größer oder gleich *n* Zeichen ist. Jede Funktion liefert *s1* zurück.

`strdup` liefert einen Zeiger auf eine neue Zeichenkette zurück, welche eine Kopie der Zeichenkette ist, auf die *s1* zeigt. Der Speicherplatz für die neue Zeichenkette wird mit `malloc`(3C) zugewiesen. Wenn die neue Zeichenkette nicht angelegt werden kann, wird ein Nullzeiger zurückgegeben.

`strlen` liefert die Anzahl der Zeichen in *s*, ausschließlich des abschließenden Nullzeichens.

`strchr` (oder `strrchr`) liefert einen Zeiger auf das erste (letzte) Auftreten von *c* (umgewandelt in `char`) in der Zeichenkette *s*; tritt *c* nicht auf, so wird ein Nullzeiger zurückgegeben. Das Nullzeichen, welches eine Zeichenkette abschließt, wird als Teil der Zeichenkette betrachtet.

`strpbrk` liefert einen Zeiger auf das erste Auftreten eines Zeichens aus *s2* in der Zeichenkette *s1*; ist in *s1* kein Zeichen aus *s2* enthalten, so wird ein Nullzeiger zurückgegeben.

`strspn` liefert die Länge der ersten Teilzeichenkette aus der Zeichenkette *s1*, die nur aus Zeichen aus der Zeichenkette *s2* besteht.

`strcspn` liefert die Länge der ersten Teilzeichenkette aus der Zeichenkette *s1*, die nur aus Zeichen besteht, die nicht in *s2* vorkommen.

`strtok` verarbeitet eine Zeichenkette *s1*, die aus einer Folge von null oder mehr Texttokens besteht, die durch eine oder mehrere Zeichen aus der Trennzeichenkette *s2* getrennt sind. Der erste Aufruf, bei dem die Zeichenkette *s1* angegeben wird, liefert einen Zeiger auf das erste Zeichen des ersten Tokens zurück und schreibt ein Nullzeichen in die Zeichenkette *s1* sofort hinter das zurückgelieferte Token. Die Funktion verwaltet die Position in der Zeichenkette zwischen mehreren Aufrufen, so daß spätere Aufrufe (die als erstes Argument einen Nullzeiger enthalten) die Zeichenkette *s1* sofort nach dem Token bearbeiten. Auf diese Weise kann die komplette Zeichenkette *s1* verarbeitet werden, bis kein Token mehr verfügbar ist. Die Trennzeichenkette *s2* kann von Aufruf zu Aufruf variieren. Ist kein Token mehr in *s1* vorhanden, wird ein Nullzeiger zurückgeliefert.

`strstr` findet das erste Auftreten der Zeichenkette *s2* innerhalb der Zeichenkette *s1* (ausschließlich des abschließenden Nullzeichens). `strstr` liefert einen Zeiger auf die gefundene Zeichenkette oder einen Nullzeiger zurück, abhängig davon, ob die Zeichenkette gefunden wurde oder nicht. Wenn *s2* auf eine Zeichenkette der Länge 0 zeigt (d.h. die leere Zeichenkette), liefert die Funktion *s1* zurück.

SIEHE AUCH

`malloc`(3C), `setlocale`(3C), `strxfrm`(3C).

HINWEIS

Jede der Funktionen nimmt die voreingestellte lokale C-Umgebung an. Bei einigen lokalen Umgebungen sollte vor der Übergabe der Zeichenketten an die Funktionen die Funktion `strxfrm` auf die Zeichenketten angewendet werden.

strtod, atof - Zeichenkette in Zahl doppelter Genauigkeit umwandeln

```
#include <stdlib.h>
double strtod (const char *nptr, char **endptr);
double atof (const char *nptr);
```

`strtod` liefert den Wert der Zeichenkette *nptr* als Gleitkommazahl mit doppelter Genauigkeit zurück (`double`). Die Zeichenkette wird bis zum ersten nichtidentifizierbaren Zeichen verarbeitet.

`strtod` erkennt optionale Leerzeichen (wie durch `isspace` in `ctype`(3C) definiert), dann ein optionales Vorzeichen, dann eine Zeichenkette mit Ziffern, welche einen Dezimalpunkt enthalten darf, und schließlich einen Exponententeil, der `e` oder `E`, gefolgt von einem optionalen Vorzeichen und einer ganzen Zahl, enthält.

Ist der Wert von *endptr* nicht `(char **)NULL`, so wird in *endptr* ein Zeiger auf das Zeichen zurückgeliefert, welches die Bearbeitung beendet hat. Können keine Zahlen erzeugt werden, wird **endptr* auf *nptr* gesetzt und Null zurückgeliefert.

`atof(nptr)` ist äquivalent zu:

```
strtod(nptr, (char **)NULL).
```

SIEHE AUCH

`ctype`(3C), `scanf`(3S), `strtol`(3C).

ERGEBNIS

Wenn der korrekte Wert einen Überlauf verursachen würde, wird `±HUGE` zurückgeliefert (entsprechend des Vorzeichens), und `errno` wird auf `ERANGE` gesetzt.

Würde der korrekte Wert einen Unterlauf verursachen, wird Null zurückgeliefert und `errno` auf `ERANGE` gesetzt.

Im ANSI-Modus (`cc -kansi`), wird `HUGE_VAL` anstelle von `HUGE` zurückgegeben.

strtol - Zeichenkette in ganze Zahl umwandeln

```
#include <stdlib.h>
long strtol (const char *str, char **ptr, int base);
unsigned long strtoul (const char *str, char **ptr, int base);
long atol (const char *str);
int atoi (const char *str);
```

`strtol` liefert den Wert der Zeichenkette `str` als `long int` zurück. Die Zeichenkette wird bis zum ersten Zeichen verarbeitet, welches inkonsistent mit der Basis ist. Vorangestellte Leerzeichen (wie durch `isspace` in `ctype`(3C) definiert) werden ignoriert.

Wenn der Wert von *ptr* nicht `(char **)NULL` ist, wird in *ptr* ein Zeiger auf das Zeichen gesetzt, welches die Verarbeitung abgebrochen hat. Kann keine ganze Zahl gebildet werden, wird *ptr* auf `str` gesetzt und Null zurückgegeben.

Ist *base* positiv (und nicht größer als 36), wird dieser Wert als Basis für die Umwandlung verwendet. Nach einem optionalen Vorzeichen werden führende Nullen ignoriert; ist *base* gleich 16, so werden die Zeichen "0x" oder "0X" ebenfalls ignoriert.

Wenn *base* gleich Null ist, bestimmt die Zeichenkette selbst die Basis; dies geschieht folgendermaßen: Nach einem optionalen Vorzeichen zeigt eine Null an, daß eine oktale Umwandlung durchgeführt werden soll; wird die Zeichenkette "0x" oder "0X" erkannt, wird eine hexadezimale Umwandlung durchgeführt; ansonsten wird eine dezimale Umwandlung ausgeführt.

Die Umwandlung von `long` nach `int` kann durch Zuweisung oder durch eine explizite Typkonvertierung stattfinden.

Würde der durch *str* dargestellte Wert einen Überlauf verursachen, wird `LONG_MAX` oder `LONG_MIN` zurückgeliefert (abhängig vom Vorzeichen), und `errno` wird auf `ERANGE` gesetzt.

`strtoul` ähnelt `strtol` mit der Ausnahme, daß `strtoul` eine vorzeichenlose Zahl vom Typ `long int` für *str* zurückgibt. Würde der Wert aus *str* einen Überlauf verursachen, wird `ULONG_MAX` zurückgeliefert und `errno` auf den Wert `ERANGE` gesetzt.

Abgesehen vom Verhalten beim Auftreten von Fehlern ist `atol(str)` äquivalent mit: `strtol(str, (char **)NULL, 10)`.

Abgesehen vom Verhalten beim Auftreten von Fehlern ist `atoi(str)` äquivalent mit: `(int) strtol(str, (char **)NULL, 10)`.

ERGEBNIS

Wenn `strtol` mit dem Argument *base* größer als 36 aufgerufen wird, wird 0 zurückgegeben und `errno` auf `EINVAL` gesetzt.

SIEHE AUCH

`ctype`(3C), `scanf`(3S), `strtod`(3C).

HINWEIS

`strtol` akzeptiert als gültige Eingaben keine Werte mehr, die größer als `LONG_MAX` sind. In diesem Fall ist `strtoul` zu verwenden.

strxfrm - Transformation von Zeichenketten durchführen

```
#include <string.h>
size_t strxfrm (char *s1, const char *s2, size_t n);
```

`strxfrm` transformiert die Zeichenkette *s2* und schreibt die resultierende Zeichenkette in das Feld *s1*. Die Transformation läuft so ab, daß bei Anwendung von `strcmp` auf zwei transformierte Zeichenketten dasselbe Resultat zurückgegeben wird, wie wenn `strcoll` auf die beiden originalen Zeichenketten angewendet würde. Die Transformation hängt von der lokalen Einstellung der Kategorie `LC_COLLATE` des Programms ab (siehe `setlocale`(3C)).

Nicht mehr als *n* Zeichen (einschließlich des abschließenden Nullzeichens) werden in das Feld geschrieben, auf das *s1* zeigt. Wenn *n* gleich 0 ist, dann darf *s1* ein Nullzeiger sein. Wenn zwei sich überlappende Objekte kopiert werden, ist das Verhalten undefiniert.

`strxfrm` liefert die Länge der transformierten Zeichenkette (ausschließlich des abschließenden Nullzeichens) zurück. Ist der zurückgelieferte Wert gleich *n* oder größer, ist der Inhalt des Felds *s1* undefiniert.

BEISPIEL

Der Wert des folgenden Ausdrucks ist die Feldlänge, welche benötigt wird, um die Transformation der Zeichenkette *s* aufzunehmen:

```
1 + strxfrm(NULL, s, 0);
```

DATEIEN

`/usr/lib/locale/`*locale*`/LC_COLLATE` Datenbasis für *locale*.

SIEHE AUCH

`setlocale`(3C), `strcoll`(3C), `string`(3C), `environ`(5).
`colltbl`(1M) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Bei einem Fehler liefert `strxfrm` den Wert `(size_t) -1`.

swab - Bytes austauschen

```
#include <stdlib.h>
void swab (const char *von, char *nach, int Anzahl);
```

swab kopiert *Anzahl* Byte, auf die *von* zeigt, in das Feld, auf das *nach* zeigt, wobei benachbarte gerade und ungerade Bytes ausgewechselt werden. *Anzahl* muß eine gerade nichtnegative Zahl sein. Wenn *Anzahl* eine ungerade und positive Zahl ist, verwendet swab statt dessen *Anzahl* -1. Wenn *Anzahl* negativ ist, führt swab nichts aus.

sysconf - konfigurierbare Systemvariablen lesen

```
#include <unistd.h>
long sysconf(int name);
```

Mit `sysconf` können die aktuellen Werte eines konfigurierbaren Systemlimits oder einer Option (Variablen) bestimmt werden.

Das Argument *name* stellt die Systemvariable dar, die abgefragt werden soll. Die folgende Tabelle stellt die Menge der Systemvariablen aus den Dateien `limits.h` und `unistd.h` dar, welche durch `sysconf` abgefragt werden können; die symbolischen Konstanten aus der Datei `unistd.h` stellen die entsprechenden Werte dar, welche für *name* verwendet werden:

NAME	RÜCKGABEWERT
_SC_ARG_MAX	ARG_MAX
_SC_CHILD_MAX	CHILD_MAX
_SC_CLK_TCK	CLK_TCK
_SC_NGROUPS_MAX	NGROUPS_MAX
_SC_OPEN_MAX	OPEN_MAX
_SC_PASS_MAX	PASS_MAX
_SC_PAGESIZE	PAGESIZE
_SC_JOB_CONTROL	_POSIX_JOB_CONTROL
_SC_SAVED_IDS	_POSIX_SAVED_IDS
_SC_VERSION	_POSIX_VERSION
_SC_XOPEN_VERSION	_XOPEN_VERSION
_SC_LOGNAME_MAX	LOGNAME_MAX

Der Wert von `CLK_TCK` kann variabel sein; es sollte nicht angenommen werden, daß `CLK_TCK` eine Konstante zur Übersetzungszeit ist. Der Wert von `CLK_TCK` entspricht dem Wert von `sysconf(_SC_CLK_TCK)`.

SIEHE AUCH

`fpathconf`(3C).

ERGEBNIS

Ist *name* ein ungültiger Wert, liefert `sysconf` den Wert -1 zurück und setzt `errno`, um den Fehler anzuzeigen. Schlägt `sysconf` fehl, weil der Wert von *name* auf dem System nicht definiert ist, wird der Wert -1 zurückgegeben, ohne daß `errno` geändert wird.

HINWEIS

Ein Aufruf von `setrlimit` kann den Wert von `OPEN_MAX` ändern.

system - Shell-Kommando absetzen

```
#include <stdlib.h>
int system (const char *string);
```

system bewirkt, daß die Zeichenkette *string* als Eingabe an die Shell (siehe sh(1)) übergegeben wird, als ob die Zeichenkette als Kommando an einem Terminal eingegeben worden wäre. Der aktuelle Prozeß wartet, bis das Shell-Kommando ausgeführt worden ist und gibt dann den Endestatus der Shell in dem durch waitpid angegebenen Format zurück.

Wenn *string* ein Nullzeiger ist, überprüft system, ob /sbin/sh vorhanden und ausführbar ist. Wenn /sbin/sh verfügbar ist, gibt system einen Wert ungleich null zurück; sonst gibt es null zurück.

system scheitert, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

EAGAIN — Die systembedingte Obergrenze der Gesamtzahl von Prozessen, die unter einem einzigen Benutzer ausgeführt werden, würde überschritten.

EINTR — system wurde durch ein Signal unterbrochen.

ENOMEM — Der neue Prozeß verlangt mehr Speicherplatz als vom systembedingten Maximum MAXMEM erlaubt wird.

SIEHE AUCH

exec(2), wait(3C).
sh(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

system erzeugt einen neuen Sohnprozeß, der seinerseits /sbin/sh durch exec aktiviert, um *string* auszuführen. Scheitert fork oder exec, so gibt system einen negativen Wert zurück und setzt errno.

tcsetpgrp - Vordergrund-Prozeßgruppennummer setzen

```
#include <unistd.h>
int tcsetpgrp (int fildes, pid_t pgid)
```

`tcsetpgrp` stellt die Vordergrund-Prozeßgruppennummer des Terminals *fildes* auf *pgid*. Die Datei, die mit *fildes* verknüpft ist, muß dabei das steuernde Terminal des aufrufenden Prozesses darstellen, und das steuernde Terminal muß mit der Sitzung des aufrufenden Prozesses verknüpft sein. Der Wert von *pgid* muß der Prozeßgruppennummer eines Prozesses entsprechen, der sich in derselben Sitzung wie der aufrufende Prozeß befindet.

`tcsetpgrp` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EBADF` Das Argument *fildes* ist kein gültiger Dateideskriptor.

`EINVAL` Das Argument *fildes* bezeichnet ein Terminal, welches `tcsetpgrp` nicht unterstützt, oder *pgid* ist keine gültige Prozeßgruppennummer.

`ENOTTY` Der aufrufende Prozeß besitzt kein steuerndes Terminal, oder die Datei ist nicht das steuernde Terminal, oder das steuernde Terminal ist nicht länger mit der Sitzung des aufrufenden Prozesses verknüpft.

`EPERM` *pgid* entspricht nicht der Prozeßgruppennummer eines existierenden Prozesses aus derselben Sitzung wie der aufrufende Prozeß.

SIEHE AUCH

`tcsetpgrp`(3C), `tcsetsid`(3C).
`termio`(7) in "Referenzhandbuch für Systemverwalter".

ERGEBNIS

Nach erfolgreicher Ausführung liefert `tcsetpgrp` den Wert 0 zurück. Ansonsten wird -1 zurückgegeben und `errno` gesetzt, um den Fehler anzuzeigen.

tmpfile - temporäre Datei erstellen

```
#include <stdio.h>
FILE *tmpfile (void);
```

`tmpfile` erstellt eine temporäre Datei unter Verwendung eines Namens, der von der `tmpnam`(3S)-Programmroutine erstellt wird, und gibt einen entsprechenden Dateizeiger `FILE` zurück. Wenn die Datei nicht geöffnet werden kann, wird eine Fehlermeldung unter Verwendung von `perror` ausgegeben und ein Nullzeiger zurückgegeben. Die Datei wird automatisch gelöscht, sobald der Prozeß, der sie verwendet, beendet wird, oder wenn die Datei geschlossen wird. Die Datei wird für Änderungen geöffnet ("w+").

SIEHE AUCH

`creat`(2), `open`(2) `unlink`(2), `fopen`(3S), `mktemp`(3C), `perror`(3C), `stdio`(3S), `tmpnam`(3S).

tmpnam, tempnam - Name für temporäre Datei erstellen

```
#include <stdio.h>
char *tmpnam (char *s);
char *tempnam (const char *dir, const char *pfx);
```

Mit diesen Funktionen werden Dateinamen erstellt, die für eine temporäre Datei gültig sind.

`tmpnam` generiert immer einen Dateinamen unter Verwendung des Pfad-Präfixes, der als `P_tmpdir` in der Include-Datei `stdio.h` definiert ist. Wenn *s* `NULL` ist, schreibt `tmpnam` das Resultat in einen internen statischen Bereich und gibt einen Zeiger auf diesen Bereich zurück. Mit dem nächsten Aufruf von `tmpnam` wird der Inhalt dieses Bereichs zerstört. Wenn *s* nicht `NULL` ist, wird davon ausgegangen, daß es die Adresse eines Feldes von wenigstens `L_tmpnam` Bytes ist, wobei `L_tmpnam` eine in `stdio.h` definierte Konstante ist; `tmpnam` setzt das Ergebnis in dieses Feld und gibt *s* zurück.

`tempnam` ermöglicht die Steuerung der Dateiverzeichniswahl. Das Argument *dir* zeigt auf den Namen des Dateiverzeichnisses, in dem die Datei erstellt werden soll. Wenn *dir* `NULL` ist oder auf eine Zeichenkette weist, die nicht der Name für ein entsprechendes Dateiverzeichnis ist, wird das als `P_tmpdir` in der Include-Datei `stdio.h` definierte Pfad-Präfix verwendet. Kann auf dieses Dateiverzeichnis nicht zugegriffen werden, wird `/tmp` als letzte Möglichkeit verwendet. Dieser gesamte Ablauf kann durch Bereitstellen einer Umgebungsvariablen `TMPDIR` verbessert werden. Der Wert von `TMPDIR` ist der Name des gewünschten Dateiverzeichnisses der temporären Datei.

Bei vielen Anwendungen ist es vorteilhaft, wenn die temporären Dateien bestimmte bevorzugte Anfangsbuchstaben in ihren Namen aufweisen. Hierfür verwendet man das Argument *pfx*. Dieses Argument kann `NULL` sein oder auf eine Zeichenkette von maximal fünf Zeichen zeigen, die als die ersten Zeichen des Namens der temporären Datei eingesetzt werden.

`tempnam` verwendet `malloc`, um Speicherplatz für den erzeugten Dateinamen zu erhalten und gibt einen Zeiger auf diesen Bereich zurück. Daher kann jeder von `tempnam` zurückgegebene Zeigerwert als Argument für `free` dienen (siehe `malloc`(3C)). Wenn `tempnam` aus irgendeinem Grunde das erwartete Ergebnis nicht liefern kann, d.h. wenn `malloc` erfolglos war oder kein geeignetes Dateiverzeichnis gefunden wurde, wird ein Nullzeiger zurückgegeben.

`tempnam` ist erfolglos, wenn nicht genug Speicher vorhanden ist.

SIEHE AUCH

`creat`(2), `unlink`(2), `fopen`(3S), `malloc`(3C), `mktemp`(3C), `tmpfile`(3S).

HINWEIS

Diese Funktionen generieren bei jedem Aufruf einen anderen Dateinamen.

Dateien, die unter Verwendung dieser Funktionen und entweder von `fopen` oder `creat` erstellt wurden, sind nur insofern temporär, als sie sich in einem Dateiverzeichnis befinden, das für temporären Gebrauch bestimmt ist, und ihre Namen eindeutig sind. Es liegt in der Verantwortung des Benutzers, die Datei zu löschen, wenn diese nicht mehr gebraucht wird. Wenn diese Funktionen mehr als `TMP_MAX` (definiert in `stdio.h`) mal in einem einzigen Prozeß aufgerufen werden, werden vorher benutzte Namen wieder verwendet.

Es ist möglich, daß während des Zeitraums von der Erstellung eines Dateinamens bis zum Öffnen der Datei ein anderer Prozeß eine Datei mit dem gleichen Namen erstellt. Dies kann jedoch nicht eintreten, wenn der andere Prozeß diese Funktionen oder `mktemp` verwendet und die Dateinamen so gewählt werden, daß ihre Duplizierung auf andere Weise unwahrscheinlich ist.

truncate, ftruncate - Datei auf angegebene Länge setzen

```
#include <unistd.h>
int truncate (const char *path, off_t length);
int ftruncate (int fildes, off_t length);
```

Die Länge der Datei mit dem Namen *path* oder dem Deskriptor *fildes* wird auf eine Länge von *length* Bytes gesetzt.

Wenn die Datei vorher länger als *length* Bytes war, kann auf die Bytes hinter der Position *length* nicht länger zugegriffen werden. War die Datei vorher kürzer, so werden die Bytes zwischen der Dateiende-Marke vor dem Aufruf und der Dateiende-Marke nach dem Aufruf mit Nullen gefüllt. Die effektive Benutzernummer des Prozesses muß das Schreibrecht für die Datei besitzen; bei `ftruncate` muß die Datei zum Schreiben geöffnet sein.

`truncate` schlägt fehl, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

`EACCES` Für eine Komponente des Pfadpräfixes existiert keine Sucherlaubnis.

`EACCES` Für die Datei *path* existiert keine Schreiberlaubnis.

`EFAULT` *path* zeigt außerhalb des zugewiesenen Adreßbereichs des Prozesses.

`EINTR` Während der Ausführung von `truncate` wurde ein Signal empfangen.

`EINVAL` *path* ist keine gewöhnliche Datei.

`EIO` Beim Lesen oder Schreiben des Dateisystems trat ein E/A-Fehler auf.

`EISDIR` Die Datei *path* ist ein Verzeichnis.

`ELOOP` Beim Übersetzen von *path* traten zu viele symbolische Verweise auf.

`EMFILE` Die maximale Anzahl der erlaubten Dateideskriptoren des Prozesses wurde erreicht.

`EMULTIHOP` Teile von *path* erfordern den Sprung auf mehrere ferne Rechner, und der Dateisystemtyp erlaubt dies nicht.

`ENAMETOOLONG` Die Länge einer Komponente aus *path* überschreitet `NAME_MAX` Zeichen, oder die Länge von *path* überschreitet `PATH_MAX` Zeichen.

`ENFILE` Es konnte nicht mehr Speicherplatz für die Dateitabelle des Systems zugewiesen werden.

`ENOENT` Entweder existiert eine Komponente des Pfadpräfixes nicht oder die Datei *path* existiert nicht.

ENOLINK *path* zeigt auf einen fernen Rechner, und der Verweis auf diesen Rechner ist nicht länger aktiv.

ENOTDIR Eine Komponente des Pfadpräfixes aus *path* ist kein Verzeichnis.

EROFS Die Datei *path* befindet sich in einem Dateisystem, das nur lesbar ist.

ETXTBSY Die Datei *path* ist eine mehrfach benutzbare Prozedur-Textdatei, die ausgeführt wird.

`ftruncate` schlägt fehlt, wenn wenigstens eine der folgenden Bedingungen erfüllt ist:

EAGAIN Die Datei existiert, logische Datei-/Satzsperren sind verhängt, und es existieren noch ausstehende Satzsperren für die Datei (siehe `chmod`(2)).

EBADF *fildes* ist kein Dateideskriptor, der zum Schreiben geöffnet ist.

EINTR Während der Ausführung von `ftruncate` wurde ein Signal empfangen.

EIO Beim Lesen oder Schreiben des Dateisystems trat ein E/A-Fehler auf.

ENOLINK *fildes* zeigt auf einen fernen Rechner, und der Verweis auf diesen Rechner ist nicht mehr aktiv.

EINVAL *fildes* entspricht keiner gewöhnlichen Datei.

SIEHE AUCH

`fcntl`(2), `open`(2).

ERGEBNIS

Nach erfolgreicher Ausführung wird 0 zurückgegeben. Ansonsten wird -1 zurückgeliefert und `errno` gesetzt, um den Fehler anzuzeigen.

tsearch, tfind, tdelete, twalk - binäre Suchbäume verwalten

```
#include <search.h>

void *tsearch (const void *key, void **rootp, int (*compar)
    (const void *, const void *));

void *tfind (const void *key, void * const *rootp, int (*compar)
    (const void *, const void *));

void *tdelete (const void *key, void **rootp, int (*compar)
    (const void *, const void *));

void twalk (void *root, void(*action) (void *, VISIT, int));
```

`tsearch`, `tfind`, `tdelete` und `twalk` sind Funktionen für die Verwaltung von binären Suchbäumen. Sie sind Verallgemeinerungen der Algorithmen T und D von Knuth (6.2.2). Alle Vergleiche werden durch eine vom Benutzer gelieferte Funktion ausgeführt. Diese Funktion wird mit zwei Argumenten aufgerufen, d.h. mit den Zeigern auf die Elemente, die verglichen werden. Sie gibt eine ganze Zahl zurück, die kleiner, gleich oder größer als 0 ist, je nachdem, ob das erste Argument kleiner, gleich oder größer als das zweite Argument ist. Die Vergleichsfunktion braucht nicht jedes Byte zu vergleichen, und daher können außer den Werten, die verglichen werden, auch willkürliche Daten in den Elementen enthalten sein.

`tsearch` wird zum Aufbau des Baums und für den Zugriff auf den Baum verwendet. *key* ist ein Zeiger auf einen Wert, auf den zugegriffen bzw. der gespeichert werden soll. Wenn der Baum einen Wert aufweist, der gleich **key* (der Wert, auf den der Schlüssel zeigt) ist, wird ein Zeiger auf diesen gefundenen Wert zurückgegeben. Andernfalls wird **key* eingefügt und ein auf diesen *key* weisender Zeiger zurückgegeben. Es werden nur Zeiger kopiert, und daher müssen die Daten von der Aufrufroutine gespeichert werden. *rootp* zeigt auf eine Variable, die auf die Wurzel des Baums zeigt. Ein `NULL`-Wert für die Variable, auf die *rootp* zeigt, gibt einen leeren Baum an; in diesem Fall wird die Variable so gesetzt, daß sie auf den Wert zeigt, der sich an der Wurzel des neuen Baums befindet.

Wie `tsearch` sucht auch `tfind` nach einem Wert im Baum und gibt einen Zeiger auf diesen Wert zurück, falls dieser gefunden wird. Wird der Wert nicht gefunden, gibt `tfind` einen Nullzeiger zurück. Die Argumente für `tfind` sind dieselben wie für `tsearch`.

Mit `tdelete` wird ein Knoten in einem binären Suchbaum gelöscht. Die Argumente sind dieselben wie für `tsearch`. Die Variable, auf die *rootp* zeigt, ändert sich, wenn der gelöschte Knoten die Wurzel des Baums war. `tdelete` gibt einen Zeiger auf den Vaterknoten des gelöschten Knotens zurück oder einen Nullzeiger, wenn der Knoten nicht gefunden wurde.

`twalk` durchläuft einen binären Suchbaum. *root* ist die Wurzel des Baums, der durchgelaufen werden soll. Jeder Knotenpunkt im Baum kann als Wurzel für ein Durchlaufen

des Baums unterhalb dieses Knotens verwendet werden. *action* ist der Name einer Funktion, die an jedem Knoten aufgerufen werden soll. Diese Funktion wird mit drei Argumenten aufgerufen. Das erste Argument ist die Adresse des besuchten Knotens. Das zweite Argument ist ein Wert des Aufzählungstyps *typedef enum { preorder, postorder, endorder, leaf } VISIT*; (definiert in der Include-Datei `search.h`), abhängig davon, ob es sich um den ersten, zweiten oder dritten Besuch des Knotens handelt, bei einem Durchlauf des Baums in die Tiefe, von links nach rechts, oder ob der Knoten ein Blatt ist. Das dritte Argument stellt die Stufe des Knotens im Baum dar, wobei die Wurzel die Stufe Null ist.

Die Zeiger auf den Schlüssel und die Wurzel des Baums sollen vom Typ 'Zeiger-auf-Element' sein und in den Typ 'Zeiger-auf-Zeichen' konvertiert werden. Genauso soll der als Typ 'Zeiger-auf-Zeichen' deklarierte zurückgegebene Wert in den Typ 'Zeiger-auf-Element' konvertiert werden.

BEISPIEL

Das folgende Programm liest Zeichenketten ein und speichert Strukturen ab, die einen Zeiger auf eine Zeichenkette und ihre Länge enthalten. Es durchläuft dann den Baum und gibt die gespeicherten Zeichenketten und ihre Längen in alphabetischer Reihenfolge aus.

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

struct node {
      char *string;
      int length;
};
char string_space[10000];
struct node nodes[500];
void *root = NULL;

int node_compare(const void *node1, const void *node2) {
      return strcmp(((const struct node *) node1)->string,
                    ((const struct node *) node2)->string);
}

void print_node(void **node, VISIT order, int level) {
      if (order == preorder || order == leaf) {
            printf("length=%d, string=%20s\n",
            (*(struct node **)node)->length,
            (*(struct node **)node)->string);
      }
}

main() {
      char *strptr = string_space;
      struct node *nodeptr = nodes;
      int i = 0;
      while (gets(strptr) != NULL && i++ < 500) {
            nodeptr->string = strptr;
            nodeptr->length = strlen(strptr);
            (void) tsearch((void *)nodeptr, &root, node_compare);
            strptr += nodeptr->length + 1;
            nodeptr++;
      }
      twalk(root, print_node);
}
```

SIEHE AUCH

`bsearch`(3C), `hsearch`(3C), `lsearch`(3C).

ERGEBNIS

Ein Nullzeiger wird von `tsearch` zurückgegeben, wenn nicht ausreichend Speicher zur Erstellung eines neuen Knotens zur Verfügung steht.

Ein Nullzeiger wird von `tfind` und `tdelete` zurückgegeben, wenn *rootp* zu Beginn `NULL` ist.

Wird der Wert gefunden, geben `tsearch` und `tfind` einen Zeiger auf den Wert zurück. Wird der Wert nicht gefunden, gibt `tfind` `NULL` zurück, und `tsearch` gibt einen Zeiger auf die eingefügte Position zurück.

HINWEIS

Das Argument `root` für `twalk` ist um eine Stufe der indirekten Adressierung niedriger als die Argumente *rootp* für `tsearch` und `tdelete`.

Es gibt zwei Nomenklaturen für die Reihenfolge, in der die Knoten eines Baums durchlaufen werden. `tsearch` verwendet die Begriffe "preorder", "postorder" und "endorder", um auszudrücken, daß ein Knoten vor seinen Söhnen oder nach dem linken und vor dem rechten Sohn oder nach seinen Söhnen besucht wird. Die andere Nomenklatur verwendet "preorder", "inorder" und "postorder", um diese Reihenfolgen zu bezeichnen, wobei "postorder" eine andere Bedeutung hat.

Wenn die aufrufende Funktion den Zeiger auf die Wurzel ändert, werden die Ergebnisse unvorhersagbar.

ttyname, isatty - Name eines Terminals abfragen

```
#include <stdlib.h>
char *ttyname (int fildes);
int isatty (int fildes);
```

`ttyname` gibt einen Zeiger auf eine Zeichenkette zurück, die den mit Null-Byte endenden Pfadnamen des Terminals enthält, das zum Dateideskriptor *fildes* gehört.

`isatty` gibt 1 zurück, wenn *fildes* mit einem Terminal verbunden ist; andernfalls wird Null zurückgegeben.

DATEIEN

`/dev/*`

ERGEBNIS

`ttyname` gibt einen Nullzeiger zurück, wenn *fildes* kein Terminal im Dateiverzeichnis `/dev` beschreibt.

HINWEIS

Der Rückgabewert weist auf statische Daten, die bei jedem Aufruf überschrieben werden.

ttyslot - Eintrag des aktuellen Benutzers in der utmp-Datei finden

```
#include <stdlib.h>
int ttyslot (void);
```

`ttyslot` gibt den Index des Eintrags des aktuellen Benutzers in der Datei `/var/adm/utmp` zurück. Der zurückgegebene Index wird durch Abfragen von Dateien in `/dev` auf den Namen des Terminals, der mit der Standardeingabe, der Standardausgabe oder der Fehlerausgabe (0, 1 oder 2) verbunden ist, bestimmt.

DATEIEN

`/var/adm/utmp`

SIEHE AUCH

`getut`(3C), `ttyname`(3C).

ERGEBNIS

Wenn bei der Suche nach dem Terminalnamen ein Fehler gefunden wurde, oder wenn keiner der obigen Dateideskriptoren einem Terminal zugeordnet wurde, wird der Wert -1 zurückgegeben.

ungetc - Zeichen in Eingabestrom zurückstellen

```
#include <stdio.h>
int ungetc (int c, FILE *stream);
```

ungetc schreibt das Zeichen *c* (umgeformt zu einem unsigned char) in den Puffer zurück, der der Datei *stream* zugeordnet ist (siehe intro(3)). Dieses Zeichen *c* wird dann vom nächsten getc(3S)-Aufruf für *stream* wieder zurückgegeben. ungetc gibt als Ergebnis *c* zurück; die zu *stream* gehörende Datei bleibt unverändert.

Ein erfolgreicher Aufruf von ungetc löscht die EOF-Anzeige des Streams.

Es wird Speicherplatz für vier Bytes zurückgestellte Zeichen garantiert.

Der Wert des Datei-Positionsanzeigers wird nach dem Lesen oder Ablegen aller rückgeführten Zeichen für *stream* der gleiche sein, wie zu dem Zeitpunkt, bevor die Zeichen zurückgeführt wurden.

Wenn *c* gleich EOF ist, schreibt ungetc nicht in den Puffer und gibt EOF zurück.

fseek, rewind (beide unter fseek(3S)) beschreiben und setpos löscht den Speicherplatz für zurückgestellte Zeichen auf den Streams, auf denen sie angewendet werden.

SIEHE AUCH

fseek(3S), setpos(3C), getc(3S), setbuf(3S), stdio(3S).

ERGEBNIS

ungetc gibt EOF zurück, wenn das Zeichen nicht zurückgestellt werden kann.

vprintf - variable Argumentenliste formatiert ausgeben

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

int vprintf(const char *format, va_list ap);

int vfprintf(FILE *stream, const char *format, va_list ap);

int vsprintf(char *s, const char *format, va_list ap);
```

`vprintf`, `vfprintf` und `vsprint` sind funktionsgleich mit `print`,`fprint` und `sprint`, werden aber, anstatt mit einer variablen Reihe von Argumenten mit einer in der Include-Datei `stdarg.h` definierten Argumentliste aufgerufen.

Die `stdarg.h` Include-Datei definiert die `va_list`-Typen und eine Reihe von Makros, um eine Liste von Parametern, deren Anzahl und Typen unterschiedlich sein können, durchgehen zu können. Der Parameter *ap* ist vom Typ `va_list`. Dieser Parameter wird zusammen mit den Makros `va_start`, `va_arg` und `va_end` aus der Include-Datei `stdarg.h` benutzt (siehe `stdarg`(5)).

BEISPIEL

Der folgende Programmausschnitt veranschaulicht die Verwendung von `vfprintf` zum Schreiben einer `error`-Routine:

```
#include <stdio.h>
#include <stdarg.h>
 /*     error sollte folgendermaßen aufgerufen werden:
  *     error(Funktionsname, Format, Arg1, ...);          */

void error(char *function_name, char *format, ...)

{
      va_list ap;

      va_start(ap,format);
      /* gib Name der Funktion aus, welche error aufruft */
      (void)fprintf(stderr, "FEHLER in %s: ", function_name);
      va_arg(ap, char*);
      /* gib Rest der Meldung aus */
      (void) vfprintf(stderr, format, ap);
      va_end(ap);
      (void) abort;
}
```

SIEHE AUCH

`printf`(3S), `stdarg`(5).

ERGEBNIS

`vprintf` und `vfprint` geben die Anzahl der übertragenen Zeichen zurück oder -1, falls ein Fehler aufgetreten war.

intro - Einführung in die ELF-Bibliothek

Mit den Funktionen der ELF-Bibliothek können in einem Programm Objektdateien, Archivdateien und Archivkomponenten, die sich im ELF-Format (Executable and Linking Format) befinden, manipuliert werden. Die Include-Datei `libelf.h` liefert die Deklaration der Typen und der Funktionen für alle Bibliotheksdienste.

Programme kommunizieren mit vielen höheren Funktionen über einen sogenannten *ELF-Deskriptor*. Sobald das Programm beginnt, mit einer Datei zu arbeiten, erzeugt die Funktion `elf_begin` einen ELF-Deskriptor, über den das Programm die Strukturen und Informationen aus der Datei manipulieren kann. Diese ELF-Deskriptoren können sowohl zum Lesen als auch zum Schreiben von Dateien verwendet werden. Nachdem ein Programm für eine Datei einen ELF-Deskriptor angelegt hat, können zur Manipulation der einzelnen Dateiabschnitte *Abschnittsdeskriptoren* angelegt werden (siehe `elf_getscn`(3E)). Abschnitte enthalten den Großteil der relevanten Informationen einer Objektdatei, wie zum Beispiel Text, Daten, Symboltabelle usw. Ein Abschnittsdeskriptor 'gehört' genauso zu einem bestimmten ELF-Deskriptor, wie ein Abschnitt zu einer Datei gehört. Die *Datendeskriptoren* sind wiederum über Abschnittsdeskriptoren verfügbar; über sie kann das Programm die Informationen aus einem Abschnitt manipulieren. Ein Datendeskriptor 'gehört' zu einem Abschnittsdeskriptor.

Deskriptoren sind eigene, private Zugriffsstrukturen, mit denen ein Programm auf eine Datei und deren Inhalte zugreifen kann. Ein Datendeskriptor ist also mit einem Abschnittsdeskriptor verknüpft, der wiederum mit einem ELF-Deskriptor verbunden ist; der ELF-Deskriptor wiederum ist mit einer Datei verknüpft. Obwohl Deskriptoren nur für das Programm selbst gültig sind, ermöglichen sie den Zugriff auch auf mehrfach benutzbare Daten. Ein Programm, das beispielsweise verschiedene Eingabedateien kombiniert und andere Dateien erzeugt oder aktualisiert, kann Datendeskriptoren für einen Eingabeabschnitt und einen Ausgabeabschnitt anlegen. Mit der Aktualisierung des Ausgabedeskriptors sind die Daten bereits über den Eingabedeskriptor zu verwenden. Auf diese Weise werden zwar verschiedene Deskriptoren verwendet, auf die entsprechenden Daten wird aber gemeinsam zugegriffen. Dieser gemeinsame Zugriff vermeidet, daß zuviel Speicherplatz für getrennte Puffer angelegt wird; ferner steigt der Durchsatz, da das Kopieren von Daten wegfällt.

Dateiklassen

ELF bietet einen Rahmen zur Definition von Objektdateien, wobei verschiedene Prozessoren und Architekturen unterstützt werden. Ein wichtiges Unterscheidungskriterium zwischen Objektdateien ist die *Klasse* bzw. die Kapazität der Datei. Die 32-Bit-Klasse unterstützt Architekturen, in denen ein 32-Bit-Objekt Adressen, Dateilängen etc. wie folgt darstellen kann:

Name	Zweck
Elf32_Addr	Programmadresse (unsigned)
Elf32_Half	mittelgroße Ganzzahl (unsigned)
Elf32_Off	Datei-Offset (unsigned)
Elf32_Sword	große Ganzzahl mit Vorzeichen
Elf32_Word	große Ganzzahl (unsigned)
unsigned char	kleine Ganzzahl (unsigned)

Andere Klassen werden bei Bedarf definiert, um größere (oder kleinere) Maschinen zu unterstützen. Einige Bibliotheksdienste arbeiten nur mit Datenobjekten von bestimmten Klassen, während andere klassenunabhängig sind. Um diesen Unterschied hervorzuheben, sind die Namen der Bibliotheksfunktionen so gewählt, daß ihr Status erkennbar ist. Dies wird unten näher beschrieben.

Datendarstellung

Zwei parallele Objektmengen realisieren typischerweise eine Umgebung zum Cross-Compilieren. Eine Menge entspricht dem Dateiinhalt, während die andere Menge dem tatsächlichen Speicherabbild des Programms entspricht, welches die Datei manipuliert. Die Typdefinitionen, die in der Include-Datei angegeben sind, arbeiten auf der realen Maschine, welche eine andere Datencodierung als die Zielmaschine (Längen, Wertigkeit der Bytes, etc.) besitzen kann. Die Speicherobjekte der tatsächlichen Maschine sollten mindestens so groß sein wie die Dateiobjekte (um Informationsverlusten vorzubeugen), dürfen aber auch größer sein, wenn dies auf der Host-Maschine naheliegend ist.

Zur Konvertierung zwischen Datei- und Speicherdarstellungen existieren Umsetzungswerkzeuge. Einige Bibliotheksroutinen konvertieren Daten automatisch, während andere die Konvertierung dem Programm überlassen. In jedem Fall müssen Programme, welche Objektdateien erzeugen, Objekte vom Dateityp in diese Dateien schreiben; Programme, welche Objektdateien lesen, gehen entsprechend vor. Siehe `elf_xlate`(3E) und `elf_fsize`(3E) für genauere Informationen.

Programme können Daten explizit umsetzen, wobei Struktur und Semantik der Objektdatei kontrolliert werden. Wenn das Programm diese Kontrolle nicht ausüben möchte, bietet die Bibliothek eine Schnittstelle auf höherem Niveau an, welche viele Details der Objektdatei versteckt. `elf_begin` und andere Funktionen erlauben einem Programm das Bearbeiten von realen Speichertypen, wobei die Konvertierung zwischen Speicherobjekten und deren Dateiäquivalenten automatisch beim Lesen oder Schreiben einer Objektdatei ausgeführt wird.

Elf-Versionen

Objektdateiversionen erlauben die Anpassung von ELF an neue Anforderungen. Drei (unabhängige) Versionen können für ein Programm von Bedeutung sein. Erstens kennt ein Anwendungsprogramm eine bestimmte Version durch seine Übersetzung mit einer bestimmten Include-Datei. Zweitens wurde bei der Übersetzung der Zugriffsbibliothek eine bestimmte Include-Datei verwendet; sie kontrolliert die Version, die von der Bibliothek verstanden wird. Drittens enthält die ELF-Objektdatei einen Wert, der ihre Version bezeichnet; diese wird durch die ELF-Version bestimmt, die dem Erzeuger der Datei bekannt ist. Im Idealfall stimmen alle drei Versionen überein, jedoch dürfen sie auch voneinander abweichen.

- Ist eine Programmversion aktueller als die Zugriffsbibliothek, ist es möglich, daß das Programm Informationen verwendet, die der Bibliothek unbekannt sind. Übersetzungsroutinen können dann nicht entsprechend arbeiten, was zu einem undefinierten Verhalten führt. Dieser Zustand erfordert die Installation einer neuen Bibliothek.
- Die Bibliotheksversion kann aktueller als die des Programms und der Datei sein. Die Bibliothek verarbeitet alte Versionen; Kompatibilitätsprobleme werden in diesem Fall umgangen.
- Schließlich kann eine Dateiversion einen aktuelleren Wert aufweisen, den weder das Programm noch die Bibliothek versteht. Unter Umständen kann das Programm dann die Datei dennoch bearbeiten, abhängig davon, ob die Datei Zusatzinformationen enthält und ob diese Informationen zuverlässig ignoriert werden können. Die sichere Alternative besteht auch hier in der Installation einer neuen Bibliothek, welche die Version der Datei bearbeiten kann.

Um die Unterschiede auszugleichen, muß ein Programm die Funktion `elf_version` verwenden, um seine Version der Bibliothek mitzuteilen; dadurch wird die *Arbeitsversion* für den Prozeß angelegt. Durch diese Vorgehensweise akzeptiert die Bibliothek Daten vom Programm und kann Daten für das Programm in korrekter Darstellung angeben. Wenn die Bibliothek Objektdateien liest, wird zur Interpretation der Daten die Version der Datei verwendet. Beim Schreiben von Dateien oder beim Konvertieren von Speichertypen, die Dateiäquivalente darstellen, verwendet die Bibliothek für die Dateidaten die Arbeitsversion des Programms.

Systemdienste

Wie bereits erwähnt, bieten `elf_begin` und andere Routinen für ELF-Dateien eine Schnittstelle auf höherem Niveau; Ein- und Ausgabe werden für das Anwendungsprogramm ausgeführt. Diese Routinen gehen davon aus, daß ein Programm ganze Dateien im Speicher halten kann, ohne daß explizit temporäre Dateien verwendet werden. Beim Lesen einer Datei bringen die Bibliotheksroutinen die Daten in den Speicher und führen dann Operationen auf der Speicherkopie der Datei aus. Lesen oder schreiben Programme große Objektdateien nach diesem Ansatz, so muß dies auf einer Maschine

geschehen, die über einen entsprechend großen virtuellen Adreßraum für den Prozeß verfügt. Wenn die zugrundeliegende systembedingte maximale Anzahl der geöffneten Dateien überschritten wird, kann ein Programm `elf_cntl` verwenden, um alle notwendigen Daten aus der Datei zu lesen. Danach wird dem Programm erlaubt, den Dateideskriptor zu schließen und anschließend wieder zu benutzen.

