Betriebssystem SINIX
CES
C-Entwicklungssystem

Beschreibung – Teil 1

Ausgabe Mai 1990 (CES V5.22)

... und Schulung?

Unsere SINIX-Kurse in
München – Berlin – Essen – Frankfurt –
Hannover – Wien – Zürich
helfen Ihnen Betriebssystem, Kommunikation und Software optimal anzuwenden und Software effizient zu entwickeln.

Zentrale Auskunft und Info-Material:
Telefon (089) 9275-3332

Siemens AG
Schule für Daten- und Informationstechnik
DI Schule S3
Postfach 830951, D-8000 München 83

Bestell-Nr. U3899-J-Z95-3
Printed in the Federal Republic of Germany
10880 AG 5902. (13600)

X/Open XPG3-konform
Warenzeichen beantragt

SINIX® ist der Name der Siemens-Version des Softwareproduktes XENIX® .
SINIX enthält Teile, die dem Copyright © von Microsoft (1980-1987) unterliegen; im übrigen unterliegt es dem Copyright © von Siemens (1989). SINIX ist ein eingetragenes Warenzeichen der Siemens AG. XENIX ist ein eingetragenes Warenzeichen der Microsoft Corporation. XENIX ist aus UNIX® -System unter Lizenz von AT&T entstanden. UNIX ist ein eingetragenes Warenzeichen AT&T.

Copyright an der Übersetzung Siemens AG, 1990, alle Rechte vorbehalten.

Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhaltes nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz.
Alle Rechte vorbehalten, insbesondere für den Fall der Patenterteilung oder GM-Eintragung.
Liefermöglichkeiten und technische Änderungen vorbehalten.

Copyright © Siemens AG 1990

Alle Rechte vorbehalten.

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

Siemens Aktiengesellschaft

Vorwort

Ein Blick auf die SINIX-Handbücher

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

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

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

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

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

Das Handbuch im Überblick

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

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

Aufteilung der Beschreibung CES C-Entwicklungssystem

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

1 Einführung

Die Einführung

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

2 Allgemeine Hinweise

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

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

3 Funktionen

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

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

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

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

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

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

4 Include-Dateien

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

Voraussetzungen dieses Handbuchs

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

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

Eine Bitte an Sie

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

Manualredaktion DI ST QM2
Otto-Hahn-Ring 6, 8 München 83

Inhalt

1 Einführung

Dieses Kapitel erklärt zunächst, wie die Beschreibungen der C-Funktionen und Include-Dateien aufgebaut sind. Danach erklärt der zweite Abschnitt wichtige, in den Beschreibungen häufig verwendete Begriffe. Der dritte Abschnitt schließlich gibt Hinweise und Ratschläge, die dazu beitragen sollen, Fehler bei der Verwendung der beschriebenen Funktionen zu vermeiden.

1.1 Beschreibungsformat

Nachdem der erste Unterabschnitt den Aufbau der einzelnen Beschreibungen erläutert hat, stellt der zweite Unterabschnitt die verwendeten Schriftarten und deren Bedeutung vor.

1.1.1 Beschreibungsrahmen

Die Beschreibungen im Handbuch folgen einem gemeinsamen Schema:

NAME
Hier finden Sie den oder die Namen der beschriebenen Funktion, ihre amerikanische Originalbezeichnung und eine stichwortartige Beschreibung ihres Verwendungszwecks.

DEFINITION

```
Die Definition stellt die Syntax der Funktion dar und ist grau
unterlegt. Sie faßt alle Angaben zusammen, die für den Gebrauch
der Funktion wichtig sind.
```

BESCHREIBUNG
Die Beschreibung erläutert die Arbeitsweise und die Definition der Funktion und beschreibt deren Parameter.

ERGEBNIS
An dieser Stelle wird der Rückkehrwert der Funktion beschrieben.

FEHLER
Hier stehen die symbolischen Fehlernamen der Werte, die bei einem fehlerhaften Aufruf oder Ablauf einer Funktion in der externen Variablen *errno* abgelegt werden.

Wenn in diesem Abschnitt festgestellt wird, daß eine Funktion fehlschlägt, so bedeutet dies, daß dieser Fehler unter den angegebenen Bedingungen immer auftritt. Wenn es dagegen heißt, daß die Funktion fehlschlagen kann, dann wird dieser Fehler unter den angegebenen Bedingungen nur dann auftreten, wenn die Funktion dies im konkreten Fall erkennt. Ein solcher Fehler kann also zwar vorliegen, wenn er jedoch nicht erkannt wird, wird auch die Variable *errno* nicht gesetzt.

Nicht alle Funktionen legen bei einem Fehler auch einen Wert in *errno* ab. In solchen Fällen finden Sie dann hier den Text:
Es sind keine Fehler definiert.

HINWEIS

Hier finden Sie Begriffserklärungen oder Informationen über das Zusammenwirken mit anderen Funktionen oder Tips für die Anwendung.

Vorsicht bzw. Achtung
Hinweise, die so gekennzeichnet sind, sollten Sie auf jeden Fall beachten.