Obwohl die `elf_begin`-Schnittstelle für die meisten Programme ausreichend und effizient ist, kann es auch Situationen geben, in denen sie nicht anwendbar ist. In diesen Fällen kann ein Anwendungsprogramm die Datenübersetzungsroutine `elf_xlate` direkt aufrufen. Diese Routinen führen keine Ein- oder Ausgabe durch; dies bleibt dem Anwendungsprogramm überlassen. Durch diese Übernahme eines größeren Aufgabenbereichs kontrolliert ein Anwendungsprogramm sein eigenes Ein-/Ausgabemodell.

Bibliotheksnamen

Die im Zusammenhang mit der Bibliothek verwendeten Namen haben folgende Formen:

`elf_`*name* Diese klassenunabhängigen Namen führen einen Dienst *name* für das Programm aus.

`elf32_`*name* Dienstnamen mit einer angegebenen Klasse, wie zum Beispiel `32`, zeigen an, daß sie nur mit der entsprechenden Klasse von Dateien arbeiten.

`Elf_`*Type* Datentypen können ebenfalls klassenunabhängig sein und werden durch *Type* unterschieden.

`Elf32_`*Type* Klassenabhängige Datentypen enthalten einen Klassennamen, wie zum Beispiel `32`.

`ELF_C_`*CMD* Einige Funktionen erhalten Kommandos, die ihre Aktionen kontrollieren. Diese Werte sind Elemente des Aufzählungstyps `Elf_Cmd`; sie reichen von Null bis `ELF_C_NUM-1`.

`ELF_F_`*FLAG* Einige Funktionen erhalten Optionen, die den Bibliotheksstatus und/oder die Aktionen kontrollieren. Die Optionen sind Bitmasken, die auch kombiniert werden können.

`ELF32_FSZ_`*TYPE*
Solche Konstanten stellen die Größe in Bytes in der Dateidarstellung der grundsätzlichen ELF-Typen für die 32-Bit-Dateiklasse dar. Siehe `elf_fsize` für weitere Informationen.

`ELF_K_`*KIND* Die Funktion `elf_kind` identifiziert *KIND* (den Typ) der Datei, die mit dem ELF-Deskriptor verknüpft ist. Diese Werte sind Werte des Aufzählungstyps `Elf_Kind`; sie reichen von Null bis `ELF_K_NUM-1`.

ELF_T_*TYPE* Wenn eine Dienstfunktion, wie zum Beispiel elf_xlate, mit mehreren Typen operiert, können Namen dieser Form den gewünschten Typ *TYPE* angeben. Beispielsweise wird ELF_T_EHDR direkt Elf32_Ehdr zugeordnet. Diese Werte sind Elemente des Aufzählungstyps Elf_Type; sie reichen von Null bis ELF_T_NUM-1.

SIEHE AUCH

cof2elf(1), elf_begin(3E), elf_cntl(3E), elf_end(3E), elf_error(3E), elf_fill(3E), elf_flag(3E), elf_fsize(3E), elf_getarhdr(3E), elf_getarsym(3E), elf_getbase(3E), elf_getdata(3E), elf_getehdr(3E), elf_getident(3E), elf_getphdr(3E), elf_getscn(3E), elf_getshdr(3E), elf_hash(3E), elf_kind(3E), elf_next(3E), elf_rand(3E), elf_rawfile(3E), elf_strptr(3E), elf_update(3E), elf_version(3E), elf_xlate(3E), a.out(4), ar(4)
Kapitel "Objektdateien" in "Leitfaden und Werkzeuge für die Programmierung mit C".

HINWEIS

Die Informationen in den ELF-Include-Dateien sind in allgemeine und prozessorabhängige Bereiche unterteilt. Ein Programm kann Informationen über den Prozessor durch das Einbinden der entsprechenden Include-Datei erhalten: <sys/elf_*NAME*.h>, wobei *NAME* dem Prozessornamen entspricht, wie er im ELF-Dateikopf verwendet wird.

Symbol	Prozessor
M32	AT&T WE 32100
SPARC	SPARC
386	Intel 80386
486	Intel 80486
860	Intel 80860
68K	Motorola 68000
88K	Motorola 88000

Andere Prozessoren werden bei Bedarf dieser Tabelle hinzugefügt. Beispielsweise kann ein Programm mit Hilfe der folgenden Programmzeilen prozessorabhängige Informationen für den Prozessor WE 32100 erhalten:

```
#include <libelf.h>
#include <sys/elf_M32.h>
```

Ohne das Einbinden von sys/elf_M32.h wären nur die allgemeinen ELF-Informationen für das Programm sichtbar.

elf_begin - Dateideskriptor erzeugen

```
#include <libelf.h>
Elf *elf_begin(int fildes, Elf_Cmd cmd, Elf *ref);
```

`elf_begin`, `elf_next`, `elf_rand`, und `elf_end` arbeiten zusammen. Sie bearbeiten ELF-Objektdateien entweder einzeln oder als Archivkomponenten. Nachdem ein ELF-Deskriptor von `elf_begin` erzeugt wurde, kann das Programm eine existierende Datei lesen, aktualisieren oder eine neue Datei erzeugen. *fildes* ist ein geöffneter Dateideskriptor, den `elf_begin` zum Lesen oder Schreiben benutzt. Der anfängliche Datei-Offset (siehe `lseek`(2)) spielt keine Rolle, und der resultierende Datei-Offset ist undefiniert.

Für *cmd* sind die folgenden Werte möglich:

`ELF_C_NULL` `elf_begin` liefert einen Nullzeiger, ohne daß ein neuer Deskriptor angelegt wird. *ref* wird bei diesem Kommando ignoriert. Siehe `elf_next`(3E) und die dazugehörigen Beispiele für weitere Informationen.

`ELF_C_READ` Wenn ein Programm den Inhalt einer existierenden Datei prüfen will, sollte es *cmd* auf diesen Wert setzen. Abhängig vom *ref*-Wert prüft dieses Kommando Archivkomponenten oder ganze Dateien. Dabei können folgende drei Fälle auftreten:

1. Wenn *ref* ein Nullzeiger ist, belegt `elf_begin` einen neuen ELF-Deskriptor und bereitet die Bearbeitung der ganzen Datei vor. Wenn die zu lesende Datei ein Archiv ist, bereitet `elf_begin` den resultierenden Deskriptor darauf vor, beim nächsten Aufruf von `elf_begin` auf die erste Archivkomponente der kompletten Datei zuzugreifen, als ob das Programm `elf_next` oder `elf_rand` zur Positionierung auf die erste Komponente verwenden würde.

2. Wenn *ref* ein mit einer Archivdatei verknüpfter Deskriptor ungleich null ist, erlaubt `elf_begin` einem Programm das Anlegen eines separaten ELF-Deskriptors, der mit einer eigenen Komponente verknüpft ist. Das Programm sollte `elf_next` oder `elf_rand` verwenden, um *ref* entsprechend zu positionieren (außer für die erste Komponente, welche `elf_begin` automatisch einstellt; siehe folgendes Beispiel). In diesem Fall sollte *fildes* derselbe Dateideskriptor sein, der auch für das Vaterarchiv verwendet wird.

3. Wenn *ref* schließlich ein Deskriptor ungleich null ist und kein Archiv darstellt, so erhöht `elf_begin` die Anzahl der Aktivierungen für den Deskriptor und liefert *ref* zurück, ohne daß ein neuer Deskriptor angelegt wird und ohne daß die Lese- und Schreibrechte des Deskriptors verändert werden. Um den Deskriptor für *ref* freizugeben, muß das Programm für jede der Aktivierungen `elf_end` einmal aufrufen. Siehe `elf_next`(3E) und die Beispiele für weitere Informationen.

`ELF_C_RDWR` Dieses Kommando führt zu denselben Aktionen wie `ELF_C_READ`, erlaubt jedoch dem Programm außerdem das Aktualisieren der Dateiinhalte (siehe `elf_update`(3E)). Dies bedeutet, daß `ELF_C_READ` nur das Lesen der Datei ermöglicht, während `ELF_C_RDWR` dem Programm das Lesen *und* Schreiben der Datei erlaubt. `ELF_C_RDWR` ist für Archivkomponenten nicht zulässig. Wenn *ref* ungleich null ist, so muß die Datei zuvor mit dem Kommando `ELF_C_RDWR` erzeugt worden sein.

`ELF_C_WRITE`
Wenn das Programm einen bereits vorhandenen Dateiinhalt ignorieren möchte, um eine neue Datei zu erzeugen, sollte *cmd* auf diesen Wert gesetzt werden. *ref* wird für dieses Kommando ignoriert.

`elf_begin` 'arbeitet' mit allen Dateien (einschließlich Dateien mit Null-Bytes), vorausgesetzt, es kann genügend Speicher für interne Strukturen belegt werden, und die notwendigen Dateiinformationen können gelesen werden. Programme, die Objektdateien auslesen, können daher `elf_kind` oder `elf_getehdr` aufrufen, um den Dateityp festzustellen (nur Objektdateien haben einen ELF-Kopf). Wenn die Datei ein Archiv ist, welches keine weiteren Komponenten zum Bearbeiten enthält, oder wenn ein Fehler auftritt, so liefert `elf_begin` einen Nullzeiger zurück. Ansonsten wird ein ELF-Deskriptor ungleich null zurückgegeben.
Vor dem ersten Aufruf von `elf_begin` muß das Programm zur Koordinierung der Versionen `elf_version` aufrufen.

Systemdienste

Bei der Bearbeitung einer Datei entscheidet die Bibliothek in Abhängigkeit von den Programmanforderungen, wann die Datei gelesen oder geschrieben wird. Normalerweise geht die Bibliothek davon aus, daß der Dateideskriptor bei Verfügbarkeit des ELF-Deskriptors ebenfalls verfügbar ist. Muß ein Programm viele Dateien gleichzeitig bearbeiten und beschränkt das zugrundeliegende Betriebssystem die Anzahl der geöffneten Dateien, so kann das Programm `elf_cntl` verwenden, um die Dateideskriptoren wiederzuverwenden. Nachdem `elf_cntl` mit entsprechenden Argumenten aufgerufen wurde, kann das Programm den Dateideskriptor schließen, ohne mit der Bibliothek in Konflikt zu kommen.
Alle Daten, die mit einem ELF-Deskriptor verknüpft sind, bleiben allokiert, bis `elf_end` die letzte Aktivierung des Deskriptors beendet hat. Nach Freigabe der Deskriptoren wird

auch der Speicher freigegeben; der Versuch, auf freigegebene Daten zuzugreifen, führt zu einem undefinierten Verhalten. Daher muß ein Programm, welches mit mehreren Ein- oder Ausgabedateien operiert, die ELF-Deskriptoren aktiv halten, bis die Bearbeitung vollständig abgeschlossen ist.

BEISPIELE

Ein Prototyp zum Lesen einer Datei ist unten angeführt. Wenn die Datei eine einfache Objektdatei ist, führt das Programm die Schleife einmal aus und erhält beim zweiten Schleifendurchlauf einen Nulldeskriptor. In diesem Fall haben `elf` und `arf` denselben Wert; der Aktivierungszähler ist 2, und das Programm ruft `elf_end` zweimal auf, um den Deskriptor freizugeben. Wenn die Datei ein Archiv ist, wird die Schleife für jede Archivkomponente durchgeführt, wobei die Dateien, die keine Objektdateien sind, ignoriert werden.

```
if (elf_version(EV_CURRENT) == EV_NONE)
{
      /* Bibliothek veraltet */
      /* Fehlerbehandlung */
}
cmd = ELF_C_READ;
arf = elf_begin(fildes, cmd, (Elf *)0);
while ((elf = elf_begin(fildes, cmd, arf)) != 0)
{
      if ((ehdr = elf32_getehdr(elf)) != 0)
      {
            /* Datei bearbeiten ... */
      }
      cmd = elf_next(elf);
      elf_end(elf);
}
elf_end(arf);
```

Alternativ dazu ist auch die wahlfreie Bearbeitung von Archiven möglich, wie das nächste Beispiel zeigt. Nachdem die Datei als Archiv identifiziert ist, bearbeitet das Programm die gewünschten Archivkomponenten. Der Einfachheit halber sind in diesem Beispiel Fehlerabfragen ausgelassen; einfache Objektdateien werden ignoriert. Außerdem bleiben in diesen Programmzeilen die ELF-Deskriptoren für alle Archivkomponenten bestehen, da `elf_end` nicht zur Freigabe aufgerufen wird.

```
elf_version(EV_CURRENT);
arf = elf_begin(fildes, ELF_C_READ, (Elf *)0);
if (elf_kind(arf) != ELF_K_AR)
{
      /* kein Archiv  */
}
/* erste Komponente bearbeiten */
/* offset = ... für gewünschten Kopf der Komponente  */
while (elf_rand(arf, offset) == offset)
{
      if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
            break;
      if ((ehdr = elf32_getehdr(elf)) != 0)
      {
            /* Archivkomponente bearbeiten ... */
      }
      /* offset = ... für gewünschten Kopf der Komponente */
}
```

Der folgende Auszug zeigt, wie eine neue ELF-Datei erzeugt werden kann. Um den allgemeinen Ablauf zu beschreiben, ist dieses Beispiel vereinfacht:

```
elf_version(EV_CURRENT);
fildes = open("pfad/name", O_RDWR|O_TRUNC|O_CREAT, 0666);
if ((elf = elf_begin(fildes, ELF_C_WRITE, (Elf *)0)) == 0)
      return;
ehdr = elf32_newehdr(elf);
phdr = elf32_newphdr(elf, count);
scn = elf_newscn(elf);
shdr = elf32_getshdr(scn);
data = elf_newdata(scn);
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

Folgender Auszug beschreibt das Aktualisieren einer bereits vorhandenen ELF-Datei. Dieses Beispiel ist ebenfalls vereinfacht, um den allgemeinen Ablauf darzustellen.

```
elf_version(EV_CURRENT);
fildes = open("pfad/name", O_RDWR);
elf = elf_begin(fildes, ELF_C_RDWR, (Elf *)0);

/* neue Informationen hinzufügen oder alte Informationen löschen... */

close(creat("pfad/name", 0666));
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

Im obigen Beispiel kürzt der Aufruf `creat` die Datei; dadurch wird sichergestellt, daß die resultierende Datei die 'richtige' Größe hat. Ohne Kürzen könnte die aktualisierte Datei genauso groß sein wie die Originaldatei, selbst wenn Informationen gelöscht wurden. Die Bibliothek kürzt, wenn möglich, die Datei durch `ftruncate` (siehe `truncate`(2)). Einige Systeme unterstützen jedoch `ftruncate` nicht; der Aufruf von `creat` schützt die Datei davor.

Bitte beachten Sie, daß beide Beispiele zum Erzeugen von Dateien jeweils mit Schreib- *und* Leserechten öffnen. Die Bibliothek verwendet `mmap` (sofern das System dies unterstützt), um den Durchsatz zu steigern, und `mmap` erfordert einen lesbaren Dateideskriptor. Die Bibliothek kann auch einen Dateideskriptor verwenden, der nur beschreibbar ist; das Anwendungsprogramm erhält dann jedoch nicht die Durchsatzvorteile von `mmap`.

SIEHE AUCH

`cof2elf`(1), `creat`(2), `lseek`(2), `mmap`(2), `open`(2), `truncate`(2), `elf`(3E), `elf_cntl`(3E), `elf_end`(3E), `elf_getarhdr`(3E), `elf_getbase`(3E), `elf_getdata`(3E), `elf_getehdr`(3E), `elf_getphdr`(3E), `elf_getscn`(3E), `elf_kind`(3E), `elf_next`(3E), `elf_rand`(3E), `elf_rawfile`(3E), `elf_update`(3E), `elf_version`(3E), `ar`(4).

HINWEIS

COFF ist ein Objektdateiformat, das vor ELF verwendet wurde. Wenn ein Programm `elf_begin` mit einer COFF-Datei aufruft, übersetzt die Bibliothek die COFF-Strukturen in die entsprechenden ELF-Äquivalente; das Programm kann auch eine COFF-Datei lesen (aber nicht schreiben), als ob es sich um eine ELF-Datei handeln würde. Diese Konversion findet nur auf der Speicherabbildung und *nicht* auf der Datei selbst statt. Nach dem Aufruf von `elf_begin` behalten die Datei-Offsets, die Adressen des ELF-Kopfes, die Programmköpfe und die Abschnittsköpfe die originalen COFF-Werte (siehe `elf_getehdr`, `elf_getphdr` und `elf_getshdr`). Ein Programm kann `elf_update` aufrufen, um diese Werte anzupassen (ohne die Datei zu schreiben). Die Bibliothek liefert dann eine konsistente ELF-Abbildung der Datei. Daten, die über `elf_getdata` gelesen werden, werden übersetzt (die COFF-Symboltabelle wird im ELF-Format dargestellt etc.). Daten, die über `elf_rawdata` gelesen werden, unterliegen keiner Umwandlung; das Programm kann somit den tatsächlichen Dateiinhalt lesen.

Einige Debugging-Informationen von COFF werden nicht übersetzt; dies beeinflußt die Semantik eines laufenden Programms nicht.

Obwohl die ELF-Bibliothek COFF unterstützt, wird den Programmierern dringend geraten, ihre Programme neu zu übersetzen, um dadurch ELF-Objektdateien zu erhalten.

elf_cntl - Dateideskriptor kontrollieren

```
#include <libelf.h>
int elf_cntl(Elf *elf, Elf_Cmd cmd);
```

`elf_cntl` weist die Bibliothek an, ihr Verhalten bezüglich eines ELF-Deskriptors *elf* zu ändern. Wie unter `elf_begin`(3E) beschrieben, kann ein ELF-Deskriptor mehrfache Aktivierungen besitzen, und mehrere ELF-Deskriptoren können gemeinsam einen Dateideskriptor verwenden. Grundsätzlich gelten die `elf_cntl`-Kommandos für alle Aktivierungen von *elf*. Darüber hinaus werden, wenn der ELF-Deskriptor mit einer Archivdatei verknüpft ist, die Deskriptoren für Komponenten aus dem Archiv, wie unten beschrieben, beeinflußt. Soweit keine andere Aussage gemacht wird, beeinträchtigen Operationen mit Archivkomponenten nicht den Deskriptor für das umfassende Archiv.

Das Argument *cmd* gibt an, welche Aktionen ausgeführt werden sollen; folgende Werte sind möglich:

`ELF_C_FDDONE`
: Dieser Wert teilt der Bibliothek mit, daß der Dateideskriptor, der mit *elf* verknüpft ist, nicht verwendet werden soll. Ein Programm sollte dieses Kommando verwenden, wenn alle notwendigen Informationen gelesen wurden und aus Leistungsgründen der Rest der Datei nicht gelesen werden soll. Der Speicher für alle ausgeführten Operationen bleibt gültig; weitere Dateioperationen, wie zum Beispiel der erste Aufruf von `elf_getdata` für einen Abschnitt, schlagen jedoch fehl, wenn sich die betreffenden Daten noch nicht im Speicher befinden.

`ELF_C_FDREAD`
: Dieses Kommando ist `ELF_C_FDDONE` ähnlich; die Bibliothek wird jedoch dazu gezwungen, den Rest der Datei zu lesen. Ein Programm sollte dieses Kommando verwenden, wenn der Dateideskriptor geschlossen werden muß, aber noch nicht alle Informationen aus der Datei gelesen wurden. Nachdem `elf_cntl` das `ELF_C_FDREAD`-Kommando ausgeführt hat, verwenden weitere Operationen, wie zum Beispiel `elf_getdata`, die Speicherversion der Datei, ohne den Dateideskriptor benutzen zu müssen.

Wenn `elf_cntl` erfolgreich ist, wird Null zurückgeliefert. Ansonsten war *elf* ein Nulldeskriptor oder ein Fehler trat auf; in diesem Falle liefert die Funktion -1.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_getdata`(3E), `elf_rawfile`(3E).

HINWEIS

Wenn das Programm die 'Raw' Operationen (siehe `elf_rawdata`, wie unter `elf_getdata`(3E) beschrieben, und `elf_rawfile`(3E)) nach dem Deaktivieren des Dateideskriptors mit `ELF_C_FDDONE` oder `ELF_C_FDREAD` verwenden möchte, muß es die Rohoperationen ausdrücklich vorher ausführen. Ansonsten schlagen die rohen Dateioperationen fehl. Der Aufruf von `elf_rawfile` macht die gesamte Speicherabbildung verfügbar; nachfolgende `elf_rawdata` Aufrufe werden dadurch unterstützt.

elf_end - Benutzung einer Objektdatei beenden

```
#include <libelf.h>
int elf_end(Elf *elf);
```

Ein Programm verwendet `elf_end`, um den ELF-Deskriptor *elf* und die mit ihm verknüpften Daten freizugeben. Solange das Programm einen Deskriptor noch nicht freigegeben hat, bleiben die mit dem Deskriptor verknüpften Daten allokiert. *elf* sollte ein Wert sein, der zuvor durch `elf_begin` zurückgegeben wurde; ein Nullzeiger ist zur Erleichterung der Fehlerbehandlung als Argument erlaubt. Sollen durch ein Programm Daten, die mit dem ELF-Deskriptor verknüpft sind, in die Datei geschrieben werden, muß `elf_update` vor dem Aufruf `elf_end` verwendet werden.

Wie unter `elf_begin`(3E) schon angeführt wurde, kann ein Deskriptor mehr als eine Aktivierung besitzen. Der Aufruf von `elf_end` entfernt eine Aktivierung und liefert den resultierenden Wert des Aktivierungszählers zurück. Die Bibliothek gibt einen Deskriptor erst frei, wenn der Aktivierungszähler gleich Null wird. Ein Rückgabewert von Null zeigt daher an, daß ein ELF-Deskriptor nicht länger gültig ist.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_update`(3E).

elf_errmsg, elf_errno - Fehlerbehandlung

```
#include <libelf.h>
const char *elf_errmsg(int err);
int elf_errno(void);
```

Wenn eine ELF-Bibliotheksfunktion fehlschlägt, kann das Programm `elf_errno` aufrufen, um die bibliotheksinterne Fehlernummer zu erhalten. Als Nebeneffekt dieser Funktion wird die interne Fehlernummer auf Null (kein Fehler) gesetzt.

`elf_errmsg` benötigt die Fehlernummer *err* und liefert dafür eine durch ein Nullzeichen beendete Fehlermeldung (ohne angehängten Zeilenvorschub), welche das Problem beschreibt. Wird für *err* Null übergeben, so wird die Meldung für den letzten Fehler zurückgegeben. Ist kein Fehler aufgetreten, ist der Rückgabewert ein Nullzeiger (nicht etwa ein Zeiger auf eine Nullzeichenkette). Der Wert -1 für *err* ergibt ebenfalls die letzte Fehlermeldung, jedoch wird ein Rückgabewert ungleich Null garantiert, selbst wenn kein Fehler aufgetreten ist. Wenn keine Fehlermeldung für die angegebene Fehlernummer verfügbar ist, liefert `elf_errmsg` einen Zeiger auf eine aussagefähige Meldung zurück. Diese Funktion setzt nicht als Nebeneffekt die interne Fehlernummer zurück.

BEISPIEL

Der folgende Programmauszug löscht zunächst die interne Fehlernummer und überprüft später auf Fehler. Wenn nach dem ersten Aufruf von `elf_errno` kein Fehler auftritt, liefert der nächste Aufruf den Wert Null.

```
(void)elf_errno();
while (more_to_do)
{
        /* Bearbeitung... */
        if ((err = elf_errno()) != 0)
        {
                msg = elf_errmsg(err);
                /* Meldung ausgeben */
        }
}
```

SIEHE AUCH

`elf`(3E), `elf_version`(3E).

elf_fill - Füll-Byte einstellen

```
#include <libelf.h>
void elf_fill(int fill);
```

Beschränkungen bezüglich der Ausrichtung von ELF-Dateien erfordern manchmal die Anwesenheit von 'Löchern'. Wenn beispielsweise die Daten für einen Abschnitt auf einem 8-Byte-Block beginnen müssen und der vorhergehende Abschnitt zu 'kurz' ist, muß die Bibliothek die dazwischenliegenden Bytes auffüllen. Der Wert dieser Bytes wird auf den Wert *fill* gesetzt. Die Bibliothek verwendet Null-Bytes, wenn das Anwendungsprogramm keinen anderen Wert angibt. Siehe `elf_getdata`(3E) für weitere Informationen über 'Löcher'.

SIEHE AUCH

`elf`(3E), `elf_getdata`(3E), `elf_flag`(3E), `elf_update`(3E).

HINWEIS

Ein Anwendungsprogramm kann durch Setzen des `ELF_F_LAYOUT`-Bits die Organisation der Objektdatei kontrollieren (siehe `elf_flag`(3E)). In diesem Falle werden die Löcher *nicht* gefüllt.

elf_flag - Schalter manipulieren

```
#include <libelf.h>
unsigned elf_flagdata(Elf_Data *data, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagehdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagelf(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagphdr(Elf *elf, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagscn(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);
unsigned elf_flagshdr(Elf_Scn *scn, Elf_Cmd cmd, unsigned flags);
```

Diese Funktionen manipulieren die Schalter, die mit den verschiedenen Strukturen einer ELF-Datei verknüpft sind. Gegeben sei ein ELF-Deskriptor *elf*, ein Datendeskriptor *data* oder ein Abschnittsdeskriptor *scn*. Die Funktionen können dann die entsprechenden Status-Bits setzen oder löschen; als Ergebnis wird der modifizierte Wert der Bits zurückgeliefert. Ein Nulldeskriptor ist erlaubt, um die Fehlerbehandlung zu vereinfachen; alle Funktionen liefern in diesem Fall den Wert Null als Ergebnis.

cmd kann die folgenden Werte annehmen:

`ELF_C_CLR` Die Funktionen löschen die in *flags* angegebenen Bits. Nur die in *flags* gesetzten Bits werden gelöscht; Null-Bits ändern den Status des Deskriptors nicht.

`ELF_C_SET` Die Funktionen setzen die in *flags* angegebenen Bits. Nur die in *flags* gesetzten Bits werden gesetzt; Null-Bits ändern den Status des Deskriptors nicht.

Die Beschreibungen zu den definierten Bits für *flags* lauten:

`ELF_F_DIRTY`
Wenn das Programm eine ELF-Datei schreiben will, garantiert dieses Bit, daß die entsprechenden Informationen in die Datei geschrieben werden. Möchte ein Programm beispielsweise den ELF-Dateikopf einer Datei verändern, wird `elf_flagehdr` mit diesem Bit in *flags* und *cmd* gleich `ELF_C_SET` aufgerufen. Ein nachfolgender Aufruf von `elf_update` schreibt dann den derart markierten Dateikopf in die Datei.

ELF_F_LAYOUT

Normalerweise entscheidet die Bibliothek, wie eine Ausgabedatei organisiert wird. Dies bedeutet, daß automatisch entschieden wird, wo Abschnitte plaziert werden, wie sie in der Datei angeordnet werden etc. Wenn dieses Bit für einen ELF-Deskriptor gesetzt ist, übernimmt das Programm die Verantwortung für die Bestimmung aller Dateipositionen. Dieses Bit ist nur für `elf_flagelf` relevant und gilt für die gesamte Datei, die mit dem Deskriptor verknüpft ist.

Wird ein Schalterbit für ein Objekt gesetzt, so werden alle Unterobjekte ebenfalls beeinflußt. Setzt das Programm beispielsweise das `ELF_F_DIRTY` Bit mit `elf_flagelf`, so befindet sich die gesamte logische Datei in diesem Modus.

BEISPIEL

Der folgende Programmauszug demonstriert, wie ein ELF-Dateikopf markiert werden kann, um in die Ausgabedatei geschrieben zu werden.

```
ehdr = elf32_getehdr(elf);
/* dirty ehdr ... */
elf_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

SIEHE AUCH

`elf`(3E), `elf_end`(3E), `elf_getdata`(3E), `elf_getehdr`(3E), `elf_update`(3E).

elf_fsize: elf32_fsize - Länge einer Objektdatei zurückgeben

```
#include <libelf.h>
size_t elf32_fsize(Elf_Type type, size_t count, unsigned ver);
```

`elf32_fsize` liefert die Bytelänge der 32-Bit-Dateidarstellung von *count* Datenobjekten des gegebenen Typs *type* zurück. Die Bibliothek benutzt die Version *ver* zur Längenberechnung (siehe `elf`(3E) und `elf_version`(3E)).

Für die Länge der grundlegenden Typen sind konstante Werte verfügbar.

Elf_Type	Dateilänge	Speichergröße
ELF_T_ADDR	ELF32_FSZ_ADDR	sizeof(Elf32_Addr)
ELF_T_BYTE	1	sizeof(unsigned char)
ELF_T_HALF	ELF32_FSZ_HALF	sizeof(Elf32_Half)
ELT_T_OFF	ELF32_FSZ_OFF	sizeof(Elf32_Off)
ELF_T_SWORD	ELF32_FSZ_SWORD	sizeof(Elf32_Sword)
ELF_T_WORD	ELF32_FSZ_WORD	sizeof(Elf32_Word)

`elf32_fsize` liefert Null, wenn der Wert von *type* oder *ver* unbekannt ist. Unter `elf_xlate`(3E) finden Sie eine Liste der Werte für *type*.

SIEHE AUCH

`elf`(3E), `elf_version`(3E), `elf_xlate`(3E).

elf_getarhdr - Mitglied einer Archivdatei auslesen

```
#include <libelf.h>
Elf_Arhdr *elf_getarhdr(Elf *elf);
```

`elf_getarhdr` liefert einen Zeiger auf den Kopf eines Mitglieds einer Archivdatei, vorausgesetzt, über den angegebenen ELF-Deskriptor *elf* ist eine solcher verfügbar. Andernfalls ist entweder kein Kopf eines Mitglieds der Archivdatei vorhanden, es trat ein Fehler auf, oder *elf* war gleich Null; `elf_getarhdr` liefert in diesem Fall den Wert Null als Ergebnis. Der Kopf enthält folgende Komponenten:

```
char            *ar_name;
time_t          ar_date;
long            ar_uid;
long            ar_gid;
unsigned long   ar_mode;
off_t           ar_size;
char            *ar_rawname;
```

Der Name eines Elements des Archivs, angesprochen durch `ar_name`, ist eine durch ein Nullzeichen beendete Zeichenkette, bei der die `ar`-Format-Steuerzeichen entfernt wurden. Die Komponente `ar_rawname` enthält eine durch ein Nullzeichen beendete Zeichenkette, die den originalen Namen in der Datei darstellt, einschließlich des abschließenden Schrägstrichs und der führenden Leerzeichen, wie sie im Archivformat vorgesehen sind.

Zusätzlich zu den 'normalen' Archivmitgliedern definiert das Archivformat einige spezielle Mitglieder. Die Namen aller besonderen Mitglieder beginnen mit einem Schrägstrich (`/`). Dadurch unterscheiden sie sich von regulären Mitgliedern (deren Namen keine Schrägstriche enthalten). Diese besonderen Mitglieder haben folgende Namen (`ar_name`).

`/` Dies ist die Symboltabelle der Archivdatei. Wenn sie vorhanden ist, wird sie das erste Mitglied der Bibliotheksdatei sein. Ein Programm kann auf die Symboltabelle des Archivs über die Funktion `elf_getarsym` zugreifen. Die Information in der Symboltabelle ist für die wahlfreie Bearbeitung der Archivdatei von Nutzen (siehe `elf_rand`(3E)).

`//` Dieses Mitglied enthält, falls vorhanden, die Zeichenkettentabelle für die langen Namen von Mitgliedern des Archivs. Der Kopf eines Archivmitglieds enthält einen 16-Byte-Bereich für dessen Namen. Diese Grenze kann in einigen Dateisystemen überschritten werden. Die Bibliothek verwendet automatisch für solch lange Namen die Zeichenkettentabelle; `ar_name` wird dabei auf den richtigen Wert gesetzt.

In einigen Fehlersituationen kann es vorkommen, daß der Name eines Mitglieds nicht zur Verfügung steht. Auch wenn in diesem Fall `ar_name` für diese Bibliothek auf Null gesetzt wird, wird `ar_rawname` wie üblich gesetzt.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_getarsym`(3E), `elf_rand`(3E), `ar`(4).

elf_getarsym - Symboltabelle eines Archivs lesen

```
#include <libelf.h>
Elf_Arsym *elf_getarsym(Elf *elf, size_t *ptr);
```

`elf_getarsym` liefert einen Zeiger auf die Symboltabelle des Archivs, falls diese für den ELF-Deskriptor *elf* verfügbar ist. Ansonsten hat das Archiv keine Symboltabelle, es trat ein Fehler auf, oder *elf* ist gleich Null. In diesem Fall liefert `elf_getarsym` den Wert Null. Die Symboltabelle ist ein Feld von Strukturen, bestehend aus den folgenden Komponenten:

```
char            *as_name;
size_t          as_off;
unsigned long   as_hash;
```

Diese Komponenten haben die folgende Semantik:

`as_name` ein Zeiger auf einen durch ein Nullzeichen beendeten Symbolnamen

`as_off` Dieser Wert ist ein Byte-Offset vom Anfang des Archivs auf den Kopf der Archivkomponente. Die Archivkomponente am angegebenen Offset definiert das entsprechende Symbol. Die Werte für `as_off` können als Argumente an `elf_rand` übergeben werden, um auf die gewünschten Archivkomponenten zuzugreifen.

`as_hash` Dies ist ein Hash-Wert für den Namen, der durch `elf_hash` berechnet wird.

Wenn *ptr* ungleich Null ist, speichert die Bibliothek die Anzahl der Tabelleneinträge an der Adresse ab, auf die *ptr* zeigt. Dieser Wert wird auf Null gesetzt, wenn der Rückgabewert Null ist. Der letzte Tabelleneintrag, der noch im Zähler angegeben ist, besitzt für `as_name` den Wert Null, für `as_off` den Wert Null und für `as_hash` den Wert `~0UL`.

SIEHE AUCH

`elf`(3E), `elf_getarhdr`(3E), `elf_hash`(3E), `elf_rand`(3E), `ar`(4).

elf_getbase - Basis-Offset einer Objektdatei lesen

```
#include <libelf.h>
off_t elf_getbase(Elf *elf);
```

`elf_getbase` liefert den Datei-Offset des ersten Bytes der Datei oder der Archivkomponente, die mit *elf* verknüpft ist; ist der Offset nicht bekannt oder kann er nicht berechnet werden, wird -1 zurückgegeben. Für *elf* ist ein Nulldeskriptor erlaubt, um die Fehlerbehandlung zu vereinfachen; der Rückgabewert ist in diesem Fall -1. Der Basis-Offset einer Archivkomponente stellt den Anfang der Informationen der Komponente dar, und *nicht* den Anfang des Kopfes der Archivkomponente.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `ar`(4).

elf_getdata, elf_newdata, elf_rawdata - Abschnittsdaten lesen

```
#include <libelf.h>
Elf_Data *elf_getdata(Elf_Scn *scn, Elf_Data *data);
Elf_Data *elf_newdata(Elf_Scn *scn);
Elf_Data *elf_rawdata(Elf_Scn *scn, Elf_Data *data);
```

Diese Funktionen lesen und manipulieren die Daten, die mit dem Abschnittsdeskriptor *scn* verknüpft sind. Beim Lesen einer bereits vorhandenen Datei ist ein Abschnitt mit einem einzelnen Datenpuffer verknüpft. Ein Programm kann einen neuen Abschnitt aus Teilen zusammensetzen, wobei die Daten aus mehreren Datenpuffern kombiniert werden. Aus diesem Grund sollten die Daten eines Abschnitts als Liste von Puffern betrachtet werden, welche jeweils durch einen Datendeskriptor bearbeitet werden.

Mit `elf_getdata` ist ein Programm in der Lage, der Reihe nach auf die Datenabschnitte einer Liste zuzugreifen. Wenn der übergebene Datendeskriptor *data* Null ist, liefert die Funktion den ersten Puffer zurück, der mit dem Abschnitt verknüpft ist. Ansonsten sollte *data* ein Datendeskriptor sein, der mit *scn* verknüpft ist; in diesem Fall erhält das Programm Zugriff auf das nächste Datenelement des Abschnitts. Wenn *scn* Null ist oder wenn ein Fehler auftritt, liefert `elf_getdata` einen Nullzeiger als Ergebnis.

`elf_getdata` übersetzt die Daten aus den Dateidarstellungen in Speicherdarstellungen (siehe `elf_xlate`(3E)); das Programm sieht Objekte von Speicherdatentypen in Abhängigkeit von der Dateiklasse (siehe `elf`(3E)). Die Arbeitsversion der Bibliothek (siehe `elf_version`(3E)) gibt an, welche Version der Speicherstrukturen von `elf_getdata` dem Programm gegenüber verwendet werden soll.

`elf_newdata` erzeugt einen neuen Datendeskriptor für einen Abschnitt; sind bereits Datenelemente für den Abschnitt vorhanden, so wird der Datendeskriptor an das Ende angehängt. Wie noch beschrieben wird, ist der neue Datendeskriptor leer und zeigt damit an, daß das Element noch keine Daten enthält. Der Typ des Deskriptors (`d_type` weiter unten) wird zunächst auf `ELF_T_BYTE` gesetzt, da dies am gebräuchlichsten ist; die Version `d_version` weiter unten) wird auf die Arbeitsversion gesetzt. Das Programm ist für die Einstellung und die Änderung der Deskriptorkomponenten verantwortlich. Diese Funktion setzt implizit das `ELF_F_DIRTY`-Bit für die Abschnittsdaten (siehe `elf_flag`(3E)). Wenn *scn* gleich Null ist oder wenn ein Fehler auftritt, liefert `elf_newdata` einen Nullzeiger als Ergebnis zurück.

`elf_rawdata` unterscheidet sich von `elf_getdata` dadurch, daß nur die nicht interpretierten Bytes, unabhängig vom Typ des Abschnitts, zurückgegeben werden. Diese Funktion sollte üblicherweise dazu verwendet werden, einen Dateiabschnitt zu lesen und zwar nur dann, wenn das Programm die automatische Datenübersetzung, wie sie unten beschrieben ist, vermeiden soll. Ein Programm darf weiterhin den Dateideskriptor, der

mit *elf* verknüpft ist, vor der ersten Raw-Operation weder schließen noch deaktivieren (siehe `elf_cntl`(3E)), da `elf_rawdata` die Daten aus der Datei lesen könnte, um sicherzustellen, daß keine Konflikte mit der Funktion `elf_getdata` auftreten. Eine ähnliche Möglichkeit, die für ganze Dateien anwendbar ist, ist unter `elf_rawfile`(3E) beschrieben. Wenn `elf_getdata` die richtige Übersetzung anbietet, so sollte diese Funktion gegenüber `elf_rawdata` bevorzugt werden. Wenn *scn* gleich Null ist, oder wenn ein Fehler auftritt, liefert `elf_rawdata` einen Nullzeiger als Ergebnis zurück.

Die Struktur `Elf_Data` enthält folgende Komponenten:

```
void            *d_buf;
Elf_Type        d_type;
size_t          d_size;
off_t           d_off;
size_t          d_align;
unsigned        d_version;
```

Diese Komponenten stehen zur direkten Manipulation durch das Programm zur Verfügung. Es folgt die Beschreibung der Komponenten.

`d_buf` Zeiger auf den Datenpuffer; ein Datenelement ohne Daten enthält einen Nullzeiger.

`d_type` Der Wert dieser Komponente gibt den Typ der Daten an, auf die `d_buf` zeigt. Der Typ eines Abschnitts bestimmt, wie der Inhalt interpretiert wird. Dies wird unten näher ausgeführt.

`d_size` Diese Komponente enthält die Gesamtlänge des Speichers (in Bytes), die die Daten beanspruchen. Dieser Wert kann sich von der Länge der Darstellung in der Datei unterscheiden. Die Länge ist gleich Null, wenn keine Daten vorhanden sind (Siehe die Erläuterungen zu `SHT_NOBITS`).

`d_off` Diese Komponente liefert den Offset innerhalb eines Abschnitts, an dem sich der Puffer befindet. Dieser Offset ist relativ zum Abschnitt der Datei, nicht zum Speicherobjekt.

`d_align` Diese Komponente enthält die benötigte Ausrichtung des Puffers, vom Anfang des Abschnitts an gerechnet. Dies bedeutet, daß `d_off` ein Vielfaches des Wertes dieser Komponente sein muß. Ist der Wert dieser Komponente beispielsweise vier, dann wird der Anfang des Puffers innerhalb des Abschnitts auf eine 4-Byte-Grenze ausgerichtet. Die Ausrichtung des gesamten Abschnitts ergibt sich aus dem Maximalwert seiner einzelnen Elemente; dadurch wird garantiert, daß die Ausrichtung eines Puffers innerhalb des Abschnitts und innerhalb einer Datei korrekt ist.

`d_version` Diese Komponente enthält die Versionsnummer des Objekts aus dem Puffer. Wenn die Bibliothek die Daten aus der Objektdatei erstmals liest, wird die Arbeitsversion verwendet, um die Übersetzung der Speicherobjekte zu kontrollieren.

Ausrichtung der Daten

Wie oben bereits erwähnt, gelten für die Pufferausrichtungen innerhalb von Abschnitten explizite Vorschriften. Daher kann es dazu kommen, daß zwischen den Puffern eines Abschnitts 'Löcher' entstehen. Programme, die Ausgabedateien erzeugen, können diese Löcher auf zwei Arten behandeln.

Zum einen kann das Programm `elf_fill` verwenden, um der Bibliothek mitzuteilen, auf welchen Wert die Bytes in diesen Löchern gesetzt werden sollen. Werden Löcher in der Datei erzeugt, wird das Füll-Byte verwendet, um diese Daten zu initialisieren. Das voreingestellte Füll-Byte der Bibliothek ist Null; es kann mit `elf_fill` geändert werden.

Zum anderen kann das Anwendungsprogramm eigene Datenpuffer anlegen, um diese Löcher zu belegen; in den Löchern können dann Werte abgelegt werden, die dem Abschnitt entsprechen. Ein Programm kann sogar verschiedene Füllwerte für verschiedene Abschnitte verwenden. Beispielsweise kann der Inhalt von Bytes in Textabschnitten auf Leeranweisungen (No-Operation) gesetzt werden, während Löcher in Datenabschnitten mit Null-Bytes gefüllt werden. Bei Verwendung dieser Methode findet die Bibliothek keine Löcher zum Füllen, da das Anwendungsprogramm diese eliminiert hat.

Abschnitts- und Speichertypen

`elf_getdata` interpretiert die Abschnittsdaten abhängig vom Abschnittstyp. Der Typ eines Abschnitts ist im Kopf eines Abschnitts festgehalten. Die folgende Tabelle zeigt die Abschnittstypen und die Art und Weise, wie die Bibliothek sie durch Speicherdatentypen für die 32-Bit-Dateiklasse darstellt. Für andere Klassen würde es ähnliche Tabellen geben. Impliziert ist, daß die Speicherdatentypen die Übersetzung durch `elf_xlate` kontrollieren.

Abschnittstyp	Elf_Typ	32-Bit Typ
SHT_DYNAMIC	ELF_T_DYN	Elf32_Dyn
SHT_DYNSYM	ELF_T_SYM	Elf32_Sym
SHT_HASH	ELF_T_WORD	Elf32_Word
SHT_NOBITS	ELF_T_BYTE	unsigned char
SHT_NOTE	ELF_T_BYTE	unsigned char
SHT_NULL	*keiner*	*keiner*
SHT_PROGBITS	ELF_T_BYTE	unsigned char
SHT_REL	ELF_T_REL	Elf32_Rel
SHT_RELA	ELF_T_RELA	Elf32_Rela
SHT_STRTAB	ELF_T_BYTE	unsigned char
SHT_SYMTAB	ELF_T_SYM	Elf32_Sym
andere	ELF_T_BYTE	unsigned char

`elf_rawdata` erzeugt einen Puffer vom Typ `ELF_T_BYTE`.

Wie bereits erwähnt, kontrolliert die Arbeitsversion des Programms, welche Strukturen die Bibliothek für das Anwendungsprogramm erzeugt. Auf die gleiche Weise interpretiert die Bibliothek die Abschnittstypen anhand der Versionen. Wenn ein Abschnittstyp zu einer Version 'gehört', die aktueller als die Arbeitsversion des Programms ist, übersetzt

die Bibliothek die Abschnittsdaten nicht. Da das Anwendungsprogramm das Datenformat in diesem Fall nicht kennen kann, liefert die Bibliothek einen unübersetzten Puffer des ELF_T_BYTE zurück; für einen nicht erkannten Abschnittstyp würde genauso vorgegangen.

Ein Abschnitt mit dem speziellen Typ SHT_NOBITS belegt keinen Platz in einer Objektdatei, selbst wenn der Abschnittskopf eine Größe ungleich Null angibt. elf_getdata und elf_rawdata 'arbeiten' auf solch einem Abschnitt, wobei die Struktur *data* als Pufferzeiger einen Nullzeiger und als Typ den oben angegebenen erhält. Obwohl keine Daten verfügbar sind, wird d_size auf die Größe aus dem Abschnittskopf gesetzt. Wenn ein Programm einen neuen Abschnitt des Typs SHT_NOBITS anlegt, sollte elf_newdata verwendet werden, um dem Abschnitt Datenpuffer hinzuzufügen. Diese 'leeren' Datenpuffer sollten für die Komponenten d_size die gewünschte Größe und für d_buf den Wert Null enthalten.

BEISPIEL

Der folgende Programmauszug liest die Tabelle mit den Namen der Abschnitte (die Fehlerbehandlung ist ausgelassen). Unter elf_strptr(3E) ist eine alternative Behandlung der String-Tabelle angeführt.

```
ehdr = elf32_getehdr(elf);
scn = elf_getscn(elf, (size_t)ehdr->e_shstrndx);
shdr = elf32_getshdr(scn);
if (shdr->sh_type != SHT_STRTAB)
{
      /* keine String-Tabelle */
}
data = 0;
if ((data = elf_getdata(scn, data)) == 0 || data->d_size == 0)
{
      /* Fehler oder keine Daten */
}
```

Die Komponente e_shstrndx in einem ELF-Kopf enthält den Abschnittstabellenindex der String-Tabelle. Das Programm holt einen Abschnittsdeskriptor für diesen Abschnitt, prüft, ob es sich um eine String-Tabelle handelt und liest dann die Daten aus. Am Ende dieses Programmausschnitts zeigt data->d_buf auf das erste Byte der String-Tabelle, und data->d_size enthält die Länge der String-Tabelle in Bytes.

SIEHE AUCH

elf(3E), elf_cntl(3E), elf_fill(3E), elf_flag(3E), elf_getehdr(3E), elf_getscn(3E), elf_getshdr(3E), elf_rawfile(3E), elf_version(3E), elf_xlate(3E).

elf_getehdr - klassenabhängigen Objektdateikopf lesen

```
#include <libelf.h>
Elf32_Ehdr *elf32_getehdr(Elf *elf);
Elf32_Ehdr *elf32_newehdr(Elf *elf);
```

`elf32_getehdr` liefert für eine Datei der 32-Bit-Klasse einen Zeiger auf einen ELF-Kopf zurück, sofern ein solcher für den ELF-Deskriptor *elf* verfügbar ist. Wenn kein Kopf für den Deskriptor existiert, belegt `elf32_newehdr` einen neuen. Ansonsten verhält sich die Funktion genauso wie `elf32_getehdr`. Sie belegt keinen neuen Kopf, wenn bereits einer existiert. Wenn kein Kopf existiert (bei `elf_getehdr`), keiner erzeugt werden kann (bei `elf_newehdr`), ein Systemfehler auftritt, die Datei keine Datei der 32-Bit-Klasse ist, oder wenn *elf* gleich Null ist, liefern beide Funktionen einen Nullzeiger als Ergebnis.

Der Kopf enthält die folgenden Komponenten:

```
unsigned char   e_ident[EI_NIDENT];
Elf32_Half      e_type;
Elf32_Half      e_machine;
Elf32_Word      e_version;
Elf32_Addr      e_entry;
Elf32_Off       e_phoff;
Elf32_Off       e_shoff;
Elf32_Word      e_flags;
Elf32_Half      e_ehsize;
Elf32_Half      e_phentsize;
Elf32_Half      e_phnum;
Elf32_Half      e_shentsize;
Elf32_Half      e_shnum;
Elf32_Half      e_shstrndx;
```

`elf32_newehdr` setzt automatisch das `ELF_F_DIRTY`-Bit (siehe `elf_flag`(3E)). Ein Programm kann `elf_getident` verwenden, um die Identifikationsbytes aus einer Datei zu überprüfen.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_flag`(3E), `elf_getident`(3E).

elf_getident - Identifkationsdaten einer Datei lesen

```
#include <libelf.h>
char *elf_getident(Elf *elf, size_t *ptr);
```

Wie unter elf(3E) beschrieben, bietet ELF einen Rahmen für verschiedene Klassen von Dateien, wobei die grundlegenden Objekte 32 Bits, 64 Bits etc. umfassen. Um diese Unterschiede auszugleichen, ohne die größeren Objekte auf kleineren Maschinen einzuschränken, beinhalten die ersten Bytes einer ELF-Datei Identifikationsinformationen, die für alle Dateiklassen gleich definiert sind. Jeder e_ident-Eintrag aus einem ELF-Kopf besitzt EI_NIDENT-Bytes, welche folgende Bedeutung haben:

e_ident Index	Wert	Zweck
EI_MAG0 EI_MAG1 EI_MAG2 EI_MAG3	ELFMAG0 ELFMAG1 ELFMAG2 ELFMAG3	Dateiidentifikation
EI_CLASS	ELFCLASSNONE ELFCLASS32 ELFCLASS64	Dateiklasse
EI_DATA	ELFDATANONE ELFDATA2LSB ELFDATA2MSB	Datencodierung
EI_VERSION	EV_CURRENT	Dateiversion
7-15	0	nicht benutzt, wird auf Null gesetzt

Andere Arten von Dateien (siehe elf_kind(3E)) können ebenfalls Identifikationsdaten enthalten, jedoch entsprechen diese nicht e_ident.

elf_getident liefert einen Zeiger auf die ersten Bytes der Datei. Wenn die Bibliothek die Datei erkennt, kann eine Konversion der Dateiabbildung in eine Speicherabbildung erfolgen. In jedem Fall wird garantiert, daß die Identifikationsbytes nicht modifiziert werden, obwohl die Größe des unveränderten Bereichs vom Dateityp abhängt. Wenn *ptr* ungleich Null ist, speichert die Bibliothek die Anzahl der Identifikationsbytes an der Adresse, auf die *ptr* zeigt. Wenn keine Daten verfügbar sind, *elf* gleich Null ist, oder wenn ein Fehler auftritt, so ist der Rückgabewert ein Nullzeiger, wobei gegebenenfalls unter *ptr* der Wert Null gespeichert wird.

SIEHE AUCH

elf(3E), elf_begin(3E), elf_getehdr(3E), elf_kind(3E), elf_rawfile(3E).

elf_getphdr - klassenabhängige Programmkopftabelle lesen

```
#include <libelf.h>
Elf32_Phdr *elf32_getphdr(Elf *elf);
Elf32_Phdr *elf32_newphdr(Elf *elf, size_t count);
```

`elf32_getphdr` liefert für eine Datei der 32-Bit-Klasse einen Zeiger auf die Programmkopftabelle, sofern eine solche für den ELF-Deskriptor *elf* verfügbar ist.

`elf32_newphdr` allokiert eine neue Tabelle mit *count*-Einträgen, unabhängig davon, ob bereits eine solche existierte, und setzt für die Tabelle das `ELF_F_DIRTY`-Bit (siehe `elf_flag`(3E)). Durch die Angabe von Null für *count* wird eine bereits vorhandene Tabelle gelöscht. Bitte beachten Sie, daß sich dieses Verhalten von `elf32_newehdr` unterscheidet (siehe `elf32_getehdr`(3E)), denn es ermöglicht dem Programm, die Programmkopftabelle zu ersetzen oder zu löschen, und falls nötig, dabei die Größe anzupassen.

Falls keine Programmkopftabelle existiert, die Datei keine Datei der 32-Bit-Klasse ist, ein Fehler auftrat oder *elf* den Wert Null enthielt, liefern beide Funktionen einen Nullzeiger als Ergebnis zurück. Außerdem liefert `elf32_newphdr` einen Nullzeiger, wenn *count* gleich Null ist.

Die Tabelle ist ein Feld aus `Elf32_Phdr`-Strukturen, wobei jede von ihnen folgende Komponenten enthält:

```
Elf32_Word      p_type;
Elf32_Off       p_offset;
Elf32_Addr      p_vaddr;
Elf32_Addr      p_paddr;
Elf32_Word      p_filesz;
Elf32_Word      p_memsz;
Elf32_Word      p_flags;
Elf32_Word      p_align;
```

Die Komponente `e_phnum` aus dem ELF-Dateikopf gibt an, wieviele Einträge die Programmkopftabelle enthält (siehe `elf_getehdr`(3E)). Ein Programm kann diesen Wert abfragen, um die Größe einer bereits vorhandenen Tabelle zu bestimmen. `elf32_newphdr` setzt den Wert dieser Komponente automatisch auf *count*. Wenn das Programm eine neue Datei erzeugt, ist es dafür verantwortlich, daß der ELF-Kopf vor der Programmkopftabelle erzeugt wird.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_flag`(3E), `elf_getehdr`(3E).

elf_getscn - Abschnittsinformationen lesen

```
#include <libelf.h>
Elf_Scn *elf_getscn(Elf *elf, size_t index);
size_t elf_ndxscn(Elf_Scn *scn);
Elf_Scn *elf_newscn(Elf *elf);
Elf_Scn *elf_nextscn(Elf *elf, Elf_Scn *scn);
```

Diese Funktionen bieten einen indizierten und sequentiellen Zugriff auf die Abschnitte, die zu dem ELF-Deskriptor *elf* gehören. Wenn ein Programm eine neue Datei erstellt, liegt es in dessen Verantwortung, zunächst einen ELF-Kopf für die Datei zu erzeugen, bevor Abschnitte definiert werden; siehe dazu auch `elf_getehdr`(3E).

`elf_getscn` erwartet in der Abschnittskopftabelle einer Datei einen *Index* und liefert als Ergebnis eines Abschnittsdeskriptor. Bitte beachten Sie, daß der erste 'echte' Abschnitt den Index 1 hat. Ein Programm kann zwar einen Abschnittsdeskriptor für einen Abschnitt mit *Index* 0 (`SHN_UNDEF`, der undefinierte Abschnitt) erhalten, jedoch enthält dieser Abschnitt keine Daten, und der Abschnittskopf ist 'leer' (aber vorhanden). Wenn der angegebene Abschnitt nicht existiert, ein Fehler auftritt, oder wenn *elf* gleich Null ist, liefert `elf_getscn` einen Nullzeiger als Ergebnis.

`elf_newscn` erzeugt einen neuen Abschnitt und hängt ihn an das Ende der Liste für *elf*. Da der `SHN_UNDEF`-Abschnitt benötigt wird, ihn die Anwendungen jedoch nicht 'interessieren', erzeugt ihn die Bibliothek automatisch. Der erste Aufruf von `elf_newscn` für einen ELF-Deskriptor ohne existierende Abschnitte liefert deshalb einen Deskriptor für Abschnitt 1. Wenn ein Fehler auftritt, oder wenn *elf* gleich Null ist, liefert `elf_newscn` einen Nullzeiger als Ergebnis zurück.

Nachdem ein neuer Abschnittsdeskriptor erzeugt wurde, kann ein Programm mit Hilfe von `elf_getshdr` den neu erzeugten, 'sauberen' Abschnittskopf anfordern. Dem neuen Abschnittsdeskriptor sind keine Daten zugeordnet (siehe auch `elf_getdata`(3E)). Wenn ein neuer Abschnitt auf diese Weise erzeugt wird, so aktualisiert die Bibliothek die Komponente `e_shnum` des ELF-Kopfes und setzt das `ELF_F_DIRTY`-Bit für den Abschnitt (siehe `elf_flag`(3E)). Bei der Erstellung einer neuen Datei liegt es im Verantwortungsbereich des Programms, daß zunächst ein ELF-Kopf für die Datei angelegt wird (siehe `elf_getehdr`(3E)) und erst danach neue Abschnitte erzeugt werden.

`elf_nextscn` erwartet einen Deskriptor für einen bereits existierenden Abschnitt *scn* und liefert als Ergebnis einen Abschnittsdeskriptor für den nächsthöheren Abschnitt. Durch die Verwendung von Null für *scn* erhält man einen Abschnittsdeskriptor für den Abschnitt mit Index 1 (der Abschnitt mit Index `SHN_UNDEF` wird dabei übersprungen). Wenn keine weiteren Abschnitte vorhanden sind, oder wenn ein Fehler auftritt, liefert `elf_nextscn` als Ergebnis einen Nullzeiger.

`elf_ndxscn` erwartet einen Deskriptor eines existierenden Abschnitts *scn* und liefert als Ergebnis dessen Index in der Abschnittstabelle. Wenn *scn* gleich Null ist, oder wenn ein Fehler auftritt, liefert `elf_ndxscn` den Wert `SHN_UNDEF` als Ergebnis.

BEISPIEL

Im folgenden Beispiel wird sequentiell auf die Abschnitte einer Datei zugegriffen. Mit jedem Durchlauf der Schleife wird ein Abschnitt der Datei bearbeitet; die Schleife endet, wenn alle Abschnitte bearbeitet wurden.

```
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
      /* Abschnitt bearbeiten */
}
```

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_flag`(3E), `elf_getdata`(3E), `elf_getehdr`(3E), `elf_getshdr`(3E).

elf_getshdr: elf32_getshdr - klassenabhängigen Abschnittskopf lesen

```
#include <libelf.h>
Elf32_Shdr *elf32_getshdr(Elf_Scn *scn);
```

`elf32_getshdr` liefert für eine Datei der 32-Bit-Klasse einen Zeiger auf einen Abschnittskopf zu dem Abschnittsdeskriptor *scn*. Andernfalls ist die Datei keine Datei der 32-Bit-Klasse, *scn* ist gleich Null, oder ein Fehler trat auf; `elf32_getshdr` liefert in diesem Fall Null als Ergebnis zurück.

Der Kopf enthält folgende Komponenten:

```
Elf32_Word      sh_name;
Elf32_Word      sh_type;
Elf32_Word      sh_flags;
Elf32_Addr      sh_addr;
Elf32_Off       sh_offset;
Elf32_Word      sh_size;
Elf32_Word      sh_link;
Elf32_Word      sh_info;
Elf32_Word      sh_addralign;
Elf32_Word      sh_entsize;
```

Wenn das Programm eine neue Datei erzeugt, ist es dafür verantwortlich, daß der ELF-Kopf für die Datei vor den Abschnitten angelegt wird.

SIEHE AUCH

`elf`(3E), `elf_flag`(3E), `elf_getscn`(3E), `elf_strptr`(3E).

elf_hash - Hash-Wert berechnen

```
#include <libelf.h>
unsigned long elf_hash(const char *name);
```

`elf_hash` berechnet einen Hash-Wert zu der Nullzeichen-terminierten Zeichenkette *name*. Der zurückgelieferte Hash-Wert h kann als Bucket-Index verwendet werden, meist nach der Berechnung von `h mod` x, um die Einhaltung der entsprechenden Grenzen sicherzustellen.

Hash-Tabellen können auf einer Maschine erzeugt und auf einer anderen verwendet werden, da `elf_hash` vorzeichenlose Arithmetikoperationen benutzt, um mögliche Unterschiede in der vorzeichenbehafteten Arithmetik zwischen unterschiedlichen Maschinen zu umgehen. Obwohl *name* oben als `char*` angegeben ist, behandelt `elf_hash` diesen Wert als `unsigned char*`, um Unterschiede bei der Behandlung des Vorzeichens zu vermeiden. Die Benutzung von `char*` vermeidet Typkonflikte bei Ausdrücken wie `elf_hash("name")`.

Die Hash-Tabellen von ELF-Dateien werden mit dieser Funktion berechnet (siehe `elf_getdata`(3E) und `elf_xlate`(3E)). Es ist sichergestellt, daß der zurückgegebene Hash-Wert nicht dem Bitmuster mit nur gesetzten Bits (`~0UL`) entspricht.

SIEHE AUCH

`elf`(3E), `elf_getdata`(3E), `elf_xlate`(3E).

elf_kind - Dateityp bestimmen

```
#include <libelf.h>
Elf_Kind elf_kind(Elf *elf);
```

Diese Funktion liefert einen Wert zur Identifizierung der Art jener Datei, welche mit einem ELF-Deskriptor *elf* verknüpft ist. Die zur Zeit definierten Werte sind unten angeführt.

`ELF_K_AR` Die Datei ist ein Archiv (siehe `ar`(4)). Ein ELF-Deskriptor kann auch mit einer Archivkomponente verknüpft werden. In diesem Fall identifiziert `elf_kind` den Typ der Komponente.

`ELF_K_COFF` Die Datei ist eine COFF-Objektdatei. Unter `elf_begin`(3E) ist die Behandlung der COFF-Dateien durch die Bibliothek beschrieben.

`ELF_K_ELF` Die Datei ist eine ELF-Datei. Das Programm kann `elf_getident` verwenden, um die Klasse zu bestimmen. Weitere Funktionen, wie `elf_getehdr`, sind verfügbar, um die Dateiinformationen auszulesen.

`ELF_K_NONE` Dieser Wert zeigt an, daß die Art der Datei der Bibliothek unbekannt ist.

Weitere Werte sind reserviert, um neuen Dateiarten zugeordnet zu werden, sobald dies notwendig wird. *elf* sollte einen Wert haben, der zuvor durch `elf_begin` zurückgeliefert wurde. Ein Nullzeiger ist erlaubt, um die Fehlerbehandlung zu erleichtern; in diesem Fall liefert `elf_kind` `ELF_K_NONE` zurück.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_getehdr`(3E), `elf_getident`(3E), `ar`(4).

elf_next - auf Archivkomponenten sequentiell zugreifen

```
#include <libelf.h>
Elf_Cmd elf_next(Elf *elf);
```

elf_next, elf_rand, und elf_begin manipulieren einfache Objektdateien und Archive. *elf* ist ein ELF-Deskriptor, der zuvor von elf_begin zurückgegeben wurde.

elf_next bietet sequentiellen Zugriff auf die nächste Archivkomponente. Wenn der ELF-Deskriptor *elf* mit einer Archivkomponente verknüpft ist, bereitet elf_next das umfassende Archiv auf den Zugriff des Programms auf die nächste Komponente durch den Aufruf von elf_begin vor. Wenn das Archiv erfolgreich auf die nächste Komponente positioniert wurde, liefert elf_next den Wert ELF_C_READ zurück. Handelt es sich bei der geöffneten Datei nicht um ein Archiv, enthält *elf* den Wert Null; trat ein Fehler auf, so wird ELF_C_NULL zurückgegeben. In jedem Fall kann der Rückgabewert als Argument an elf_begin übergeben werden, um die entsprechende Aktion zu veranlassen.

SIEHE AUCH

elf(3E), elf_begin(3E), elf_getarsym(3E), elf_rand(3E), ar(4).

elf_rand - auf Archivkomponenten wahlfrei zugreifen

```
#include <libelf.h>

size_t elf_rand(Elf *elf, size_t offset);
```

elf_rand, elf_next und elf_begin manipulieren einfache Objektdateien und Archive. *elf* ist ein ELF-Deskriptor, der zuvor von elf_begin zurückgegeben wurde.

elf_rand erlaubt die wahlfreie Bearbeitung von Archivkomponenten. *elf* wird auf den Zugriff auf eine beliebige Archivkomponente vorbereitet. *elf* muß ein Deskriptor für das Archiv selbst sein und nicht etwa für eine Komponente innerhalb des Archivs. *offset* gibt den Byte-Offset des Archivkopfs der gewünschten Komponente an, gerechnet vom Anfang des Archivs. Unter elf_getarsym(3E) finden Sie weitere Informationen über die Offsets von Archivkomponenten. Wenn der Aufruf von elf_rand gelingt, wird *offset* zurückgegeben. Wenn ein Fehler auftrat, *elf* gleich Null war, oder wenn die Datei kein Archiv war (keine Archivkomponente kann einen Null-Offset haben), wird 0 zurückgegeben. Ein Programm kann die wahlweise und sequentielle Archivbearbeitung gemischt verwenden.

BEISPIEL

Ein Archiv beginnt mit einer 'magischen Zeichenkette' mit der Länge von SARMAG-Bytes; darauf folgt sofort die erste Archivkomponente. Ein Anwendungsprogramm kann daher die folgende Funktion verwenden, um ein Archiv zurückzusetzen (die Funktion liefert -1 bei Fehler; ansonsten wird 0 zurückgegeben).

```
#include <ar.h>
#include <libelf.h>

int
rewindelf(Elf *elf)
{
        if (elf_rand(elf, (size_t)SARMAG) == SARMAG)
                return 0;
        return -1;
}
```

SIEHE AUCH

elf(3E), elf_begin(3E), elf_getarsym(3E), elf_next(3E), ar(4).

elf_rawfile - nicht interpretierten Dateiinhalt lesen

```
#include <libelf.h>
char *elf_rawfile(Elf *elf, size_t *ptr);
```

`elf_rawfile` liefert einen Zeiger auf eine nicht interpretierte Speicherdarstellung der Datei. Diese Funktion sollte nur verwendet werden, um eine gerade gelesene Datei wieder zu erhalten. Beispielsweise kann ein Programm `elf_rawfile` verwenden, um die unveränderten Bytes einer Archivkomponente zu erhalten.

Ein Programm darf den Dateideskriptor, der mit *elf* verknüpft ist, nicht schließen oder deaktivieren (siehe `elf_cntl`(3E)), bevor der erste Aufruf von `elf_rawfile` gemacht wurde, da `elf_rawfile` die Daten unter Umständen noch aus der Datei lesen muß, wenn die Daten noch nicht im Speicher stehen. Im allgemeinen ist diese Funktion für unbekannte Dateitypen effizienter als für Objektdateien. Die Bibliothek übersetzt die Objektdateien implizit im Speicher, während unbekannte Dateien unverändert bleiben. Die Anforderung einer uninterpretierten Abbildung einer Objektdatei kann daher zu einer doppelten Kopie der Datei im Speicher führen.

`elf_rawdata` (siehe `elf_getdata`(3E)) ist eine verwandte Funktion, welche den Zugriff auf Abschnitte innerhalb einer Datei erlaubt.

Ist *ptr* ungleich Null, so speichert die Bibliothek die Dateigröße (in Bytes) an der Adresse, auf die *ptr* zeigt. Sind keine Daten verfügbar, ist *elf* gleich Null; tritt ein Fehler auf, ist der Rückgabewert ein Nullzeiger; in *ptr* wird Null abgelegt.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_cntl`(3E), `elf_getdata`(3E), `elf_getehdr`(3E), `elf_getident`(3E), `elf_kind`(3E).

HINWEIS

Ein Programm, das `elf_rawfile` benutzt und das die gleiche Datei als Objektdatei interpretiert, hat möglicherweise zwei Kopien einer Datei im Speicher. Wenn solch ein Programm die nicht interpretierte Speicherabbildung vor den übersetzten Informationen anfordert (durch Funktionen wie `elf_getehdr`, `elf_getdata` etc.), 'friert' die Bibliothek die ursprüngliche Speicherkopie für die nicht interpretierte Abbildung ein. Die eingefrorene Kopie wird als Quelle zur Erzeugung von übersetzten Objekten verwendet, ohne daß die Datei neu gelesen wird. Die Anwendung sollte die nicht interpretierte Dateiabbildung, die durch `elf_rawfile` zurückgegeben wird, lediglich als lesbaren Puffer behandeln, es sei denn, es soll die Betrachtungsweise der zu übersetzenden Daten geändert werden. In jedem Fall kann das Anwendungsprogramm die übersetzten Objekte ändern, ohne daß die Daten der nicht interpretierten Abbildung verändert werden.

Mehrfaches Aufrufen von `elf_rawfile` mit demselben ELF-Deskriptor liefert den gleichen Wert zurück; die Bibliothek erzeugt keine doppelten Dateikopien.

elf_strptr - Zeiger auf Zeichenkette erzeugen

```
#include <libelf.h>
char *elf_strptr(Elf *elf, size_t section, size_t offset);
```

Diese Funktion konvertiert den Abschnitt mit Zeichenketten bei *offset* in einen Zeiger auf eine Zeichenkette. *elf* bezeichnet die Datei, in der sich der Zeichenkettenabschnitt befindet, und *section* gibt den Abschnittstabellenindex für die Zeichenketten an. `elf_strptr` liefert normalerweise einen Zeiger auf eine Zeichenkette; ein Nullzeiger wird zurückgegeben, wenn *elf* gleich Null ist, *section* ungültig ist, oder wenn es sich nicht um einen Abschnitt des Typs `SHT_STRTAB` handelt, die Abschnittsdaten nicht gelesen werden können, *offset* ungültig ist, oder wenn ein Fehler auftritt.

BEISPIEL

Eine Vorlage zum Lesen von Abschnittsnamen ist im folgenden angeführt. Der Dateikopf definiert die Tabelle mit den Abschnittsnamen in der Komponente `e_shstrndx`. Der folgende Programmauszug durchläuft die Abschnitte und gibt deren Namen aus.

```
if ((ehdr = elf32_getehdr(elf)) == 0)
{
      /* Fehlerbehandlung */
      return;
}
ndx = ehdr->e_shstrndx;
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
      char    *name = 0;
      if ((shdr = elf32_getshdr(scn)) != 0)
        name = elf_strptr(elf, ndx, (size_t)shdr->sh_name);
      printf("'%s'\n", name? name: "(Null)");
}
```

SIEHE AUCH

`elf`(3E), `elf_getdata`(3E), `elf_getshdr`(3E), `elf_xlate`(3E).

HINWEIS

Ein Programm kann `elf_getdata` aufrufen, um einen kompletten Abschnitt mit Zeichenketten zu lesen. Für einige Anwendungsprogramme ist dies sowohl effizienter als auch gebräuchlicher als die Verwendung von `elf_strptr`.

elf_update - ELF-Deskriptor aktualisieren

```
#include <libelf.h>

off_t elf_update(Elf *elf, Elf_Cmd cmd);
```

`elf_update` veranlaßt die Bibliothek zur Prüfung der Informationen, die mit dem ELF-Deskriptor *elf* verknüpft sind, und zur Neuberechnung der strukturellen Daten, die zur Generierung der Dateidarstellung notwendig sind.

cmd kann folgende Werte annehmen:

`ELF_C_NULL` Dieser Wert teilt `elf_update` mit, die verschiedenen Werte neu zu berechnen und nur die Speicherstrukturen des ELF-Deskriptors zu aktualisieren. Alle modifizierten Strukturen werden mit dem `ELF_F_DIRTY`-Bit markiert. Ein Programm kann daher die strukturellen Informationen aktualisieren und sie dann erneut überprüfen, ohne daß die Datei, die mit dem ELF-Deskriptor verknüpft ist, verändert wird. Da dies die Datei nicht ändert, kann der ELF-Deskriptor zum Lesen, Schreiben oder zu beidem geöffnet sein (siehe `elf_begin`(3E)).

`ELF_C_WRITE`

Enthält *cmd* diesen Wert, so geht `elf_update` genauso vor wie bei `ELF_C_NULL` und schreibt zusätzlich sämtliche markierten ('dirty') Informationen, die mit dem ELF-Deskriptor verknüpft sind, in die Datei. Wenn also ein Programm `elf_getdata` oder `elf_flag` benutzt hat, um neue Informationen für den ELF-Deskriptor anzulegen (oder alte Informationen zu aktualisieren), so werden diese Daten geprüft, abgestimmt, bei Bedarf umgesetzt (siehe `elf_xlate`(3E)) und dann in die Datei geschrieben. Wenn Teile der Datei geschrieben werden, werden deren `ELF_F_DIRTY`-Bits gelöscht; dies bedeutet, daß diese Objekte nicht länger in die Datei geschrieben werden müssen (siehe `elf_flag`(3E)). Die Abschnittsdaten werden in der Reihenfolge der Abschnittskopfeinträge geschrieben, und die Abschnittskopftabelle wird an das Dateiende geschrieben.

Wurde der ELF-Deskriptor mit `elf_begin` angelegt, muß er das Schreiben der Datei erlauben. Dies bedeutet, daß das entsprechende Kommando für `elf_begin` entweder `ELF_C_RDWR` oder `ELF_C_WRITE` sein muß.

Wenn `elf_update` erfolgreich ausgeführt wird, wird die Gesamtlänge der Dateidarstellung zurückgeliefert (nicht der Speicherdarstellung). Ansonsten ist ein Fehler aufgetreten, und die Funktion liefert -1.

Beim Aktualisieren der internen Strukturen setzt `elf_update` selbst einige Komponenten. Die unten aufgelisteten Komponenten unterliegen der Verantwortung des Anwendungsprogramms und beinhalten diejenigen Werte, welche durch das Programm vorgegeben wurden.

	Komponente	Hinweise
ELF-Kopf	e_ident[EI_DATA]	Bibliothek kontrolliert andere e_ident-Werte
	e_type	
	e_machine	
	e_version	
	e_entry	
	e_phoff	nur bei Angabe von ELF_F_LAYOUT
	e_shoff	nur bei Angabe von ELF_F_LAYOUT
	e_flags	
	e_shstrndx	

	Komponente	Hinweise
Programmkopf	p_type	Das Anwendungsprogramm kontrolliert alle Programmkopfeinträge.
	p_offset	
	p_vaddr	
	p_paddr	
	p_filesz	
	p_memsz	
	p_flags	
	p_align	

	Komponente	Hinweise
Abschnittskopf	sh_name	
	sh_type	
	sh_flags	
	sh_addr	
	sh_offset	nur bei Angabe von ELF_F_LAYOUT
	sh_size	nur bei Angabe von ELF_F_LAYOUT
	sh_link	
	sh_info	
	sh_addralign	nur bei Angabe von ELF_F_LAYOUT
	sh_entsize	

	Komponente	Hinweise
Datendeskriptor	d_buf	
	d_type	
	d_size	
	d_off	nur bei Angabe von ELF_F_LAYOUT
	d_align	
	d_version	

Zu beachten ist, daß das Programm für zwei besonders wichtige Komponenten (neben den anderen) im ELF-Kopf verantwortlich ist. Die Komponente `e_version` kontrolliert die Version der Datenstrukturen, die in die Datei geschrieben werden. Ist die Version `EV_NONE`, so verwendet die Bibliothek die eigene interne Version. Der Eintrag `e_ident[EI_DATA]` kontrolliert die Datencodierung der Datei. Als Spezialfall darf dieser Wert `ELFDATANONE` annehmen, um die herkömmliche Datencodierung der lokalen Maschine auszuwählen. Ein Fehler tritt dann auf, wenn die herkömmliche Dateicodierung keiner Dateicodierung entspricht, die die Bibliothek kennt.

Weiterhin ist zu beachten, daß das Programm für die Komponente `sh_entsize` des Abschnittskopfs verantwortlich ist. Obwohl die Bibliothek diese Komponente für Abschnitte bekannten Typs setzt, ist nicht garantiert, daß der korrekte Wert für alle Abschnitte bekannt ist. Daher verläßt sich die Bibliothek darauf, daß das Programm die entsprechenden Werte für unbekannte Abschnittstypen selbst einträgt. Wenn die Eintragsgröße unbekannt oder nicht anwendbar ist, sollte der Wert auf Null gesetzt werden.

Bei der Entscheidung, wie die Ausgabedatei erzeugt werden soll, richtet sich `elf_update` bei der Erzeugung der Ausgabeabschnitte nach den Ausrichtungen der einzelnen Datenpuffer. Der Datenpuffer mit der strengsten Ausrichtungsbeschränkung kontrolliert die Ausrichtung des gesamten Abschnitts. Falls notwendig, fügt die Bibliothek auch Leerräume zwischen Puffern ein, um somit die korrekte Ausrichtung jedes Puffers zu garantieren.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_flag`(3E), `elf_fsize`(3E), `elf_getdata`(3E), `elf_getehdr`(3E), `elf_getshdr`(3E), `elf_xlate`(3E).

HINWEIS

Wie bereits erwähnt, übersetzt das Kommando `ELF_C_WRITE` bei Bedarf die Daten, bevor sie in die Datei geschrieben werden. Diese Übersetzung ist für das Anwendungsprogramm *nicht* immer transparent. Wenn ein Programm Zeiger für den Zugriff auf Daten einer Datei unterhält (siehe beispielsweise `elf_getehdr`(3E) und `elf_getdata`(3E)), sollte das Programm die Zeiger nach dem Aufruf von `elf_update` neu aufbauen.

Wie unter `elf_begin`(3E) beschrieben, kann ein Programm eine COFF-Datei 'aktualisieren', um das Abbild konsistent mit dem ELF-Format darstellen zu können. Das Kommando `ELF_C_NULL` aktualisiert nur die Speicherabbildung; man kann das Kommando `ELF_C_WRITE` dazu verwenden, die Datei zu modifizieren. Absolute, ausführbare Dateien (`a.out`-Dateien) erfordern eine spezielle Ausrichtung, welche zwischen dem COFF- und dem ELF-Format nicht beibehalten werden kann. Daher kann eine ausführbare COFF-Datei mit dem Kommando `ELF_C_WRITE` nicht aktualisiert werden (`ELF_C_NULL` ist jedoch erlaubt).

elf_version - Versionen abgleichen

```
#include <libelf.h>
unsigned elf_version(unsigned ver);
```

Wie unter `elf`(3E) erläutert, besitzen das Programm, die Bibliothek und eine Objektdatei verschiedene Ansichten über die 'neueste' ELF-Version. `elf_version` erlaubt einem Programm die Feststellung der *internen Version* der ELF-Bibliothek. Ferner wird dem Programm ermöglicht, die verwendeten Speichertypen anzugeben; dies geschieht durch die Übergabe der *Arbeitsversion ver* an die Bibliothek. Jedes Programm, das die ELF-Bibliothek benutzt, muß die Versionen, wie beschrieben, abgleichen.

Die Include-Datei `libelf.h` stellt dem Programm die Version über das Makro `EV_CURRENT` zur Verfügung. Wenn die bibliotheksinterne Version (die höchste, der Bibliothek bekannte Version) kleiner als die dem Progamm bekannte Version ist, könnten der Bibliothek semantische Informationen fehlen, die vom Programm vorausgesetzt werden. Daher akzeptiert `elf_version` keine Arbeitsversion, die der Bibliothek unbekannt ist.

Die Übergabe von *ver* gleich `EV_NONE` veranlaßt `elf_version` dazu, die bibliotheksinterne Version zurückzuliefern, ohne daß die Arbeitsversion geändert wird. Wenn die Version *ver* der Bibliothek bekannt ist, liefert `elf_version` die vorherige Arbeitsversionsnummer zurück. Ansonsten bleibt die Arbeitsversion unverändert und `elf_version` liefert `EV_NONE`.

BEISPIEL

Der folgende Programmauszug aus einem Anwendungsprogramm schützt davor, eine ältere Bibliothek zu verwenden.

```
if (elf_version(EV_CURRENT) == EV_NONE)
{
      /* Bibliothek veraltet */
      /* Fehlerbehandlung */
}
```

HINWEIS

Die Arbeitsversion sollte für alle Operationen mit einem bestimmten ELF-Deskriptor identisch sein. Das Ändern der Version zwischen den Operationen eines Deskriptors führt unter Umständen nicht zu dem erwarteten Ergebnis.

SIEHE AUCH

`elf`(3E), `elf_begin`(3E), `elf_xlate`(3E).

elf_xlate - klassenabhängige Datenumsetzung

```
#include <libelf.h>
Elf_Data *elf32_xlatetof(Elf_Data *dst, const Elf_Data *src, unsigned encode);
Elf_Data *elf32_xlatetom(Elf_Data *dst, const Elf_Data *src, unsigned encode);
```

`elf32_xlatetom` setzt Datenstrukturen der 32-Bit-Klasse aus der Dateidarstellung in ihre Speicherdarstellung um. `elf32_xlatetof` führt den umgekehrten Prozeß durch. Diese Konversion ist für Cross-Entwicklungsumgebungen (Cross Compilation) sehr wichtig. *src* ist ein Zeiger auf den Quellpuffer, der die originalen Daten enthält; *dst* ist ein Zeiger auf den Zielpuffer, der die übersetzten Daten aufnimmt. *encode* gibt die Byte-Ordnung an, in der die Dateiobjekte dargestellt werden (sollen); die Byte-Ordnung muß ein Wert sein, der für den Eintrag `e_ident[EI_DATA]` (siehe `elf_getident`(3E)) des ELF-Kopfes definiert ist. Wenn die Daten umgesetzt werden können, liefern die Funktionen *dst* zurück. Ansonsten wird Null zurückgegeben, da ein Fehler auftrat, wie zum Beispiel inkompatible Typen, Zielpufferüberlauf etc.

Der `Elf_Data`-Deskriptor wird unter `elf_getdata`(3E) näher beschrieben; er wird durch die Übersetzungsroutinen wie folgt verwendet:

`d_buf` Sowohl die Quelle als auch das Ziel müssen gültige Pufferzeiger enthalten.

`d_type` Diese Komponente gibt den Typ der Daten an, auf die `d_buf` zeigt, und den Typ der Daten, die im Zielpuffer angelegt werden. Das Programm übergibt einen `d_type`-Wert für die Quelle; die Bibliothek setzt den `d_type` des Ziels auf den gleichen Wert. Diese Werte werden unten erläutert.

`d_size` Diese Komponente enthält die Gesamtlänge des durch die Quelldaten belegten und die Größe des für die Zieldaten allokierten Speichers. Wenn der Zielpuffer nicht groß genug ist, ändern die Routinen den ursprünglichen Inhalt nicht. Die Umsetzungsroutinen setzen die Komponente `d_size` des Ziels nach der erfolgten Umsetzung auf die tatsächlich benötigte Größe. Die Größen von Quelle und Ziel können unterschiedlich sein.

`d_version` Diese Komponente enthält die (gewünschte) Versionsnummer der Objekte im Puffer. Die Quell- und Zielversionen sind unabhängig.

Die Übersetzungsroutinen erlauben die Identität von Quell- und Zielpuffer. Dies bedeutet, daß `dst->d_buf` gleich `src->d_buf` sein kann. In anderen Fällen, wo Quell- und Zielpuffer überlappen, ist das resultierende Verhalten undefiniert.

Elf_Type	32-Bit-Speichertypen
ELF_T_ADDR	Elf32_Addr
ELF_T_BYTE	unsigned char
ELF_T_DYN	Elf32_Dyn
ELF_T_EHDR	Elf32_Ehdr
ELF_T_HALF	Elf32_Half
ELT_T_OFF	Elf32_Off
ELF_T_PHDR	Elf32_Phdr
ELF_T_REL	Elf32_Rel
ELF_T_RELA	Elf32_Rela
ELF_T_SHDR	Elf32_Shdr
ELF_T_SWORD	Elf32_Sword
ELF_T_SYM	Elf32_Sym
ELF_T_WORD	Elf32_Word

Das Umsetzen von Puffern des Typs `ELF_T_BYTE` ändert die Byte-Ordnung nicht.

SIEHE AUCH

`elf`(3E), `elf_fsize`(3E), `elf_getdata`(3E), `elf_getident`(3E).

nlist - Einträge einer Namensliste lesen

```
#include <nlist.h>
int nlist (const char *filename, struct nlist *nl);
```

`nlist` untersucht die Namensliste der ausführbaren Datei mit dem Namen *filename* und extrahiert eine Liste von Werten in einem Feld von `nlist`-Strukturen bei *nl*. Die Namensliste *nl* setzt sich aus einem Feld von Strukturen zusammen, bestehend aus den Namen von Variablen, Typen und Werten. Die Liste wird durch einen leeren Namen beendet, d.h. durch eine Struktur, die als Namenskomponente eine Nullzeichenkette enthält. Jeder Name einer Variablen wird in der Namensliste der Datei gesucht. Wird der Name gefunden, so werden Typ, Wert, Speicherklasse und Abschnittsnummer des Namens in den anderen Komponenten eingetragen. Die Typ-Komponente kann auf 0 gesetzt werden, wenn bei der Übersetzung der Datei die Option `-g` von `cc`(1) nicht verwendet wurde. `nlist` liefert immer die Informationen für ein externes Symbol zu einem Namen, wenn der Name in der Datei vorkommt. Existiert kein externes Symbol mit dem angegebenen Namen, und gibt es in der Datei mehr als ein (lokales) Symbol mit dem angegebenen Namen (wie zum Beispiel bei Symbolen, die in verschiedenen Dateien `static` definiert wurden), beziehen sich die zurückgegebenen Werte auf das letzte Auftreten des Namens in der Datei. Wenn der Name nicht gefunden wird, werden alle Komponenten der Struktur mit Ausnahme von `n_name` auf 0 gesetzt.

Diese Funktion ist beim Zugriff auf die System-Namenstabelle in der Datei `/stand/unix` nützlich. Auf diese Weise können Programme die aktuellen Werte wichtiger Systemadressen erfahren.

SIEHE AUCH

`a.out`(4).

ERGEBNIS

Alle Komponenten werden auf den Wert 0 gesetzt, wenn die Datei nicht gelesen werden kann oder sie keine gültige Namensliste enthält.

`nlist` liefert bei Erfolg den Wert 0, bei Auftreten eines Fehlers den Wert -1.

basename - letztes Element eines Pfadnamens zurückgeben

```
#include <libgen.h>
char *basename (char *Pfad);
```

Wenn man basename einen Zeiger auf eine mit Null beendete Zeichenkette übergibt, die einen Pfadnamen enthält, gibt basename einen Zeiger auf das letzte Element von *Pfad* zurück. Abschließende /-Zeichen werden gelöscht.

Wenn *Pfad* oder **Pfad* Null ist, wird ein Zeiger auf eine Konstante '.' zurückgegeben.

BEISPIELE

Eingabezeichenkette	Ausgabezeiger
/usr/lib	lib
/usr/	usr
/	/

SIEHE AUCH

dirname(3G).
basename(1) in "SINIX V5.41 Kommandos".

bgets - Stream bis zum nächsten Begrenzer lesen

```
#include <libgen.h>
char *bgets (char *Puffer, size_t *Anzahl, FILE *Stream, const char *Abbruchfolge);
```

bgets liest Zeichen vom *Stream* in den *Puffer*, bis entweder *Anzahl* Zeichen gelesen worden sind oder eines der Zeichen in *Abbruchfolge* im Stream angetroffen wird. Die gelesenen Daten werden mit einem Null-Byte abgeschlossen (\0), und es wird ein Zeiger auf das abschließende Null-Byte zurückgegeben. Wenn ein Zeichen der *Abbruchfolge* angetroffen wird, ist das letzte nicht-Nullzeichen der Begrenzer, der das Einlesen abschließt.

Beachten Sie, daß abgesehen davon, daß der zurückgegebene Wert auf das Ende statt auf den Anfang der gelesenen Zeichenkette zeigt, folgende Aufrufe identisch sind:

```
bgets (Puffer, sizeof Puffer, Stream, "\n")

fgets (Puffer, sizeof Puffer, Stream);
```

Im Puffer wird immer genug Platz für das abschließene Nullzeichen reserviert.

Wenn *Abbruchfolge* ein Nullzeiger ist, wird der Wert von *Abbruchfolge* von dem vorherigen Aufruf verwendet. Wenn *Abbruchfolge* beim ersten Aufruf Null ist, werden keine Begrenzer verwendet.

BEISPIELE

```
#include  <libgen.h>

char Puffer[8];
/* den ersten Benutzernamen aus /etc/passwd einlesen */
fp = fopen("/etc/passwd","r");
bgets(Puffer, 8, fp, ":");
```

ERGEBNIS

Bei einem Fehler oder am Dateiende wird NULL zurückgegeben. Der Fehlerstatus wird erst beim nächsten Aufruf ausgegeben, wenn Zeichen gelesen, aber noch nicht zurückgegeben wurden.

SIEHE AUCH

gets(3S).

bufsplit - Puffer in Felder aufteilen

```
#include <libgen.h>
size_t bufsplit (char *Puffer, size_t n, char **a);
```

`bufsplit` untersucht den *Puffer* und weist dem Zeiger-Array *a* Werte zu, so daß die Zeiger auf die ersten *n* Felder in *Puffer* zeigen, die durch Tabulatoren oder Neue-Zeile-Zeichen voneinander getrennt sind.

Um die Zeichen zum Trennen der Felder zu ändern, rufen Sie `bufsplit` mit *Puffer* auf, der auf die Zeichenkette zeigt, und mit *n* und *a*, die auf Null gesetzt sind. Um zum Beispiel :, . und , als Trennzeichen zusammen mit Tabulator und Neue-Zeile-Zeichen zu verwenden, geben Sie ein:

```
bufsplit (":.,\t\n", 0, (char**)0);
```

BEISPIELE

```
/*
 * set a[0] = "Dies", a[1] = "ist", a[2] = "ein",
 * a[3] = "Test"
 */
bufsplit("Dies\tist\tein\tTest\n", 4, a);
```

HINWEIS

`bufsplit` ändert die Begrenzer in *Puffer* auf Null-Bytes.

ERGEBNIS

Die Anzahl von zugewiesenen Feldern in dem Array *a* wird zurückgegeben. Wenn *Puffer* `NULL` ist, ist der Rückgabewert 0, und das Array bleibt unverändert. Sonst ist der Wert mindestens 1. Den in dem Array verbleibenden Elementen wird die Adresse des Null-Bytes am Ende des Puffers zugewiesen.

copylist - Datei in den Speicher kopieren

```
#include <libgen.h>
char *copylist (const char *Dateiname, off_t *szptr);
```

`copylist` kopiert eine Liste von Zeilen aus einer Datei in neu zugewiesenen Speicher, wobei Neue-Zeile-Zeichen durch Nullzeichen ersetzt werden. `copylist` erwartet zwei Argumente: einen Zeiger *Dateiname* auf den Namen der zu kopierenden Datei und einen Zeiger *szptr* auf eine Variable, in der die Größe der Datei gespeichert wird.

Wenn `copylist` erfolgreich ist, wird ein Zeiger auf den zugewiesenen Speicherbereich zurückgegeben. Ansonsten wird `NULL` zurückgegeben, wenn `copylist` Schwierigkeiten hat, die Datei zu finden, `malloc` aufzurufen oder die Datei zu öffnen.

BEISPIELE

```
/* "Datei" nach Puffer lesen */
off_t Groesse;
char *Puffer;
Puffer = copylist("Datei", &Groesse);
for (i = 0; i < Groesse; i++)
      if(Puffer[i])
            putchar(Puffer[i]);
      else
            putchar('\n');
```

SIEHE AUCH

`malloc`(3C).

dirname - Name des übergeordneten Dateiverzeichnisses ausgeben

```
#include <libgen.h>
char *dirname (char *Pfad);
```

Wenn `dirname` ein Zeiger auf eine mit einem Nullzeichen abgeschlossene Zeichenkette übergeben wird, die einen Pfadnamen des Dateisystems enthält, gibt `dirname` einen Zeiger auf eine `static`-Zeichenkette zurück, die das übergeordnete Verzeichnis dieser Datei bezeichnet. Manchmal fügt `dirname` ein Null-Byte vor das letzte Element ein, daher muß der Inhalt von *Pfad* verfügbar sein. Abschließende /-Zeichen in dem Pfad werden nicht als Teil des Pfads gewertet.

Wenn *Pfad* oder **Pfad* null ist, wird ein Zeiger auf eine static-Konstante '.' zurückgegeben.

`dirname` und `basename` liefern zusammen einen vollständigen Pfadnamen.
`dirname(`*Pfad*`)` ist das Verzeichnis, in dem `basename(`*Pfad*`)` gefunden wird.

BEISPIELE

Ein einfacher Dateiname und die Zeichenketten '.' und '..' haben alle '.' als Rückgabewert.

Eingabefolge	Ausgabefolge
/usr/lib	/usr
/usr/	/
usr	.
/	/
.	.
..	.

Der folgende Programmtext bewirkt, daß ein Pfadname gelesen, in das entsprechende Verzeichnis gewechselt (siehe `chdir`(2)) und die Datei geöffnet wird.