PORTABILITÄT

Hier finden Sie Informationen darüber, ob die Funktion im X/Open-Standard enthalten ist oder nicht. Unter SINIX gibt es eine Reihe von Funktionen, die zusätzlich zu den in diesem Standard verfügbaren Funktionen vorhanden sind. Portable Anwendungen sollten diese – und auch die Funktionen, die im X/Open-Standard als optional definiert sind – nicht verwenden.

Portabilitäts-Hinweis:
Wenn es zwischen SINIX und anderen X/Open-kompatiblen Systemen Unterschiede geben kann, dann werden Bemerkungen dazu so gekennzeichnet.

BEISPIEL

Ein in der Regel kurz gehaltenes Beispiel soll eine Anwendung der Funktion zeigen.

SIEHE AUCH

Hier finden Sie Querverweise auf verwandte Funktionen, Include-Dateien und Abschnitte im Kapitel 2.

1.1.2 Bedeutung der Schriftart

Halbfett
Halbfette Schrift ist ausschließlich dem Rahmen der Beschreibung vorbehalten. Dies soll Ihnen eine schnelle Orientierung ermöglichen.

Kursiv
In Kursivschrift sind gesetzt:

- Kommandos, z.B. *echo*, *cc*
- Systemaufrufe, C-Funktionen, Makros und Dateien, z.B. *printf()*
- Parameter von Funktionen und Variablennamen, z.B. *pfad*, *errno*
- Pfad- und Dateinamen, z.B. */.../rumpeldipumpel*. Drei Punkte im Pfadnamen der Datei *rumpeldipumpel* sollen zeigen, daß der Pfadname beliebig sein kann.

<kursiv>
In Kursivschrift, die von spitzen Klammern eingerahmt ist, werden Include-Dateien angegeben, z.B. *<stdlib.h>*.

[normal]
Normale Schrift, die von eckigen Klammern eingerahmt wird, bezeichnet einen Parameter, der nicht angegeben werden muß (wahlfreier Parameter).

```
[normal]
{normal}
```

Zwei Parameter in normaler Schrift, die übereinander stehen und von einer gemeinsamen geschweiften Klammer umfaßt werden, sind Parameter, die alternativ verwendet werden können.

GROSS
In Großbuchstaben sind symbolische Konstanten und die symbolischen Namen von Signalen geschrieben, z.B. SIGABRT.

[GROSS]
Großbuchstaben, die von eckigen Klammern eingeschlossen sind, bezeichnen Fehlernamen, z.B. [EINVAL].

{GROSS}
Großbuchstaben, die von geschweiften Klammern eingerahmt sind, bezeichnen implementierungsabhängige Konstanten, z.B. {INT_MAX}. Die konkreten Werte für diese Konstanten können in der Datei *<limits.h>* nachgesehen werden!

1.2 Glossar

In den Schnittstellendefinitionen werden viele Fachausdrücke verwendet. Die Beschreibung dieser Ausdrücke folgt.

Absoluter Pfadname (Absolute Pathname)

Siehe Abschnitt *Pfadnamen-Auflösung.*

Adreßraum (Adress Space)

Der Speicherbereich, auf den von einem Prozeß aus zugegriffen werden kann.

Auftragskontrolle (Job Control)

Auftragskontrolle erlaubt es den Benutzern, die Ausführung einzelner Prozesse zu stoppen (oder auszusetzen) und zu einem späteren Zeitpunkt fortzusetzen. Der Benutzer verwendet diese Fähigkeit typischerweise über die interaktive Schnittstelle, die gemeinsam durch den Ein- und Ausgabetreiber der Datensichtstation und durch einen Kommando-Interpreter angeboten wird. x/open-kompatible Systeme können die Auftragskontroll-Fähigkeit optional unterstützen; das Vorhandensein dieser Option wird einer Anwendung zur Übersetzungs- oder Laufzeit durch die Definition des Symbols {_POSIX_JOB_CONTROL} angezeigt. SINIX unterstützt die Auftragskontrolle nicht.

Benutzerklasse Eigentümer (File Owner Class)

Ein Prozeß ist in der Benutzerklasse Eigentümer einer Datei, wenn die effektive Benutzernummer des Prozesses zur Benutzernummer der Datei paßt.

Benutzerklasse Gruppe (File Group Class)

Ein Prozeß ist in der Benutzerklasse Gruppe einer Datei, wenn der Prozeß nicht in der Benutzerklasse Eigentümer ist und wenn die effektive Gruppennummer oder eine der weiteren Gruppennummern des Prozesses zu der Gruppennummer der Datei paßt. Andere Mitglieder dieser Klasse können von X/Open-kompatiblen Implementierungen definiert werden. Unter SINIX sind keine weiteren Mitglieder dieser Klasse definiert.

Benutzerklasse übrige Benutzer (File Other Class)

Ein Prozeß ist in der Benutzerklasse übrige Benutzer einer Datei wenn der Prozeß nicht in der Benutzerklasse Eigentümer oder in der Benutzerklasse Gruppe ist.