```
char Pfad[100], *Pfadkopie;
int fd;
gets (Pfad);
Pfadkopie = strdup (Pfad);
chdir (dirname (Pfadkopie) );
fd = open (basename (Pfad), O_RDONLY);
```

SIEHE AUCH

`chdir`(2), `basename`(3G).
`basename`(1) in "SINIX V5.41 Kommandos".

gmatch - globaler Mustervergleich in der Shell

```
#include <libgen.h>
int gmatch (const char *str, const char *Muster);
```

`gmatch` überprüft, ob die mit einem Nullzeichen abgeschlossene Zeichenkette *str* mit dem mit einem Nullzeichen abgeschlossenen Zeichenketten-Muster *Muster* übereinstimmt. `gmatch` gibt einen Wert ungleich null zurück, wenn die Zeichenkette auf das Muster paßt, und sonst null. In Muster-Zeichenketten wird ein Gegenschrägstrich `\` als Entwertungszeichen verwendet.

BEISPIEL

```
char *s;
gmatch (s, "*[a\-]" );
```

`gmatch` gibt für alle Zeichenketten mit einem `a` oder `-` als letztes Zeichen einen Wert ungleich null zurück.

SIEHE AUCH

`sh`(1) in "SINIX V5.41 Kommandos".

isencrypt - Zeichen-Puffer-Verschlüsselung feststellen

```
#include <libgen.h>
int isencrypt (const char *fbuf, size_t ninbuf);
```

Mit `isencrypt` kann festgestellt werden, ob ein Zeichen-Puffer verschlüsselt ist.

`isencrypt` nimmt an, daß eine Datei nicht verschlüsselt ist, wenn alle Zeichen im ersten Block ASCII-Zeichen sind. Wenn unter den ersten *ninbuf* Zeichen Nicht-ASCII-Zeichen vorkommen und wenn die Kategorie `setlocale LC_CTYPE` auf `C` oder `ascii` gesetzt ist, wird der Puffer nicht verschlüsselt sein.

Wenn die Kategorie `LC_CTYPE` auf einen anderen Wert als `C` oder `ascii` gesetzt ist, verwendet `isencrypt` eine Heuristik, um eine Verschlüsselung festzustellen. Wenn der Puffer mindestens 64 Zeichen hat, wird ein Chi-Quadrat-Test verwendet, um festzustellen, ob die Bytes im Puffer eine gleichmäßige Verteilung haben; `isencrypt` nimmt an, daß der Puffer verschlüsselt ist, wenn dies der Fall ist. Wenn der Puffer weniger als 64 Zeichen enthält, wird ein Test auf Nullzeichen und ein abschließendes Neue-Zeile-Zeichen gemacht, um festzustellen, ob der Puffer verschlüsselt ist.

ERGEBNIS

Wenn der Puffer verschlüsselt ist, wird 1 zurückgegeben, sonst wird 0 zurückgegeben.

SIEHE AUCH

`setlocale`(3C).

mkdirp, rmdirp - Verzeichnisse in einem Pfad erzeugen/entfernen

```
#include <libgen.h>
int mkdirp (const char *Pfad, mode_t Modus);
int rmdirp (char *Pfad1, char *Pfad2);
```

`mkdirp` erzeugt alle fehlenden Verzeichnisse in dem angegebenen *Pfad* mit dem angegebenen *Modus* (siehe `chmod`(2) für die Werte von *Modus*).

`rmdirp` entfernt Verzeichnisse im Pfad *Pfad1*. Dieses Entfernen beginnt am Ende des Pfads und geht so weit wie möglich zurück bis zur Wurzel. Wenn ein Fehler auftritt, wird der verbleibende Pfad in *Pfad2* gespeichert. `rmdirp` gibt nur dann 0 zurück, wenn die Funktion in der Lage ist, jedes Verzeichnis in dem Pfad zu entfernen.

BEISPIELE

```
/*  Test-Verzeichnisse erzeugen */
if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == -1) {
      fprintf(stderr, "kann kein Verzeichnis erzeugen");
      exit(1);
}
chdir("/tmp/sub1/sub2/sub3");
.
.
.
/* cleanup */
chdir("/tmp");
rmdirp("sub1/sub2/sub3");
```

SIEHE AUCH

`mkdir`(2), `rmdir`(2).

ERGEBNIS

Wenn ein benötigtes Verzeichnis nicht erzeugt werden kann, gibt `mkdirp` -1 zurück und setzt `errno` auf eine der Fehlernummern von `mkdir`. Wenn alle Verzeichnisse erzeugt worden sind, oder schon existieren, wird null zurückgegeben.

HINWEIS

`mkdirp` verwendet `malloc`, um der Zeichenkette temporären Speicherplatz zuzuweisen.

`rmdirp` gibt -2 zurück, wenn ein '.' oder '..' in dem Pfad vorkommt, und -3, wenn ein Versuch gemacht wird, das aktuelle Verzeichnis zu entfernen. Wenn ein anderer als einer der beiden obigen Fehler auftritt, wird -1 zurückgegeben.

p2open, p2close - Pipe zu Kommando öffnen/schließen

```
#include <libgen.h>
int p2open (const char *cmd, FILE *fp[2]);
int p2close (FILE *fp[2]);
```

p2open erzeugt und startet eine Shell-Prozedur, um die Kommandozeile auszuführen, auf die *cmd* zeigt. Bei der Rückkehr zeigt fp[0] auf einen FILE-Zeiger, um auf die Standardeingabe des Kommandos zu schreiben, und fp[1] zeigt auf einen FILE-Zeiger, um von der Standardausgabe des Kommandos zu lesen. Damit hat das Programm die Kontrolle über die Ein- und die Ausgabe des Kommandos.

Wenn die Funktion erfolgreich ist, gibt sie 0 zurück; sonst wird -1 zurückgegeben.

p2close wird verwendet, um die Dateizeiger zu schließen, die p2open geöffnet hat. p2close wartet darauf, daß der Prozeß endet und gibt dann den Prozeßstatus zurück. Bei Erfolg wird 0 zurückgegeben, sonst -1.

BEISPIELE

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

main(argc,argv)
int argc;
char **argv;
{
        FILE *fp[2];
        pid_t pid;
        char buf[16];

        pid=p2open("/usr/bin/cat", fp);
        if ( pid == 0 ) {
                fprintf(stderr, "p2open ist gescheitert\n");
                exit(1);
        }
        write(fileno(fp[0]),"Dies ist ein Test\n", 16);
        if(read(fileno(fp[1]), buf, 16) <=0)
                fprintf(stderr, "p2open ist gescheitert\n");
        else
                write(1, buf, 16);
        (void)p2close(fp);
}
```

SIEHE AUCH

fclose(3S), popen(3S), setbuf(3S).

ERGEBNIS

Häufig sind zu wenig Dateideskriptoren vorhanden. p2close gibt -1 zurück, wenn die beiden Dateizeiger nicht vom selben p2open stammen.

HINWEIS

Bei gepuffertem Schreiben nach `fp[0]` kann es sein, daß das Kommando scheinbar nicht ausgeführt wird. Ebenso kann es problematisch sein, wenn mit einer Pipe verbundene Kommandos eine ungepufferte Ausgabe verwenden. Verwenden Sie `fflush`-Aufrufe oder nicht gepuffertes `fp[0]` (siehe auch `fclose`(3S)).

pathfind - Verzeichnisse nach einer Datei durchsuchen

```
#include <libgen.h>
char *pathfind (const char *Pfad, const char *Name, const char *Modus);
```

`pathfind` durchsucht die in *Pfad* angegebenen Verzeichnisse nach der Datei *Name*. Die in *Pfad* angegebenen Verzeichnisse werden durch Semikolon getrennt. *Modus* ist eine Folge von Optionsbuchstaben, die folgende Bedeutung haben:

Buchstabe	Bedeutung
r	lesbar
w	beschreibbar
x	ausführbar
f	normale Datei
b	blockorientierte Gerätedatei
c	zeichenorientierte Gerätedatei
d	Verzeichnis
p	FIFO (Pipe)
u	Bit der Benutzernummer setzen
g	Bit der Gruppennummer setzen
k	Sticky-Bit
s	Größe ungleich 0

Die Lese-, Schreib- und Ausführoptionen werden bezüglich der realen Benutzer- und Gruppennummer des aktuellen Prozesses überprüft.

Wenn die Datei *Name* mit allen durch *Modus* angegebenen Eigenschaften in irgendeinem der durch *Pfad* angegebenen Verzeichnisse gefunden wird, gibt `pathfind` einen Zeiger auf eine Zeichenkette zurück, die den Inhalt von *Pfad*, gefolgt von einem Schrägstrich `/`, gefolgt von *Name*, enthält.

Wenn *Name* mit einem Schrägstrich beginnt, wird er als absoluter Pfadname betrachtet, und *Pfad* wird ignoriert.

Ein leeres *Pfad*-Element wird als das aktuelle Verzeichnis betrachtet. `./` wird nicht vor den Namen der gefundenen Datei geschrieben; es wird nur der Inhalt von *Name* zurückgegeben.

BEISPIELE

Das `ls`-Kommando mit Hilfe der Umgebungsvariablen `PATH` finden:

```
pathfind (getenv ("PATH"), "ls", "rx")
```

SIEHE AUCH

`access`(2), `mknod`(2), `stat`(2), `getenv`(3C).
`sh`(1), `test`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Wenn keine passende Datei gefunden wird, gibt `pathname` einen Nullzeiger, `((char *) 0)`, zurück.

HINWEIS

Die Zeichenkette, auf die der zurückgegebene Zeiger zeigt, wird in einem `static`-Bereich gespeichert, der bei nachfolgenden Aufrufen von `pathfind` wieder verwendet wird.

regcmp, regex - regulären Ausdruck übersetzen und ausführen

```
#include <libgen.h>
char *regcmp (const char *string1 [, char *string2, ...], (char *)0);
char *regex (const char *re, const char *subject [, char *ret0, ...]);
extern char *__loc1;
```

`regcmp` übersetzt einen regulären Ausdruck, der aus verketteten Argumenten besteht, und gibt einen Zeiger auf die übersetzte Form zurück. `malloc`(3C) wird zur Bereitstellung von Speicherplatz für die übersetzte Form verwendet. Der Benutzer ist für die Freigabe von unbenötigtem Platz verantwortlich. Eine Rückgabe `NULL` von `regcmp` zeigt an, daß ein ungültiges Argument vorliegt.

`regex` sucht ein übersetztes Muster in der Zeichenkette *subject*. Zusätzliche Argumente werden übergeben, um Rückgabewerte zu erhalten. Bei Nichterfolg gibt `regex` `NULL` zurück und bei Erfolg einen Zeiger auf das nächste Zeichen, für das keine Übereinstimmung gefunden wird. Ein globaler Zeichenzeiger `__loc1` weist auf die Stelle, an der die Übereinstimmung beginnt. `regcmp` und `regex` wurden weitgehend vom Editor, `ed`(1) verwendet, wobei Syntax und Semantik jedoch leicht verändert wurden. Die gültigen Symbole und ihre jeweilige Bedeutung sind wie folgt:

`[]*.^` Diese Symbole haben dieselbe Bedeutung wie in `ed`(1).

`$` Dieses Symbol entspricht dem Ende der Zeichenkette (`\n` entspricht einem Neue-Zeile-Zeichen).

`-` Das von Klammern umschlossene Minuszeichen bedeutet *einschließlich*. So ist beispielsweise `[a-z]` gleichbedeutend mit `[abcd...xyz]`. Das `-` kann nur dann für sich selbst stehen, wenn es als das erste oder letzte Zeichen verwendet wird. So paßt beispielsweise der Ausdruck `[]-]` zu den Zeichen `]` und `-`.

`+` Ein regulärer Ausdruck mit nachfolgendem `+` bedeutet *einmal oder mehrere Male*. So ist zum Beispiel `[0-9]+` gleichbedeutend mit `[0-9] [0-9]*`.

`{`*m*`}` `{`*m*`,}` `{`*m*`,`*u*`}`
Mit `{}` umschlossene ganzzahlige Werte zeigen die Häufigkeit an, mit der der vorangehende reguläre Ausdruck angewendet werden soll. Der Wert *m* ist die Mindestanzahl und *u* das Maximum. *u* muß kleiner als 256 sein. Wenn nur *m* vorhanden ist (z.B. `{`*m*`}`), wird damit genau angegeben, wie oft der reguläre Ausdruck angewendet werden soll. Der Wert `{`*m*`,}` ist analog zu `{`*m*`,`*Unendlich*`}`. Die Operationen mit dem Plus-Zeichen `+` und dem Stern `*` sind gleichbedeutend mit `{1,}` bzw. `{0,}`.

`(` ... `)$`*n* Der Wert des geklammerten regulären Ausdrucks soll zurückgegeben wer-

den. Der Wert wird im $(n+1)$ten Argument nach dem Argument *subject* gespeichert. Es sind höchstens zehn geklammerte reguläre Ausdrücke zulässig. `regex` führt die Zuweisungen auf jeden Fall aus.

`( ... )` Für Gruppierungen werden Klammern verwendet. Ein Operator, z.B. `*`, `+`, `{}`, kann auf Einzelzeichen oder auf einen von Klammern umschlossenen regulären Ausdruck angewendet werden. Beispiel: `(a*(cb+)*)$0`.

Notwendigerweise sind alle oben definierten Symbole Sonderzeichen. Daher müssen sie mit einem Gegenschrägstrich `\` gekennzeichnet werden, wenn sie für sich stehen sollen.

BEISPIELE

Das folgende Beispiel sucht ein führendes Neue-Zeile-Zeichen in der Zeichenkette `subject`, auf die `cursor` zeigt.

```
char *cursor, *newcursor, *ptr;
 ...
newcursor = regex((ptr = regcmp("^\n", (char *)0)), cursor);
free(ptr);
```

Das folgende Beispiel sucht nach der Zeichenkette `Testing3` und gibt die Adresse des Zeichens hinter dem letzten passenden Zeichen (dem Zeichen `4`) zurück. Die Zeichenkette `Testing3` wird in das Zeichenfeld `ret0` kopiert.

```
char ret0[9];
char *newcursor, *name;
 ...
name = regcmp("([A-Za-z][A-za-z0-9]{0,7})$0", (char *)0);
newcursor = regex(name, "012Testing345", ret0);
```

Bei diesem Beispiel wird ein vorübersetzter regulärer Ausdruck in `file.i` (siehe `regcmp`(1)) gegen *string* geprüft.

```
#include "file.i"
char *string, *newcursor;
 ...
newcursor = regex(name, string);
```

SIEHE AUCH

`regcmp`(1), `malloc`(3C).
`ed`(1) in "SINIX V5.41 Kommandos".

HINWEIS

Das Benutzerprogramm kann möglicherweise keinen Speicherplatz mehr zur Verfügung stellen, wenn `regcmp` iterativ ohne Freigabe der nicht mehr benötigten Vektoren aufgerufen wird.

regexpr - regulären Ausdruck übersetzen und anwenden

```
#include <regexpr.h>
char *compile (const char *instring, char *expbuf, char *endbuf);
int step (const char *string, char *expbuf);
int advance (const char *string, char *expbuf);
extern char *loc1, *loc2, *locs;
extern int nbra, regerrno, reglength;
extern char *braslist[], *braelist[];
```

Diese Routinen werden verwendet, um reguläre Ausdrücke zu übersetzen und die übersetzten Ausdrücke mit Zeilen zu vergleichen. Die übersetzten regulären Ausdrücke haben die von `ed` verwendete Form.

`compile` wird wie folgt verwendet:

Der Parameter *instring* ist eine mit einem Nullzeichen abschließende Zeichenkette, die einen regulären Ausdruck repräsentiert.

Der Parameter *expbuf* zeigt auf den Speicherplatz, an den der übersetzte reguläre Ausdruck gebracht werden soll. Wenn *expbuf* `NULL` ist, verwendet `compile` `malloc`, um Speicherplatz für den übersetzten regulären Ausdruck zuzuweisen. Wenn ein Fehler auftritt, wird dieser Platz wieder freigegeben. Es liegt in der Verantwortung des Benutzers, den Speicherplatz freizugeben, wenn der übersetzte reguläre Ausdruck nicht länger benötigt wird.

Der Parameter *endbuf* zeigt auf einen Speicherplatz über der höchsten Adresse, an der der übersetzte reguläre Ausdruck untergebracht werden kann. Dieses Argument wird ignoriert, wenn *expbuf* `NULL` ist. Wenn der übersetzte Ausdruck nicht in
(*endbuf* - *expbuf*)
Bytes paßt, gibt `compile` `NULL` zurück, und `regerrno` (siehe unten) wird auf 50 gesetzt.

Wenn `compile` Erfolg hat, gibt es einen Zeiger ungleich `NULL` zurück, dessen Wert von *expbuf* abhängt. Wenn *expbuf* ungleich `NULL` ist, gibt `compile` einen Zeiger auf das Byte hinter dem letzten Byte in dem übersetzten regulären Ausdruck zurück. Die Länge des übersetzten regulären Ausdrucks wird in `reglength` gespeichert. Sonst gibt `compile` einen Zeiger auf den von `malloc` zugewiesenen Speicherplatz zurück.

Wenn beim Übersetzen des regulären Ausdrucks ein Fehler entdeckt wird, wird von `compile` ein Nullzeiger zurückgegeben, und `regerrno` wird auf einen der unten angezeigten Fehlernummern ungleich Null gesetzt:

FEHLER	BEDEUTUNG
11	zu großer Endpunkt des Bereichs
16	falsche Zahl
25	'\Ziffer' außerhalb des Gültigkeitsbereich
36	ungültiger oder fehlender Begrenzer
41	keine gespeicherte Suchfolge
42	\(\) nicht ausgeglichen
43	zu viele \(
44	mehr als 2 Zahlen in \{ \}\
45	} nach \ erwartet
46	erste Zahl ist größer als zweite in \{ \}.
49	[ ] nicht ausgeglichen
50	Überlauf des regulären Ausdrucks

`step` wird wie folgt verwendet:
Der erste Parameter von `step` ist ein Zeiger auf eine Zeichenkette, die verglichen werden soll. Die Zeichenkette sollte mit einem Nullzeichen abschließen.

Der Parameter *expbuf* ist der übersetzte reguläre Ausdruck, der durch den Aufruf der Funktion `compile` erhalten wird.

Die Funktion `step` gibt einen Wert ungleich Null zurück, wenn die übergebene Zeichenkette auf den regulären Ausdruck paßt, und Null, wenn das nicht der Fall ist. Wenn der Vergleich paßt, werden zwei externe Zeichen-Zeiger als Nebeneffekt des `step`-Aufrufs gesetzt. Die in `step` gesetzte Variable ist `loc1`. `loc1` ist ein Zeiger auf das erste Zeichen, das auf den regulären Ausdruck paßt. Die Variable `loc2` zeigt auf das Zeichen hinter dem letzten Zeichen, das auf den regulären Ausdruck paßt. Wenn also der reguläre Ausdruck auf die ganze Zeile paßt, zeigt `loc1` auf das erste Zeichen von *string*, und `loc2` zeigt auf das Nullzeichen am Ende von *string*.

Der Zweck von `step` besteht darin, schrittweise durch das Argument *string* zu laufen, bis ein passendes Zeichen gefunden wird oder bis das Ende von *string* erreicht wird. Wenn der reguläre Ausdruck mit ^ beginnt, versucht `step` den Vergleich mit dem regulären Ausdruck nur am Anfang der Zeichenkette.

Die Funktion `advance` hat dieselben Argumente und Nebeneffekte wie `step`, aber diese Funktion schränkt den Vergleich immer auf den Anfang der Zeichenkette ein.

Wenn man nach aufeinanderfolgenden Mustern in derselben Zeichenkette sucht, sollte `locs` auf denselben Wert wie `loc2` gesetzt werden, und `step` sollte mit *string* gleich `loc2` aufgerufen werden. `locs` wird von Kommandos wie `ed` und `sed` verwendet, so daß globale Ersetzungen wie `s/y*//g` nicht endlos laufen, und ist standardmäßig `NULL`.

Die externe Variable nbra wird verwendet, um die Anzahl von Unterausdrücken in dem übersetzten regulären Ausdruck zu bestimmen. braslist und braelist sind Felder von Zeigern auf Zeichen, die auf den Anfang und das Ende der nbra-Unterausdrücke in der passenden Zeichenkette zeigen. So zeigt zum Beispiel nach dem Aufruf von step oder advance mit der Zeichenkette sabcdefg und dem regulären Ausdruck \(abcdef\), braslist[0] auf a und braelist[0] auf g. Diese Felder werden von Kommandos wie ed und sed verwendet, um wiederholte Ersetzungen mit Mustern durchzuführen, die die Notation \n für Unterausdrücke enthalten.

Beachten Sie, daß es nicht notwendig ist, die externen Variablen regerrno, nbra, loc1, loc2 locs, braelist und braslist zu verwenden, wenn man nur überprüfen möchte, ob eine Zeichenkette auf einen regulären Ausdruck paßt oder nicht.

BEISPIELE

Folgender Programmauszug entspricht dem regulären Ausdruck von grep:

```
#include <regexpr.h>

. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
      regerr(regerrno);
. . .
if (step(linebuf, expbuf))
      succeed();
```

SIEHE AUCH

regexp(5).
ed(1), grep(1), sed(1) in "SINIX V5.41 Kommandos".

str: strfind, strrspn, strtrns - Zeichenketten ändern

```
#include <libgen.h>
int strfind (const char *as1, const char *as2);
char *strrspn (const char *string, const char *tc);
char * strtrns (const char *str, const char *alt, const char *neu, char *Ergebnis);
```

`strfind` gibt den Offset der zweiten Zeichenkette *as2* an, wenn *as2* eine Teilfolge von *as1* ist.

`strrspn` gibt einen Zeiger auf das erste Zeichen in der Zeichenkette, die beschnitten werden soll, zurück (alle Zeichen vom ersten bis zum letzten von *string* sind in *tc*).

`strtrns` wandelt `str` um und kopiert diese Folge nach *Ergebnis*. Alle Zeichen, die in *alt* vorkommen, werden durch das Zeichen in derselben Position in *neu* ersetzt. *neu* wird als Ergebnist zurückgegeben.

BEISPIELE

```
/* einen Zeiger auf die Teilfolge "hello" in as1 finden */
i = strfind(as1, "hello");

/* Müll vom Ende der Zeichenkette abtrennen */
s2 = strrspn(s1, "*?#$%");
*s2 = '\0'

/* Klein- in Großschrift umwandeln */
a1[] = "abcdefghijklmnopqrstuvwxyz";
a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
s2 = strtrns(s1, a1, a2, s2);
```

SIEHE AUCH

`string`(3C).

ERGEBNIS

Wenn die zweite Zeichenkette keine Teilfolge der ersten ist, gibt `strfind` -1 zurück.

strccpy, streadd, strcadd, strecpy - Zeichenketten kopieren

```
#include <libgen.h>
char *strccpy (char *Ausgabe, const char *Eingabe);
char *strcadd (char *Ausgabe, const char *Eingabe);
char *strecpy (char *Ausgabe, const char *Eingabe, const char *Ausnahmen);
char *streadd (char *Ausgabe, const char *Eingabe, const char *Ausnahmen);
```

`strccpy` kopiert die Zeichenkette *Eingabe* bis zu einem Null-Byte in die Zeichenkette *Ausgabe* und komprimiert die Escape-Sequenzen der Sprache C (zum Beispiel `\n`, `\001`) in das äquivalente Zeichen. An die Ausgabe wird ein Null-Byte angehängt. Das Argument *Ausgabe* muß auf einen Speicherplatz zeigen, der groß genug ist, um das Ergebnis aufzunehmen. Wenn er so groß ist wie der Speicherplatz, auf den *Eingabe* zeigt, ist er mit Sicherheit groß genug. `strccpy` gibt das Argument *Ausgabe* zurück.

`strcadd` ist genau wie `strccpy`, nur daß hier ein Zeiger auf das Null-Byte, das die Ausgabe abschließt, zurückgegeben wird.

`strecpy` kopiert die Zeichenkette *Eingabe* bis zu einem Null-Byte zur Zeichenkette *Ausgabe* und erweitert nichtgrafische Zeichen in ihre äquivalenten Escape-Sequenzen der Sprache C (zum Beispiel `\n`, `\001`). Das Argument *Ausgabe* muß auf einen Speicherplatz zeigen, der groß genug ist, um das Ergebnis aufzunehmen; ein Speicherplatz der viermal so groß ist wie der, auf den *Eingabe* zeigt, ist mit Sicherheit groß genug (schlimmstenfalls kann aus jedem Zeichen `\` mit drei Ziffern werden). Zeichen in der Zeichenkette *Ausnahmen* werden nicht erweitert. Das Argument *Ausnahmen* kann `NULL` sein, das heißt, daß alle nicht-grafischen Zeichen erweitert werden. `strecpy` gibt das Argument *Ausgabe* zurück.

`streadd` entspricht `strecpy`, nur daß hier ein Zeiger auf das Null-Byte zurückgegeben wird, das die Ausgabe abschließt.

BEISPIELE

```
/* alle Zeichen außer Neue-Zeile und Tabulator erweitern */
strecpy( Ausgabe, Eingabe, "\n\t");

/* einige Zeichenketten zusammenfügen und komprimieren */
cp = strcadd( Ausgabe, Eingabe1 );
cp = strcadd( cp, Eingabe2 );
cp = strcadd( cp, Eingabe3 );
```

SIEHE AUCH

`string`(3C), `str`(3G).

intro - Einführung in die mathematische Bibliothek

Dieser Abschnitt beschreibt die Funktionen der mathematischen Bibliothek `libm`. Deklarationen für diese Funktionen erhalten Sie, indem Sie die Datei `math.h` mit der Anweisung `#include` in Ihren Quelltext einbinden. Dort sind auch einige allgemein nützliche Konstanten definiert (siehe `intro`(3) und `math`(5)).

Die mathematische Bibliothek wird nicht automatisch vom C-Entwicklungssystem dazugebunden. Benutzen Sie die Option `-l` des Kommandos `cc`, um die oben angegebenen Bibliotheken zu verwenden.

`libm` besteht aus einem vollständigen Satz von Routinen mit doppelter Genauigkeit und einigen Routinen mit einfacher Genauigkeit (gekennzeichnet durch ein nachgestelltes `f`). Die Routinen mit einfacher Genauigkeit benötigen weniger Rechenzeit bei einer geringeren Genauigkeit des Ergebnisses. Einige ausgewählte Routinen sind auf minimale Laufzeit optimiert. Die optimierten Routinen sind: `sin`, `cos`, `tan`, `atan`, `atan2`, `exp`, `log`, `log10`, `pow` und `sqrt`, sowie deren Entsprechungen mit einfacher Genauigkeit.

Unter `intro`(3) finden Sie Definitionen zur Sprache C.

DATEIEN

LIBDIR	üblicherweise `/usr/ccs/lib`
LIBDIR`/libm.a`	

SIEHE AUCH

`cc`(1), `intro`(2), `intro`(3), `math`(5).
Abschnitt "Der Binder" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ERGEBNIS

Die Fehlerbehandlung ist abhängig vom verwendeten Übersetzungsmodus. Im K&R-Modus (`cc -kcc`) liefern diese Funktionen die vereinbarten Werte `0`, `±HUGE` oder `NaN`, wenn die Funktion für die gegebenen Argumente nicht definiert oder das Ergebnis nicht darstellbar ist. Im ANSI-Modus (`cc -kansi`) wird `±HUGE_VAL` anstelle von `±HUGE` verwendet. (`HUGE_VAL` und `HUGE` sind definiert in `math.h`, als Darstellung für 'Unendlich' bzw. für den größten darstellbaren Wert einer Zahl mit einfacher Genauigkeit.) In jedem Falle wird die externe Variable `errno` (siehe `intro`(2)) auf den Wert `EDOM` oder `ERANGE` gesetzt; der Wert kann dabei jedoch abhängig vom gewählten Übersetzungsmodus variieren. Siehe dazu auch die Tabelle unter `matherr`(3M) weiter unten.

bessel - Bessel-Funktionen

```
#include <math.h>
double j0 (double x);
double j1 (double x);
double jn (int n, double x);
double y0 (double x);
double y1 (double x);
double yn (int n, double x);
```

`j0`, `j1` und `jn` liefern Bessel-Funktionen der ersten Art für x der Ordnungen 0, 1 und n.

`y0`, `y1` und `yn` liefern Bessel-Funktionen der zweiten Art für x der Ordnungen 0, 1 und n. Der Wert von x muß positiv sein.

SIEHE AUCH

`matherr`(3M).

ERGEBNIS

Nicht positive Argumente bewirken, daß `y0`, `y1` und `yn` den Wert `-HUGE` zurückgeben und `errno` auf `EDOM` setzen. Außerdem wird auf der Standard-Fehlerausgabe eine Meldung ausgegeben, die den `DOMAIN`-Fehler anzeigt.

Argumente, die zu groß sind, bewirken, daß `j0`, `j1`, `y0` und `y1` null zurückgeben und `errno` auf `ERANGE` setzen. Außerdem wird eine Meldung zur Anzeige des Fehlers `TLOSS` auf der Standard-Fehlerausgabe ausgegeben.

Im K&R-Modus (`cc -kcc`) kann diese Fehlerbehandlung mit der Funktion `matherr` geändert werden. Im ANSI-Modus (`cc -kansi`) wird `HUGE_VAL` anstelle von `HUGE` zurückgegeben, und es wird keine Fehlermeldung ausgegeben.

erf, erfc - Fehlerfunktion und komplementäre Fehlerfunktion

```
#include <math.h>
double erf (double x);
double erfc (double x);
```

`erf` liefert den Wert der Fehlerfunktion von x, der folgendermaßen definiert ist:

$$\frac{2}{\sqrt{\Pi}} \int_0^x e^{-t^2}\, dt$$

Die Funktion `erfc`, die den Wert 1.0 - `erf(`x`)` liefert, wird wegen des großen Genauigkeitsverlusts zur Verfügung gestellt, der entsteht, wenn `erf(`x`)` mit großen Werten aufgerufen und das Ergebnis später von 1.0 subtrahiert wird (für x = 5 gehen 12 Stellen verloren).

SIEHE AUCH

`exp`(3M).

exp - Exponential-, Logarithmus-, Potenz-, Quadratwurzelfunktionen

```
#include <math.h>
double exp (double x);
float expf (float x);
double cbrt (double x);
double log (double x);
float logf (float x);
double log10 (double x);
float log10f (float x);
double pow (double x, double y);
float powf (float x, float y);
double sqrt (double x);
float sqrtf (float x);
```

`exp` und `expf` geben e^x zurück.

`cbrt` gibt die Kubikwurzel von x zurück.

`log` und `logf` geben den natürlichen Logarithmus von x zurück. Der Wert von x muß positiv sein.

`log10` und `log10f` geben den Logarithmus zur Basis 10 von x zurück. Der Wert von x muß positiv sein.

`pow` und `powf` geben x^y zurück. Wenn x gleich 0 ist, muß y positiv sein. Wenn x negativ ist, muß y eine ganze Zahl sein.

`sqrt` und `sqrtf` geben die nichtnegative Quadratwurzel von x zurück. Der Wert von x darf nicht negativ sein.

SIEHE AUCH

`hypot`(3M), `matherr`(3M), `sinh`(3M).

ERGEBNIS

`exp` und `expf` geben `HUGE` zurück, wenn der Wert zu einem Überlauf führen würde oder 0, wenn der Wert einen Unterlauf ergeben würde. `errno` wird auf `ERANGE` gesetzt.

`log`, `logf`, `log10` und `log10f` geben `-HUGE` zurück und setzen `errno` auf `EDOM`, wenn x nicht positiv ist. In diesem Fall wird eine Nachricht, die einen `DOMAIN`-Fehler anzeigt, auf der Standard-Fehlerausgabe ausgegeben.

`pow` und `powf` geben 0 zurück und setzen `errno` auf `EDOM`, wenn x 0 ist und y nicht positiv ist, oder wenn x negativ und y keine ganze Zahl ist. In diesen Fällen wird eine Nachricht auf der Standard-Fehlerausgabe ausgegeben, die einen `DOMAIN`-Fehler anzeigt. Wenn der korrekte Wert für `pow` oder `powf` zu einem Überlaufen oder zu einem Unterlaufen führen würde, geben diese Funktionen `±HUGE` bzw. 0 zurück und setzen `errno` auf `ERANGE`.

`sqrt` und `sqrtf` geben 0 zurück und setzen `errno` auf `EDOM`, wenn x negativ ist. Eine Nachricht, die den Fehler `DOMAIN` anzeigt, wird in diesem Fall auf der Standard-Fehlerausgabe ausgegeben.

Im K&R-Modus (`cc -kcc`) kann diese Fehlerbehandlung mit der Funktion `matherr` geändert werden. Im ANSI-Modus (`cc -kansi`) wird `HUGE_VAL` statt `HUGE` zurückgegeben, und es werden keine Fehlermeldungen ausgegeben. `pow` und `powf` melden 1 bei keinem Fehler, wenn x und y 0 sind; wenn x 0 ist und y negativ, geben sie `-HUGE_VAL` zurück und setzen `errno` auf `EDOM`. Im K&R-Modus geben `log` und `logf` `-HUGE_VAL` zurück und setzen `errno` auf `ERANGE`, wenn x 0 ist. `sqrt` und `sqrtf` geben `NaN` (not a number) zurück, wenn x negativ ist.

floor - Abrunden, Aufrunden, Rest bei Division, Absolutbetrag

```
#include <math.h>
double floor (double x);
float floorf (float x);
double ceil (double x);
float ceilf (float x);
double copysign (double x, double y);
double fmod (double x, double y);
float fmodf (float x, float y);
double fabs (double x);
float fabsf (float x);
double rint (double x);
double remainder (double x, double y);
```

`floor` und `floorf` geben den größten ganzzahligen Wert (als Zahl vom Typ `double` bzw. `float`) zurück, der nicht größer als x ist.

`ceil` und `ceilf` geben die kleinste ganze Zahl zurück, die nicht kleiner als x ist.

`copysign` gibt x mit dem Vorzeichen von y zurück.

`fmod` und `fmodf` geben den Gleitkomma-Restwert der Division von x durch y zurück. Genauer gesagt, geben sie die Zahl f mit dem gleichen Vorzeichen wie x zurück, so daß $x = iy + f$ für eine Ganzzahl i und $|f| < |y|$.

`fabs` und `fabsf` geben den Absolutwert von x zurück, $|x|$.

`rint` gibt den Ganzzahlwert, der am nächsten an seinem Gleitkomma-Argument x liegt, als Gleitkommazahl mit doppelter Genauigkeit zurück. Der zurückgegebene Wert wird entsprechend dem aktuell gesetzten Rundungsmodus des Rechners gerundet. Wenn der Standardmodus 'round-to-nearest' gesetzt ist und die Differenz zwischen dem Funktionsargument und dem gerundeten Ergebnis genau 0.5 ist, wird das Resultat auf die nächste gerade Ganzzahl gerundet.

`remainder` gibt den Gleitkommarest der Division x durch y zurück. Genauer gesagt, gibt es den Wert $r = x - yn$ zurück, wobei n die ganze Zahl ist, die am dichtesten beim exakten Wert x/y liegt. Wann immer $|n - x/y| = {}^1/_2$, ist n gerade.

SIEHE AUCH

`abs`(3C), `matherr`(3M).

ERGEBNIS

`fmod` und `fmodf` geben x zurück, wenn y 0 ist, und setzen `errno` auf `EDOM`. `remainder` gibt `NaN` zurück, wenn y 0 ist, und setzt `errno` auf `EDOM`. Im K&R-Modus (`cc -kcc`) wird eine Meldung auf der Standard-Fehlerausgabe ausgegeben, die einen `DOMAIN`-Fehler anzeigt. Diese Fehlerbehandlung kann mit der Funktion `matherr` geändert werden.

gamma, lgamma - Logarithmus der Gammafunktion

```
#include <math.h>
double gamma (double x);
double lgamma (double x);
extern int signgam;
```

`gamma` und `lgamma` berechnen `ln(| gamma(x) |)`

`gamma(x)` ist definiert als

$$\int_0^{\infty} e^{-t}\ t^{x-1}\ dt$$

Das Vorzeichen von gamma (x) wird in der externen Ganzzahl `signgam` zurückgegeben. Das Argument x muß eine positive ganze Zahl sein.

Der nachstehende C-Programmteil kann zur Berechnung von gamma verwendet werden:

```
if ((y = gamma(x)) > LN_MAXDOUBLE)
 error();
y = signgam * exp(y);
```

`LN_MAXDOUBLE` ist hierbei der niedrigste Wert, der dazu führt, daß `exp` einen Bereichsfehler zurückgibt. Dieser Wert ist in der Include-Datei `values.h` definiert.

SIEHE AUCH

`exp`(3M), `matherr`(3M), `values`(5).

ERGEBNIS

Für nicht positive Ganzzahlargumente wird `HUGE` zurückgegeben, und `errno` wird auf `EDOM` gesetzt. Auf der Standard-Fehlerausgabe wird eine Meldung ausgegeben, die den Fehler `SING` anzeigt.

Wenn der korrekte Wert zu einem Überlauf führen würde, geben `gamma` und `lgamma` `HUGE` zurück und setzen `errno` auf `ERANGE`.

Im K&R-Modus (`cc -kcc`) kann die Fehlerbehandlung mit der Funktion `matherr` geändert werden. Im ANSI-Modus (`cc -kansi`) wird `HUGE_VAL` anstelle von `HUGE` zurückgeliefert, und es wird keine Fehlermeldung ausgegeben.

hypot - euklidischer Abstand

```
#include <math.h>
double hypot (double x, double y);
```

`hypot` berechnet

```
sqrt(x * x + y * y)
```

Unerwünschte Überläufe werden dabei vermieden.

SIEHE AUCH

`matherr`(3M).

ERGEBNIS

Wenn der korrekte Wert ein Überlaufen ergeben würde, liefert `hypot` `HUGE` zurück und setzt `errno` auf `ERANGE`.

Im K&R-Modus (`cc -kcc`) kann die Fehlerbehandlung mit der Funktion `matherr` geändert werden. Im ANSI-Modus (`cc -kansi`) wird gegebenenfalls `HUGE_VAL` anstelle von `HUGE` zurückgeliefert.

matherr - Fehlerbehandlungsfunktion

```
#include <math.h>
int matherr (struct exception *x);
```

matherr wird von Funktionen der mathematischen Bibliothek zur Festellung von Fehlern aufgerufen. Beachten Sie, daß matherr nicht aufgerufen wird, wenn die Übersetzeroption -kansi verwendet wird. Die Benutzer können eigene Prozeduren zur Behandlung von Fehlern definieren, indem sie eine Funktion mit dem Namen matherr in den jeweiligen Programme vorsehen. matherr muß die obenstehende Form aufweisen. Bei Auftreten eines Fehlers wird ein Zeiger auf die Struktur *x* zur Beschreibung der Ausnahmesituation an die vom Benutzer bereitgestellte Funktion matherr weitergereicht. Diese Struktur, die in der Include-Datei math.h definiert ist, hat folgende Form:

```
struct exception {
      int type;
      char *name;
      double arg1, arg2, retval;
};
```

Das Element type ist eine Zahl vom Typ int, die den Typ des aufgetretenen Fehlers anhand der nachstehenden Liste (in der Include-Datei definiert) von Konstanten beschreibt:

DOMAIN	Argument außerhalb des Definitionsbereichs
SING	Singularität des Arguments
OVERFLOW	Überlauf des Ergebnisses
UNDERFLOW	Unterlauf des Ergebnisses
TLOSS	totaler Verlust der Genauigkeit
PLOSS	teilweiser Verlust der Genauigkeit

Das Element name weist auf eine Zeichenkette mit dem Namen der Funktion, in der der Fehler aufgetreten ist. Die Variablen arg1 und arg2 sind die Argumente, mit denen die Funktion aufgerufen wurde. retval wird auf den Standardwert gesetzt, der von der Funktion zurückgegeben wird, wenn es nicht von der Funktion matherr auf einen anderen Wert gesetzt wird.

Wenn die Funktion matherr einen Wert ungleich null zurückgibt, werden keine Fehlermeldungen ausgegeben, und errno wird nicht gesetzt.

Wenn matherr nicht vom Benutzer bereitgestellt wird, dann werden bei Auftreten eines Fehlers die bei den betroffenen mathematischen Funktionen beschriebenen Standard-Fehlerbehandlungs-Prozeduren aufgerufen. Diese Prozeduren sind in der nachstehenden Tabelle zusammengefaßt. In jedem Fall wird errno auf EDOM oder ERANGE gesetzt, und das Programm läuft weiter.

Standardprozeduren für Fehlerbehandlung						
	Fehlertypen					
Typ	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE	ERANGE
BESSEL: y0, y1, yn(arg<0)	– M, -H	– –	– –	– –	M, 0 –	– –
EXP, EXPF:	–	–	H	0	–	–
LOG, LOG10: LOGF, LOG10F: (arg < 0) (arg = 0)	 M, -H M, -H	 – –	 – –	 – –	 – –	 – –
POW, POWF: neg ** nicht-int 0 ** nicht-pos	– M, 0 M, 0	– – –	±H – –	0 – –	– – –	– – –
SQRT, SQRTF:	M, 0	–	–	–	–	–
FMOD, FMODF: (arg2 = 0)	 M, X	 –	 –	 –	 –	 –
REMAINDER: (arg2 = 0)	 M, N	 –	 –	 –	 –	 –
GAMMA, LGAMMA:	–	M, H	H	–	–	–
HYPOT:	–	–	H	–	–	–
SINH, SINHF:	–	–	±H	–	–	–
COSH, COSHF:	–	–	H	–	–	–
ASIN, ACOS, ATAN2: ASINF, ACOSF, ATAN2F:	 M, 0	 –	 –	 –	 –	 –
ACOSH:	M, N	–	–	–	–	–
ATANH: (|arg| > 1) (|arg| = 1)	 M, N –	 – M, N	 – –	 – –	 – –	 – –

Abkürzungen

M	Meldung wird ausgegeben (nicht im ANSI-Modus).
H	HUGE wird zurückgegeben (HUGE_VAL im ANSI-Modus)
–H	–HUGE wird zurückgegeben (–HUGE_VAL im ANSI-Modus)
±H	HUGE oder –HUGE wird zurückgegeben (HUGE_VAL oder –HUGE_VAL im ANSI-Modus).
0	0 wird zurückgegeben.
X	*arg1* wird zurückgegeben.
N	NaN wird zurückgegeben.

BEISPIEL

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

int
matherr(register struct exception *x);
{
      switch (x->type) {
      case DOMAIN:
              /* sqrt verändern, so daß sqrt(-arg1) zurückgegeben wird,*/
              /* nicht 0 */
              if (!strcmp(x->name, "sqrt")) {
                      x->retval = sqrt(-x->arg1);
                      return (0); /* Meldung ausgeben und errno setzen */
              }
      case SING:
              /* Bei allen anderen DOMAIN- oder SING-Fehlern, Meldung */
              /* ausgeben und abbrechen (abort) */
              fprintf(stderr, "Gültigkeitsbereichsfehler in %s\n", x->name);
              abort( );
      case PLOSS:
              /* detaillierte Fehlermeldung ausgeben */
              fprintf(stderr, "Genauigkeitsverlsut in %s(%g)=%g\n"
                      x->name, x->arg1, x->retval);
              return (1); /* keine weiteren Maßnahmen ergreifen */
      }
      return (0); /* bei allen anderen Fehlern die Standardprozedur */
                  /* ausführen */
}
```

sinh - Hyperbel-Funktionen

```
#include <math.h>
double sinh (double x);
float sinhf (float x);
double cosh (double x);
float coshf (float x);
double tanh (double x);
float tanhf (float x);
double asinh (double x);
double acosh (double x);
double atanh (double x);
```

`sinh`, `cosh` und `tanh` geben jeweils den Hyperbel-Sinus, -Kosinus bzw. -Tangens ihres Arguments zurück. `asinh`, `acosh` und `atanh` geben jeweils den inversen Hyperbel-Sinus, den inversen -Kosinus bzw. den inversen -Tangens ihres Arguments zurück.

SIEHE AUCH

`matherr`(3M).

ERGEBNIS

`sinh`, `sinhf`, `cosh` und `coshf` geben `HUGE` zurück (und `sinh` und `sinhf` können bei negativem x `-HUGE` zurückgeben, wenn der korrekte Wert überlaufen würde, und setzen `errno` auf `ERANGE`).

`acosh` gibt `NaN` zurück und setzt `errno` auf `EDOM`, wenn das Argument x kleiner als 1 ist. Eine Meldung, die einen `DOMAIN`-Fehler anzeigt, wird auf der Standard-Fehlerausgabe ausgegeben.

`atanh` gibt `NaN` zurück und setzt `errno` auf `EDOM`, wenn $|x| \geq 1$. Wenn $|x| = 1$, wird eine Meldung auf der Standard-Fehlerausgabe ausgegeben, die einen `SING`-Fehler anzeigt; wenn $|x| > 1$ zeigt die Meldung einen `DOMAIN`-Fehler an.

Im K&R-Modus (`cc -kcc`) kann diese Fehlerbehandlung mit der Funktion `matherr` verändert werden. Im ANSI-Modus (`cc -kansi`) wird `HUGE_VAL` statt `HUGE` zurückgegeben, und es wird keine Fehlermeldung ausgegeben.

trig - Trigonometriefunktionen

```
#include <math.h>
double sin (double x);
float sinf (float x);
double cos (double x);
float cosf (float x);
double tan (double x);
float tanf (float x);
double asin (double x);
float asinf (float x);
double acos (double x);
float acosf (float x);
double atan (double x);
float atanf (float x);
double atan2 (double y, double x);
float atan2f (float y, float x);
```

`sin`, `cos`, `tan` und die Versionen mit einfacher Genauigkeit `sinf`, `cosf` und `tanf` geben jeweils den Sinus, Kosinus bzw. Tangens ihres Arguments x, gemessen in Radian, zurück.

`asin` und `asinf` geben den Arkussinus von x im Bereich $(-\pi/2, +\pi/2)$ zurück.

`acos` und `acosf` geben den Arkuskosinus von x im Bereich $(0, +\pi)$ zurück.

`atan` und `atanf` geben den Arkustangens von x im Bereich $(-\pi/2, +\pi/2)$ zurück.

`atan2` und `atan2f` geben den Arkustangens von y/x im Bereich $(-\pi, +\pi)$ zurück, wobei sie die Vorzeichen von beiden Argumenten verwenden, um den Quadranten des Rückgabewerts zu bestimmen.

SIEHE AUCH

`matherr`(3M).

ERGEBNIS

Wenn die Höhe des Arguments von `asin`, `asinf`, `acos` oder `acosf` 1 übersteigt, oder wenn beide Argumente von `atan2` oder `atan2f` 0 sind, wird 0 zurückgegeben, und `errno` wird auf `EDOM` gesetzt. Zusätzlich wird eine Meldung auf die Standard-Fehlerausgabe geschrieben, die einen `DOMAIN`-Fehler anzeigt.

Im K&R-Modus (`cc -kcc`) kann diese Fehlerbehandlung mit der Funktion `matherr` verändert werden. Im ANSI-Modus (`cc -kansi`) werden keine Fehlermeldungen ausgegeben.

assert - Zusicherung im Programm überprüfen

```
#include <assert.h>
void assert (int expression);
```

Dieses Makro ist für das Einbinden einer Diagnose in das Programm nützlich. Wenn bei der Ausführung der Anweisung der Ausdruck *expression* den Wert null ergibt, gibt `assert` die Meldung

'`Assertion failed:` *expression*, `file` *xyz*, `line` *nnn*'

auf der Standard-Fehlerausgabe aus und bricht das Programm ab. In der Fehlermeldung ist *xyz* der Name der Quelldatei und *nnn* die Zeilennummer der Anweisung. Für diese Werte werden die Präprozessor-Makros `__FILE__` und `__LINE__` verwendet.

Durch Übersetzen mit der Präprozessor-Steueranweisung `#define NDEBUG` vor der Anweisung `#include assert.h` kann verhindert werden, daß solche Überprüfungen mit übersetzt werden.

SIEHE AUCH

`abort`(3C).

HINWEIS

Da `assert` als Makro realisiert ist, darf der Ausdruck *expression* keine konstanten Zeichenketten enthalten.

crypt - Verschlüsselungsfunktionen für Paßwörter und Dateien

```
#include <crypt.h>
char *crypt (const char *key, const char *salt);
void setkey (const char *key);
void encrypt (char *block, int flag);
char *des_crypt (const char *key, const char *salt);
void des_setkey (const char *key);
void des_encrypt (char *block, int flag);
int run_setkey (int *p, const char *key);
int run_crypt (long offset, char *buffer, unsigned int count, int *p);
int crypt_close(int *p);
```

`des_crypt` ist die Paßwort-Verschlüsselungsfunktion. Sie beruht auf einem Einweg-Hash-Algorithmus mit Variationen, die (unter anderem) die Verwendung von Hardware-Implementierungen für Schlüsselsuchverfahren verhindern sollen.

key ist das eingegebene Paßwort eines Benutzers. *salt* ist eine Zeichenkette der Länge zwei aus [`a-zA-Z0-9./`]. Diese Zeichenkette wird zur Veränderung des Verschlüsselungsalgorithmus auf eine von 4096 Arten verwendet; danach wird das Paßwort als Schlüssel zum wiederholten Verschlüsseln einer konstanten Zeichenkette benutzt. Der jeweils zurückgegebene Wert zeigt auf das verschlüsselte Paßwort. Die ersten beiden Zeichen sind gleich *salt*.

Die Funktionen `des_setkey` und `des_encrypt` ermöglichen den Zugriff auf den aktuellen Verschlüsselungsalgorithmus. Das Argument von `des_setkey` ist ein Zeichenfeld mit einer Länge von 64, das nur die Zeichen mit den numerischen Werten 0 und 1 enthält. Bei einer Aufteilung dieser Zeichenkette in Gruppen von je 8 wird das niederwertige Bit in jeder Gruppe ignoriert; hieraus ergibt sich ein Schlüssel mit 56 Bits, der eingetragen wird. Dies ist dann der Schlüssel, der von dem Algorithmus zum Verschlüsseln der Zeichenkette `block` bei der Funktion `des_encrypt` verwendet wird.

Das Argument für `des_encrypt` ist ein Zeichenfeld der Länge 64, das nur die Zeichen mit den numerischen Werten 0 und 1 enthält. Das Argumentfeld wird anstelle eines ähnlichen Feldes geändert und repräsentiert dabei die Bits des Arguments, nachdem es unter Verwendung des von `des_setkey` gesetzten Schlüssels der Hash-Funktion des Arguments unterworfen worden ist. Wenn *flag* gleich Null ist, wird das Argument verschlüsselt, sonst wird es entschlüsselt.

Beachten Sie, daß die Entschlüsselung in der internationalen Version von crypt(3X) nicht zur Verfügung steht. Die internationale Version ist Teil des C-Entwicklungssystems, und die Inlandversion ist Teil des Sicherheitspakets. Wenn versucht wird, die Entschlüsselung mit der internationalen Version von des_encrypt aufzurufen, wird eine Fehlermeldung ausgegeben.

crypt, setkey und encrypt sind Benutzerprogramme, die je nachdem des_crypt, des_setkey und des_encrypt aufrufen.

Die Routinen run_setkey und run_crypt sind für Applikationen (wie ed(1) und vi(1)) gedacht, die Verschlüsselungsfähigkeiten benötigen, die mit crypt(1) auf Benutzerebene kompatibel sein sollen. run_setkey erstellt eine bidirektionale Pipe-Verbindung zu crypt, wobei *key* als Paßwort-Argument verwendet wird. run_setkey nimmt einen Block Zeichen und transformiert mit crypt(1) den Klartext oder Chiffriertext in Chiffriertext bzw. Klartext. *offset* ist die relative Byte-Position, gerechnet vom Beginn der Datei an, von der der durch *buffer* zur Verfügung gestellte Textblock stammt. *count* ist die Anzahl der Zeichen in *buffer*, und *p* ist ein Feld, das Verweise auf eine Tabelle von Ein- und Ausgabedatei-Streams enthält. Wenn die Verschlüsselung beendet ist, wird crypt_close verwendet, um die Verbindung mit crypt(1) zu beenden.

run_setkey gibt als Ergebnis den Wert -1 zurück, wenn eine Verbindung mit crypt(1) nicht aufgebaut werden kann. Das passiert bei internationalen Versionen des SINIX-Systems, auf denen crypt(1) nicht verfügbar ist. Wenn an run_setkey ein leerer Schlüssel übergeben wird, wird 0 zurückgegeben und sonst 1. run_crypt gibt -1 zurück, wenn es von der Pipe, die an crypt angehängt ist, nicht lesen oder nicht auf sie schreiben kann. Sonst wird 0 zurückgegeben.

Das Programm muß für den Zugriff auf Objektdateien mit der Bibliothek libcrypt.a gebunden werden.

SIEHE AUCH

getpass(3C), passwd(4).
crypt(1), login(1), passwd(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

In der internationalen Version von crypt(3X) wird ein Wert von 1 für flag beim Aufruf von des_encrypt nicht akzeptiert. errno wird in diesem Falle auf den Wert ENOSYS gesetzt, um anzuzeigen, daß die Funktionalität nicht verfügbar ist.

HINWEIS

Der Rückgabewert in crypt zeigt auf Daten der Speicherklasse static, die bei jedem Aufruf überschrieben werden.

dlclose - mehrfach benutzbares Objekt schließen

```
#include <dlfcn.h>
int dlclose(void *handle);
```

`dlclose` löst die Verbindung eines mehrfach benutzbaren Objekts, das zuvor durch `dlopen` geöffnet wurde. Nachdem ein Objekt durch `dlclose` geschlossen wurde, stehen dessen Symbole für `dlsym` nicht weiter zur Verfügung. Alle Objekte, die aufgrund des Aufrufs von `dlopen` mit dem fraglichen Objekt automatisch geladen wurden (siehe `dlopen`(3X)), werden ebenfalls geschlossen. *handle* ist der Wert, der von einem früheren Aufruf von `dlopen` zurückgegeben wurde.

SIEHE AUCH

`dlerror`(3X), `dlopen`(3X), `dlsym`(3X).

ERGEBNIS

Wenn das bezeichnete Oojekt erfolgreich geschlossen werden konnte, liefert `dlclose` das Ergebnis 0. Konnte das Objekt nicht geschlossen werden, oder bezeichnet *handle* kein geöffnetes Objekt, so liefert `dlclose` einen Wert ungleich null als Ergebnis zurück. Genauere Informationen über die Fehlerursache stehen über `dlerror` zur Verfügung.

HINWEIS

Ein erfolgreicher Aufruf von `dlclose` garantiert nicht, daß das zu *handle* gehörende Objekt auch tatsächlich vom Adreßraum des Prozesses entfernt wird. Objekte, die durch einen Aufruf von `dlopen` geladen wurden, können außerdem auch durch einen anderen Aufruf von `dlopen` geladen worden sein. Außerdem kann dasselbe Objekt auch mehrfach geladen worden sein. Ein Objekt wird nicht aus dem Adreßraum entfernt, bevor nicht alle Referenzen, die durch ein explizites `dlopen` erzeugt worden sind, wieder geschlossen wurden, und alle anderen Objekte, die dieses Objekt implizit referenzieren, ebenfalls geschlossen wurden.

Nach dem Schließen eines Objekts durch `dlclose` führt die Verwendung von Symbolen in diesem Objekt zu einem undefinierten Verhalten.

dlerror - Informationen zur Fehlerursache erfragen

```
#include <dlfcn.h>
char *dlerror(void);
```

`dlerror` liefert als Ergebnis eine durch ein Nullzeichen abgeschlossene Zeichenkette (ohne abschließendes Zeilenendezeichen), die den letzten während des dynamischen Bindens aufgetretenen Fehler beschreibt. Sind seit dem letzten Aufruf von `dlerror` keine Fehler im dynamischen Binder aufgetreten, liefert `dlerror` den Wert `NULL` als Ergebnis. So wird, wenn `dlerror` nach einem Aufruf direkt noch einmal aufgerufen wird, `NULL` als Ergebnis geliefert.

SIEHE AUCH

`dlerror`(3X), `dlopen`(3X), `dlsym`(3X).

HINWEIS

Die Meldung, die von `dlerror` zurückgegeben wird, kann in einem statisch definierten Puffer stehen, der mit jedem Aufruf von `dlerror` überschrieben wird. Anwendungsprogramme sollten nicht in diesen Puffer schreiben. Programme, die die Fehlermeldung aufbewahren wollen, sollten sich eine eigene Kopie der Meldung anlegen.

dlopen - mehrfach benutzbares Objekt öffnen

```
#include <dlfcn.h>
void *dlopen(char *pathname, int mode);
```

`dlopen` ist eine der Funktionen, mit denen der Benutzer direkten Zugriff auf die Möglichkeiten des dynamischen Binders hat. Diese Funktionen sind über eine Bibliothek verfügbar, die geladen wird, wenn beim Aufruf von `cc` oder `ld` die Option `-ldl` verwendet wird.

`dlopen` ermöglicht einem laufenden Prozeß den Zugriff auf ein mehrfach benutzbares Objekt. `dlopen` liefert als Ergebnis an den Prozeß einen *handle*, den der Prozeß in folgenden Aufrufen von `dlsym` und `dlclose` verwenden kann. Dieser Wert sollte von dem Prozeß in keiner Weise interpretiert werden. *pathname* ist der Pfadname des zu öffnenden Objekts; es kann sich dabei um einen absoluten Pfad handeln, oder er kann relativ zum aktuellen Verzeichnis angegeben werden. Ist der Wert von *pathname* gleich 0, so macht `dlopen` die Symbole in der ursprünglichen Datei `a.out` und alle Objekte, die zusammen mit `a.out` zum Zeitpunkt des Programmstarts geladen werden, für `dlsym` verfügbar.

Wenn ein mehrfach benutzbares Objekt in den Adreßraum eines Prozesses geladen wird, kann es Verweise auf Symbole enthalten, deren Adresse nicht bekannt ist, bis das Objekt geladen wird. Diese Verweise müssen aufgelöst werden, bevor auf die Symbole zugegriffen werden kann. Der Parameter *mode* steuert, wann diese Auflösung stattfindet. Folgende Werte sind möglich:

`RTLD_LAZY` In diesem Modus werden lediglich die Verweise auf Datensymbole beim Laden des Objekts aufgelöst. Verweise auf Funktionen werden nicht aufgelöst, bevor die Funktion zum ersten Mal aufgerufen wird. Dieser Modus sollte zu einem besseren Systemverhalten führen, da ein Prozeß in der Regel nicht alle Funktionen eines gegebenen, mehrfach benutzbaren Objekts benutzen dürfte.

`RTLD_NOW` In diesem Modus werden alle Verweise beim ersten Laden des Objekts aufgelöst. Dies kann dazu führen, daß einige Berechnungen umsonst vorgenommen werden, wenn Adressen von Funktionen aufgelöst werden, die niemals aufgerufen werden. Der Modus ist jedoch nützlich, wenn Anwendungen sicherstellen müssen, daß nach dem Laden eines Objekts alle während der Laufzeit referenzierten Symbole auch verfügbar sind.

SIEHE AUCH

cc(1), ld(1), sh(1), exec(2), dlclose(3X), dlerror(3X), dlsym(3X).
Kapitel "Das C-Übersetzungssystem" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ERGEBNIS

Wenn *pathname* nicht gefunden wird, nicht zum Lesen geöffnet werden kann, kein mehrfach benutzbares Objekt ist, oder wenn während der Bearbeitung oder während des Ladens von *pathname* oder während der Auflösung seiner symbolischen Verweise ein Fehler auftritt, gibt dlopen den Wert NULL als Ergebnis zurück. Genauere Informationen über die Fehlerursachen stehen über die Funktion dlerror zur Verfügung.

HINWEIS

Wenn bei der Erstellung von *pathname* andere Objekte zusammen mit *pathname* gebunden wurden, so werden diese Objekte automatisch durch dlopen geladen. Der Suchpfad für *pathname* und andere benötigte Objekte kann durch Setzen der Umgebungsvariablen LD_LIBRARY_PATH angegeben werden. Diese Umgebungsvariable sollte eine durch Doppelpunkte getrennte Liste von Verzeichnissen im dem gleichen Format wie die Variable PATH (siehe sh(1)) enthalten. LD_LIBRARY_PATH wird ignoriert, wenn der Prozeß mit gesetztem setuid- oder setgid-Bit läuft (siehe exec(2)), oder wenn der angegebene Namen kein einfacher Dateiname ist (d.h. er enthält ein /-Zeichen). Objekte, die unter demselben absoluten oder relativen Pfad gefunden werden, können beliebig oft durch dlopen geöffnet werden. Das angegebene Objekt wird jedoch nur einmal in den Adreßraum des laufenden Prozesses geladen. Wird dasselbe Objekt jedoch durch zwei verschiedene Pfadnamen gefunden, kann es auch mehrere Male geladen werden. Als Beispiel betrachten wir das Objekt /usr/home/me/mylibs/mylib.so und gehen davon aus, daß /usr/home/me/workdir das aktuelle Verzeichnis ist.

```
...
void *handle1;
void *handle2;

handle1 = dlopen("../mylibs/mylib.so", RTLD_LAZY);
handle2 = dlopen("/usr/home/me/mylibs/mylib.so", RTLD_LAZY);
...
```

Dies führt dazu, daß mylibs.so zweimal für den laufenden Prozeß geladen wird. Andererseits wird mylibs.so bei Verwendung desselben Objektes und bei demselben aktuellen Verzeichnis nur einmal geladen, wenn LD_LIBRARY_PATH=/usr/home/me/mylibs ist.

```
...
void *handle1;
void *handle2;

handle1 = dlopen("mylib.so", RTLD_LAZY);
handle2 = dlopen("/usr/home/me/mylibs/mylib.so", RTLD_LAZY);
...
```

Objekte, die durch einen einzelnen Aufruf von `dlopen` geladen werden, können Symbole voneinander verwenden oder von jedem Objekt, das zur Startzeit des Programms automatisch geladen wird. Objekte, die durch einen Aufruf von `dlopen` geladen werden, können jedoch nicht auf Symbole eines Objekts zugreifen, das durch einen anderen Aufruf von `dlopen` geladen wird. Diese Symbole stehen indirekt durch die Verwendung von `dlsym` zur Verfügung.

Benutzer, die einen Zugriff auf die Symboltabelle von `a.out` selbst durch die Verwendung von `dlsym(0,` *mode*`)` anstreben, sollten sich darüber im klaren sein, daß einige der in `a.out` definierten Symbole für den dynamischen Binder möglicherweise nicht zur Verfügung stehen. Die durch `ld` erstellte Symboltabelle für die Verwendung durch den dynamischen Binder kann unter Umständen nur eine Teilmenge der in `a.out` definierten Symbole enthalten, und zwar genau jene, die von den mehrfach benutzbaren Objekten verwendet werden, mit denen `a.out` gebunden wurde.

dlsym - Adresse von Symbol in mehrfach benutzbarem Objekt berechnen

```
#include <dlfcn.h>
void *dlsym(void *handle, char *name);
```

dlsym erlaubt einem Prozeß, die Adresse eines Symbols zu berechnen, das in einem mehrfach benutzbaren Objekt definiert ist. Das Objekt muß zuvor durch dlopen geöffnet worden sein. *handle* ist der Wert, den der Aufruf von dlopen als Ergebnis lieferte; das entsprechende mehrfach benutzbare Objekt darf seither nicht durch dlclose geschlossen worden sein. *name* ist der Name des Symbols als Zeichenkette. dlsym sucht nach dem angegebenen Symbol in allen mehrfach benutzbaren Objekten, die durch das Laden des durch *handle* bezeichneten Objekts automatisch geladen wurden (siehe dlopen(3X)).

BEISPIEL

Das folgende Beispiel zeigt, wie dlopen und dlsym dazu verwendet werden können, auf Funktionen oder auf Datenobjekte zuzugreifen. Der Einfachheit halber wurde die Fehlerbehandlung weggelassen.

```
void *handle;
int i, *iptr;
int (*fptr)(int);

/* Öffnen des benötigten Objekts */
handle = dlopen("/usr/mydir/libx.so", RTLD_LAZY);

/* Suchen der Adresse einer Funktion und eines Datenobjekts */
fptr = (int (*)(int))dlsym(handle, "some_function");

iptr = (int *)dlsym(handle, "int_object");

/* Aufruf der Funktion mit Übergabe des Datenobjekts als Parameter */

i = (*fptr)(*iptr);
```

SIEHE AUCH

dlerror(3X), dlopen(3X), dlsym(3X).

ERGEBNIS

Wenn *handle* kein gültiges Objekt bezeichnet, das durch dlopen geöffnet wurde, oder wenn das angegebene Symbol in den mit *handle* im Zusammenhang stehenden Objekten nicht gefunden werden kann, liefert dlsym den Wert NULL als Ergebnis. Genauere Informationen über die Fehlerursache stehen über die Funktion dlerror zur Verfügung.

libwindows - Funktionsbibliothek für grafikfähiges Terminal

```
int openagent (void);
int New (int cntlfd, int origin_x, int origin_y, int corner_x, int corner_y);
int Newlayer (int cntlfd, int origin_x, int origin_y, int corner_x, int corner_y);
int openchan (int chan);
int Runlayer (int chan, char *command);
int Current (int cntlfd, int chan);
int Delete (int cntlfd, int chan);
int Top (int cntlfd, int chan);
int Bottom (int cntlfd, int chan);
int Move (int cntlfd, int chan, int origin_x, int origin_y);
int Reshape (int cntlfd, int chan, int origin_x, int origin_y, int corner_x, int corner_y);
int Exit (int cntlfd);
```

Diese Funktionsbibliothek ermöglicht einem Programm auf einem SINIX-System die Ausführung von Operationen auf Bildschirmfenstern (siehe `layers`(5)).

Die Funktion `openagent` eröffnet den Steuerkanal der `xt`(7)-Kanalgruppe, zu der der aufrufende Prozeß gehört. Bei erfolgreichem Abschluß liefert `openagent` einen Dateideskriptor als Ergebnis, der an jede der anderen `libwindows`-Funktionen, mit Ausnahme von `openchan` und `Runlayer`, übergeben werden kann. (Der Dateideskriptor kann auch mit dem Systemaufruf `close` verwendet werden.) Andernfalls wird der Wert -1 zurückgeliefert.

Die Routine `New` erstellt ein neues (Shell)-Fenster mit einer separaten Shell. Die Argumente *origin_x, origin_y, corner_x* und *corner_y* sind die Koordinaten des Fenster-Rechtecks. Wenn alle Koordinatenargumente 0 sind, muß der Benutzer das Rechteck des Fensters interaktiv festlegen. Das Fenster liegt über allen sich überschneidenden Fenstern. Das Fenster wird nicht zum aktuellen Fenster (d.h. die Tastatur ist nicht mit dem neuen Fenster verbunden). Nach erfolgreicher Beendigung gibt `New` die zum Fenster gehörende `xt`(7)-Kanalnummer zurück.

Mit der Routine `Newlayer` wird ein neues Fenster ohne eigene Shell erstellt. Ansonsten ist die Funktion identisch zu `New`, wie oben beschrieben.

Die Routine `openchan` öffnet den Kanal *chan*, der von der Funktion `New` oder `Newlayer` erhalten wurde. Nach erfolgreicher Beendigung gibt `openchan` einen Dateideskriptor als Resultat zurück, der als Eingabe für `write`(2) oder `close`(2) verwendet werden kann. Andernfalls wird der Wert -1 zurückgegeben.

Die Funktion `Runlayer` führt das Kommando *command* in dem Fenster aus, der dem Kanal *chan* zugeordnet ist. Dieses Fenster wurde üblicherweise mit einem vorangegangenen Aufruf von `Newlayer` erzeugt. Alle Prozesse, die zu diesem Zeitpunkt mit diesem Fenster verbunden sind, werden abgebrochen, und der neue Prozeß hat danach die Umgebung des Prozesses `layers`.

Die Funktion `Current` macht den zum Kanal *chan* gehörenden Fenster zum aktuellen Fenster (d.h. an die Tastatur angeschlossen).

Die Funktion `Delete` löscht den zum Kanal *chan* gehörenden Fenster und bricht alle Host-Prozesse in Zusammenhang mit diesem Fenster ab.

Die Funktion `Top` bewirkt, daß das dem Kanal *chan* zugeordnete Fenster über allen sich überschneidenden Fenstern erscheint.

Die Funktion `Bottom` legt das zum Kanal *chan* gehörende Fenster unter alle sich überschneidenden Fenster.

Die Funktion `Move` verlagert das zum Kanal *chan* gehörende Fenster von seinem aktuellen Platz auf dem Bildschirm zu einer neuen Stelle auf dem Bildschirm am Ursprungspunkt (*origin_x, origin_y*). Größe und Inhalt des Fensters werden beibehalten.

Die Funktion `Reshape` verändert das zum Kanal *chan* gehörende Fenster. Die Argumente *origin_x, origin_y, corner_x* und *corner_y* sind die neuen Koordinaten des Fenster-Rechtecks. Wenn alle Koordinatenargumente 0 sind, darf der Benutzer das Rechteck des Fensters interaktiv festlegen.

Die Funktion `Exit` bewirkt, daß das Programm `layers`(5) beendet wird und dabei alle mit ihm in Zusammenhang stehenden Prozesse abgebrochen werden.

DATEIEN

ULIBDIR`libwindows.a`	Funktionsbibliothek für Bildschirmfenster
ULIBDIR	meist `/usr/lib`

SIEHE AUCH

`close`(2), `write`(2), `jagent`(5).
`layers`(1) in "SINIX V5.41 Kommandos".

ERGEBNIS

Bei erfolgreichem Abschluß liefern `Runlayer`, `Current`, `Delete`, `Top`, `Bottom`, `Move`, `Reshape` und `Exit` den Wert 0 als Ergebnis, während `openagent`, `New`, `Newlayer` und `openchan` Werte zurückliefern, die oben bei den einzelnen Funktionen beschrieben wurden. Wenn ein Fehler auftritt, wird -1 zurückgegeben.

HINWEIS

Die Werte für die Koordinaten der Fenster-Rechtecke sind abhängig vom Typ des Terminals. Diese Abhängigkeit betrifft die Funktionen, die Koordinaten von Fenster-Rechtecken verwenden: `Move`, `New`, `Newlayer` und `Reshape`. Einige Terminals erwarten für diese Werte Zeichenpositionen (Bytes), andere gehen davon aus, daß die Information in Pixeln (Bits) ausgedrückt ist.

Es wird empfohlen, bei Anwendungen beim Zugriff auf den `xt`-Treiber `/dev/xt/??[0-7]` anstelle von `/dev/xt??[0-7]` zu verwenden.

maillock - Sperrdatei für Mailbox eines Benutzers verwalten

```
#include <maillock.h>
int maillock (const char *user, int retrycnt);
int mailunlock (void);
```

Die Funktion `maillock` versucht, für die Mail-Datei eines Benutzers eine Sperrdatei anzulegen. Wenn bereits eine Sperrdatei existiert, geht `maillock` davon aus, daß der Inhalt der Datei die Prozeßnummer des Prozesses angibt (als `ASCII`-Zeichenkette, die durch ein Nullzeichen abgeschlossen ist), der die Sperrdatei angelegt hat (vermutlich durch einen Aufruf von `maillock`). Wenn der Prozeß, der die Sperrdatei angelegt hat, noch immer vorhanden ist, wartet `maillock` *retrycnt* mal, bevor ein Fehler signalisiert wird. Dabei wird fünf Sekunden mal die angegebene Anzahl gewartet. Das bedeutet, das erste Warten dauert fünf Sekunden, das nächste Warten dauert zehn Sekunden, usw., bis die Anzahl Versuche den Wert von *retrycnt* erreicht. Wenn die Sperrdatei nicht länger benötigt wird, sollte sie durch den Aufruf der Funktion `mailunlock` entfernt werden.

user bezeichnet den Login-Namen des Benutzers, für dessen Mailbox eine Sperrdatei angelegt wird. `maillock` geht davon aus, daß sich die Mail-Dateien des Benutzers am vorgesehenen Speicherplatz befinden, wie in `maillock.h` definiert.

Die folgenden Definitionen von Ergebniswerten sind in `maillock.h` enthalten:

```
#define     L_SUCCESS       0       /* Sperrdatei angelegt oder entfernt */
#define     L_NAMELEN       1       /* Name des Empfängers > 13 Zeichen */
#define     L_TMPLOCK       2       /* Erzeugung der tmp. Datei nicht möglich */
#define     L_TMPWRITE      3       /* Schreiben der PID in Sperrdatei nicht möglich */
#define     L_MAXTRYS       4       /* Nach retrycnt Versuchen gescheitert */
#define     L_ERROR         5       /* Überprüfe errno für den Grund */
#define     L_MANLOCK       6       /* Pflichtsperre konnte nicht gesetzt werden */
```

DATEIEN

LIBDIR`/llib-mail.ln`
LIBDIR`/libmail.a`
`/var/mail/*`
`/var/mail/*.lock`

HINWEIS

`mailunlock` entfernt nur die Sperrdatei, die durch den neuesten Aufruf von `maillock` angelegt wurde. Der Aufruf von `maillock` für verschiedene Benutzer, ohne dazwischenliegende Aufrufe von `mailunlock`, führt dazu, daß die zuerst angelegten Sperrdateien bestehen bleiben und eventuell die weitere Auslieferung von Nachrichten verhindert wird, solange der momentane Prozeß nicht terminiert.

malloc - Hauptspeicher zuweisen

```
#include <stdlib.h>
#include <malloc.h>
void *malloc (size_t size);
void free (void *ptr);
void *realloc (void *ptr, size_t size);
void *calloc (size_t nelem, size_t elsize);
#include <malloc.h>
int mallopt (int cmd, int value);
struct mallinfo mallinfo (void);
```

`malloc` und `free` bilden ein einfaches, allgemein einsetzbares Paket zur Speicherzuweisung.

`malloc` gibt als Ergebnis einen Zeiger auf einen Block zurück, der wenigstens *size* Bytes groß ist.

Das Argument für `free` ist ein Zeiger auf einen Block, der zuvor von `malloc` belegt wurde. Nach der Ausführung von `free` wird dieser Speicherplatz für eine neue Zuteilung zur Verfügung gestellt und sein Inhalt zerstört (dieses Verhalten läßt sich jedoch ändern; siehe `mallopt`). Wenn *ptr* ein Nullzeiger ist, geschieht nichts.

Wenn der von `malloc` zugewiesene Speicherplatz überschritten wird, oder wenn `free` eine ungültige Adresse übergeben wird, sind die Ergebnisse unvorhersehbar.

`realloc` ändert die Größe des Blocks, auf den *ptr* zeigt, auf *size* Bytes und gibt einen Zeiger auf den (möglicherweise verlagerten) Block zurück. Der Inhalt bleibt bis zur geringeren Größe unverändert.

`calloc` weist Speicherplatz für ein Feld von *nelem* Elementen der Größe *elsize* zu. Dieser Speicherplatz wird mit Nullen initialisiert.

`mallopt` bietet eine Steuerungsmöglichkeit über den Zuweisungsalgorithmus. Die verfügbaren Werte für *cmd* sind in der Include-Datei `malloc.h` definiert und haben folgende Bedeutung:

`M_MXFAST` *maxfast* auf den Wert *value* setzen
Der Algorithmus weist alle Blöcke unterhalb der Größe von *maxfast* in großen Gruppen zu und teilt sie dann sehr schnell aus. Der Standardwert für *maxfast* ist 24.

M_NLBLKS *numlblks* auf den Wert *value* setzen
Die oben erwähnten 'großen Gruppen' enthalten jeweils *numlblks* Blöcke. *numlblks* muß größer als 0 sein. Der Standardwert für *numlblks* ist 100.

M_GRAIN *grain* auf den Wert *value* setzen
Die Größen der Blöcke, die kleiner als *maxfast* sind, gelten als auf das nächste Mehrfache von *grain* abgerundet, das größer als 0 sein muß. Der Standardwert von *grain* ist die kleinste Anzahl von Bytes, die die Ausrichtung jedes Datentyps noch zuläßt. Der Wert wird beim Setzen von *grain* auf ein Vielfaches des Standardwerts aufgerundet.

M_KEEP Daten in einem freigesetzten Block bis zum nächsten `malloc` , `realloc` oder `calloc` aufbewahren
Diese Option wird lediglich aus Gründen der Kompatibilität mit der früheren Version von `malloc` beibehalten. Von ihrer Verwendung wird abgeraten.

M_MEMSZ Der Wert von *value* wird auf das nächsthöhere Vielfache von 1K, mindestens aber auf 2K (2048), aufgerundet. Neuer Speicherplatz wird vom System immer mit diesem Wert angefordert.

M_FREE steuert die Behandlung von freien Blöcken
Wenn *value* gleich 0 ist, dann werden alle frei gewordenen Blöcke an den Anfang der freien Liste gesetzt (Standardeinstellung). Andernfalls werden alle freien Blöcke an das Ende der freien Liste gesetzt.

`mallopt` kann wiederholt aufgerufen werden, jedoch nicht mehr, nachdem der erste kleine Block zugeteilt wurde.

`mallinfo` liefert Informationen über den Speicherplatzverbrauch. Es gibt die folgende Struktur zurück:

```
struct mallinfo  {
     int arena;      /* Gesamtplatz im Bereich */
     int ordblks;    /* Anzahl der normalen Blöcke */
     int smblks;     /* Anzahl der kleinen Blöcke */
     int hblkhd;     /* Speicher für Blockköpfe */
     int hblks;      /* Anzahl von Blöcken */
     int usmblks;    /* Für kleine Blöcke benutzter Speicher */
     int fsmblks;    /* Für kleine Blöcke freier Speicher */
     int uordblks;   /* Für normale Blöcke benutzter Speicher */
     int fordblks;   /* Für normale Blöcke freier Speicher */
     int keepcost;   /* Zusätzlicher Speicherbedarf bei Verwendung der keep-Option */
}
```

Diese Struktur ist in der Include-Datei `malloc.h` definiert.

Jede der Belegungsroutinen gibt einen Zeiger auf einen geeignet ausgerichteten Speicherplatz zurück, der (nach eventueller Zeigertyp-Anpassung) für die Speicherung jedes beliebigen Objekttyps geeignet ist.

SIEHE AUCH

`brk`(2), `malloc`(3C).

ERGEBNIS

`malloc`, `realloc` und `calloc` geben einen Nullzeiger zurück, wenn nicht genügend Speicher zur Verfügung steht. Wenn `realloc` `NULL` zurückgibt, bleibt der Block, auf den *ptr* weist, unverändert. Wenn `mallopt` nach einer Zuordnung aufgerufen wird, oder wenn *cmd* bzw. *value* ungültig sind, wird ein Wert ungleich null zurückgegeben. Andernfalls wird null zurückgegeben.

HINWEIS

Es ist zu beachten, daß dieses Paket im Gegensatz zu `malloc`(3C) den Inhalt eines Blocks bei Freigabe nicht bewahrt, wenn die Option `M_KEEP` von `mallopt` nicht benutzt wird.

Nichtdokumentierte Merkmale von `malloc`(3C)
sind nicht dupliziert worden.

Funktionsprototypen für `malloc`, `realloc`, `calloc` und `free` werden auch in der Include-Datei `malloc.h` definiert, um die Kompatibilität mit alten Anwendungen zu erreichen. Neue Anwendungen sollten für den Zugriff auf die Prototypen dieser Funktionen `stdlib.h` verwenden.

mpcntl - Standardschnittstelle für Multiprozessor-Verarbeitung

```
#include <sys/types.h>
#include <sys/mpcntl.h>
mpcntl(int req, void *arg)
```

`mpcntl` stellt dem Programmierer verschiedene Dienste zur Implementierung seiner Anwendungen in einer Multiprozessor-Umgebung zur Verfügung.

Bindung: Ein Prozeß kann eine Bindung an eine bestimmte CPU (Zentraleinheit) explizit aufbauen. Dabei kann es sich um eine exklusive Bindung handeln, bei der mehrere Prozesse eine Bindung zu einer CPU eingehen und alle nicht explizit gebundenen Prozesse keinen Zugriff auf diese CPU haben. Im anderen Fall, dem Standardfall, teilen sich mehrere Prozesse eine CPU (ohne Ausschluß anderer Prozesse).

Prozeßsteuerung: Deaktiviert oder aktiviert das Standardverhalten des Kernel zur Prozeßbearbeitung, das zum Ziel hat, den jeweiligen Cache-Status zu erhalten. Bei deaktiviertem Standardverhalten steht ein Prozeß jedem Prozessor zur Bearbeitung zur Verfügung.

Eingeschränkte Bearbeitungssteuerung: Ein Prozeß kann ein Vorrecht anderer Prozesse auf Ablauf vorübergehend aussetzen, bis er Prozeß blockiert oder dieses Vorrecht wieder aktiviert. Gleichzeitig reagiert der Prozeß auf keine Signale mehr (außer auf SIGKILL). Der Prozeß kann die CPU auch freigeben, so daß andere Prozesse bearbeitet werden können. Wenn der Prozeß eine hohe Priorität hat, wird er nicht unterbrochen.

CPU-Status: Es wird die angeforderte Information über die Anzahl der CPUs im Rechner und über spezielle CPUs ausgegeben. Es kann auch die Status-Information einer CPU geändert werden.

req gibt die gewünschte Anforderung an.
arg ist ein Zeiger, der in bestimmten Fällen ein Nullzeiger ist. Üblicherweise wird *arg* als Zeiger auf eine `mpcntl`-Struktur interpretiert. Sie wird in der Datei `mpcntl.h` folgendermaßen definiert:

```
typedef  uint_t  cpumask_t;

typedef  struct  mpcreq {
    cpumask_t   mpc_cpu;        /* CPU-Bezeichner */
    pid_t       mpc_pid;        /* gültige PID */
} mpcreq_t;
```

Das Feld *mpc_cpu* ist eine *cpumask_t*-Struktur (`int`), bei der jedes Bit eine logische CPU darstellt (Bit 0 stellt die CPU 0 dar, Bit 1 die CPU 1 usw.). So ist es möglich, gleichzeitig mehr als eine CPU-Nummer an das System weiterzugeben. Das Makro `CPUNO_TO_LCPUID` wird zur Konvertierung von CPU-Nummern und einem Bit in der Maske *cpumask_t* verwendet. Es ist nicht nötig, mehr als eine CPU anzugeben.

mcp_pid ist eine *pid_t*-Struktur mit einer gültigen Prozeßnummer (dem Rückgabewert eines Aufrufs von `getpid()`).

Durch eine unsachgemäße Anwendung einiger `mpcntl`-Operationen kann der Systemablauf beeinträchtigt werden, oder es kann unter Umständen sogar zu einem Deadlock kommen. Zur Vermeidung solcher Situationen sollten diese `mpcntl`-Optionen nur vom Systemverwalter (Benutzernummer 0) bzw. nur von Benutzern mit besonderen `mpcntl`-Rechten verwendet werden.

Es folgen die möglichen Anforderungen an `mpcntl` mit der jeweiligen *arg*-Beschreibung:

`MPCNTL_BIND` und `MPCNTL_BINDXCLU`

Eine (exklusive) Bindung zwischen dem angegebenen Prozeß und mindestens einer angegebenen CPU wird hergestellt. Eine exklusive Bindung unterscheidet sich von einer regulären Bindung dadurch, daß bei einer exklusiven Bindung die CPU tatsächlich nur die gebundenen Prozesse ausführt. Man könnte dies als eine Art "primitive CPU/Prozeßverwaltung" bezeichnen. *arg* zeigt auf die Datenstruktur *mpcreq_t*, die logische Ziel-CPUs und eine gültige Prozeßnummer angibt. Mit `MPCNTL_BIND` und `MPCNTL_BINDXCLU` kann eine Bindung zwischen einem Prozeß und mehreren CPUs hergestellt werden, wenn in *mpc_cpu* mehrere Bit gesetzt sind. Dadurch werden alle Prozessoren für diese Bindung reserviert und es wird ausgeschlossen, daß diese CPUs andere, nicht explizit gebundene Prozesse ausführen. Sie sollten daher auf einen gezielten Einsatz achten.

Bei asymmetrischen Systemen sollten Sie darauf achten, daß Prozesse immer Zugriff auf die notwendigen Ressourcen haben (zum Beispiel auf das Gleitpunktregister). Sollte ein Prozeß nicht auf eine notwendige Ressource zugreifen können, versucht das System diesen Fehler zu beseitigen, indem die Bindung geändert oder gelöscht wird. Nach erfolgreicher Ausführung des Aufrufs läuft der Prozeß auf einer der Ziel-CPUs (z.B. bei bereits vorliegender Bindung).

Die Attribute der Bindung werden auch mit den Systemaufrufen `fork`(2) und `exec`(2) vererbt. Die Ausführung ist unter Umständen nicht erfolgreich, wenn durch `MPCNTL_CPUINFOSET` eine Änderung an einer Ziel-CPU vorgenommen wurde, um die angeforderte Operation auszuschließen (z.B. wenn die CPU als nicht verfügbar gekennzeichnet wurde (`CPIFS_NAVAIL`) oder mit `CPIFS_BINDOFF` oder `CPIFS_XBINDOFF` dieser Bindungstyp ausgeschlossen wurde). `MPCNTL_BINDXCLU` kann nur vom Systemverwalter benutzt werden.

MPCNTL_UNBIND

Die Bindung des angegebenen Prozesses an mindestens eine CPU wird gelöst. Es wird also das Gegenteil von MPCNTL_BIND bzw. MPCNTL_BINDXCLU bewirkt. Bei der Lösung einer exklusiven Bindung eines Prozesses wird der entsprechende Prozessor wieder für alle Prozesse verfügbar, wenn die gelöste Bindung die letzte exklusive Bindung dieser CPU war. *arg* zeigt auf die Datenstruktur *mpcreq_t*, die die Ziel-Prozeßnummer angibt. Bestehende Bindungen können mit aufeinander folgenden MPCNTL_BIND- bzw. MPCNTL_BINDXCLU-Aufrufen geändert werden. So können alte Bindungen gelöscht und neue eingerichtet werden, ohne MPCNTL_UNBIND aufzurufen.

MPCNTL_AFFINOFF und MPCNTL_AFFINON

Deaktiviert bzw. aktiviert das Standardverhalten, das eine schwache Bindung zwischen einem Prozeß und der CPU, auf der der Prozeß zuletzt lief, bewirkt. Wenn dieses Verhalten deaktiviert ist, läuft der Prozeß auf der CPU, die als nächste frei wird (wobei die üblichen Prioritäten berücksichtigt werden). *arg* zeigt auf die Datenstruktur *mpcreq_t*, die die Ziel-Prozeßnummer angibt. Nach einem Aufruf von exec(2) wird die Standardeinstellung MPCNTL_AFFINON wiederhergestellt. Andererseits wird eine Deaktivierung des Standardverhaltens vom Systemaufruf fork(2) vererbt. Diese Aufrufe bleiben wirkungslos, wenn das Standardverhalten durch eine Systemeinstellung deaktiviert ist.

MPCNTL_NOPRMPT und MPCNTL_PRMPT

Das Vorrecht des aktuellen Prozesses wird deaktiviert bzw. aktiviert und der Prozeß reagiert nicht mehr auf Signale (optional). Mit dieser Option ist es möglich, einen Prozeß in einem kritischen Bereich ablaufen zu lassen, ohne daß er unterbrochen werden kann. Ein Anwendungsfall wäre zum Beispiel beim Vorliegen einer "spinlock"-Situation. Dieses Verhalten kann mit argp modifiziert werden. Folgende Werte sind möglich:

MPCNTL_NOPRMPT_SIGS:

Der Vorrecht-Status des Prozesses beeinflußt die Signalbehandlung nicht (so werden zum Beispiel gehaltene/ignorierte Signale weiterhin gehalten/ignoriert, abgefangene Signale werden bearbeitet, und ignorierte Signale werden ignoriert).

Wenn ein Prozeß läuft, kann er nicht unterbrochen werden. Dieses Verhalten bleibt sowohl bei einem beabsichtigten (zum Beispiel einem Systemaufruf), als auch bei einem unbeabsichtigten (zum Beispiel einem Paging-Fehler) Blockieren des Prozesses erhalten. Wenn ein nicht unterbrechbarer Prozess die CPU freigibt, konkurriert er anschließend mit den anderen Prozessen um einen Zugriff auf eine CPU. Der Prozeß ist dann wieder nicht unterbrechbar, bis bis er erneut blockiert.

MPCNTL_NOPRMPT_NOSIGS:
Maskierte, ignorierte oder abgefangene Signale beeinflussen die Signalbehandlung für den Prozeß. Ein Verhalten, bei dem der Prozeß nicht unterbrochen werden kann, gilt nur so lange, bis der Kernel involviert ist (durch einen Systemaufruf oder eine Ausnahme). Ein Interrupt beeinträchtigt das Vorrecht eines Prozesses nicht direkt und wird auch nicht als Kernel-Aktion interpretiert. Der erste MPCNTL_NOPRMPT-Aufruf aktiviert diesen Mechanismus. Weitere Aufrufe von MPCNTL_PRMPT und MPCNTL_NOPRMPT führen zu bibliotheks-spezifischen Operationen (so aktualisiert die Bibliothek zum Beispiel den Wert dieser Adresse im Benutzerbereich, ohne daß die Betriebssystem-Ebene aufgerufen wird). Diese Optimierung soll das Vorrecht-Verhalten beschleunigen. Bei Erfolg wird der vorherige Vorrecht-Status des Prozesses zurückgegeben (> 0 wenn nicht unterbrechbar, 0 wenn unterbrechbar). Andernfalls wird -1 zurückgegeben.
Für diese Option sind zusätzliche Rechte erforderlich. Sie wird mit `fork` oder `exec` nicht vererbt.

MPCNTL_GETFLAG
Die `mpcntl`-Optionen und Bindungsinformation des Zielprozesses werden zurückgegeben. *arg* zeigt sowohl auf die Datenstruktur *mpcreq_t*, die die Ziel-Prozeßnummer angibt, als auch auf die Datenstruktur *cpumask_t*, mit der die Bindungsinformation des Zielprozesses zurückgegeben wird. Bei Erfolg gibt MPCNTL_GETFLAG die Optionen zurück, und das Feld *mpc_cpu* der Struktur, auf die *arg* zeigt, enthält die Bindungsinformation. Andernfalls wird -1 zurückgegeben.
Eine Interpretation der Optionen ist mit Hilfe der `#define`-Anweisungen in der Datei `sys/proc.h` möglich. Relevant sind PBIND, PBINDXCLU, PAFFINOFF sowie PNOPRMPT (unterteilt in PNOPRMPTSIG und PNOPRMPTNSIG).

MPCNTL_RDTIMER
Der 'high resolution-Timer' (ein Zeitgeber mit hoher Genauigkeit) des Systems wird gelesen, und der abgelesene Wert wird zurückgegeben. Die Einheit für den Timer ist etwa eine Mikrosekunde. Der Benutzer muß mit einer 'wrap-around condition' rechnen, da der Wertebereich begrenzt sein kann.
Die Werte für den Timer haben keinen Bezug zur Uhrzeit und haben nur als Angabe der Dauer eines Intervalls einen Sinn. Bei Implementationen, bei denen die Hardware diese Option nicht unterstützt, wird u.U. -1 zurückgegeben. Der Parameter *arg* wird ignoriert. Diese Option kann ohne besondere Rechte verwendet werden.

MPCNTL_YIELD

Der aufrufende Prozeß gibt die CPU frei. Der Systemaufruf kehrt erst zurück, wenn der Prozeß entsprechend neu eingereiht wurde. Wenn dies der einzige Prozeß oder der Prozeß mit der höchsten Priorität ist, kehrt der Aufruf sofort zurück.

MPCNTL_CPUCNT

Die Anzahl der CPUs im System wird zurückgegeben. *arg* zeigt auf die Datenstruktur *cpumask_t* im Benutzerbereich, die die CPUs anzeigt, die aktiv und nicht als gesperrt gekennzeichnet sind. Der Rückgabewert des mpcntl-Aufrufs gibt die Gesamtzahl der konfigurierten CPUs an.
Das Spektrum der logischen CPUs reicht lückenlos von 0 bis (cpucnt - 1). Alle CPUs werden beim Laden des Systems konfiguriert und gestartet. Wenn *arg* null ist, wird nur die Anzahl zurückgegeben.

MPCNTL_CPUINFOGET

Es wird eine Datenstruktur *cpuinfo_t* zurückgegeben, die weitere Informationen zu einer bestimmten CPU enthält. Die Datenstruktur *cpuinfo_t* ist folgendermaßen definiert:

```
typedef struct cpuinfo {
        cpuid_t         cpi_cpuid;      /* logische CPU-Nummer */
        int             cpi_cpustatus;  /* var. CPU-Statusoptionen */
        int             cpi_cpuattr;    /* CPU-Attribute */
        enum cputype    cpi_cputype;    /* Prozessortyp */
        int             cpi_cpurev;     /* Revision der CPU */
        enum fputype    cpi_fptype;     /* Gleitpunkt-Coprozessor */
        int             cpi_fprev;      /* Revision des Gleitpunkt-Coproz. */
        enum fputype    cpi_fptype2;    /* Gleitpunkt-Coprozessor [2] */
        int             cpi_fprev2;     /* Revision des Gleitpunkt-Coproz. [2] */
        int             cpi_cachsize;   /* Größe (in Byte) d. sek. Cache */
        int             cpi_cpuclock;   /* Geschwind. der CPU in MHz */
        int             cpi_bind;       /* # Prozesse explizit gebunden */
        int             cpi_xbind;      /* # Prozesse exklusiv gebunden */
        int             cpi_drvbind;    /* # Treiber an CPU gebunden */
        int             cpi_physcpuid;  /* physikal. CPU-Nummer */
        int             cpi_physintbind;
                        /* physikal. Geräte-Int. an CPU gebunden */
        int             cpi_pad[8];     /* reserviert */
} cpui   nfo_t;
```

MPCNTL_CPUINFOSET

Diese Option wird zum Setzen der *cpi_cpustatus*-Optionen (z.B. Lösen einer Bindung oder Deaktivieren einer CPU) verwendet. Der Rest der Struktur *cpuinfo_t* wird ignoriert. Typische Aktionen sind die Ausführung von MPCNTL_CPUINFOGET, eine Änderung von Optionen nach Bedarf sowie der Aufruf von MPCNTL_CPUINFOSET. Folgende Optionen stehen zur Verfügung:

CPIFS_INSVC:
Die CPU ist aktiv. Mit dem Löschen der Option wird die CPU inaktiv. Nicht alle Hardware-Umgebungen unterstützen diese Option. Im Standardfall ist diese Option gesetzt. Wenn eine CPU inaktiv wird, werden Bindungen an diese CPU gelöst.

CPIFS_NAVAIL:
Die CPU steht für eine Prozeßausführung nicht zur Verfügung. Es wird allerdings nichts darüber ausgesagt, ob die CPU aktiv ist. Im Standardfall ist diese Option nicht gesetzt. Wenn der Status einer CPU so verändert wird, daß sie nicht mehr zur Verfügung steht, werden die Bindungen an diese CPU gelöst.

CPIFS_BINDOFF:
Die CPU steht für normale Bindungsoperationen (MPCNTL_BIND) nicht zur Verfügung, aber existierende Bindungen werden nicht verändert. Im Standardfall ist diese Option nicht gesetzt. Vererbte Bindungsattribute werden (bei den Systemaufrufen fork bzw. exec) dahingehend beeinflußt, daß sie ignoriert werden, bis die CPU Bindungen wieder zuläßt.

CPIFS_XBINDOFF:
Die CPU steht für exklusive Bindungsoperationen (MPCNTL_BINDXCLU) nicht zur Verfügung, aber existierende Bindungen werden nicht verändert. Im Standardfall ist diese Option nicht gesetzt. Vererbte Bindungsattribute werden (bei den Systemaufrufen fork bzw. exec) ignoriert, bis die CPU Bindungen wieder zuläßt.

CPIFS_CACHEOFF:
Die angegebene CPU-Cache wird ausgeschaltet. Diese Option wird nicht von allen Hardware-Umgebungen unterstützt. Im Standardfall ist diese Option nicht gesetzt.

CPIFS_IOINTROFF:
Die angegebene CPU empfängt keine E/A-Interrupt-Ereignisse. Diese Option wird nicht von allen Hardware-Umgebungen unterstützt. Im Standardfall ist diese Option nicht gesetzt. Für diese Option sind Systemverwalter-Rechte erforderlich.

ERGEBNIS

Bei Erfolg gibt `mpcntl` Null bzw. den Rückgabewert aus (z.B. bei `MPCNTL_RDTIMER`). Andernfalls gibt `mpcntl` -1 zurück. In diesem Fall gibt die Fehlernummer einen Hinweis auf die Fehlerursache.

`EINVAL` Mindestens eines der Argumente für `mpcntl` ist ungültig. Der Grund kann zum Beispiel eine ungültige Prozeßnummer oder Option sein.

`EPERM` Die Benutzernummer ist nicht die des Systemverwalters (0), oder es fehlen die notwendigen Rechte. Dieser Fehler kann auch auftreten, wenn für einen Zielprozeß, den der Benutzer nicht verändern darf, neue Optionen angegeben werden.

`ENXIO` Die Hardware-Umgebung unterstützt die Option nicht, es liegt eine Bereichsüberschreitung für die CPU-Nummer vor, oder der Status der CPU läßt die Operation nicht zu.

`EFAULT` Es wurde ein ungültiger Zeiger auf ein Argument (z.B. *arg*) übergeben.

`EBUSY` Die CPU hat einen Bindungstyp, den die Optionen ausdrücklich nicht zulassen, bzw. eine exklusive Operation versucht, einen nicht exklusiven Status herzustellen.

SIEHE AUCH

`getpid`(2), `signal`(2).

HINWEIS

Beim Binden muß die Bibliothek `libmpoc` dazugebunden werden (`cc -lmproc`).

spinlock - Benutzersynchronisation

```
#include <ulocks.h>
void initspin(spinlock_t *lck, long spincnt);
void spinlock(spinlock_t *lck);
void spinunlock(spinlock_t *lck);
int cspinlock(spinlock_t *lck);
void yield();
```

Die einfachste Synchronisationsmethode für eine Multiprozessor-Umgebung ist die sogenannte "spinlock"-Methode, die auch anderen Synchronisationsmethoden zugrundeliegt. Diese Methode ist abhängig von einer unteilbaren Lesen-Modifizieren-Schreiben-Operation, die vom System zur Verfügung gestellt wird. Die Methode wird auf unterschiedlichen CPU-Architekturen unterschiedlich implementiert; unterschiedliche Systemarchitekturen unterstützen sie auch auf unterschiedliche Art und Weise. Normalerweise werden diese systemnahen Operationen in Machinensprache codiert. Es werden einige Grundmethoden zur Verfügung gestellt, um nicht auf die einzelnen Implementationsformen auf den unterschiedlichen Rechnern eingehen zu müssen.

Eine Anwendung erfolgt in der Regel durch kooperierende Prozesse, die bestimmte Adressbereiche (wie gemeinsame Speicherbereiche) gemeinsam nutzen. Hierdurch wird der Zugriff auf gemeinsame Ressourcen und Datenstrukturen gesteuert. Die Ziel-Datenstruktur sieht folgendermaßen aus:

```
typedef struct spinlock {
    volatile int   sl_lock;
    long           sl_scount;
} spinlock_t;
```

`initspin` initialisiert eine Zielsperre, die von diesen Methoden verwendet wird. `initspin` hat zwei Argumente: *lck* ist ein Zeiger auf die Ziel-Datenstruktur *spinlock_t*, *spincnt* ist die Anzahl der Wartezyklen, die auf ein gesperrtes Ziel angewandt werden, bevor der aufrufende Prozeß die CPU freigibt. Der Wert -1 für *spincnt* bedeutet eine unendliche Anzahl an Wartezyklen (der Prozeß gibt die CPU nicht freiwillig frei).

`spinlock` versucht, die durch *lck* angegebene Sperre zu erzielen. Entweder diese Funktion wird eine bestimmte Anzahl an Wartezyklen ausführen (festgelegt durch den Initialisierungsparameter `spincnt`), bevor die CPU schließlich freigegeben wird, oder sie wird unendlich oft wiederholt (wenn *spincnt* den Wert -1 hat), bis die Sperre erzielt worden ist. Das Betriebssystem kann die zur Verfügung stehende Zeit allerdings jederzeit beenden.

`spinunlock` gibt die Sperre, die durch *lck* angegeben ist, frei. `spinunlock` bewirkt das Gegenteil von `spinlock`.

Die bedingte "spinlock"-Operation `cspinlock` versucht, die angegebene Sperre *lck* zu erzielen, und gibt den Status der Sperre zurück. Wenn die Sperre bereits besteht, werden keine Wartezyklen ausgeführt, und es wird ein Fehler gemeldet (Rückgabewert ungleich Null). Wenn die Sperre erzielt werden kann, wird Null zurückgegeben.

`yield` ermöglicht die Freigabe der CPU ohne Blockieren (implementiert mit der Option *mpcntl_yield* von `mpcntl`). Diese Funktion kehrt sofort zurück, wenn der aufrufende Prozeß eine ausreichend hohe Priorität hat, so daß die Ausführung fortgesetzt werden kann.

SIEHE AUCH

`mpcntl`(3X)

HINWEIS

Beim Binden muß die Bibliothek `libmpoc` dazugebunden werden (`cc -lmporc`).

sputl, sgetl - maschinenunabhängig auf lange Ganzzahlen zugreifen

```
#include <ldfcn.h>
void sputl (long value, char *buffer);
long sgetl (const char *buffer);
```

`sputl` nimmt die vier Bytes der Langzahl *value* und schreibt sie in den Speicher, auf den *buffer* zeigt. Die Anordnung der Bytes ist auf allen Rechnern gleich.

`sgetl` liest die vier Bytes aus der Adresse, auf die *buffer* zeigt, und gibt die Langzahl in der Form des lokalen Rechners zurück.

`sputl` und `sgetl` bieten eine maschinenunabhängige Form zum Speichern langer Ganzzahlen in Binärform in einer Datei, ohne daß eine Umsetzung in Zeichen erforderlich ist.

HINWEIS

Programme, die `sputl` oder `sgetl` verwenden, müssen mit der Option `-lld` übersetzt werden.

Dateiformate

intro - Einführung in die Dateiformate

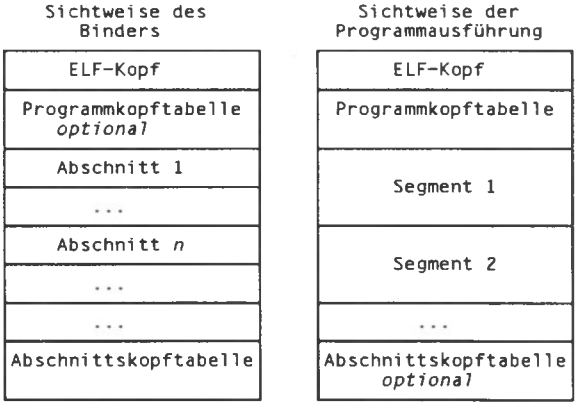
In diesem Abschnitt werden die Formate verschiedener Dateien beschrieben. In den betreffenden Fällen werden die Deklarationen der C-Strukturen für die Dateiformate angegeben. Man findet die Include-Dateien, die diese Strukturdeklarationen enthalten, normalerweise im Dateiverzeichnis `/usr/include` oder `/usr/include/sys`. Zur Einbindung in C-Programme ist die Syntax `#include <`*filename.h*`>` oder `#include <sys/`*filename.h*`>` zu verwenden.

a.out - ELF (Executable and Linking Format)-Dateien

```
#include <elf.h>
```

Der Dateiname `a.out` ist die Standard-Ausgabedatei des Binders `ld`(1). Der Binder macht eine `a.out`-Datei ausführbar, wenn beim Binden keine Fehler auftraten.

Programme, die ELF-Dateien verändern, sollten die Bibliothek verwenden, die in `elf`(3E) beschrieben wird. Es folgt ein Überblick über das Dateiformat. Ausführliche Informationen finden Sie auch in der angegebenen Literatur.



Der ELF-Kopf steht am Anfang und enthält eine Beschreibung des Dateiaufbaus. Abschnitte enthalten Informationen eines Objekts für den Binder: Anweisungen, Daten, Symboltabellen, Verschiebe-Informationen usw. Segmente enthalten die notwendigen Informationen, um eine Objektdatei auszuführen. Ein Segment kann mehrere Abschnitte enthalten.

Eine Programmkopftabelle gibt dem System an, wie der Prozeß zu erzeugen ist. Dateien, die als Prozeß ablaufen sollen, müssen eine Programmkopftabelle haben. Verschiebbare Dateien müssen keine solche Tabelle haben. In einer Abschnittskopftabelle werden die Abschnitte der Datei beschrieben. Jeder Abschnitt hat hier einen Eintrag. Dieser Eintrag enthält den Abschnittsnamen, die Abschnittsgröße usw. Dateien, die gebunden werden sollen, müssen eine Abschnittskopftabelle beinhalten. Andere Objektdateien müssen diese Tabelle nicht haben.

In der Abbildung steht die Programmkopftabelle direkt hinter dem ELF-Kopf und die Abschnittskopftabelle hinter den Abschnitten. Diese Reihenfolge ist nicht zwingend. Segmente und Abschnitte haben keine vorgegebene Reihenfolge. Nur der ELF-Kopf muß am Anfang der Datei stehen.

Wenn eine `a.out`-Datei ausgeführt wird, werden drei logische Segmente eingerichtet: das Textsegment, das Datensegment (nichtinitialisierte Daten stehen hinter initialisierten Daten, wobei nichtinitialisierte Daten mit Null initialisiert werden) und das Stapelsegment. Das Textsegment kann nicht vom Programm beschrieben werden. Wird das gleiche `a.out` nochmal aufgerufen, greifen die Prozesse auf das gleiche Textsegment zu.

Das Datensegment beginnt an der nächsten maximalen Seitengrenze hinter dem Textsegment. Wenn ein Prozeß erzeugt wird, kann der Teil der Datei, in dem das Ende des Textsegments und der Anfang des Datensegments stehen, zweimal geladen werden. Der doppelte Teil des Textsegments, der am Anfang des Datensegments steht, wird nie ausgeführt. Er wird einfach dupliziert, damit das Betriebssystem den Datenbereich in ganzen Seiten einlagern kann, ohne den Datenabschnitt auf eine Seitengrenze schieben zu müssen. Die erste Datenadresse ist daher die nächste Seitengrenze hinter dem Textsegment plus die Größe des Textsegments modulo der maximalen Seitengröße. Falls das Textsegment ein Vielfaches der maximalen Seitengröße ist, muß kein Teil des Textsegments dupliziert werden. Der Stapel wird automatisch den Erfordernissen entsprechend erweitert. Das Datensegment wird durch Aufrufe des `brk`(2)-Systemaufrufs erweitert.

SIEHE AUCH

`cc`(1), `ld`(1), `brk`(2), `elf`(3E).
Kapitel "Objektdateien" in "Leitfaden und Werkzeuge für die Programmierung mit C".

ar - Format der Archivdatei

```
#include <ar.h>
```

Das Archivkommando `ar` wird zum Zusammenfassen mehrerer Dateien in eine Datei verwendet. Archive werden hauptsächlich als Bibliotheken verwendet, die vom Binder `ld` durchsucht werden.

Jedes Archiv beginnt mit der magischen Archivzeichenkette.

```
#define ARMAG "!<arch>\n"       /* magische Zeichenkette */
#define SARMAG 8                /* Länge der magischen Zeichenkette */
```

Nach der magischen Archivzeichenkette folgen die Archivdateielemente. Vor jedem Dateielement steht ein Dateielement-Kopf in folgendem Format:

```
#define  ARFMAG    "`\n"

struct  ar_hdr                  /* Dateikopf*/
{
    char    ar_name[16];        /* '/' abgeschlossener Dateiname */
    char    ar_date[12];        /* Datum der Datei */
    char    ar_uid[6];          /* Dateibenutzernummer */
    char    ar_gid[6];          /* Dateigruppennumer */
    char    ar_mode[8];         /* Dateimodus (oktal) */
    char    ar_size[10];        /* Dateigröße*/
    char    ar_fmag[2];         /* Abschlußzeichenkette des Kopfes*/
};
```

Alle Daten in den Dateiköpfen sind abdruckbarer ASCII-Code. Die numerischen Informationen in den Köpfen werden als Dezimalzahlen gespeichert (mit Ausnahme von `ar_mode`, der in Oktaldarstellung angegeben wird). Wenn das Archiv abdruckbare Dateien enthält, ist es selbst auch abdruckbar.

Wenn der Dateiname paßt, enthält das Feld `ar_name` den Namen direkt, wird durch einen Schrägstrich (/) beendet und rechts mit Leerstellen aufgefüllt. Wenn der Name nicht paßt, enthält `ar_name` einen Schrägstrich (/), gefolgt von der dezimalen Darstellung des Relativzeigers des Namens in der unten beschriebenen Zeichenkettentabelle des Archives.

Das Feld `ar_date` ist das Änderungsdatum der Datei zum Zeitpunkt der Eintragung in das Archiv. Archive in diesem Format können von einem System zum anderen übertragen werden, solange das portierbare Archivkommando `ar` verwendet wird.

Jedes Archivdateielement beginnt an einer geraden Byte-Grenze; falls nötig, wird ein Neue-Zeile-Zeichen (new line) zwischen den Dateien eingefügt. Die angegebene Größe gibt die tatsächliche Größe der Datei ohne Füllzeichen an.

Es ist zu beachten, daß in einer Archivdatei keine Möglichkeit für leere Bereiche gegeben ist.

Jedes Archiv, das Objektdateien enthält (siehe `a.out`(4)), enthält eine Archivsymboltabelle. Diese Symboltabelle wird vom Binder `ld` verwendet, um zu bestimmen, welche Teile des Archivs während des Bindens geladen werden müssen. Die Archivsymboltabelle, sofern vorhanden, ist immer die erste Datei im Archiv (wird aber nie aufgelistet) und wird automatisch von `ar` erzeugt und/oder aktualisiert.

Die Archivsymboltabelle hat einen Namen der Länge 0 (d.h. `ar_name[0]` ist `'/'`), `ar_name[1]==' '`, usw.). Alle 'Wörter' in dieser Symboltabelle haben vier Bytes und verwenden die unten gezeigte, rechnerunabhängige Codierung. Alle Rechner verwenden die hier beschriebene Codierung für die Symboltabelle, selbst wenn die 'natürliche' Bytefolge anders ist.

0x01020304

0	1	2	3
01	02	03	04

Der Inhalt dieser Datei ist folgendermaßen:

1. die Anzahl der Symbole; Länge: 4 Bytes
2. das Feld der Relativzeiger in die Archivdatei; Länge: 4 Bytes * 'Anzahl der Symbole'
3. die Namenstabelle; Länge: *ar_size* - 4 Bytes * ('Anzahl der Symbole' + 1)

Als Beispiel definiert die folgende Symboltabelle vier Symbole. Das Archivmitglied beim Datei-Offset 114 definiert `Name` und `Objekt`. Das Archivmitglied beim Offset 426 definiert `Funktion` und eine zweite Version von `Name`.

Offset	+0	+1	+2	+3	
0	4				4 Offset-Einträge
4	114				Name
8	114				Objekt
12	426				Funktion
16	426				Name
20	n	a	m	e	
24	\0	o	b	j	
28	e	c	t	\0	
32	f	u	n	c	
36	t	i	o	n	
40	\0	n	a	m	
44	e	\0			

Die Anzahl der Symbole und das Feld der Relativzeiger werden mit `sgetl` und `sputl` verwaltet. Die Namenstabelle enthält genau soviele mit Null-Byte endende Zeichenketten, wie Elemente im Relativzeigerfeld vorhanden sind. Jeder Relativzeiger entspricht einem Namen in der Namenstabelle (in derselben Reihenfolge). Die Bezeichnungen in der Namenstabelle sind alle die definierten globalen Symbole, die in den Objektdateien im Archiv zu finden sind. Jeder Relativzeiger ist die Adresse des Archivkopfes für das zugehörige Symbol.

Wenn der Name eines Archivmitglieds länger als 15 Bytes ist, enthält ein besonderes Archivmitglied eine Tabelle von Dateinamen, denen jeweils ein Schrägstrich und eine Neue-Zeile-Zeichen folgt. Das Mitglied der Namenstabelle (Zeichenkettentabelle), sofern vorhanden, steht vor allen 'normalen' Archivmitgliedern. Die besondere Archivsymboltabelle ist kein 'normales' Mitglied und muß ganz vorne stehen, wenn sie existiert. Der Eintrag `ar_name` des Kopfes der Zeichenkettentabelle enthält einen Namen der Länge Null `ar_name[0]=='/'`, gefolgt von einem abschließenden Schrägstrich (`ar_name[1]=='/'`), gefolgt von Leerzeichen (`ar_name[2]=='\ '`, usw.). Offsets in die Zeichenkettentabelle beginnen bei Null. Beispiele für `ar_name`-Werte für kurze und lange Dateinamen finden sie unten.

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	f	i	l	e	_	n	a	m	e	_
10	s	a	m	p	l	e	/	\n	l	o
20	n	g	e	r	f	i	l	e	n	a
30	m	e	x	a	m	p	l	e	/	\n

Mitgliedsname	ar_name	Bemerkung
short-name	short-name/	Nicht in der Zeichenkettentabelle
file_name_sample	/0	Offset 0 in der Zeichenkettentabelle
longerfilenamexample	/18	Offset 18 in der Zeichenkettentabelle

SIEHE AUCH

`ar`(1), `ld`(1), `strip`(1), `sputl`(3X), `a.out`(4).

HINWEIS

`strip` entfernt alle Symboltabellen-Einträge des Archivs. Die Symboltabellen-Einträge müssen dann wieder über die Option `-ts` des Kommandos `ar` hergestellt werden, bevor das Archiv von dem Binder `ld` verwendet werden kann.

limits - Include-Datei für implementierungsabhängige Konstanten

#include <limits.h>

Die Include-Datei `limits.h` ist eine Liste von Größenbegrenzungen, die durch eine spezielle Implementierung des Betriebssystems auferlegt werden. Alle Werte sind dezimal angegeben.

```
ARG_MAX       5120                  /* max. Länge der Argumente für exec */
CHAR_BIT      8                     /* # Bits in einem char */
CHAR_MAX      127                   /* max. ganzzahliger Wert eines char */
CHAR_MIN      -128                  /* kleinster ganzzahliger Wert eines char */
CHILD_MAX     25                    /* max. # Prozesse pro Benutzer */
CLK_TCK       _sysconf(3)           /* # von Uhrtakten je Sekunde */
DBL_DIG       15                    /* Ziffernstellen eines double */
DBL_MAX       1.7976931348623157E+308  /* max. Dezimalwert eines double */
DBL_MIN       2.2250738585072014E-308  /* min. Dezimalwert eines double*/
FCHR_MAX      1048576               /* max. Größe einer Datei in Bytes */
FLT_DIG       6                     /* Ziffernstellen eines float */
FLT_MAX       3.40282347E+38F       /* max. Dezimalwert eines float */
FLT_MIN       1.17549435E-38F       /* min. Dezimalwert eines float */
INT_MAX       2147483647            /* max. Dezimalwert eines int */
INT_MIN       -2147483648           /* min. Dezimalwert eines int */
LINK_MAX      1000                  /* max. # Verweise auf eine Datei */
LOGNAME_MAX   8                     /* max. # Zeichen in einem Login-Namen */
LONG_BIT      32                    /* max. # Bits in einem long */
LONG_MAX      2147483647            /* max. Dezimalwert einer long */
LONG_MIN      -2147483648           /* min. Dezimalwert einer long */
MAX_CANON     256                   /* max. Anzahl Bytes einer Zeile mit
                                       kanonischer Verarbeitung */
MAX_INPUT     512                   /* max. Größe eines Eingabepuffers für char */
MB_LEN_MAX    5                     /* max. # Bytes in einem Mehrbytezeichen */
NAME_MAX      14                    /* max. # Zeichen in einem Dateinamen */
NGROUPS_MAX   16                    /* max. # Gruppen für einen Benutzer */
NL_ARGMAX     9                     /* max. Wert von "Ziffern" in Aufrufen von
                                       NLS printf() und scanf() */
NL_LANGMAX    14                    /* max. # Bytes in LANG Namen */
NL_MSGMAX     32767                 /* max. Anzahl Nachrichten */
NL_NMAX       1                     /* max. # Bytes bei N-zu-1 Zeichen Abbildungen */
NL_SETMAX     255                   /* max. set number */
NL_TEXTMAX    255                   /* max. # Bytes in einer Nachricht */
NZERO         20                    /* Standard Prozeß Priorität */
OPEN_MAX      60                    /* max. # Dateien, die ein Prozeß öffnen kann */
PASS_MAX      8                     /* max. # Zeichen in einem Paßwort */
PATH_MAX      1024                  /* max. # Zeichen in einem Pfadnamen */
PID_MAX       30000                 /* max. Wert für eine Prozeßnummer */
PIPE_BUF      5120                  /* max. # Bytes als Schreibeinheit auf eine Pipe */
PIPE_MAX      5120                  /* max. # Bytes die in eine Pipe
                                       geschrieben werden können */
SCHAR_MAX     127                   /* max. Dezimalwert eines signed char */
SCHAR_MIN     (-128)                /* min. Dezimalwert eines signed char */
SHRT_MAX      32767                 /* max. Dezimalwert eines short int */
SHRT_MIN      (-32768)              /* min. Dezimalwert eines short int */
STD_BLK       1024                  /* # Bytes in einem physischen E/A-Block */
SYS_NMLN      257                   /* 4.0 Größe der utsname Elemente */
                                    /* auch in sys/utsname.h definiert */
SYSPID_MAX    1                     /* max. Prozeßnummer der Systemprozesse */
TMP_MAX       17576                 /* max. # von einmaligen Namen generiert
                                       durch tmpnam */
UCHAR_MAX     255                   /* max. Dezimalwert eines unsigned char */
UID_MAX       60000                 /* max. Wert für eine Benutzer- oder Gruppennummer */
UINT_MAX      4294967295            /* max. Dezimalwert eines unsigned int */
ULONG_MAX     4294967295            /* max. Dezimalwert eines unsigned long int */
USHRT_MAX     65535                 /* max. Dezimalwert eines unsigned short int */
USI_MAX       4294967296            /* max. Dezimalwert einer unsigned */
WORD_BIT      32                    /* # Bits in einem "Wort" oder int */
```

Die folgenden POSIX-Definitionen sind die Grenzwerte, die von einer Anwendung höchstens verwendet werden dürfen, wenn sie POSIX-konform sein will. Konforme Implementierungen stellen Werte mindestens dieser Größe zur Verfügung.

```
_POSIX_ARG_MAX      4096   /* max. Länge der Argumente für exec */
_POSIX_CHILD_MAX    6      /* max. # Prozesse pro Benutzernummer */
_POSIX_LINK_MAX     8      /* max. # Verweise auf eine Datei */
_POSIX_MAX_CANON    255    /* max. # Bytes in einer Eingabezeile */
_POSIX_MAX_INPUT    255    /* max. # Bytes in einer Terminal
                              Eingabewarteschlange */
_POSIX_NAME_MAX     14     /* # Bytes in einem Dateinamen */
_POSIX_NGROUPS_MAX  0      /* max. # Gruppen in einem Prozeß */
_POSIX_OPEN_MAX     16     /* max. # geöffneter Dateien eines Prozesses */
_POSIX_PATH_MAX     255    /* max. # Zeichen in einem Pfadnamen */
_POSIX_PIPE_BUF     512    /* max. # Bytes einer atomaren Schreib-
                              operation in eine Pipe */
```

sccsfile - Format der SCCS-Datei

Eine SCCS-Datei (Source Code Control System - Quellcode-Verwaltung) ist eine ASCII-Datei. Sie besteht aus sechs logischen Teilen: der Prüfsumme, der Delta-Tabelle (Informationen über jedes Delta), den Benutzernamen (Login-Namen und/oder numerische Gruppennummern von Benutzern, die Deltas hinzufügen können), den Anzeigern (Definitionen der internen Schlüsselwörter), den Kommentaren (beliebige beschreibende Angaben über die Datei) und dem Hauptteil (die tatsächlichen Textzeilen, die mit Steuerzeilen vermischt sind).

Überall in einer SCCS-Datei gibt es Zeilen, die mit dem Zeichen ASCII SOH (oktal 001) beginnen. Dieses Zeichen wird im nachfolgenden Text als Steuerzeichen bezeichnet und grafisch durch @ dargestellt. Bei jeder unten beschriebenen Zeile, bei der das Steuerzeichen nicht angegeben ist, wird verhindert, daß sie mit dem Steuerzeichen beginnen kann.

Einträge der Form *DDDDD* stellen eine Fünf-Ziffern-Zeichenkette (eine Zahl zwischen 00000 und 99999) dar.

Jeder logische Teil einer SCCS-Datei wird nachstehend im einzelnen beschrieben.

Prüfsumme

Die Prüfsumme ist die erste Zeile einer SCCS-Datei. Die Zeile hat folgende Form:

@h*DDDDD*

Der Wert der Prüfsumme ist die Summe aller Zeichen mit Ausnahme der Zeichen in der ersten Zeile. Das @h liefert die magische Zahl (oktal) 064001, abhängig von der Bytefolge.

Delta-Tabelle

Die Delta-Tabelle enthält eine variable Anzahl von Einträgen in einer der folgenden Formen:

@s *DDDDD/DDDDD/DDDDD*
@d *<Typ> <SCCS ID> Ja/Mo/Ta St:Mi:Se <Benutzer> DDDDD DDDDD*
@i *DDDDD ...*
@x *DDDDD ...*
@g *DDDDD ...*
@m *<MR Nummer>*
. . .
@c *<Kommentare>* . . .
. . .
@e

Die erste Zeile (@s) enthält die Anzahl der jeweils eingefügten oder gelöschten bzw. unveränderten Zeilen. Die zweite Zeile (@d) enthält den Typ des Deltas (normal: D oder entfernt: R), die SID-Nummer des Deltas, Datum und Uhrzeit der Erstellung des Deltas, den Login-Namen, der der realen Benutzernummer zum Zeitpunkt der Erstellung des Deltas entspricht, und die SID-Nummern des Deltas bzw. der vorherigen Deltas.

Die Zeilen @i, @x und @g enthalten die SID-Nummern der eingeschlossenen, ausgeschlossenen bzw. ignorierten Deltas. Diese Zeilen sind optional.

Die Zeilen @m (optional) enthalten jeweils eine MR-Nummer, die zum Delta gehört; die Zeilen @c enthalten Kommentare im Zusammenhang mit dem Delta. Die Zeile @e beendet den Delta-Tabelleneintrag.

Benutzernamen

Die diese Login-Namen und/oder numerischen Gruppennummern enthaltenen Zeilen sind von den Zeilen @u und @U umgeben. Eine leere Liste gestattet jedem die Erstellung eines Deltas. Jede mit einem ! beginnende Zeile verbietet der nachfolgenden Gruppe bzw. dem nachfolgenden Benutzer die Erstellung von Deltas.

Kennzeichen

Intern benutzte Schlüsselwörter. Weitere Informationen über ihre Verwendung sind in admin(1) zu finden. Jede Kennzeichenzeile hat folgende Form:

@f <*Kennzeichen*> <*optionaler Text*>

Folgende Kennzeichen sind definiert:

@f t	<Programmtyp>
@f v	<Programmname>
@f i	<Schlüsselwort>
@f b	
@f m	<Modulbezeichnung>
@f f	<Untergrenze>
@f c	<Obergrenze>
@f d	<Standard-SID>
@f n	
@f j	
@f l	<gesperrte Versionen>
@f q	<vom Benutzer definiert>
@f z	<für Verwendung in Schnittstellen reserviert>

Das Kennzeichen t definiert den Ersatztext für das Schlüsselkennwort %Y%. Das Kennzeichen v steuert die Anforderung von MR-Nummern zusätzlich zu Kommentaren; wenn das optionale Argument vorhanden ist, definiert es ein Programm zur Prüfung der MR-Nummer. Das Kennzeichen i steuert, ob Warnung oder Fehler bei der Meldung

'No id keywords' erfolgt. Wenn das Kennzeichen i nicht vorhanden ist, stellt diese Meldung lediglich eine Warnung dar; wenn das Kennzeichen i vorhanden ist, kennzeichnet diese Meldung einen schwerwiegenden Fehler (die Datei kann nicht mit get 'geholt' werden, oder das Delta wird nicht ausgeführt). Wenn das Kennzeichen b vorhanden ist, kann die Option -b im Kommando get zur Auslösung einer Verzweigung im Deltabaum verwendet werden. Das Kennzeichen m definiert die erste Wahl für den Austauschtext des Schlüsselkennworts %M%. Das Kennzeichen f definiert die niedrigste Version, d.h. die Version, unterhalb der keine Deltas hinzugefügt werden dürfen. Das Kennzeichen c definiert die höchste Version, d.h. die Version, über der keine Deltas hinzugefügt werden dürfen. Das Kennzeichen d definiert die Standard-SID, die zu verwenden ist, wenn bei einem Kommando get keine angegeben wurde. Das Kennzeichen n bewirkt, daß delta ein Null-Delta (ein Delta, das keine Änderungen anwendet) in die Versionen einsetzt, die übersprungen werden, wenn ein Delta in einer neuen Version ausgeführt wird (z.B. wenn Delta 5.1 nach Delta 2.7 ausgeführt wird, werden Versionen 3 und 4 übersprungen). Das Fehlen des Anzeigers n bewirkt, daß übersprungene Versionen völlig leer sind. Das Kennzeichen j bewirkt, daß get gleichzeitige Bearbeitungen derselben Ausgangs-SID zuläßt. Das Kennzeichen l definiert eine *Liste* von Versionen, die gegen ein Editieren gesperrt sind. Das Kennzeichen q definiert den Ersatz für das Schlüsselkennwort %Q%. Das Kennzeichen z wird bei besonderen Schnittstellenprogrammen verwendet.

Kommentare

Beliebiger Text wird von den Zeilen @t und @T eingerahmt. Der Kommentarteil enthält normalerweise eine Beschreibung des Zweckes der Datei.

Hauptteil

Der Hauptteil besteht aus Textzeilen und Steuerzeilen. Im Gegensatz zu den Steuerzeilen fangen die Textzeilen nicht mit einem Steuerzeichen an. Es gibt drei Arten von Steuerzeilen: einfügen, löschen und beenden, die dementsprechend durch:

@I *DDDDD*
@D *DDDDD*
@E *DDDDD*

dargestellt werden.

Die Ziffernfolge ist die SID-Nummer, die dem Delta für die Steuerzeile entspricht.

SIEHE AUCH

admin(1), delta(1), get(1), prs(1).

strftime - Sprachenabhängige Zeichenketten

Für eine Umgebung kann nur eine druckbare Datei existieren, in der die Informationen über die Formatierung von Datum und Uhrzeit stehen. Diese Datei muß im Verzeichnis `/usr/lib/locale/<locale>/LC_TIME` stehen. Sie hat folgenden Inhalt:

1. Abkürzungen der Monatsnamen (sortiert)
2. Monatsnamen (sortiert)
3. Abkürzungen der Wochentage (sortiert)
4. Wochentage (sortiert)
5. Standardzeichenketten für die Angabe der lokalen Zeit (`%X`) und des lokalen Datums (`%x`).
6. Standardformat für `cftime`, wenn kein Argument angegeben wurde, oder wenn das Argument Null ist.
7. AM (ante meridian) Zeichenkette
8. PM (post meridian) Zeichenkette

Jede Zeichenkette steht alleine auf einer Zeile. Alle Leerzeichen sind signifikant. Die Reihenfolge der Zeichenketten in der Datei muß der oben angegebenen Reihenfolge entsprechen.

BEISPIEL

```
/usr/lib/locale/C/LC_TIME

Jan
Feb
 ...
January
February
 ...
Sun
Mon
 ...
Sunday
Monday
 ...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
```

DATEIEN

`/usr/lib/locale/<locale>/LC_TIME`

SIEHE AUCH

`ctime`(3C), `setlocale`(3C), `strftime`(3C).

timezone - Standard-Zeitzone des Systems

/etc/TIMEZONE

Mit dieser Datei wird die Zeitzonen-Umgebungsvariable `TZ` eingestellt und exportiert.

Die Datei wird in andere Dateien 'eingeblendet', denen die Zeitzone bekannt sein muß.

BEISPIELE

`/etc/TIMEZONE` für Mitteleuropa

```
#     Zeitzone
TZ=MEZ-1MSZ-2
export TZ
```

SIEHE AUCH

`ctime`(3C), `environ`(5).
`rc2`(1M), `profile`(4) in "Referenzhandbuch für Systemverwalter".

utmp, wtmp - Einsprungformate

```
#include <utmp.h>
```

Diese Dateien, die Benutzer- und Abrechnungsdaten für Kommandos wie `who`, `write` und `login` enthalten, haben folgende Struktur, die in `utmp.h` definiert ist:

```
#define  UTMP_FILE  "/var/adm/utmp"
#define  WTMP_FILE  "/var/adm/wtmp"
#define  ut_name  ut_user

struct  utmp {
  char     ut_user[8];     /* Benutzer-Login-Name */
  char     ut_id[4];       /* /sbin/inittab id (von dem Prozeß erzeugt, */
                           /* der Einträge in utmp ablegt) */
  char     ut_line[12];    /* Gerätename (Konsole, lnxx) */
  short    ut_pid;         /* Prozeßnummer */
  short    ut_type;        /* Typ des Eintrags */
  struct   exit_status {
  short    e_termination;  /* Prozeß-Ende-Status */
  short    e_exit;         /* Abbruchstatus */
  } ut_exit;               /* Ende-Status eines Prozesses, der
                              als DEAD_PROCESS gekennzeichnet ist. */
  time_t   ut_time;        /* Zeiteintrag wurde ausgeführt */
};

/* Definitionen für ut_type */
#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5 /* Prozeß von "init" erzeugt */
#define LOGIN_PROCESS  6 /* Ein "getty"-Prozeß, der auf Login wartet*/
#define USER_PROCESS   7 /* Ein Benutzerprozeß */
#define DEAD_PROCESS   8
#define ACCOUNTING     9
#define UTMAXTYPE      ACCOUNTING /* größter zulässiger Wert für ut_type */

/* Unten sind besondere Zeichenketten oder Formate angegeben, die im */
/* Feld "ut_line" angewendet werden, wenn die Abrechnung für etwas   */
/* anderes als einen Prozeß vorgenommen wird. Keine Zeichenkette für */
/* das Feld ut_line kann länger als 11 Zeichen + Null-Byte sein      */

#define!RUNLVL_MSG!"run level %c"
#define!BOOT_MSG !"system boot"
#define!OTIME_MSG !"old time"
#define!NTIME_MSG !"new time"
```

DATEIEN

`/var/adm/utmp^b`
`/var/adm/wtmp`

SIEHE AUCH

`getut`(3C)
`login`(1), `who`(1), `write`(1) in "SINIX V5.41 Kommandos".

utmpx, wtmpx - Einsprungformate

```
#include <utmpx.h>
```

utmpx(4) ist eine erweiterte Version von utmp(4).

Diese Dateien halten Benutzer und Abrechnungsinformationen für Kommandos wie who, write und login. Die folgende Struktur ist in utmpx.h definiert:

```
#define   UTMPX_FILE    "/var/adm/utmpx"
#define   WTMPX_FILE    "/var/adm/wtmpx"
#define   ut_name     ut_user
#define   ut_xtime    ut_tv.tv_sec

struct utmpx  {
   char    ut_user[32];           /* Benutzerkennung */
   char    ut_id[4];              /* inittab Nummer */
   char    ut_line[32];           /* Gerätename (console, lnxx) */
   pid_t   ut_pid;                /* Prozeßnummer */
   short   ut_type;               /* Typ des Eintrags */
   struct exit_status ut_exit;    /* Prozeß-Ende/Abbruchstatus */
   struct timeval ut_tv;          /* Zeit der Eintragserstellung */
   long    ut_session;            /* Sitzungsnummer, für Fensterprogramme */
   long    pad[5];                /* reserviert */
   short   ut_syslen;             /* gültige Länge von ut_host */
                                  /* einschließlich Nullzeichen */
   char    ut_host[257];          /* ferner Rechnername */
  } ;

      /* Definitionen für ut_type */

#define   EMPTY          0
#define   RUN_LVL        1
#define   BOOT_TIME      2
#define   OLD_TIME       3
#define   NEW_TIME       4
#define   INIT_PROCESS   5  /* Prozeß durch "init" erzeugt */
#define   LOGIN_PROCESS  6  /* Ein "getty"-Prozeß, der auf login wartet */
#define   USER_PROCESS   7  /* Ein Benutzerprozeß */
#define   DEAD_PROCESS   8
#define   ACCOUNTING     9

#define   UTMAXTYPE      ACCOUNTING  /* Größter legaler Wert von ut_type */

/* Unten sind besondere Zeichenketten oder Formate angegeben, die im */
/* Feld "ut_line" angewendet werden, wenn die Abrechnung für etwas   */
/* anderes als einen Prozeß vorgenommen wird. Keine Zeichenkette für */
/* das Feld ut_line kann länger als 11 Zeichen + Null-Byte sein      */

#define   RUNLVL_MSG     "run-level %c"
#define   BOOT_MSG       "system boot"
#define   OTIME_MSG      "old time"
#define   NTIME_MSG      "new time"
#define   MOD_WIN        10
```

DATEIEN

/var/adm/utmpx und /var/adm/wtmpx

SIEHE AUCH

getutx(3C)
login(1), who(1), write(1) in "SINIX V5.41 Kommandos".

Hilfsmittel und verschiedene Funktionen

intro - Einführung

In diesem Kapitel werden verschiedene Hilfsmittel für die Programmierung wie z.B. Makropakete, Zeichensatztabellen usw. dargestellt.

Außerdem werden einige weitere Bibliotheken und Funktionen beschrieben.

ascii - Tabelle des ASCII-Zeichensatzes

`ascii` ist eine Tabelle des ASCII-Zeichensatzes. Es wird die oktale und hexadezimale Darstellung der Zeichen angegeben.

Oktaler ASCII-Wert

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

Hexadezimaler ASCII-Wert

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

DATEIEN

`/usr/pub/ascii`

environ - Benutzer-Umgebung

Wenn ein Prozeß mit der Ausführung beginnt, stellen die exec-Routinen ein Feld von Zeichenketten zur Verfügung, das "Umgebung" genannt wird (siehe exec(2)). Konventionen zufolge haben diese Zeichenketten die Form *Variable=Wert*, zum Beispiel PATH=/sbin:/usr/sbin. Diese Umgebungsvariablen ermöglichen, daß die Informationen zur Umgebung eines Programms auch Anwendungen zur Verfügung gestellt wird. Die folgenden Umgebungsvariablen müssen in der Ziel-Laufzeit-Umgebung gesetzt sein:

HOME ist der Name des Login-Verzeichnisses des Benutzers und wird vom login(1) der Paßwortdatei gesetzt (siehe passwd(4)).

LANG ist die Zeichenkette, die verwendet wird, um lokale Informationen anzugeben, die Benutzern erlauben, mit verschiedenen nationalen Konventionen zu arbeiten. Die Funktion setlocale(3C) verwendet die Umgebungsvariable LANG, wenn sie mit "" als *locale*-Argument aufgerufen wird. LANG wird als Standardvereinbarung für die lokale Umgebung verwendet, wenn die zugehörige Umgebungsvariable für eine bestimmte Kategorie nicht gesetzt ist.

Wenn zum Beispiel setlocale als

```
setlocale(LC_CTYPE, "")
```

aufgerufen wird, wird zuerst die Umgebungsvariable LC_CTYPE abgefragt, um zu sehen, ob sie gesetzt und ungleich Null ist. Wenn LC_CTYPE nicht gesetzt oder Null ist, überprüft setlocale die Umgebungsvariable LANG, um festzustellen, ob diese gesetzt und ungleich Null ist. Wenn beide, LANG und LC_CTYPE, nicht gesetzt oder gleich Null sind, wird die standardmäßig eingestellte lokale C-Umgebung verwendet, um die Kategorie LC_CTYPE zu setzen.

Die meisten Kommandos rufen als erstes

```
setlocale(LC_ALL, "")
```

auf. Dann kann das Kommando mit verschiedenen nationalen Konventionen verwendet werden, indem einfach die passende Umgebungsvariable gesetzt wird.

Die folgenden Umgebungsvariablen werden unterstützt, um jeder Kategorie von setlocale(3C) zu entsprechen:

LC_COLLATE
: Diese Kategorie gibt die verwendete Sortierfolge an. Die diesbezüglichen Informationen werden in einer Datenbank gespeichert, die von dem Kommando colltbl(1M) erzeugt wird. Diese Umgebungsvariable beeinflußt strcoll(3C) und strxfrm(3C).

LC_CTYPE
: Diese Kategorie legt die Klassifikation von Zeichen, Umwandlung von Zeichen und die Größe von Mehrbyte-Zeichen fest. Die diesbezüglichen Informationen werden in einer Datenbank gespeichert, die von dem Kommando `chrtbl`(1M) erzeugt wird. Die Standardvereinbarung für `C` entspricht der 7-Bit-Zeichenmenge. Diese Umgebungsvariable wird von `ctype`(3C), `mbchar`(3C) und vielen Kommandos verwendet. Zum Beispiel: `cat`(1), `ed`(1), `ls`(1) und `vi`(1).

LC_MESSAGES
: Diese Kategorie gibt die Sprache an, die von der Datenbank für Meldungen verwendet wird. Zum Beispiel kann eine Anwendung eine Datenbank mit französischen Meldungen und eine andere mit deutschen Meldungen haben. Datenbanken für Meldungen werden mit dem Kommmando `mkmsgs`(1M) erzeugt. Diese Umgebungsvariable wird von `exstr`(1), `gettxt`(1), `gettxt`(3C) und `srchtxt`(1) verwendet.

LC_MONETARY
: Diese Kategorie spezifiziert die Währungssymbole und Trennzeichen für eine bestimmte Umgebung. Die diesbezüglichen Informationen werden in einer Datenbank gespeichert, die von dem Kommando `montbl`(1M) erzeugt wird. Diese Umgebungsvariable wird von `localeconv`(3C) verwendet.

LC_NUMERIC
: Diese Kategorie gibt die Trennzeichen für Dezimalstellen und Tausenderstellen an. Die diesbezüglichen Informationen werden in einer Datenbank gespeichert, die von dem Kommando `chrtbl`(1M) erzeugt wird. Die Standardvereinbarung für `C` verwendet . als Trennzeichen für Dezimalstellen und kein Zeichen für die Tausenderstellen. Diese Umgebungsvariable wird von `localeconv`(3C), `printf`(3C) und `strtod`(3C) verwendet.

LC_TIME
: Diese Kategorie spezifiziert Datums- und Zeitformate. Die diesbezüglichen Informationen werden in einer Datenbank gespeichert, die in `strftime`(4) angegeben ist. Die Standardvereinbarung für `C` entspricht den Datums- und Zeitformaten der USA. Diese Umgebungsvariable wird von vielen Kommandos und Funktionen verwendet. Zum Beispiel: `at`(1), `calendar`(1), `date`(1), `strftime`(3C) und `getdate`(3C).

MSGVERB
: steuert, welche Komponenten der Standardformate für Meldungen `fmtmsg` auswählt, wenn Meldungen an `stderr` ausgegeben werden (siehe `fmtmsg`(1) und `fmtmsg`(3C)).

SEV_LEVEL
: vereinbart Fehlerstufen und verbindet Druckformate mit ihnen im Standard-Fehlerformat (siehe `addseverity`(3C), `fmtmsg`(1) und `fmtmsg`(3C)).

NETPATH ist eine durch Doppelpunkte getrennte Liste von Netzwerkbezeichnern. Ein Netzwerkbezeichner ist eine Zeichenkette, die von der Netzwerkauswahl-Komponente des Systems verwendet wird, um anwendungsspezifische Standardsuchpfade für Netzwerke zur Verfügung zu stellen. Ein Netzwerkbezeichner muß aus Zeichen ungleich Null bestehen und muß mindestens die Länge 1 haben. Es ist keine maximale Länge festgelegt. Netzwerkbezeichner werden meistens von dem Systemverwalter gewählt. Ein Netzwerkbezeichner ist auch das erste Feld in jedem `/etc/netconfig`-Dateieintrag. `NETPATH` stellt also eine Verbindung zu der Datei `/etc/netconfig` und die Informationen über ein Netzwerk, das in dem Netzwerkeintrag enthalten ist, zur Verfügung. `/etc/netconfig` wird vom Systemverwalter gewartet. Die in `getnetpath`(3N) beschriebenen Bibliotheksroutinen greifen auf die Umgebungsvariable `NETPATH` zu.

NLSPATH enthält eine Reihe von Schablonen, die `catopen`(3C) beim Versuch, Meldungskataloge zu lokalisieren, verwendet. Jede Schablone besteht aus einem optionalen Präfix, einem oder mehreren Ersetzungsfeldern, einem Dateinamen und einem optionalen Suffix.