Benutzernummer (User ID)

Jeder Benutzer im System wird durch eine nichtnegative ganze Zahl identifiziert, die in einem *uid_t* enthalten sein kann (siehe *<sys/types.h>*. Wenn die Identität eines Benutzers einem Prozeß zugeordnet wird, dann wird auf eine Benutzernummer als auf eine reale, effektive oder gesicherte Benutzernummer zugegriffen.

Benutzernummer, effektive

Ein Attribut eines Prozesses, das verwendet wird, um verschiedene Rechte zu bestimmen, einschließlich der Zugriffsberechtigungen für eine Datei (siehe auch Abschnitt *Benutzernummer*). Dieser Wert kann sich während der Lebensdauer eines Prozesses ändern, so wie dies unter *setuid()* und *exec* beschrieben wird.

Benutzernummer, reale (Real User ID)

Das Attribut eines Prozesses, das zum Zeitpunkt der Erzeugung dieses Prozesses denjenigen Benutzer identifiziert, der diesen Prozeß erzeugt hat. Siehe auch Abschnitt *Benutzernummer*. Dieser Wert kann während der Lebensdauer des Prozesses verändert werden, wie dies unter *setuid()* beschrieben ist.

Besondere Rechte (Appropriate Privileges)

Einige der in diesem Handbuch definierten Funktionen und Funktions-Optionen verlangen spezielle Berechtigungen zu ihrem Aufruf. SINIX definiert ein oder mehrere besondere Rechte, die ein Prozess besitzen kann, der diese Funktionen aufrufen will. Dieser Begriff ersetzt gemäß X/Open-Standard den Begriff der Systemverwalter-Rechte.

Clock Tick

Die (maschinenspezifische) Anzahl der Intervalle pro Sekunde wird durch {CLK_TCK} definiert. Sie wird verwendet, um den Wert im Typ *clock_t* auszudrücken, der von *times()* geliefert wird.

C Standard

Die Abkürzung für *Draft ANSI X3.159, Programming Language C* (der sogenannte ANSI-Standard für die Programmiersprache C).

Datei (File)

Ein Objekt, auf das geschrieben und/oder von dem gelesen werden kann. Eine Datei besitzt bestimmte Attribute, einschließlich der Zugriffsberechtigungen und des Dateityps. Dateitypen schließen normale Dateien, Gerätedateien für zeichen- und blockorientierte Geräte, FIFO-Gerätedateien und Dateiverzeichnisse ein.

Andere Dateitypen können von der Implementierung definiert werden. Unter SINIX sind keine anderen Dateitypen definiert.

Dateibaum (File Hierarchie)

Dateien im System sind in einer hierarchischen Struktur organisiert. Alle Knoten, die keine Blätter sind, sind Dateiverzeichnisse (nichtterminale Knoten). Alle Knoten, die Blätter sind, sind Dateien beliebigen Dateityps (terminale Knoten).
Da sich mehrere Dateiverzeichniseinträge auf dieselbe Datei beziehen können, wird der Dateibaum sinnvoll als *gerichteter Graph* beschrieben.

Datei-Beschreibung (File Description)

Siehe Abschnitt *Datei-Beschreibung, offene.*

Datei-Beschreibung, offene (Open File Description)

Eine offene Datei-Beschreibung enthält die Informationen, wie ein Prozeß oder eine Gruppe von Prozessen auf eine Datei zugreifen. Jede Dateikennzahl verweist auf genau eine offene Datei-Beschreibung. Aber auf eine offene Datei-Beschreibung können mehr als eine Dateikennzahl verweisen. Die Dateiposition, der Dateimodus und die Zugriffsarten auf diese Datei sind Attribute einer offenen Datei-Beschreibung.

Dateikennzahl (File Descriptor)

Eine Dateikennzahl ist eine je Prozeß eindeutige ganze Zahl, die verwendet wird, um eine offene Datei zum Zweck des Dateizugriffs zu identifizieren. Der Wert einer Dateikennzahl kann von 0 bis {OPEN_MAX}-1 reichen. Ein Prozeß kann nicht mehr als {OPEN_MAX} Dateikennzahlen gleichzeitig offen haben. Siehe Abschnitt *Datei-Beschreibung, offene.*

Dateiname (Filename)

Namen, die aus 1 bis {NAME_MAX} Bytes bestehen, können verwendet werden, um eine Datei zu benennen. Die Zeichen, die einen Namen bilden, können aus dem gesamten Zeichensatz gewählt werden, mit Ausnahme der Zeichen Nullbyte (\0) und Schrägstrich (/). Die Dateinamen . und .. haben eine besondere Bedeutung, siehe auch Abschnitt *Pfadnamen-Auflösung.* Auf einen Dateinamen bezieht man sich manchmal auch als eine Pfadnamen-Komponente.

Dateinamen sollten aus dem Zeichensatz für portable Dateinamen zusammengesetzt werden, weil die Verwendung anderer Zeichen in bestimmten Zusammenhängen verwirrend oder zweifelhaft sein kann (siehe Abschnitt *Zeichensatz für portable Dateinamen*).

Datei, normale (Regular File)

Eine Datei, die eine wahlfrei zugreifbare Folge von Bytes, ohne jede weitere, vom System festgelegte, Struktur ist.

Dateinummer (File Serial Nummer)

Eine Dateinummer ist ein im Dateisystem eindeutiger Identifikator für eine Datei.

Dateimodus (File Mode)

Der Dateimodus beinhaltet die Schutzbits der Datei und andere Kennzeichen der Datei, so wie unter *<sys/stat.h>* beschrieben.

Datei, offene (Open file)

Eine Datei die derzeit einer Dateikennzahl zugeordnet ist.

Dateisystem (File System)

Eine Ansammlung von Dateien und bestimmter Attribute von Dateien. Es bietet den Namensbereich für Dateinummern, die sich auf diese Dateien beziehen.

Dateisystem (nur Lesen) (Read-only File System)

Ein Dateisystem, das eine von der Implementierung definierte, charakteristische einschränkende Änderung besitzt.

Dateiverzeichnis (Directory)

Eine Datei, die Dateiverzeichniseinträge enthält. Keine zwei Dateiverzeichniseinträge im selben Dateiverzeichnis haben denselben Namen.

Dateiverzeichniseintrag (Directory Entry)

Ein Objekt, das einer Datei einen Dateinamen zuordnet. Mehrere Dateiverzeichniseinträge können derselben Datei Namen zuordnen.

Dateiverzeichnis, aktuelles (Working or Current Working Directory)

Ein einem Prozeß zugeordnetes Dateiverzeichnis. Dieses Dateiverzeichnis wird bei der Pfadnamen-Auflösung für solche Pfadnamen verwendet, die nicht mit einem Schrägstrich beginnen.

Dateiverzeichnis, leeres (Empty Directory)

Ein Dateiverzeichnis, das höchstens die Dateiverzeichniseinträge . und .. enthält (siehe Abschnitte *Punkt* und *Punkt-Punkt*).

Dateiverzeichnisstrom (Directory Stream)

Ein für jeden Prozeß eindeutiger Wert, der benutzt wird, um auf ein offenes Dateiverzeichnis zu verweisen.

Dateizeiten-Änderung (File Times Update)

Jeder Datei sind drei Zeitwerte zugeordnet, die geändert werden, wenn auf die Daten in dieser Datei zugegriffen wurde oder die Daten oder der Dateizustand verändert wurden. Diese Werte werden in der Eigenschafts-Struktur zurückgegeben, wie unter *<sys/stat.h>* beschrieben.

Für jede Funktion in diesem Handbuch, die Daten einer Datei liest oder schreibt oder den Zustand einer Datei ändert, werden die entsprechenden, zeitbezogenen Felder als "zum Ändern markiert" bezeichnet. Zu einem Änderungszeitpunkt werden alle markierten Felder mit der aktuellen Zeit besetzt und die Änderungsmarken gelöscht. Zwei solche Änderungszeitpunkte sind, wenn eine Datei nicht länger von irgendeinem Prozeß geöffnet ist und wenn *stat()* oder *fstat()* für diese Datei ausgeführt werden. Sonstige Änderungszeitpunkte sind nicht festgelegt. Für Dateien in Dateisystemen, die nur zum Lesen eingehängt sind, werden keine Änderungen durchgeführt.

Datensichtstation (Terminal or Terminal Device)

Eine Gerätedatei für ein zeichenorientiertes Gerät, die den Vorgaben der allgemeinen Datensichtstations-Schnittstelle genügt, so wie dies in Abschnitt *2.4 Allgemeine Terminalschnittstelle* beschrieben ist.

Epochenwert (Epoch)

Die Zeit 0 Uhr, 0 Minuten und 0 Sekunden am 1. Januar 1970 (Coordinated Universal Time)

Erweiterte Sicherheitskontrollen (Extended Security Controls)

Die Zugriffskontrolle (siehe auch Abschnitt *Zugriffsberechtigungen für Dateien*) und die Rechte (siehe auch Abschnitt *Besondere Rechte*) wurden definiert, um implementierungsabhängige erweiterte Sicherheitskontrollen zuzulassen. Diese erlauben einer Implementierung, Sicherheitsmechanismen anzubieten, um Sicherheitsverfahren zu implementieren, die von den in diesem Handbuch definierten verschieden sind. Diese Sicherheitsmechanismen werden die definierte Semantik keiner der in diesem Handbuch beschriebenen Funktionen ändern oder ersetzen. Unter CES V5.22 sind keine erweiterten Sicherheitskontrollen definiert.

FIFO-Gerätedateien (oder FIFO) (FIFO Special Files)

Eine FIFO-Gerätedatei ist eine Dateiart. Daten, die auf eine FIFO-Gerätedatei geschrieben werden, werden auf first-in-first-out-Basis gelesen. Andere Eigenschaften von FIFO's werden unter *mkfifo()*, *lseek()*, *open()*, *read()* und *write()* beschrieben.

Gerät (Device)

Eine Peripheriegerät oder ein Objekt, das einer Anwendung als solches erscheint.