```
NLSPATH="/system/nlslib/%N.cat"
```

legt fest, daß `catopen` nach allen Meldungskatalogen in dem Verzeichnis `/system/nlslib` suchen soll, wobei der Name des Katalogs aus dem Parameter *Name*, der von `catopen` übergeben wird, also `%N`, sowie aus dem Suffix `.cat` zusammengesetzt werden soll.

Die Ersetzungsfelder bestehen aus einem `%`, gefolgt von einem Schlüsselwort aus einem Buchstaben. Die folgenden Schlüsselwörter sind zur Zeit definiert:

```
%N   Der Wert des Parameters Name, der an
     catopen() übergeben wurde.
%L   Wert von LANG.
%l   Sprachelement von LANG.
%t   Ortselement von LANG.
%c   Codeliste-Element von LANG.
%%   ein einziges %-Zeichen.
```

Es wird eine leere Zeichenkette eingesetzt, wenn der angegebene Wert zur Zeit nicht definiert ist. Die Trennzeichen _ und . werden in `%t`- und `%c`-Ersetzungen nicht eingeschlossen.

In `NLSPATH` definierte Schablonen werden durch Doppelpunkte getrennt. Ein führender Doppelpunkt oder zwei benachbarte Doppelpunkte `::` bedeuten dasselbe wie die Angabe von `%N`.

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

zeigt `catopen` an, daß es in *Name*, *Name*`.cat` und `/nlslib/$LANG/`*Name*`.cat` nach dem verlangten Meldungskatalog suchen soll.

PATH — Die Folge von Verzeichnis-Präfixen, die `sh`(1), `time`(1), `nice`(1), `nohup`(1), usw. bei der Suche nach einer Datei mit einem unvollständigen Pfadnamen verwenden. Die Präfixe werden durch Doppelpunkte getrennt. `login`(1) setzt `PATH=/usr/bin` (siehe `sh`(1)).

TERM — ist die Art des Terminals, für das die Ausgabe vorbereitet werden muß. Diese Informationen werden von Kommandos wie `vi`(1) verwendet, die besondere Fähigkeiten des Terminals nutzen.

TZ — enthält Informationen über die Zeitzone. Der Inhalt dieser Umgebungsvariablen wird von den Funktionen `ctime`(3C), `localtime` (siehe `ctime`(3C)), `strftime`(3C) und `mktime`(3C) verwendet, um die Standard-Zeitzone zu überschreiben. Wenn das erste Zeichen von `TZ` ein Doppelpunkt ist, wird das Verhalten von der Implementierung definiert, sonst hat `TZ` die Form:

std offset [dst [offset],[start[/time],end[/time]]]

std und *dst* enthalten drei oder mehr Bytes für die Bezeichnung der Standard-Zeitzone (*std*) und der Sommerzeitzone (*dst*). Nur *std* ist notwendig. Wenn *dst* fehlt, erscheint die Sommerzeit nicht in dieser Umgebung. Klein- und Großbuchstaben sind erlaubt. Alle Zeichen außer einem führenden Doppelpunkt, Ziffern, einem Komma, einem Minuszeichen oder einem Pluszeichen sind erlaubt.

offset zeigt den Wert an, der zur Ortszeit addiert werden muß, um zur universalen Bezugszeit zu gelangen. Der Offset hat folgende Form:

hh [:*mm* [:*ss*]]

Die Minuten (*mm*) und Sekunden (*ss*) sind optional. Die Stunde (*hh*) ist notwendig und kann aus einer einzelnen Ziffer bestehen. Der *offset* nach *std* ist notwendig. Wenn kein *offset* nach *dst* folgt, wird die Sommerzeit als eine Stunde vor der Standardzeit angenommen. Es können eine oder mehrere Ziffern verwendet werden, der Wert wird immer als eine Dezimalzahl interpretiert. Die Stunde muß zwischen 0 und 24 liegen, und die Minuten und Sekunden (soweit angegeben) zwischen 0 und 59. Werte außerhalb dieser Bereiche können zu unvorhersehbarem Verhalten führen. Wenn ein - am Anfang steht, ist die Zeitzone östlich vom Nullmeridian; sonst ist sie westlich davon (was auch durch ein optionales + Zeichen angezeigt werden kann).

start/time,end/time zeigt an, wann zur Sommerzeit und wann wieder zurück gewechselt werden soll, wobei *start/time* beschreibt, wann der Wechsel von der Standardzeit zur Sommerzeit und *end/time*, wann der Wechsel zurück stattfindet. Jedes *time*-Feld beschreibt, wann der Wechsel vorgenommen wird.

Die Formate von *start* und *end* haben eine der folgenden Formen:

`J`*n* Der Tag *n* (1 <= *n* <= 365) nach dem Julianischen Kalender; Tage, die nur in Schaltjahren auftreten, werden nicht gezählt. Also ist jedes Jahr der 28. Februar Tag 59 und der 1. März Tag 60. Es ist unmöglich, sich auf den gelegentlichen 29. Februar zu beziehen.

`n` von Null beginnender Tag (0 <= *n* <= 365) nach dem Julianischen Kalender; Tage, die nur in Schaltjahren auftreten, werden gezählt, und es ist möglich, sich auf den 29. Februar zu beziehen.

`M`*m.n.d*
Der Tag *d* (0 <= *d* <= 6) der Woche *n* des Monats *m* des Jahres (1 <= *n* <= 5, 1 <= *m* <= 12), wobei Woche 5 'der letzte *d*-Tag im Monat *m*' bedeutet, der entweder in der vierten oder fünften Woche auftreten kann). Woche 1 ist die erste Woche, in der der *d*-Tag auftritt. Tag 0 ist Sonntag.

Für *start* und *end* werden implementierungsspezifische Standardwerte angenommen, wenn diese optionalen Felder nicht angegeben werden.

time hat dasselbe Format wie *offset*, außer daß kein Vorzeichen (- oder +) erlaubt ist. Der Standard, wenn *time* nicht angegeben ist, ist 02:00:00.

Weitere Namen können durch das Kommando `export` und Argumente *Name* = *Wert* in `sh`(1) oder durch `exec`(2) in die Umgebung eingebaut werden. Es ist nicht sehr ratsam, mit Shell-Variablen in Konflikte zu geraten, die häufig von `.profile`-Dateien exportiert werden: `MAIL`, `PS1`, `PS2`, `IFS` (siehe `profile`(4)).

SIEHE AUCH

`chrtbl`(1M), `colltbl`(1M), `mkmsgs`(1M), `montbl`(1M), `netconfig`(4), `strftime`(4), `passwd`(4), `profile`(4) in "Referenzhandbuch für Systemverwalter".

`exec`(2), `addseverity`(3C), `catopen`(3C), `ctime`(3C), `ctype`(3C), `fmtmsg`(3C), `getdate`(3C), `gettxt`(3C), `localeconv`(3C), `mbchar`(3C), `mktime`(3C), `printf`(3C), `strcoll`(3C), `strftime`(3C), `strtod`(3C), `strxfrm`(3C), `strftime`(4), `timezone`(4).

`cat`(1), `date`(1), `ed`(1), `fmtmsg`(1), `ls`(1), `login`(1), `nice`(1), `nohup`(1), `sh`(1), `sort`(1), `time`(1), `vi`(1) in "SINIX V5.41 Kommandos".
`getnetpath`(3N) in "Leitfaden für Programmierer: Netzwerk-Schnittstellen".

fcntl - Optionen für die Steuerung von Dateien

```
#include <fcntl.h>
```

Die Include-Datei `fcntl.h` definiert die folgenden Anfragen und Parameter zum Gebrauch durch die Funktionen `fcntl` (siehe `fcntl`(2)) und `open` (siehe `open`(2)).

Werte für Kommandos von `fcntl` (eindeutig):

```
F_DUPFD      Dateideskriptor duplizieren
F_GETFD      Dateideskriptor-Marken holen
F_SETFD      Dateideskriptor-Marken setzen
F_GETFL      Dateistatus-Marken holen
F_SETFL      Dateistatus-Marken setzen
F_GETLK      Satzsperr-Informationen holen
F_SETLK      Satzsperr-Informationen setzen
F_SETLKW     Satzsperr-Informationen setzen; warten, falls blockiert
```

Für `fcntl` werden folgende Dateideskriptor-Marken angewendet:

```
FD_CLOEXEC   obiges Dateiverzeichnis schließen
             bei Ausführen einer exec-Funktion (siehe exec(2))
```

Werte für `l_type`, um Datensätze zu sperren mit `fcntl` (eindeutig):

```
F_RDLCK      gemeinsam benutzte Sperre oder Lesesperre
F_UNLCK      entsperren
F_WRLCK      Exklusiv- oder Schreibsperre
```

Die folgenden drei Wertegruppen sind bitweise verschieden; Werte für `oflag`, angewendet durch `open`:

```
O_CREAT      Datei erstellen, falls sie nicht existiert
O_EXCL       exklusive Benutzung
O_NOCTTY     kein steuerndes Terminal zuweisen
O_TRUNC      Datei wurde abgeschnitten
```

Dateistatus-Marke, angewendet für `open` und `fcntl`:

```
O_APPEND     Appendmodus setzen
O_NDELAY     keine (Prozeß-)Blockierung
O_NONBLOCK   Entsperrmodus (POSIX)
O_SYNC       synchrones Schreiben
```

Maske für die Anwendung mit dem Dateizugriffsverfahren:

```
O_ACCMODE  Maske für Dateizugriffsverfahren
```

Dateizugriffsverfahren, angewendet für `open` und `fcntl`:

```
O_RDONLY     Öffnen zum Lesen
O_RDWR       Öffnen zum Lesen und Schreiben
O_WRONLY     Öffnen zum Schreiben
```

Die Struktur `flock` beschreibt eine Dateisperre. Sie beinhaltet die folgenden Komponenten:

```
short   l_type;    /* Sperrtyp */
short   l_whence;  /* Offsettyp */
off_t   l_start;   /* Relativer Offset in Bytes */
off_t   l_len;     /* Größe; wenn 0, dann bis EOF (Dateiende) */
long    l_sysid;   /* zurückgegeben mit F_GETLK */
pid_t   l_pid;     /* zurückgegeben mit F_GETKL */
```

SIEHE AUCH

creat(2), exec(2), fcntl(2), open(2).

jagent - Host-Steuerung für grafikfähiges Terminal

```
#include <sys/jioctl.h>
int ioctl (int cntlfd, JAGENT, &arg);
```

Wenn der Systemaufruf `ioctl`(2) an einem `xt`(7)-Gerät mit der Anforderung `JAGENT` ausgeführt wird, wird dadurch einem Host-Programm das Senden von Informationen an ein grafikfähiges Terminal ermöglicht.

`ioctl` hat drei Argumente:

`cntlfd` der Dateideskriptor für den `xt`(7)-Steuerkanal

`JAGENT` die `xt`(7) `ioctl`(2)-Aufforderung zum Aufrufen einer `agent`-Funktion für ein grafikfähiges Terminal

`&arg` die Adresse einer `bagent`-Struktur, die wie folgt in `sys/jioctl.h` definiert ist:

```
struct  bagent {
   int size;   /* Größe von src bzw. nach Rückkehr von dest */
    char*src;   /* die src-Zeichenkette */
    char*dest;  /* die dest-Zeichenkette */
};
```

Der Zeiger `src` muß mit einer Byte-Zeichenkette initialisiert werden, die zum grafikfähigen Terminal gesendet wird (siehe `layers`(5) mit einer Liste von `JAGENT`-Zeichenkette, die von grafikfähigen Terminals akzeptiert werden). Der `dest`-Zeiger muß mit der Adresse eines Puffers initialisiert werden, in dem eine von dem Terminal zurückgegebene Byte-Zeichenkette abgelegt werden kann. Bei dem Aufruf von `ioctl`(2) muß das Argument `size` die Länge der `src`-Zeichenkette sein. Bei Rückkehr wird `size` von `ioctl`(2) auf die Länge der Zeichenkette `dest` gesetzt.

SIEHE AUCH

`ioctl`(2), `layers`(5), `libwindows`(3X).
`xt(7)` in "Leitfaden für Programmierer: STREAMS".

ERGEBNIS

Nach erfolgreicher Ausführung wird die Größe der Zeichenkette `dest` zurückgegeben. Bei Auftreten eines Fehlers wird -1 zurückgegeben.

langinfo - Konstanten für die Sprachumgebung

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

Diese Include-Datei enthält die Konstanten zur Identifikation der Datenobjekte für die Sprachumgebung. Der Modus der Objekte wird in `nl_types` gegeben.

```
DAY_1       'sunday'
DAY_2       'monday'
DAY_3       'tuesday'
DAY_4       'wednesday'
DAY_5       'thursday'
DAY_6       'friday'
DAY_7       'saturday'
ABDAY_1     'sun'
ABDAY_2     'mon'
ABDAY_3     'tue'
ABDAY_4     'wed'
ABDAY_5     'thur'
ABDAY_6     'fri'
ABDAY_7     'sat'
MON_1       'january'
MON_2       'febuary'
MON_3       'march'
MON_4       'april'
MON_5       'may'
MON_6       'june'
MON_7       'july'
MON_8       'august'
MON_9       'september'
MON_10      'october'
MON_11      'november'
MON_12      'december'
ABMON_1     'jan'
ABMON_2     'feb'
ABMON_3     'mar'
ABMON_4     'apr'
ABMON_5     'may'
ABMON_6     'jun'
ABMON_7     'jul'
ABMON_8     'aug'
ABMON_9     'sep'
ABMON_10    'oct'
ABMON_11    'nov'
ABMON_12    'dec'
RADIXCHAR   '.'
THOUSEP     ','
YESSTR      'yes'
NOSTR       'no'
CRNCYSTR    Währungssymbol der lokalen Umgebung
D_T_FMT     Standardformat der lokalen Umgebung für Datum und Zeit
D_FMT       Standardformat der lokalen Umgebung für Datum
T_FMT       Standardformat der lokalen Umgebung für Zeit
AM_STR      'AM'
PM_STR      'PM'
```

Diese Informationen werden durch `nl_langinfo` gelesen.

Die Objekte `CRNCYSTR`, `RADIXCHAR` und `THOUSEP` werden aus den Feldern `currency_symbol`, `decimal_point` und `thousands_sep` der von `localeconv` zurückgelieferten Struktur gewonnen.

Die Objekte T_FMT, D_FMT, D_T_FMT, YESSTR und NOSTR werden aus einem speziellen Meldungskatalog Xopen_info gelesen, welcher für jede unterstützte Umgebung im entsprechenden Verzeichnis angelegt werden soll (siehe gettxt(3C) und mkmsgs(1M)). Dieser Katalog enthält die Meldungen in der Reihenfolge T_FMT, D_FMT, D_T_FMT, YESSTR und NOSTR.

Alle anderen Objekte werden wie bei strftime behandelt.

SIEHE AUCH

gettxt(3C), localeconv(3C), nl_langinfo(3C), strftime(3C), cftime(4), nl_types(5). chrtbl(1), mkmsgs(1M) in "Referenzhandbuch für Systemverwalter".

layers - Protokoll für Host und grafikfähiges Terminal

`layers` sind asynchrone Fenster, die vom Betriebssystem in einem grafikfähigen Terminal unterstützt werden. Die Kommunikation zwischen den Prozessen des SINIX-Systems und den Terminal-Prozessen unter `layers` (siehe `layers`(1)) erfolgt über Multiplex-Kanäle, die von den entsprechenden Betriebssystemen verwaltet werden und ein Protokoll benutzen, das in `xtproto`(5) angegeben ist.

Der Inhalt der Pakete, die Daten zwischen einem SINIX-System-Prozeß und einem Fenster (layer) übertragen, ist asymmetrisch. Die vom SINIX-System an einen bestimmten Terminal-Prozeß gesendeten Daten sind nicht differenziert, und die Interpretation des Inhalts der Pakete ist dem Terminal-Prozeß überlassen.

Steuerdaten für Terminal-Prozesse werden über Kanal 0 gesendet. Prozeß 0 im grafikfähigen Terminal führt die vorgegebenen Funktionen im Auftrag des Prozesses aus, der an den zugewiesenen Kanal angeschlossen ist. Diese Pakete haben folgende Form:

Kommando, Kanal

mit Ausnahme von `JTIMOM`- und `JAGENT`-Daten, die folgende Form haben

Kommando, Daten...

Die Kommandos sind die niederwertigsten acht Bits der folgenden `ioctl`(2)-Codes:

`JBOOT` Vorbereitungen zum Laden eines neuen Terminal-Programms in die vorgegebenen Fenster treffen

`JTERM` heruntergefahrenes Programm abbrechen und das Standard-Fensterprogramm wiederherstellen

`JTIMO` Zeitschranken-Parameter für das Protokoll einstellen. Die Daten bestehen aus vier Bytes in zwei Gruppen: aus dem Wert der Zeitschranke für den Empfang in Millisekunden (die niederwertigsten acht Bits, denen die höchstwertigen acht Bits folgen) und dem Wert für die Zeitschranke des Sendens (in demselben Format).

`JZOMBOOT` wie `JBOOT`, jedoch darf das Programm nicht nach dem Laden ausgeführt werden.

`JAGENT` eine Quellbyte-Zeichenkette an die `agent`-Routine des Terminals senden und auf die Rückgabe einer Antwortbyte-Zeichenkette warten

Die Daten kommen aus einer `bagent`-Struktur (siehe `jagent`(5)), und sie bestehen aus einem Feld mit einer Größe von einem Byte, dem zwei Byte für den `agent`-Kommandocode und Parameter folgen. Ganze Zwei-Byte-Zahlen, die als Teil eines `agent`-Kommandos übertragen werden, werden mit dem höchstwertigen Byte zuerst gesendet. Die Antwort von dem Terminal ist im allgemeinen identisch mit dem Kommandopaket, wobei die bei-

den Kommandobytes durch den Rückgabewert ersetzt werden: 0 für Erfolg, -1 für Mißerfolg. Es ist zu beachten, daß die Routinen in der Bibliothek `libwindows`(3X) alle Parameter in einer `agentrect`-Struktur senden. Die `agent`-Kommandocodes und ihre Parameter sind wie folgt:

`A_NEWLAYER`	mit nachfolgender Zwei-Byte-Kanalnummer und einer Struktur mit dem Fensterrechteck (vier Zwei-Byte-Koordinaten)
`A_CURRENT`	mit nachfolgender Zwei-Byte-Kanalnummer
`A_DELETE`	mit nachfolgender Zwei-Byte-Kanalnummer
`A_TOP`	mit nachfolgender Zwei-Byte-Kanalnummer
`A_BOTTOM`	mit nachfolgender Zwei-Byte-Kanalnummer
`A_MOVE`	mit nachfolgender Zwei-Byte-Kanalnummer und einer gewünschten Zielposition (zwei Zwei-Byte-Koordinaten)
`A_RESHAPE`	mit nachfolgender Zwei-Byte-Kanalnummer und dem neuen Rechteck (vier Zwei-Byte-Koordinaten)
`A_NEW`	mit nachfolgender Zwei-Byte-Kanalnummer und einer Viereckstruktur (vier Zwei-Byte-Koordinaten)
`A_EXIT`	keine Parameter benötigt
`A_ROMVERSION`	keine Parameter benötigt. Das Antwortpaket enthält das Größenbyte, den Zwei-Byte-Rückgabewert, zwei unbenutzte Bytes und den Parameterteil der Zeichenkette der Terminalnummer (z.B. `8;7;3`).

`JXTPROTO` setzt den `xt`-Protokolltyp (siehe `xtproto`(5)). Die Daten bestehen aus einem Byte, das die Minimalgröße für den Datenteil von regulären `xt`-Paketen angibt, die vom Host an das Terminal gesendet werden. Diese Zahl kann bei niedrigeren Baud-Raten oder wenn die Option `-m` beim Aufruf von `layers`(1) angegeben wurde, kleiner als die Zahl sein, die von `A_XTPROTO` zurückgegeben wird. Eine Größe von 1 gibt ein Netzwerk `xt`-Protokoll an.

Die Pakete von dem grafikfähigen Terminal für das SINIX-System haben die Form:

Kommando, Daten ...

Einzelbyte-Kommandos sind wie folgt:

`C_SENDCHAR` das nächste Byte zum SINIX-System-Prozeß senden.

C_NEW eine neue SINIX-System-Prozeßgruppe für dieses Fenster erstellen; die Fenstergröße-Parameter der Fenster berücksichtigen. Die Daten für dieses Kommando weisen das in der jwinsize-Struktur beschriebene Format auf. Die Größe des Fensters wird von zwei ganzen Zwei-Byte-Zahlen vorgegeben, wobei das niederwertige Bit zuerst gesendet wird.

C_UNBLK Blockierung der Übertragung an das Fenster aufheben. Für dieses Kommando gibt es keine Daten.

C_DELETE die zu diesem Fenster gehörende SINIX-System-Prozeßgruppe löschen. Für dieses Kommando gibt es keine Daten.

C_EXIT verlassen; alle mit diesem Terminal im Zusammenhang stehenden SINIX-System-Prozeßgruppen abbrechen und die Sitzung beenden. Für dieses Kommando gibt es keine Daten.

C_DEFUNCT ein Endesignal an die mit diesem Terminal in Zusammenhang stehenden SINIX-System-Prozeßgruppen senden (das Fensterprogramm ist beendet). Für dieses Kommando gibt es keine Daten.

C_SENDNCHARS
Der Rest der Daten besteht aus Zeichen, die an den SINIX-System-Prozeß weiterzureichen sind.

C_RESHAPE Das Fenster wurde verändert. Die Fenstergrößen-Parameter sind für dieses Fenster zu ändern. Die Daten haben die gleiche Form wie für das Kommando C_NEW. An den Prozeß im Fenster wird auch ein SIGWINCH-Signal gesendet, so daß der Prozeß weiß, daß das Fenster umgeformt wurde, und er auf die neuen Fensterparameter zugreifen kann.

C_NOFLOW xt-Netzwerk-Flußkontrolle abschalten (siehe xtproto(5)).

C_YESFLOW xt-Netzwerk-Flußkontrolle einschalten (siehe xtproto(5)).

DATEIEN

/usr/include/windows.h
/usr/include/sys/jioctl.h

SIEHE AUCH

layers(1), libwindows(3X), jagent(5), xtproto(5).
xt(7) in "Leitfaden für Programmierer: STREAMS".

math - mathematische Funktionen und Konstanten

```
#include <math.h>
```

Diese Datei enthält Deklarationen aller Funktionen in der (im Abschnitt 3M beschriebenen) mathematischen Bibliothek sowie verschiedener Funktionen in der C-Bibliothek (Abschnitt 3C), die Gleitkommawerte zurückgeben.

Sie definiert den Aufbau und die Konstanten, die von den Fehlerbehandlungen `matherr`(3M) verwendet werden, einschließlich folgender Konstante, die als Fehlerrückgabewert eingesetzt wird:

`HUGE`	Höchstwert einer Gleitkommazahl einfacher Genauigkeit

Die nachstehenden mathematischen Konstanten sind zur einfacheren Anwendung durch den Benutzer wie folgt definiert:

`M_E`	Basis des natürlichen Logarithmus (e)
`M_LOG2E`	Logarithmus zur Basis 2 von e
`M_LOG10E`	Logarithmus zur Basis 10 von e
`M_LN2`	natürlicher Logarithmus von 2
`M_LN10`	natürlicher Logarithmus von 10
`M_PI`	π, Verhältnis des Umfangs eines Kreises zu seinem Durchmesser
`M_PI_2`	$\pi/2$
`M_PI_4`	$\pi/4$
`M_1_PI`	$1/\pi$
`M_2_PI`	$2/\pi$
`M_2_SQRTPI`	2/Wurzel aus π
`M_SQRT2`	positive Quadratwurzel von 2
`M_SQRT1_2`	positive Quadratwurzel von 1/2

Die folgenden mathematischen Konstanten sind auch in dieser Include-Datei definiert:

`MAXFLOAT`	Maximalwert einer Gleitkommazahl einfacher Genauigkeit
`HUGE_VAL`	positives Unendlich

Zur Definition von verschiedenen maschinenabhängigen Konstanten siehe `values`(5).

SIEHE AUCH

`intro`(3), `matherr`(3M), `values`(5).

nl_types - Datentypen für Sprachen

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

Diese Include-Datei enthält die folgenden Definitionen:

`nl_catd` wird von den Meldungskatalogfunktionen `catopen`, `catgets` und `catclose` verwendet, um einen Katalog zu identifizieren.

`nl_item` wird von `nl_langinfo` zur Identifikation der Sprachumgebung verwendet. Die Werte für Objekte vom Typ `nl_item` werden in `langinfo.h` definiert.

`NL_SETD` wird von `gencat` verwendet, wenn keine `$set`-Direktive in der Meldungsdatei enthalten ist. Diese Konstante kann bei aufeinanderfolgenden Aufrufen von `catgets` als Wert für den Satzauswahlparameter verwendet werden.

`NL_MGSMAX` maximale Anzahl von Meldungen pro Satz

`NL_SETMAX` maximale Anzahl von Sätzen pro Katalog

`NL_TEXTMAX` maximale Länge einer Meldung

`DEF_NLSPATH`
voreingestellter Suchpfad für die Kataloge

SIEHE AUCH

`catgets`(3C), `catopen`(3C), `nl_langinfo`(3C), `langinfo`(5).
`gencat`(1M) in "Referenzhandbuch für Systemverwalter".

prof - Profilerstellung innerhalb einer Funktion

```
#define MARK
#include <prof.h>
void MARK (name);
```

MARK führt eine Markierung *name* ein, die genau wie ein Einsprung einer Funktion behandelt wird. Mit der Ausführung der Markierung wird ein Wert zu dem Zähler für diese Markierung hinzugefügt, und die aufgewendete Zeit des Befehlszählers wird der unmittelbar vorhergehenden Markierung oder der Funktion (wenn innerhalb der aktiven Funktion keine vorhergehende Markierung vorhanden ist) zugerechnet.

name kann eine beliebige Kombination aus Buchstaben, Zahlen oder Unterstrichzeichen sein. Jeder *name* in einem Einzelübersetzungslauf muß eindeutig sein, darf jedoch derselbe sein wie ein beliebiges normales Programmsymbol.

Die Markierungen sind nur dann wirksam, wenn das Symbol MARK vor dem Einfügen der Include-Datei `prof.h` definiert wird, entweder durch eine Übersetzeranweisung, wie in der Übersicht, oder von einem Kommandozeile-Argument, d.h.:

```
cc -p -DMARK foo.c
```

Wenn MARK nicht definiert ist, können die Anweisungen MARK(*name*) in den Quelldateien bleiben, und sie werden dann ignoriert. `prof -g` muß verwendet werden, um Informationen über die Kennungen zu erhalten.

BEISPIEL

In diesem Beispiel können Markierungen zur Bestimmung der Zeit verwendet werden, die in jeder Schleife verbraucht wird. Wenn dieses Beispiel nicht mit definiertem MARK in der Kommandozeile übersetzt wird, werden die Markierungen ignoriert.

```
#include <prof.h>
foo( )
{
  int i, j;
  . . .
  MARK(loop1);
  for (i = 0; i < 2000; i++) {
  . . .
  }
  MARK(loop2);
  for (j = 0; j < 2000; j++) {
  . . .
  }
}
```

SIEHE AUCH

`prof`(1), `profil`(2), `monitor`(3C).

regexp - Übersetzung/Mustervergleich von regulären Ausdrücken

```
#define INIT declarations
#define GETC(void) getc code
#define PEEKC(void) peekc code
#define UNGETC(c) ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(char *instring, char *expbuf, char *endbuf, int eof);

int step(char *string, char *expbuf);

int advance(char *string, char *expbuf);

extern char *loc1, *loc2, *locs;
```

Diese Funktionen sind allgemeine Funktionen zur Behandlung von regulären Ausdrükken in Programmen, die Mustervergleiche von regulären Ausdrücken durchführen. Diese Funktionen werden in der Include-Datei `regexp.h` definiert.

Die Funktionen `step` und `advance` führen den Mustervergleich mit einer Zeichenkette und einem übersetzten regulären Ausdruck als Eingabe durch.

Die Funktion `compile` nimmt als Eingabe einen regulären Ausdruck, wie er unten definiert wird, und erzeugt einen übersetzten Ausdruck, der mit `step` oder `advance` verwendet werden kann.

Ein regulärer Ausdruck spezifiziert eine Menge von Zeichenketten. Wenn ein Element in dieser Menge liegt, wird gesagt, daß es auf den regulären Ausdruck paßt. Einige Zeichen haben eine besondere Bedeutung, wenn sie in einem regulären Ausdruck verwendet werden; andere Zeichen stehen für sich selbst.

Die regulären Ausdrücke, die mit den `regexp`-Funktionen verfügbar sind, werden folgendermaßen erzeugt:

Ausdruck	*Bedeutung*
c	das Zeichen *c*, wobei *c* kein Sonderzeichen ist
c	das Zeichen *c*, wobei *c* irgendein Zeichen ist, außer einer Ziffer im Bereich `1-9`
^	der Anfang der Zeile, auf der der Vergleich durchgeführt wird
$	das Ende der Zeile, auf der der Vergleich durchgeführt wird
.	irgendein Zeichen in der Eingabe

[*s*]	irgendein Zeichen in der Menge *s*, wobei *s* eine Folge und/oder ein Bereich von Zeichen ist, z.B. [*c*-*c*]
[^*s*]	irgendein Zeichen, das nicht in der Menge *s* liegt, wobei *s* wie oben definiert ist
*r**	null oder mehrere aufeinanderfolgende Vorkommen des regulären Ausdrucks *r*. Die längste, am weitesten links liegende, passende Zeichenkette wird verwendet.
rx	das Auftreten des regulären Ausdrucks *r*, gefolgt vom Auftreten des regulären Ausdrucks *x* (Konkatenation)
r\{*m*,*n*\}	irgendeine Anzahl zwischen *m* und *n* aufeinanderfolgenden Vorkommen des regulären Ausdrucks *r*. Der reguläre Ausdruck *r*\{*m*\} paßt bei genau *m* Vorkommen; *r*\{*m*,\} paßt bei mindestens *m* Vorkommen.
\(*r*\)	der reguläre Ausdruck *r*. Wenn *n* (wobei *n* eine Zahl größer null ist) in einem konstruierten regulären Ausdruck vorkommt, steht es für den regulären Ausdruck *x*, wobei *x* der *n*-te reguläre Ausdruck eingeschlossen in \(und \) ist, der vorher in dem konstruiert regulären Ausdruck vorkam. \(*r*\)*x*\(*y*\)*z*\2 ist z.B. die Konkatenation der regulären Ausdrücke *rxyzy*.

Folgende Zeichen haben eine besondere Bedeutung, wenn sie nicht innerhalb von eckigen Klammern auftreten oder ihnen ein \ vorangeht: ., *, [, \. Andere Sonderzeichen, wie z.B. $ haben in noch weiter eingeschränkten Umgebungen eine besondere Bedeutung.

Das Zeichen ^ am Anfang eines Ausdrucks ermöglicht einen passenden Vergleich nur unmittelbar nach einem Neue-Zeile-Zeichen, und das Zeichen $ am Ende eines Ausdrucks verlangt ein abschließendes Neue-Zeile-Zeichen.

Zwei Zeichen haben nur dann eine besondere Bedeutung, wenn sie innerhalb von eckigen Klammern verwendet werden. Das Zeichen - gibt einen Bereich an, [*c*-*c*], außer wenn es direkt nach einer öffnenden oder direkt vor einer schließenden Klammer auftritt, [-*c*] oder [*c*-]. In dem Fall hat es keine besondere Bedeutung. Bei der Verwendung innerhalb von eckigen Klammern hat das Zeichen ^ die Bedeutung *Komplement von*, wenn es unmittelbar auf die öffnende eckige Klammer folgt (Beispiel: [^c]); sonst steht es zwischen eckigen Klammern (Beispiel: [^]) für das normale Zeichen ^.

Die besondere Bedeutung des Operators \ kann nur durch das Voranstellen eines weiteren \, d.h \\, ausgeschaltet werden.

In Programmen müssen die folgenden fünf Makros vor der `#include <regexp.h>`-Anweisung deklariert sein. Diese Makros werden von der Routine `compile` verwendet. Die Makros `GETC`, `PEEKC` und `UNGETC` arbeiten auf dem regulären Ausdruck, der als Eingabe an `compile` übergeben wurde.

GETC Dieses Makro gibt den Wert des nächsten Zeichens (Bytes) im Muster des regulären Ausdrucks an. Aufeinanderfolgende Aufrufe von GETC sollten aufeinanderfolgende Zeichen des regulären Ausdrucks liefern.

PEEKC Dieses Makro gibt das nächste Zeichen (Byte) im regulären Ausdruck an. Unmittelbar aufeinanderfolgende Aufrufe von PEEKC sollten dasselbe Zeichen zurückliefern, das auch dasselbe Zeichen sein sollte, das von GETC zurückgegeben wird.

UNGETC(*c*) Dieses Makro bewirkt, daß das Argument c beim nächsten Aufruf von GETC oder PEEKC zurückgegeben wird. Es wird nie mehr als ein Zeichen zum Zurückschieben benötigt, und dieses Zeichen ist mit Sicherheit das letzte von GETC gelesene Zeichen. Der Rückgabewert von UNGETC(c) wird immer ignoriert.

RETURN(*ptr*)
Dieses Makro wird bei normalem Aussteigen aus der compile-Routine verwendet. Der Wert des Arguments *ptr* ist ein Zeiger auf das Zeichen nach dem letzten Zeichen des übersetzten regulären Ausdrucks. Das ist für solche Programme sinnvoll, die Speicherzuweisungen verwalten.

ERROR(*val*) Dieses Makro ist das nicht normale Zurückkehren der compile-Routine. Das Argument *val* ist eine Fehlernummer. Dieser Aufruf sollte nie zurückkehren.

Die Syntax der compile-Routine ist folgende:

compile(*instring, expbuf, endbuf, eof*)

instring wird nie explizit von der compile-Routine verwendet, ist aber für solche Programme sinnvoll, die verschiedene Zeiger auf Eingabezeichen übergeben. Manchmal wird er in der INIT-Deklaration (siehe unten) verwendet. Programme, die Funktionen zur Zeicheneingabe aufrufen, oder die Zeichen in einem externen Feld stehen haben, können für diesen Parameter (char *)0 einsetzen.

expbuf ist ein Zeiger auf Zeichen. Er zeigt auf den Ort, an dem der übersetzte reguläre Ausdruck untergebracht wird.

endbuf ist um eins höher als die höchste Adresse, an der der übersetzte reguläre Ausdruck untergebracht werden kann. Wenn der übersetzte Ausdruck nicht in (endbuf-expbuf) Bytes paßt, wird ERROR(50) aufgerufen.

Der Parameter *eof* ist das Zeichen, das das Ende des regulären Ausdrucks kennzeichnet. Dieses Zeichen ist normalerweise /.

Jedes Programm, das die Include-Datei `regexp.h` einschließt, muß eine `#define`-Anweisung für `INIT` haben. `INIT` wird für abhängige Deklarationen und Initialisierungen verwendet. Meistens wird es benutzt, um eine Registervariable so zu setzen, daß sie auf den Anfang eines regulären Ausdrucks zeigt, so daß diese Registervariable in den Deklarationen von `GETC`, `PEEKC` und `UNGETC` verwendet werden kann. Sonst kann es benutzt werden, um externe Variablen, die von `GETC`, `PEEKC` und `UNGETC` verwendet werden können, zu deklarieren (siehe Abschnitt `BEISPIEL` unten).

Der erste Parameter für die Funktionen `step` und `advance` ist ein Zeiger auf eine Zeichenkette, die auf Übereinstimmung überprüft werden soll. Die Zeichenkette sollte mit Null abgeschlossen sein. Der zweite Parameter, *expbuf*, ist der übersetzte reguläre Ausdruck, der durch einen Aufruf der Funktion `compile` erhalten wurde.

Die Funktion `step` gibt einen Wert ungleich null zurück, wenn eine Teilfolge von *string* auf den regulären Ausdruck in *expbuf* paßt. Sonst gibt sie Null zurück. Im ersten Fall werden zwei externe Zeichenketten als Nebeneffekt des Aufrufs von `step` gesetzt. Die Variable `loc1` zeigt auf das erste Zeichen, das auf den regulären Ausdruck paßt; die Variable `loc2` zeigt auf das Zeichen nach dem letzten Zeichen, das auf den regulären Ausdruck paßt. Wenn also die gesamte Eingabezeichenkette auf den regulären Ausdruck paßt, zeigt `loc1` auf das erste Zeichen und `loc2` auf das letzte Zeichen der Zeichenkette.

Die Funktion `advance` gibt einen Wert ungleich Null zurück, wenn die ursprüngliche Teilfolge von *string* auf den regulären Ausdruck in *expbuf* paßt. Wenn eine Übereinstimmung auftritt, wird als Nebeneffekt `loc2` (ein externer Zeiger auf Zeichen) gesetzt. Die Variable `loc2` zeigt auf das nächste Zeichen in *string* hinter dem letzten passenden Zeichen.

Wenn `advance` auf eine *- oder `\{ \}`-Folge in einem regulären Ausdruck trifft, wird es seinen Zeiger auf die zu überprüfende Zeichenkette so weit wie möglich vorschieben und sich selbst rekursiv aufrufen, um den Rest der Zeichenkette mit dem Rest des regulären Ausdrucks zu vergleichen. Solange keine Übereinstimmung da ist, versucht `advance` zurückzugehen bis es eine Übereinstimmung findet oder der Punkt erreicht wird, an dem der ursprüngliche Ausdruck * oder `\{ \}` stand. Es ist manchmal sinnvoll, das Zurückgehen abzubrechen bevor der Startpunkt innerhalb der Zeichenkette erreicht ist. Wenn der externe Zeiger auf Zeichen `locs` während des Zurückgehens gleich diesem Punkt ist, verläßt `advance` die Schleife und gibt den Wert Null zurück.

Die externen Variablen `circf`, `sed` und `nbra` sind reserviert.

ERGEBNIS

Die Funktion `compile` verwendet bei Erfolg das Makro `RETURN` und bei Mißerfolg das Makro `ERROR` (siehe oben). Die Funktionen `step` und `advance` geben einen Wert ungleich null zurück, wenn sie eine Übereinstimmung gefunden haben. Sonst geben sie Null zurück. Fehler sind:

11	Endpunkt des Bereichs zu groß
16	ungültige Zahl
25	\ *digit* außerhalb des Bereich
36	ungültiger oder fehlender Begrenzer
41	keine Suchfolge im Speicher
42	`\( \)` Ungleichgewicht
43	zuviele `\(`
44	mehr als 2 Zahlen in `\{ \}`
45	Nach \ wird } erwartet
46	In `\{ \}` ist die erste Zahl größer als die zweite
49	`[ ]` nicht ausgeglichen
50	regulärer Ausdruck zu umfangreich

BEISPIEL

Makros und Aufrufe für reguläre Ausdrücke können in einem Anwendungsprogramm folgendermaßen definiert werden:

```
#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>

  . . .
  (void) compile(*argv, expbuf, &expbuf[ESIZE],'\0');
  . . .
  if (step(linebuf, expbuf))
                      succeed;
```

siginfo - Informationen über Signalerzeugung

```
#include <siginfo.h>
```

Wenn ein Prozeß ein Signal empfängt, können Informationen abgefragt werden, die Auskunft darüber geben, warum das System ein Signal generiert hat (siehe sigaction(2)). Wenn ein Prozeß seine Sohnprozesse überwacht, können Informationen darüber eingeholt werden, warum ein Sohnprozeß seinen Zustand geändert hat (siehe waitid(2)). In jedem Fall liefert das System die Informationen in einer Struktur vom Typ siginfo_t zurück, welche die folgenden Informationen enthält:

```
int si_signo  /* Signalnummer */
int si_errno  /* Fehlernummer */
int si_code   /* Signalcode */
```

si_signo enthält die vom System erzeugte Signalnummer. (Bei der Funktion waitid(2) ist si_signo immer SIGCHLD.)

Wenn si_errno ungleich Null ist, stellt der Wert eine Fehlernummer dar, welche mit dem Signal verknüpft ist; diese Fehlernummer wird in errno.h definiert.

si_code enthält einen Code, der die Ursache des Signals beschreibt. Wenn der Wert von si_code kleiner oder gleich 0 ist, wurde das Signal von einem Benutzerprozeß erzeugt (siehe kill(2) und sigsend(2)), und die Struktur siginfo enthält die folgenden weiteren Informationen:

```
pid_t si_pid  /* Prozeßnummer des Senders */
uid_t si_uid  /* Benutzernummer des Senders */
```

Ansonsten enthält si_code einen signalspezifischen Grund für die Signalerzeugung:

Signal	Code	Grund
SIGILL	ILL_ILLOPC	ungültige Prozessor-Anweisung
	ILL_ILLOPN	ungültiger Operand
	ILL_ILLADR	ungültiger Adressierungsmodus
	ILL_ILLTRP	ungültige Unterbrechung
	ILL_PRVOPC	privilegierte Prozessor-Anweisung
	ILL_PRVREG	privilegiertes Register
	ILL_COPROC	Fehler des Coprozessors
	ILL_BADSTK	interner Stapelfehler
SIGFPE	FPE_INTDIV	ganzzahlige Division durch Null
	FPE_INTOVF	ganzzahliger Überlauf
	FPE_FLTDIV	Gleitkommadivision durch Null
	FPE_FLTOVF	Gleitkommaüberlauf
	FPE_FLTUND	Gleitkommaunterlauf
	FPE_FLTRES	ungenaues Gleitkommaergebnis
	FPE_FLTINV	ungültige Gleitkommaoperation
	FPE_FLTSUB	Index außerhalb des gültigen Bereichs
SIGSEGV	SEGV_MAPERR	Adresse nicht auf Objekt abgebildet
	SEGV_ACCERR	ungültige Zugriffsrechte für abgebildetes Objekt

SIGBUS	BUS_ADRALN	ungültige Adreßzuweisung
	BUS_ADRERR	nichtexistierende physikalische Adresse
	BUS_OBJERR	objektspezifischer Hardwarefehler
SIGTRAP	TRAP_BRKPT	Prozeß-Haltepunkt
	TRAP_TRACE	Prozeß-Trace-Unterbrechung
SIGCHLD	CLD_EXITED	Sohnprozeß über exit beendet
	CLD_KILLED	Sohnprozeß über kill beendet
	CLD_DUMPED	Sohnprozeß nicht normal beendet
	CLD_TRAPPED	überwachter Sohnprozeß empfing Signal
	CLD_STOPPED	Sohnprozeß angehalten
	CLD_CONTINUED	angehaltener Sohnprozeß läuft weiter
SIGPOLL	POLL_IN	Dateneingabe verfügbar
	POLL_OUT	Ausgabepuffer verfügbar
	POLL_MSG	Eingabemeldungen verfügbar
	POLL_ERR	E/A-Fehler
	POLL_PRI	Eingabe mit hoher Priorität verfügbar
	POLL_HUP	Verbindung mit Gerät abgebrochen

Für Signale, die vom Kernel generiert werden, sind folgende signalabhängige Informationen verfügbar:

Signal	Feld	Wert
SIGILL SIGFPE	caddr_t si_addr	Adresse der fehlerhaften Anweisung
SIGSEGV SIGBUS	caddr_t si_addr	Adresse des fehlerhaften Speicherzugriffs
SIGCHLD	pid_t si_pid int si_status	Prozeßnummer des Sohnprozesses Exit-Wert oder Signal
SIGPOLL	long si_band	Band-Ereignis für POLL_IN, POLL_OUT, oder LL_MSG

SIEHE AUCH

sigaction(2), waitid(2), signal(5).

HINWEIS

Für SIGCHLD-Signale ist si_status gleich dem Exit-Wert von Prozessen, sofern si_code gleich CLD_EXITED ist; ansonsten wird der Wert des Signals angenommen, welches den Prozeß dazu veranlaßt hat, den Zustand zu ändern. Bei einigen Implementierungen ist der exakte Wert von si_addr nicht verfügbar; in diesem Fall ist si_addr garantiert auf derselben Seite wie die fehlerhafte Anweisung oder der fehlerhafte Speicherzugriff.

signal - Signale

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

Ein Signal ist die asynchrone Benachrichtigung über ein Ereignis. Ein Signal wird für einen Prozeß erzeugt (oder einem Prozeß geschickt), wenn das mit dem Signal verbundene Ereignis zum ersten Mal auftritt. Beispiele für solche Ereignisse sind Gerätefehler, abgelaufene Uhren oder eine Aktivität am Terminal. Weiterhin können Signale das Ergebnis eines `kill`- oder `sigsend`-Systemaufrufs sein. Unter bestimmten Umständen erzeugt dasselbe Ereignis Signale für mehrere Prozesse. Ein Prozeß kann die Quelle und den genauen Grund für die Erzeugung eines Signals feststellen (siehe `siginfo`(5)).

Jeder Prozeß kann für jedes Signal ein bestimmtes Verhalten des Systems, die sogenannte Voreinstellung des Signals, als Reaktion vorsehen. Die Zuordnung von Aktionen zu Signalen erbt ein Prozeß von seinem Vaterprozeß. Eine einmal als Reaktion für ein bestimmtes Signal eingestellte Funktion bleibt üblicherweise solange eingestellt, bis durch eine der Funktionen `sigaction`, `signal` oder `sigset` eine andere Einstellung explizit vorgenommen wird, oder bis der Prozeß ein `exec` ausführt (siehe `sigaction`(2) und `signal`(2)). Wenn ein Prozeß die Anweisung `exec` ausführt, werden alle Signale, deren Voreinstellung auf Bearbeitung des Signals steht, auf `SIG_DFL` umgestellt. Alternativ dazu kann ein Prozeß vorsehen, daß das System die Voreinstellung eines Signals nach dem Empfang eines Signals auf `SIG_DFL` umstellt (siehe `sigaction`(2) und `signal`(2)).

Ein Signal wird als an einen Prozeß ausgeliefert bezeichnet, sobald die für diesen Prozeß und dieses Signal vorgesehene Aktion ausgeführt wurde. Von dem Zeitpunkt seiner Erzeugung bis zu seiner Auslieferung wird ein Signal als unbearbeitet bezeichnet (siehe `sigpending`(2)). Auf normale Art und Weise kann dieser Zustand von einer Anwendung nicht erkannt werden. Ein Signal kann jedoch von seiner Auslieferung an einen Prozeß blockiert sein (siehe `signal`(2) und `sigprocmask`(2)). Wenn für ein blockiertes Signal eine andere Aktion als das Ignorieren des Signals vorgesehen ist, und wenn dieses Signal für den Prozeß erzeugt wird, so bleibt das Signal unbearbeitet, bis es entweder freigegeben wird, oder bis sich die Voreinstellung für das Signal dahingehend ändert, daß es in Zukunft ignoriert werden soll. Wenn die Voreinstellung für ein blockiertes Signal vorsieht, daß das Signal ignoriert wird, und wenn das Signal für den Prozeß erzeugt wird, so wird das Signal sofort bei seiner Erzeugung vernichtet.

Jeder Prozeß verfügt über eine Signalmaske. Sie gibt an, welche Signale gegenwärtig von einer Auslieferung an den Prozeß blockiert sind (siehe `sigprocmask`(2)). Die Signalmaske eines Prozesses wird durch diejenige des Vaterprozesses initialisiert.

Die Entscheidung, welche Aktion in Reaktion auf ein Signal auszuführen ist, wird zum Zeitpunkt der Auslieferung vorgenommen. Alle Änderungen, die seit der Erzeugung des Signals vorgenommen wurden, sind voll wirksam. Die Bestimmung der Aktion ist dabei unabhängig von dem Anlaß für die Erzeugung des Signals.

Folgende Signale werden derzeit in `signal.h` definiert:

Name	Wert	Standard	Ereignis
SIGHUP	1	Exit	Hangup (siehe termio(7))
SIGINT	2	Exit	Unterbrechung (siehe termio(7))
SIGQUIT	3	Core	Ende (siehe termio(7))
SIGILL	4	Core	Unzulässiger Befehl
SIGTRAP	5	Core	Unterbrechung zur Laufzeitverfolgung
SIGIOT	6	Core	Abbruch
SIGEMT	7	Core	Unterbrechung für Emulator
SIGFPE	8	Core	Arithmetische Ausnahmesituation
SIGKILL	9	Exit	Bedingungsloser Abbruch
SIGBUS	10	Core	Adreßfehler
SIGSEGV	11	Core	Verletzung der Speichersegmentierung
SIGSYS	12	Core	Fehlerhafter Systemaufruf
SIGPIPE	13	Exit	Unterbrochene Pipe
SIGALRM	14	Exit	Alarmuhr
SIGTERM	15	Exit	Beendet
SIGUSR1	16	Exit	Benutzersignal 1
SIGUSR2	17	Exit	Benutzersignal 2
SIGCHLD	18	Ignore	Status eines Sohnprozesses (Änderung gemäß POSIX)
SIGPWR	19	Ignore	Spannungsunterbrechung/Neustart
SIGWINCH	20	Ignore	Änderung der Fenstergröße
SIGURG	21	Ignore	Dringender Vorfall an Socket
SIGIO	22	Ignore	E/A auf Socket möglich (entspricht SIGPOLL)
SIGSTOP	23	Stop	Angehalten (Signal)
SIGTSTP	24	Stop	Angehalten (Benutzer) (siehe termio(7))
SIGCONT	25	Ignore	Fortgesetzt
SIGTTIN	26	Stop	Angehalten (Eingabe auf TTY) (siehe termio(7))
SIGTTOU	27	Stop	Angehalten (Ausgabe auf TTY) (siehe termio(7))
SIGVTALRM	28	Exit	Ablauf eines virtuellen Zeitnehmers
SIGPROF	29	Exit	Ablauf eines Timers zur Laufzeitanalyse
SIGXCPU	30	Core	CPU-Zeitbegrenzung überschritten (siehe getrlimit(2))
SIGXFSZ	31	Core	Datei-Speicherbegrenzung überschritten (siehe getrlimit(2))

Durch die Verwendung der Systemaufrufe `signal`, `sigset` und `sigaction` kann ein Prozeß für ein Signal eine von drei möglichen Voreinstellungen vorsehen: das für dieses Signal vorgesehene Standardverhalten, das Ignorieren des Signals oder die Bearbeitung des Signals.

Standardverhalten: `SIG_DFL`

Eine Voreinstellung von `SIG_DFL` wählt das Standardverhalten. Das Standardverhalten eines jeden Signals ist in der obigen Liste festgehalten. Möglichkeiten sind:

Exit — Bei Erhalt des Signals wird der empfangende Prozeß abgebrochen, mit allen Konsequenzen, wie sie in `exit`(2) beschrieben sind.

Core — Bei Erhalt des Signals wird der empfangende Prozeß abgebrochen, mit allen Konsequenzen, wie sie in `exit`(2) beschrieben sind. Zusätzlich wird ein Speicherabzug (core) des Prozesses im aktuellen Dateiverzeichnis angelegt.

Stop — Bei Erhalt des Signals wird der empfangende Prozeß angehalten.

Ignore — Bei Erhalt des Signals wird dieses durch den empfangenden Prozeß ignoriert. Dies entspricht exakt der Voreinstellung von `SIG_IGN`.

Ignorieren des Signals: SIG_IGN

Eine Voreinstellung von SIG_IGN gibt an, daß das Signal ignoriert werden soll.

Signal bearbeiten: *Adresse einer Funktion*

Wird als Voreinstellung die Adresse einer Funktion verwendet, so bedeutet dies, daß der Prozeß bei Empfang dieses Signals das Signal-Steuerprogramm an der angegebenen Adresse ausführen soll. Normalerweise erhält das Signal-Steuerprogramm als einziges Argument die Nummer des empfangenen Signals; durch die Verwendung der Funktion sigaction zur Voreinstellung können jedoch zusätzliche Parameter vorgesehen werden (siehe sigaction(2)). Solange das Signal-Steuerprogramm keine anderen Vorkehrungen trifft, wird der Prozeß nach Beendigung des Signal-Steuerprogramms an der Stelle fortgesetzt, an der es unterbrochen wurde. Bei der Verwendung von ungültigen Funktionsadressen ist das Verhalten nicht definiert.

Wurde die Voreinstellung durch die Funktionen sigset oder sigaction vorgenommen, so wird das Signal automatisch durch das System blockiert, während der Signalempfänger aktiv ist. Wenn longjmp (siehe setjmp(3C)) verwendet werden soll, um den Signalempfänger zu verlassen, dann muß das Signal explizit durch den Benutzer freigegeben werden (siehe signal(2) und sigprocmask(2)).

Unterbricht die Ausführung eines Signal-Steuerprogramms einen nicht unterbrechbaren Systemaufruf, so wird das Signal-Steuerprogramm ausgeführt. Der unterbrochene Systemaufruf liefert danach den Wert -1 an den aufrufenden Prozeß zurück, errno wird auf den Wert EINTR gesetzt. Ist hingegen der Schalter SA_RESTART gesetzt, so wird der Systemaufruf ohne Fehlermeldung neu gestartet.

HINWEIS

Die Voreinstellungen der Signale SIGKILL und SIGSTOP können nicht geändert werden. Das System erzeugt einen Fehler, wenn dies versucht wird.

Die Signale SIGKILL und SIGSTOP können nicht blockiert werden. Die Einhaltung dieser Einschränkung wird ohne eine Meldung erzwungen.

Empfängt ein Prozeß das Signal SIGCONT, so werden, unabhängig von den momentanen Voreinstellungen, alle unbearbeiteten Signale SIGSTOP, SIGTSTP, SIGTTIN und SIGTTOU vernichtet. Zusätzlich wird der Prozeß fortgesetzt, falls er unterbrochen sein sollte.

SIGPOLL wird verwendet, wenn ein zu einer STREAMS-Datei (siehe intro(2)) gehörender Dateideskriptor ein unbearbeitetes 'auswählbares' Ereignis aufweist. Ein Prozeß muß die Zustellung dieses Signals durch einen Aufruf des I_SETSIG-Ioctl explizit anfordern. Ansonsten wird der Prozeß niemals ein SIGPOLL erhalten.

Wenn die Voreinstellung für das Signal SIGCHLD mit signal oder sigset vorgenommen wurde, oder mit sigaction und mit gesetztem Schalter SA_NOCLDSTOP, so wird dieses

Signal nur dann an den aufrufenden Prozeß gesendet, wenn seine Sohnprozesse terminieren. Ansonsten würde dieses Signal auch gesendet, wenn die Sohnprozesse aufgrund der Ablaufsteuerung angehalten wurden, oder wenn sie ihre Arbeit wieder aufnehmen.

Der Bezeichner SIGCLD wird ebenfalls in dieser Include-Datei definiert und bezeichnet dasselbe Signal wie SIGCHLD. SIGCLD wird aus Kompatibilitätsgründen weiter unterstützt, neue Anwendungen sollten jedoch SIGCHLD verwenden.

Die Voreinstellung von Signalen, die als SIG_IGN veerbt werden, sollte nicht verändert werden.

SIEHE AUCH

exit(2), getrlimit(2), intro(2), kill(2), pause(2), sigaction(2), sigaltstack(2), signal(2), sigprocmask(2), sigsend(2), sigsuspend(2), wait(2), sigsetops(3C), siginfo(5), ucontext(5).

stat - vom Systemaufruf zurückgegebene Daten

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

Die Systemaufrufe `stat`, `lstat` und `fstat` geben Daten in eine `stat`-Struktur zurück, welche in `stat.h` definiert ist.

Die Konstanten, welche in dem `st_mode`-Feld benutzt werden, sind auch in dieser Datei definiert:

```
#define S_IFMT     /* Dateityp */
#define S_IAMB     /* Zugriffsmodus-Bits */
#define S_IFIFO    /* FIFO-Datei */
#define S_IFCHR    /* zeichenorientierte Gerätedatei */
#define S_IFDIR    /* Dateiverzeichnis */
#define S_IFNAM    /* XENIX namenorientierte Datei */
#define S_INSEM    /* XENIX Semaphor-Untertyp von IFNAM */
#define S_INSHD    /* XENIX gemeinsamer Daten-Untertyp von IFNAM */
#define S_IFBLK    /* blockorientierte Gerätedatei */
#define S_IFREG    /* normal */
#define S_IFLNK    /* Symbolischer Verweis */
#define S_ISUID    /* s-Bit für Eigentümer bei Ausführung setzen */
#define S_ISGID    /* s-Bit für Gruppe bei Ausführung setzen */
#define S_ISVTX    /* Textsegment auch nach Verwendung aufbewahren */
#define S_IREAD    /* Lese-Erlaubnis, Eigentümer */
#define S_IWRITE   /* Schreiberlaubnis, Eigentümer */
#define S_IEXEC    /* Ausführungs-/Sucherlaubnis, Eigentümer */
#define S_ENFMT    /* Anzeiger, obligatorisches Datensatzsperren */
#define S_IRWXU    /* Lesen, Schreiben, Ausführen: Eigentümer */
#define S_IRUSR    /* Lese-Erlaubnis: Eigentümer */
#define S_IWUSR    /* Schreiberlaubnis: Eigentümer */
#define S_IXUSR    /* Ausführungserlaubnis: Eigentümer */
#define S_IRWXG    /* Lesen, Schreiben, Ausführen, Gruppe */
#define S_IRGRP    /* Lese-Erlaubnis: Gruppe */
#define S_IWGRP    /* Schreiberlaubnis: Gruppe */
#define S_IXGRP    /* Ausführungserlaubnis: Gruppe */
#define S_IRWXO    /* Lesen, Schreiben, Ausführen: Andere */
#define S_IROTH    /* Lese-Erlaubnis: Andere */
#define S_IWOTH    /* Schreiberlaubnis: Andere */
#define S_IXOTH    /* Ausführungserlaubnis: Andere */
```

Die folgenden Makros gelten für die POSIX-Übereinstimmung:

```
#define   S_ISBLK(mode)  blockorientierte Gerätedatei
#define   S_ISHCR(mode)  zeichenorientierte Gerätedatei
#define   S_ISDIR(mode)  Dateiverzeichnis
#define   S_ISFIFO(mode) Befehlsverknüpfungs- oder FIFO-Datei
#define   S_ISREG(mode)  normale Datei
```

SIEHE AUCH

`stat`(2), `types`(5).

stdarg - variable Argumentenliste bearbeiten

```
#include <stdarg.h>
va_list pvar;
void va_start(va_list pvar, parmN);
type va_arg(va_list pvar, type);
void va_end(va_list pvar);
```

Diese Makros erlauben das Entwickeln von portablen Prozeduren, welche eine variable Anzahl von Argumenten verschiedener Typen akzeptieren. Routinen mit variablen Argumentenlisten (wie `printf`), welche nicht `stdarg` verwenden, sind nicht portabel, da verschiedene Maschinen unterschiedliche Konventionen der Argumentübergabe verwenden.

`va_list` ist ein Typ, der für die Variable definiert wird, um die Liste zu durchlaufen.

Das Makro `va_start` wird vor dem Zugriff auf die namenlosen Argumente ausgeführt und initialisiert *pvar* zum nachfolgenden Bearbeiten durch `va_arg` und `va_end`. Der Parameter *parmN* ist der Bezeichner des letzten Parameters in der variablen Parameterliste der Funktionsdefinition. Wenn dieser Parameter deklariert wird mit der Speicherklasse `register`, mit einem Funktions- oder Feldtyp, oder gar mit einem Typ, der nicht mit dem Typ kompatibel ist, der nach der Anwendung der voreingestellten Argumentbehandlung resultiert, ist das Ergebnis undefiniert.

Der Parameter *parmN* wird bei strenger ANSI-C-Übersetzung erfordert. Unter anderen Übersetzungsmodi braucht *parmN* nicht angegeben zu werden, und der zweite Parameter für das Makro `va_start` kann ausgelassen werden (z.B. `va_start(pvar, );`). Dies erlaubt Routinen, die keine Parameter vor `...` in der variablen Parameterliste akzeptieren.

Das Makro `va_arg` wird als ein Ausdruck bewertet, der den Typ und den Wert des nächsten Arguments des Aufrufs darstellt. Der Parameter *pvar* sollte vorher durch `va_start` initialisert werden. Jeder Aufruf von `va_arg` ändert *pvar* so, daß die Werte von nachfolgenden Argumenten der Reihe nach zurückgeliefert werden. Der Parameter *type* ist der Typname des nächsten zurückgelieferten Arguments. Der Typname muß so angegeben sein, daß der Typ eines Zeigers auf ein Objekt dieses Typs über Anhängen eines * an *type* erzeugt werden kann. Gibt es kein nächstes Argument oder ist *type* nicht kompatibel mit dem Typ des nächsten Arguments (entsprechend der voreingestellten Argumentbehandlung), ist das Verhalten undefiniert.

Das Makro `va_end` wird zum Aufräumen verwendet.

Mehrfache Durchläufe, mit `va_start` beginnend und `va_end` endend, sind möglich.

BEISPIEL

Im folgenden Beispiel werden die Zeiger auf Zeichenketten über eine variable Argumentenliste in ein Feld eingelesen. Dies geschieht über die Funktion `f1`, welche allerdings nicht mehr als `MAXARGS` Argumente akzeptiert. Danach wird das Feld als einzelnes Argument an die Funktion `f2` übergeben. Die Anzahl der Zeiger wird im ersten Argument von `f1` angegeben.

```
#include <stdarg.h>
#define MAXARGS 31

void f1(int n_ptrs, ...)
{
   va_list ap;
   char *array[MAXARGS];
   int ptr_no = 0;

   if (n_ptrs > MAXARGS)
       n_ptrs = MAXARGS;
   va_start(ap, n_ptrs);
   while (ptr_no < n_ptrs)
       array[ptr_no++] = va_arg(ap, char*);
   va_end(ap);
   f2(n_ptrs, array);
}
```

Bei jedem Aufruf von `f1` sollte die Definition der Funktion oder eine Deklaration wie

```
void f1(int, ...)
```

verfügbar sein.

SIEHE AUCH

`vprintf`(3S).

HINWEIS

Es liegt bei der aufrufenden Routine, anzugeben, wieviele Argumente übergeben werden, da die Festellung der Anzahl der Argumente aus dem Stapel nicht immer möglich ist. Beispielsweise wird `execl` ein Nullzeiger übergeben, um das Ende der Liste anzuzeigen. `printf` erkennt anhand des Formats die Anzahl der Argumente. Die Angabe eines zweiten Arguments `char`, `short` oder `float` für `va_arg` ist nicht portabel, da die von der aufgerufenen Funktion sichtbaren Argumente nicht `char`, `short` oder `float` sind. C konvertiert `char`- und `short`-Argumente in `int` und wandelt `float`-Argumente in `double` um, bevor sie an eine Funktion übergeben werden.

types - einfache Systemdatentypen

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

Die in der Include-Datei definierten Datentypen werden vom SINIX-System verwendet; auf einige Daten dieser Typen kann vom Benutzer zugegriffen werden.

```
typedef long                    daddr_t;
typedef char                    caddr_t;
typedef unsigned char           unchar;
typedef unsigned short          ushort;
typedef unsigned int            uint;
typedef unsigned long           ulong;
typedef ushort                  ino_t;
typedef short                   cnt_t;
typedef long                    time_t;
typedef struct { int val[6]; }  label_t;
typedef short                   dev_t;
typedef long                    off_t;
typedef long                    paddr_t;
typedef int                     key_t;
typedef unsigned char           use_t;
typedef short                   sysid_t;
typedef short                   index_t;
typedef short                   lock_t;
typedef unsigned int            size_t;
typedef unsigned short          sel_t;
```

`daddr_t` wird für Plattenadressen benutzt, wobei ein Indexeintrag auf der Platte jedoch eine Ausnahme bildet (siehe `fs`(4)). Die Zeiten werden in Sekunden seit 00:00:00 GMT, 1. Januar 1970
angegeben. Major- und Minor-Teil einer Gerätezahl geben die Gerätenummer eines Geräts an und sind von der Installation abhängig. Offsets werden in Bytes vom Beginn einer Datei an gemessen. Die Variablen `label_t` werden zum Sichern des Prozessorstatus verwendet, während ein anderer Prozeß abläuft.

SIEHE AUCH

`fs`(4).

ucontext - Benutzerkontext

```
#include <ucontext.h>
```

Die Struktur `ucontext` definiert den Kontrollkontext innerhalb eines ablaufenden Prozesses.

Diese Struktur beinhaltet mindestens die folgenden Komponenten:

```
ucontext_t  uc_link
sigset_t    uc_sigmask
stack_t     uc_stack
mcontext_t  uc_mcontext
```

`uc_link` ist ein Zeiger auf den Kontext, der wieder aufgenommen werden soll, wenn der aktuelle Kontext beendet ist. Wenn `uc_link` gleich Null ist, dann ist der aktuelle Kontext der Hauptkontext; der Prozeß wird beendet, wenn dieser Kontext beendet wird.

`uc_sigmask` definiert die Signalmengen, welche blockiert werden, wenn dieser Kontext aktiv ist (siehe `sigprocmask`(2)).

`uc_stack` definiert den Stapel für diesen Kontext (siehe `sigaltstack`(2)).

`uc_mcontext` enthält die gesicherten Maschinenregister und implementierungsspezifische Kontextdaten. Portable Anwendungen sollten `uc_mcontext` nicht modifizieren.

SIEHE AUCH

`getcontext`(2), `sigaction`(2), `sigprocmask`(2), `sigaltstack`(2), `makecontext`(3C).

values - maschinenabhängige Werte

```
#include <values.h>
```

Diese Datei enthält einen Satz für bestimmte Prozessorarchitekturen verschieden definierter Konstanten.

Die für ganze Zahlen angenommene Darstellung ist eine Binärdarstellung (Einer- oder Zweier-Komplement), in der das Vorzeichen durch den Wert des höchstwertigen Bits dargestellt wird.

`BITS`*(type)* Anzahl der Bits in einem angegebenen Typ (z.B.,`int`)

`HIBITS` Wert eines `short integer`, bei dem nur das höchstwertige Bit gesetzt ist

`HIBITL` Wert eines `long integer`, bei dem nur das höchstwertige Bit gesetzt ist

`HIBITI` Wert eines `integer`, bei dem nur das höchstwertige Bit gesetzt ist

`MAXSHORT` Höchstwert eines `short integer` mit Vorzeichen

`MAXLONG` Höchstwert eines `long integer` mit Vorzeichen

`MAXINT` Höchstwert eines normalen `integer` mit Vorzeichen

`MAXFLOAT, LN_MAXFLOAT`
Höchstwert einer Gleitkommazahl einfacher Genauigkeit und ihr natürlicher Logarithmus

`MAXDOUBLE, LN_MAXDOUBL`
Höchstwert einer Gleitkommazahl doppelter Genauigkeit und ihr natürlicher Logarithmus

`MINFLOAT, LN_MINFLOAT`
Kleinster positiver Wert einer Gleitkommazahl einfacher Genauigkeit und ihr natürlicher Logarithmus

`MINDOUBLE, LN_MINDOUBLE`
kleinster positiver Wert einer Gleitkommazahl doppelter Genauigkeit und ihr natürlicher Logarithmus

`FSIGNIF` Zahl der signifikanten Bits in der Mantisse einer Gleitkommazahl einfacher Genauigkeit

`DSIGNIF` Zahl der signifikanten Bits in der Mantisse einer Gleitkommazahl doppelter Genauigkeit

SIEHE AUCH

`intro`(3), `limits`(4), `math`(5).

varargs - variable Argumentenlisten bearbeiten

```
#include <varargs.h>
va_alist
va_dcl
va_list pvar;
void va_start(va_list pvar);
type va_arg(va_list pvar, type);
void va_end(va_list pvar);
```

Dieser Satz von Makros ermöglicht das Schreiben portabler Prozeduren, die variable Argumentenlisten akzeptieren. Funktionen, die variable Argumentenlisten aufweisen (wie z.B. `printf`(3S)), jedoch `varargs` nicht verwenden, sind von Natur aus nicht portabel, da verschiedene Maschinen verschiedene Methoden zum Weitergeben von Argumenten verwenden.

`va_alist` wird als Parameterliste in einem Funktionskopf verwendet.

`va_dcl` ist eine Deklaration für `va_alist`. Auf `va_dcl` darf kein Semikolon folgen.

`va_list` ist ein Typ, der für die Variable definiert ist, die zum Durchlaufen der Liste verwendet wird.

`va_start` wird zum Initialisieren von `pvar` auf den Anfang der Liste aufgerufen.

`va_arg` gibt das nächste Argument in der Liste zurück, auf das `pvar` weist. *type* ist der Typ, den das Argument darstellen soll. Verschiedene Typen können gemischt werden, doch ist es dann der Funktion überlassen zu wissen, welcher Argumenttyp erwartet wird, da dies nicht zur Laufzeit ermittelt werden kann.

`va_end` wird zum abschließenden Aufräumen verwendet.

Mehrfache Durchgänge sind möglich, wobei jeder Durchgang mit `va_start` und `va_end` umschlossen ist.

BEISPIEL

Dieses Beispiel zeigt eine mögliche Realisierung von `execl` (siehe `exec`(2)).

```
#include <unistd.h>
#include <varargs.h>
#define MAXARGS      100
/* execl wird aufgerufen mit
      execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
   va_list ap;
   char *file;
   char *args[MAXARGS];/* angenommen, es ist groß genug*/
   int argno = 0;
   va_start(ap);
   file = va_arg(ap, char *);
   while ((args[argno++] = va_arg(ap, char *)) != 0)
   ;
   va_end(ap);
   return execv(file, args);
}
```

SIEHE AUCH

`exec`(2), `printf`(3S), `vprintf`(3S), `stdard`(5).

HINWEIS

Die Angabe der Anzahl der vorliegenden Argumente ist Aufgabe der aufrufenden Funktion, da die Bestimmung der Anzahl der Argumente nicht immer vom Stapelrahmen des Laufzeitkellers aus möglich ist. So wird beispielsweise ein Nullzeiger zum Signalisieren des Endes der Liste an `execl` weitergegeben. `printf` kann anhand des Formats feststellen, wieviele Argumente vorliegen.

Die Angabe eines zweiten Arguments vom Typ `char`, `short` oder `float` für `va_arg` ist nicht portabel, da die von der aufgerufenen Funktion gesehenen Argumente nicht `char`, `short` oder `float` sind. C wandelt `char`- und `short`-Argumente in `int`- und `float`-Argumente in `double` um, bevor diese an eine Funktion übergeben werden.

`stdarg` ist die bevorzugte Schnittstelle.

wstat - Status beim Warten

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

Wenn ein Prozeß auf den Status seiner Sohnprozesse mittels `wait` oder `waitpid` wartet, kann der zurückgelieferte Status mit den in `sys/wait.h` definierten Makros ausgewertet werden. Diese Makros werten ganzzahlige Ausdrücke aus. Das Argument *stat* stellt dabei den ganzzahligen Wert dar, der von `wait` oder `waitpid` zurückgegeben wird.

`WIFEXITED(`*stat*`)`
: liefert ungleich Null, wenn der Status eines normal beendeten Sohnprozesses zurückgeliefert wurde.

`WEXITSTATUS(`*stat*`)`
: liefert bei einem `WIFEXITED(`*stat*`)`-Wert von ungleich Null den Exit-Code, den der Sohnprozeß an `_exit` oder `exit` übergeben hat, oder den Wert, der von `main` zurückgegeben wurde.

`WIFSIGNALED(`*stat*`)`
: liefert ungleich Null, wenn der Status eines Sohnprozesses zurückgegeben wurde, welcher aufgrund eines Signals beendet wurde.

`WTERMSIG(`*stat*`)`
: gibt bei einem `WIFSIGNALED(`*stat*`)`-Wert von ungleich Null die Nummer des Signals an, welches die Beendigung des Sohnprozesses verursacht hat.

`WIFSTOPPED(`*stat*`)`
: Ein Wert ungleich Null gibt an, daß der Status eines Sohnprozesses zurückgeliefert wurde, der momentan angehalten ist.

`WSTOPSIG(`*stat*`)`
: Wenn der Wert von `WIFSTOPPED(`*stat*`)` ungleich Null ist, liefert dieser Makro die Nummer des Signals zurück, welches den Sohnprozeß zum Anhalten gebracht hat.

`WIFCONTINUED(`*stat*`)`
: liefert einen Wert ungleich Null zurück, wenn der Status eines wieder aufgenommenen Sohnprozesses zurückgeliefert wurde.

`WCOREDUMP(`*stat*`)`
: Ist der Wert von `WIFSIGNALED(`*stat*`)` ungleich Null, liefert das Makro einen Wert ungleich Null, wenn ein Speicherabzug für den beendeten Sohnprozeß erzeugt wurde.

SIEHE AUCH

`exit`(2), `wait`(2), `waitpid`(3C).

xtproto - vom Treiber xt verwendetes Protokoll für Multiplex-Kanäle

Dieses xt-Protokoll wird für die Kommunikation zwischen mehreren SINIX-System-Rechnerprozessen und einem unter dem Kommando layers arbeitenden grafikfähigen Terminal verwendet (siehe xt(7)). Es ist ein Multiplex-Protokoll, das die Datenübertragung zwischen Rechnerprozessen und grafikfähigen Terminals ermöglicht und dabei mehrere virtuelle Terminal-Sitzungen über eine einzige Verbindung erlaubt. Das Protokoll ist durch den xt-Rechnertreiber und entsprechende Firmware in einem Terminal mit Fenster-Darstellung implementiert.

Der xt-Treiber implementiert zwei verschiedene Protokolle auf niedriger Ebene. Welches Protokoll verwendet wird, hängt von dem Medium ab, das für die Kommunikation mit dem Terminal verwendet wird. Das reguläre xt-Protokoll wird verwendet, wenn über ein unzuverlässiges Medium wie RS-232 kommuniziert wird. Das reguläre xt-Protokoll stellt eine Flußkontrolle und Fehlerbehebung zur Verfügung und garantiert dabei eine fehlerfreie Datenübertragung. Das Netzwerk-xt-Protokoll wird verwendet, wenn über ein zuverlässiges Medium, wie z.B. einem lokalen Netzwerk, kommuniziert wird. Um maximalen Durchsatz zu erreichen, benutzt das Netzwerk-xt-Protokoll die Flußkontrolle und die Fehlerkorrektur des darunterliegenden Netzwerks.

Das Kommando layers befragt das grafikfähige Terminal, ob das reguläre oder das Netzwerk-xt-Protokoll beim Systemaufruf A_XTPROTO JAGENT ioctl verwendet werden soll (siehe layers(5)). Das Kommando layers entscheidet dann, abhängig vom Rückgabewert von A_XTPROTO, der Baud-Rate und der Option -m von layers, welches Protokoll verwendet werden soll.

Das reguläre xt-Protokoll verwendet Pakete mit einem 2-Byte-Vorsatz, der eine 3-Bit-Sequenznummer, 3-Bit-Kanalnummer, Steueranzeiger und einem Byte für die Datengröße. Der Datenteil von Paketen, die vom Rechner an das Terminal gesendet werden, darf nicht größer als 252 Bytes sein. Die maximale Größe für den Datenteil kann bei niedrigeren Baud-Raten, oder wenn die Option -m von layers angegeben war, kleiner als 252 sein. Wenn mit einem älteren grafikfähigen Terminal kommuniziert wird, ist die maximale Größe des Datenteils auf 32 Bytes festgelegt. Die maximale Größe des Datenteils von Paketen, die vom Terminal an den Rechner gesendet werden, ist immer auf 32 Bytes festgelegt. Jeder Kanal ist doppelt gepuffert.

Der Reihenfolge nach korrekt empfangene reguläre xt-Pakete werden mit einem Steuerpaket bestätigt, das ein ACK enthält; jedoch erzeugen Pakete außerhalb der Reihenfolge ein Steuerpaket, das ein NAK enthält, das die erneute Übertragung aller nichtquittierten Pakete in korrekter Reihenfolge veranlaßt.

Nichtquittierte reguläre xt-Pakete werden nach einem bestimmten Zeitintervall, das von der Baud-Rate abhängt, noch einmal gesendet. Ein weiterer Zeitparameter gibt die Zeitspanne an, nach der unvollständig erhaltene Pakete gelöscht werden.

Das Netzwerk-xt-Protokoll verwendet einen 3-Byte-Vorsatz, der eine 3-Bit-Kanalnummer,

verschiedene Steueranzeiger und 2 Bytes für die Datengröße enthält. Der Datenteil der Pakete, die vom Rechner zum Terminal gesendet werden, kann beliebig groß sein. Der Datenteil der Pakete, die vom Terminal zum Rechner gesendet werden, darf höchstens 1025 Bytes enthalten.

Da das Netzwerk-xt-Protokoll auf dem zugrundeliegenden Medium aufbaut, um eine fehlerfreie Datenlieferung zu garantieren, werden keine CRC-Codes oder Zeitschranken benötigt.

Das Netzwerk-xt-Protokoll stellt einfache Flußkontroll-Mechanismen zur Verfügung, um die Menge von Daten, die an ein Fenster im Terminal gesendet werden, bevor vom Rechner die Bestätigung NETWORK XT ACK empfangen wird, zu begrenzen. Die Absicht dieser Flußkontrolle ist es, die Menge von Daten zu begrenzen, die an ein Fenster im Terminal gesendet werden, das seine Eingabe gerade nicht liest, weil zum Beispiel der Benutzer gerade die Scroll-Lock-Taste (zum Anhalten der Ausgabe) gedrückt hält. Das ist notwendig, um einen Datenstau zu verhindern und andere Daten, die an andere Fenster gesendet werden, zu blockieren. Um den allgemeinen Durchsatz zu erhöhen, kann die Flußkontrolle vom Netzwerk-xt durch Prozesse im Terminal deaktiviert werden, die ihre Eingabe immer schnell lesen.

DATEIEN

/usr/include/sys/xtproto.h	Protokolldefinitionen für Kanäle in Multiplex-Schaltung

SIEHE AUCH

jagent(5), layers(5).
layers(1) in "SINIX V5.41 Kommandos".
xt(7) in "Leitfaden für Programmierer: STREAMS".

Literatur

Literatur der Siemens Nixdorf Informationssysteme AG

CES (SINIX)
Leitfaden und Werkzeuge für die Programmierung mit C
Benutzerhandbuch

Zielgruppe
C-Programmierer, die unter dem Betriebssystem SINIX arbeiten
Inhalt
Im Handbuch werden das C-Übersetzungssystem (Präprozessor, Binder, Include-Dateien, Bibliotheken) und Dienstprogramme zur Entwicklung, Verwaltung, Wartung und Generierung von C-Programmen beschrieben.

CES (SINIX)
cc-Kommando für RM600
Beschreibung

Zielgruppe
C-Programmierer, die unter SINIX auf einem RM600 arbeiten.
Inhalt
Vollständige Beschreibung des cc-Kommandos mit Optionen für CES unter SINIX auf RM600.

CES (SINIX)
cc-Kommando für MX300 und WX200
Beschreibung

Zielgruppe
C-Programmierer, die unter SINIX auf einem MX300 oder WX200 arbeiten.
Inhalt
Vollständige Beschreibung des cc-Kommandos mit Optionen für CES unter SINIX auf MX300 oder WX200.

SINIX V5.41
Kommandos Band 1, A - K
Beschreibung

Zielgruppe
SINIX-Anwender mit Grundkenntnissen
Inhalt
Nachschlagewerk für die Benutzerkommandos (A - K) des Betriebssystems SINIX V5.41.

SINIX V5.41
Kommandos Band 2, L - Z
Beschreibung

Zielgruppe
SINIX-Anwender mit Grundkenntnissen
Inhalt
Nachschlagewerk für die Benutzerkommandos (L - Z) des Betriebssystems SINIX V5.41.

SINIX V5.41
Kommandos Band 3
Tabellen und Verzeichnisse
Beschreibung

Zielgruppe
SINIX-Anwender mit Grundkenntnissen
Inhalt
Gesamtinhaltsverzeichnis, Tabellensammlung (Reguläre Ausdrücke, Gerätedateien für Datenträger u.a.), Fachwort-, Literatur- und Stichwortverzeichnis.

DBX (SINIX)
Benutzerhandbuch

Zielgruppe
C++-Programmierer, C-Programmierer, FORTRAN77-Programmierer
Inhalt
DBX ist eine interaktive symbolische Testhilfe für C++, C und FORTRAN77-Programme. Das Handbuch gibt eine Einführung in DBX und beschreibt die DBX-Kommandos.

MX500 (SINIX V5.40)
MX300 (SINIX V5.41)
Referenzhandbuch für Systemverwalter

Zielgruppe
Systemverwalter
Inhalt
Beschreibt Kommandos und Anwendungsprogramme zur Systempflege sowie Dateiformate, spezielle Dateien zur Systemverwaltung und gibt Diagnosehinweise.

Leitfaden für Programmierer Netzwerk-Schnittstellen (SINIX V5.40)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Beschreibung der TLI-Schnittstelle und ihrer Anbwendungsmöglichkeiten, sowie deren Entwicklungswerkzeuge für Netze.

Leitfaden für Programmierer STREAMS (SINIX V5.40)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Einführung in die Programmierung und Verwendung von STREAMS.

Leitfaden für die Umstellung auf ANSI C (SINIX V5.40)
Benutzerhandbuch

Zielgruppe
Systemprogrammierer
Inhalt
Möglichkeiten werden dargestellt, bereits existierende C-Programme mit ANSI-C-Spezifikationen auszubauen und neue Programme mit ANSI C zu schreiben.

Kernighan/Ritchie
Programmieren in C
Zweite Ausgabe ANSI-C

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Geschäftsstelle. Dort können Sie auch die Handbücher bestellen.

Sonstige Literatur

Kernighan/Ritchie
The C Programming Language
Bell Laboratories, 1978

Verzeichnis der Kommandos und Funktionen

F

G

M

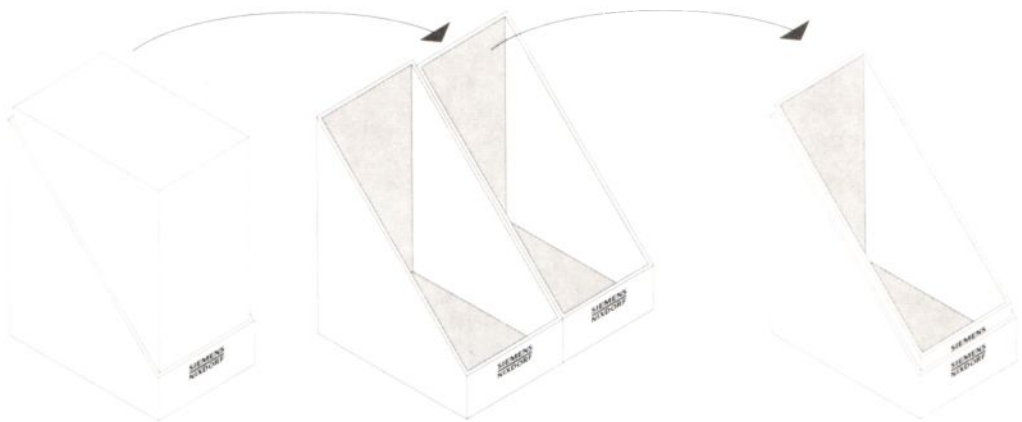
S

T

Sammelboxen

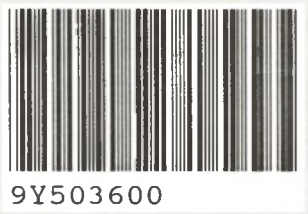
Für Handbücher des vorliegenden Formates bieten wir zweiteilige Sammelboxen in zweierlei Größen an. Der Bestellvorgang entspricht dem für Handbücher.

Breite: ca. 5 cm
Bestellnummer: U3775-J-Z18-1



Breite: ca. 10 cm
Bestellnummer: U3776-J-Z18-1

960297 MC&D



Herausgegeben von/Published by
Siemens Nixdorf Informationssysteme AG
Postfach 2160, W-4790 Paderborn
Postfach 830951, W-8000 München 83

Bestell-Nr./Order No. ***U6401-J-Z145-2***
Printed in the Federal Republic of Germany
13260 AG 5924. (16580)