Gerätedatei für blockorientierte Geräte (Block Special File)

Eine Datei, die sich auf ein Gerät bezieht. Eine Gerätedatei für blockorientierte Geräte unterscheidet sich von einer Gerätedatei für zeichenorientierte Geräte dadurch, daß sie den Zugriff auf das Gerät in einer Art und Weise bietet, die die Hardware-Eigenschaften des Geräts verbirgt.

Gerätedatei für zeichenorientierte Geräte (Character Special File)

Eine Datei, die sich auf ein Gerät bezieht. Ein Beispiel für eine solche Datei ist die Gerätedatei für eine Datensichtstation, deren Zugriff unter Abschnitt *2.4 Allgemeine Terminalschnittstelle* beschrieben wird.

Gerätenummer (Device ID)

Eine nichtnegative ganze Zahl, die verwendet wird, um ein Gerät zu identifizieren.

Gesicherte Benutzernummer (Saved Set User ID)

Die gesicherte Benutzernummer ist ein Attribut eines Prozesses, das mehr Flexibilität bei der Zuteilung des Attributs effektive Benutzernummer erlaubt, so wie dies unter *setuid()* und *exec* beschrieben wird.

Gesicherte Gruppennummer (Saved Set User ID)

Die gesicherte Gruppennummer ist ein Attribut eines Prozesses, das mehr Flexibilität bei der Zuteilung des Attributs effektive Gruppennummer erlaubt, so wie dies unter *setgid()* und *exec* beschrieben wird.

Hintergrund-Prozeßgruppe (Background Process Group)

Jede Prozeßgruppe, die Mitglied einer Sitzung ist, die eine Verbindung zu einem kontrollierenden Terminal hergestellt hat und die nicht in der Vordergrund-Prozeßgruppe ist.

Gruppennummer (Group ID)

Jeder Benutzer am System ist Mitglied zumindest einer Gruppe. Eine Gruppe wird durch eine Gruppennummer identifiziert, eine nichtnegative ganze Zahl, die in einem *gid_t* abgelegt sein kann (siehe *<sys/types.h>*). Wenn die Identität einer Gruppe einem Prozeß zugeordnet wird, dann wird der Wert einer Gruppennummer als eine reale Gruppennummer, als eine effektive Gruppennummer, als eine der *optionalen* weiteren Gruppennummern oder als eine gesicherte Gruppennummer angesprochen.

Gruppennummer, effektive (Effective Group ID)

Ein Attribut eines Prozesses, das verwendet wird, um verschiedene Rechte zu bestimmen, einschließlich der Zugriffsberechtigungen für eine Datei (siehe auch Abschnitt *Gruppennummer*). Dieser Wert kann sich während der Lebensdauer eines Prozesses ändern, so wie dies unter *setgid()* und *exec* beschrieben wird.

Gruppennummer, reale (Real Group ID)

Die Eigenschaft eines Prozesses, die zum Zeitpunkt seiner Erzeugung die Gruppe des Benutzers identifiziert, der diesen Prozeß erzeugt hat. Siehe Abschnitt *Gruppennummer*. Dieser Wert kann sich während der Lebensdauer des Prozesses ändern, so wie dies unter *setgid()* beschrieben ist.

Kommando-Interpreter (Command Interpreter)

Für Anwendungen ist es möglich, Dienstprogramme über eine Reihe von Schnittstellen aufzurufen, für die man annimmt, daß sie sich wie Kommando-Interpreter verhalten. Die offensichtlichsten dieser Schnittstellen sind *sh* (*Betriebssystem SINIX V5.22 Kommandos* [1]) und *system()*, obwohl auch *popen()* und die verschiedenen Formen von *exec* ebenfalls als Interpreter verstanden werden können.

Kontrollierender Prozeß (Controlling Process)

Der Sitzungsführer, der die Verbindung zum kontrollierenden Terminal hergestellt hat. Wenn die Datensichtstation aufhört, ein kontrollierendes Terminal für diese Sitzung zu sein, dann hört auch der Sitzungsführer auf, der kontrollierende Prozeß zu sein.

Kontrollierendes Terminal (Controlling Terminal)

Eine Datensichtstation, die einer Sitzung zugeordnet ist. Jede Sitzung kann höchstens ein kontrollierendes Terminal besitzen, das ihr zugeordnet ist; ein kontrollierendes Terminal ist genau einer Sitzung zugeordnet. Bestimmte Eingabesequenzen vom kontrollierenden Terminal (siehe Abschnitt *2.4 Allgemeine Terminalschnittstelle*) bewirken, daß Signale an alle Prozesse gesendet werden, die sich in der Prozeßgruppe befinden, die diesem kontrollierenden Terminal zugeordnet ist.

Makro für den Test von Eigenschaften (Feature Test Macro)

Ein Makro, das verwendet wird, um zu entscheiden, ob eine bestimmte Menge von Eigenschaften aus einer Include-Datei eingebunden wird.

Meldungskatalog (Message Catalogue)

Eine Datei oder ein Speicherbereich, der Programmmeldungen, Eingabeaufforderungen und Antworten darauf für eine bestimmte Landessprache, ein bestimmtes Gebiet und einen bestimmten Zeichensatz enthält.

Meldungskatalog-Deskriptor (Message Catalogue Descriptor)

Ein je Prozeß eindeutiger Wert des Typs *nl_catd*, der verwendet wird, um einen offenen Meldungskatalog zu identifizieren (siehe *nl_types.h*>.

Modus einer Datei (File Mode)

Der Modus einer Datei ist eine Ansammlung von Attributen, die den Dateityp und ihre Zugriffsberechtigungen angeben.

NaN (Not a Number)

Ein Wert, der in einer Gleitpunktzahl abgespeichert werden kann, aber keine gültige Gleitpunktzahl ist. Ein Beispiel für solch ein Bitmuster unter SINIX ist eine Gleitpunktzahl, deren Exponenten-Bits alle gleich 1 gesetzt sind.

Nullbyte (Null Character)

Ein Byte, in dem alle Bits auf 0 gesetzt sind.

Nullzeiger (Null Pointer)

Dies ist der Wert, den man erhält, wenn man die Zahl 0 in einen Zeiger umwandelt, z.B. *(char *) 0*. Die Programmiersprache C garantiert, daß dieser Wert keinem gültigen Zeiger entspricht, daher wird er von vielen Funktionen verwendet, die Zeiger zurückgeben, um einen Fehler anzuzeigen.

Pfadnamen-Anfang (Pathname Prefix)

Ein Pfadname, der optional mit einem Schrägstrich endet, und der auf ein Dateiverzeichnis verweist.

Pfadname (Pathname)

Eine Zeichenkette, die verwendet wird, um eine Datei zu identifizieren. Sie besteht aus höchstens {PATH_MAX} Bytes, einschließlich des abschließenden Nullbytes. Sie besteht aus einem optionalen, führenden Schrägstrich, gefolgt von keinem oder mehr Dateinamen, die wiederum durch Schrägstriche getrennt sind. Wenn der Pfadname sich auf ein Dateiverzeichnis bezieht, dann kann er auch einen oder mehrere abschließende Schrägstriche enthalten. Mehrere aufeinanderfolgende Schrägstriche werden als ein Schrägstrich angenommen. Ein Pfadname, der mit zwei Schrägstrichen beginnt, kann von einigen X/Open-kompatiblen Implementierungen in besonderer Weise interpretiert werden, obwohl mehr als zwei führende Schrägstriche als ein einziger Schrägstrich behandelt werden. Die Interpretation des Pfadnamens wird beschrieben im Abschnitt *Pfadnamen-Auflösung*.

Pfadnamen-Auflösung (Pathname Resolution)

Pfadnamen-Auflösung wird für einen Prozeß durchgeführt, um einen Pfadnamen bis zu einer bestimmten Datei im Dateibaum aufzulösen. Es kann mehrere Pfadnamen geben, die sich bis zur selben Datei auflösen lassen.

Jeder Dateiname im Pfadnamen wird ausfindig gemacht im Dateiverzeichnis, das durch seinen Vorgänger angegeben wird (z.B. für Pfadnamenfragment *a/b* wird *b* im Dateiverzeichnis *a* ausfindig gemacht). Die Pfadnamen-Auflösung schlägt fehl, wenn dies nicht durchgeführt werden kann. Wenn der Pfadname mit einem Schrägstrich beginnt, dann wird der Vorgänger des ersten Dateinamens im Pfadnamen als das Root-Dateiverzeichnis des Prozesses angenommen (solche Pfadnamen werden auch als absolute Pfadnamen bezeichnet).

Wenn der Pfadname nicht mit einem Schrägstrich beginnt, dann wird als Vorgänger des ersten Dateinamens im Pfadname das aktuelle Dateiverzeichnis des Prozesses angenommen (solche Pfadnamen werden auch als relative Pfadnamen bezeichnet).

Die Interpretation einer Pfadnamen-Komponente hängt von den Werten {NAME_MAX} und {_POSIX_NO_TRUNC} ab, die dem Pfadnamen-Anfang dieser Komponente zugeordnet sind. Wenn eine Pfadnamen-Komponente länger als {NAME_MAX} ist, und {_POSIX_NO_TRUNC} für den Pfadnamen-Anfang dieser Komponente aktiv ist (siehe *pathconf()*), dann gilt dies als Fehler. Andernfalls werden nur die ersten {NAME_MAX} Bytes der Pfadnamen-Komponente berücksichtigt.

Der besondere Dateiname . verweist auf das Dateiverzeichnis, das durch seinen Vorgänger angegeben wird. Der besondere Dateiname .. verweist auf das übergeordnete Dateiverzeichnis seines Vorgängers. Als Sonderfall kann .. im Root-Dateiverzeichnis auf das Root-Dateiverzeichnis selbst verweisen.

Ein Pfadname, der nur aus einem einzelnen Schrägstrich besteht, benennt das Root-Dateiverzeichnis des Prozesses. Ein leerer Pfadname ist ungültig.

Pfadnamen-Komponente (Pathname Component)

Siehe Abschnitt *Dateiname*.

Pipe

Ein Objekt auf das über eine der beiden Dateikennzahlen zugegriffen wird, die durch die Funktion *pipe()* erzeugt worden sind. Einmal erzeugt, können diese Dateikennzahlen zu seiner Manipulation verwendet werden und es verhält sich genauso wie eine FIFO-Gerätedatei, wenn auf diese Art darauf zugegriffen wird. Es hat im Dateibaum keinen Namen.

Portable Pfadnamen (Portable Pathname)

Damit ein Pfadname unter X/Open-kompatiblen Systemen portabel ist, sollte er aus höchstens {PATH_MAX} Bytes bestehen, einschließlich des abschließenden Nullbytes. Es sollte ein Pfadname sein, der aus einem optionalen, führenden Schrägstrich sowie keinem oder mehr portablen Dateinamen bestehen, die durch Schrägstriche voneinander getrennt sind.

Dateiposition (File Offset)

Die Dateiposition gibt an, wieviele Bytes vom Dateianfang entfernt die nächste Ein- oder Ausgabeoperation beginnt (1. Byte = 1). Jede offene Datei-Beschreibung, die zu einer normalen Datei, einer Gerätedatei für blockorientierte Geräte oder einem Dateiverzeichnis gehört, besitzt eine Position. Eine Gerätedatei für ein zeichenorientiertes Gerät, das keine Datensichtstation ist, kann eine Position besitzen. Es gibt keine Position in Pipes und FIFOs.

Prozeß (Process)

Ein Adreßraum und einzelner Programmcode der in diesem Adreßraum ausgeführt wird, sowie die von diesem benötigten Betriebsmittel des Systems. Ein Prozeß wird von einem anderen Prozeß durch den Aufruf der Funktion *fork()* erzeugt. Der Prozeß, der *fork()* aufruft, heißt Vaterprozeß und der neue, durch *fork()* erzeugte Prozeß heißt Sohnprozeß.

Prozeßgruppe (Process Group)

Jeder Prozeß im System ist Mitglied einer Prozeßgruppe, die durch eine Prozeßgruppennummer identifiziert wird. Diese Gruppierung von Prozessen erlaubt es, verwandten Gruppen von Prozessen Signale zu senden. Ein neu erzeugter Prozeß gehört der Prozeßgruppe seines Erzeugers an.

Prozeßgruppennummer (Process Group ID)

Jede Prozeßgruppe im System wird während ihrer Lebensdauer eindeutig durch eine positive ganze Zahl identifiziert, die in einem *pid_t* enthalten sein kann und Prozeßgruppennummer genannt wird (sieh *<sys/types.h>*). Eine Prozeßgruppennummer kann von System nicht erneut verwendet werden, bevor die Lebensdauer der Prozeßgruppe endet.

Prozeßgruppenchef (Process Group Leader)

Ein Prozeß, dessen Prozeßnummer zu seiner Prozeßgruppennummer identisch ist.

Prozeßgruppe, Lebensdauer (Process Group Lifetime)

Ein Zeitraum, der dann beginnt, wenn eine Prozeßgruppe erzeugt wird und der dann endet, wenn der letzte verbleibende Prozeß dieser Prozeßgruppe diese Prozeßgruppe verläßt. Das Verlassen einer Prozeßgruppe erfolgt entweder durch die Beendigung des Prozesses oder durch den Aufruf einer der Funktionen *setsid()* oder *setpgid()*.

Prozeß, Lebensdauer (Process Lifetime)

Nachdem ein Prozeß mit der Funktion *fork()* erzeugt wurde, gilt er als aktiv. Sein Kontrollbereich und Adreßraum existieren, bis er sich beendet. Dann gelangt er in einen inaktiven Zustand, in dem bestimmte Betriebsmittel an das System zurückgegeben werden können, obwohl einige Betriebsmittel, wie z.B. die Prozeßnummer, noch immer verwendet werden. Wenn ein anderer Prozeß eine der Funktionen *wait()* oder *waitpid()* für einen inaktiven Prozeß ausführt, dann werden die übrigen Betriebsmittel an das System zurückgegeben. Das letzte an das System zurückgegebene Betriebsmittel ist die Prozeßnummer. Zu diesem Zeitpunkt endet die Lebensdauer des Prozesses.

Prozeßnummer (Process ID)

Jeder aktive Prozeß im System wird während seiner Lebensdauer eindeutig durch eine positive ganze Zahl identifiziert, die in einem *pid_t* enthalten sein kann und die Prozeßnummer genannt wird (siehe *<sys/types.h>*). Eine Prozeßnummer darf vom System nicht vor Ende der Lebensdauer des Prozesses wiederverwendet werden. Zusätzlich darf, wenn eine Prozeßgruppe existiert, deren Prozeßgruppennummer gleich dieser Prozeßnummer ist, diese nicht wiederverwendet werden, bis die Lebensdauer der Prozeßgruppe endet. Kein Prozeß, der nicht ein Systemprozeß ist, sollte die Prozeßnummer 1 besitzen.

Punkt (Dot)

Der Dateiname, der nur aus einem einzigen Zeichen '.' (Punkt) besteht. (Siehe auch Abschnitt *Pfadnamen-Auflösung*.)

Punkt-Punkt (Dot-dot)

Der Dateiname, der aus den zwei Zeichen '..' (Punkt) besteht. (Siehe auch Abschnitt *Pfadnamen-Auflösung*.)

Rechte (Privilege)

Siehe Abschnitt *Besondere Rechte*.

Root-Dateiverzeichnis (Root Directory)

Ein Dateiverzeichnis, das einem Prozeß zugeordnet ist und das bei der Pfadnamen-Auflösung für Pfadnamen verwendet wird, die mit einem Schrägstrich beginnen.

Schrägstrich (Slash)

Der Begriff Schrägstrich wird verwendet, um das einzelne Zeichen '/' darzustellen. Dieses Zeichen ist im Amerikanischen auch unter dem Namen *solidus* in *ISO 8859/1* bekannt.

Schutzbits einer Datei (File Permission Bits)

Die Schutzbits einer Datei werden zusammen mit anderen Informationen benutzt, um zu entscheiden, ob ein Prozeß Lese-, Schreib- oder Ausführungs-/Durchsuchrecht für eine Datei besitzt. Die Bits sind in drei Abschnitte eingeteilt: Eigentümer, Gruppe und übrige Benutzer. Jeder Abschnitt wird in Verbindung mit der entsprechenden Benutzerklasse der Prozesse verwendet. Diese Bits sind im Dateimodus enthalten, wie unter *<sys/stat.h>* beschrieben. Der Gebrauch der Schutzbits für eine Datei in Zugriffsentscheidungen wird detailliert unter Zugriffsberechtigungen für Dateien beschrieben.

Signal

Ein Mechanismus, durch den ein Prozeß von einem im System auftretenden Ereignis benachrichtigt oder beeinflußt werden kann. Beispiele für solche Ereignisse schließen Hardware-Ausnahmen und besondere Aktionen von Prozessen ein. Der Begriff Signal wird auch für die Ereignisse selbst verwendet.

Sitzung (Session)

Jede Prozeßgruppe ist Mitlied einer Sitzung. Für einen Prozeß wird angenommen, daß er ein Mitglied derjenigen Sitzung ist, in der seine Prozeßgruppe Mitglied ist. Ein neu erzeugter Prozeß gehört der Sitzung seines Erzeugers an. Ein Prozeß kann die Mitgleidschaft in einer Sitzung ändern (siehe *setsid()*). Implementierungen, die *setpgid()* unterstützen, können mehrere Prozeßgruppen in derselben Sitzung haben.

Sitzung, Lebensdauer (Session Lifetime)

Der Zeitraum zwischen der Erzeugung einer Sitzung und dem Ende der Lebensdauer aller Prozeßgruppen, die Mitglieder dieser Sitzung bleiben.

Sitzungsführer (Session Leader)

Ein Sitzungsführer ist ein Prozeß, der eine Sitzung erzeugt hat (siehe auch unter *setsid()*).

Sohnprozeß (Child Process)

Siehe Abschnitt *Prozeß*.

System

Der Begriff System wird in diesem Handbuch verwendet, um eine Implementierung der X/Open-Systemschnittstellen zu bezeichnen.

Systemprozeß (System Process)

Ein Objekt, das anders als ein Prozeß der eine Anwendung ausführt, vom System definiert ist. Ein Systemprozeß besitzt eine Prozeßnummer.

Übergeordnetes Dateiverzeichnis (Parent Directory)

Das Dateiverzeichnis, das einen Dateiverzeichniseintrag für die betreffende Datei enthält. Dieses Konzept findet für . und .. keine Anwendung.

Vaterprozeß (Parent Process)

Siehe Abschnitt *Prozeß*.

Vaterprozeßnummer (Parent Process ID)

Ein neuer Prozeß wird von einem derzeit aktiven Prozeß erzeugt. Die Vaterprozeßnummer eines Prozesses ist die Prozeßnummer seines Erzeugers, solange dieser lebt. Nachdem die Lebensdauer des Erzeugers geendet hat, ist die Vaterprozeßnummer die Prozeßnummer eines von der Implementierung definierten Prozesses. Unter SINIX ist dies die Prozeßnummer des *init*-Prozesses.

Verwaiste Prozeßgruppe (Orphaned Process Group)

Eine Prozeßgruppe, in der der Vaterprozeß jedes Mitglieds selbst Mitglied der Gruppe oder nicht Mitglied der Sitzung dieser Gruppe ist.

Verweis (Link)

Siehe Abschnitt *Dateiverzeichniseintrag*.

Verweiszähler (Link Count)

Der Verweiszähler einer Datei ist die Anzahl der Dateiverzeichniseinträge, die sich auf diese Datei beziehen.

Vordergrund-Prozeßgruppe (Foreground Process Group)

Jede Sitzung, die eine Verbindung zu einem kontrollierenden Terminal aufgebaut hat, besitzt exakt eine Prozeßgruppe dieser Sitzung als Vordergrund-Prozeßgruppe dieses kontrollierenden Terminals. Die Vordergrund-Prozeßgruppe besitzt bestimmte Rechte, wenn sie auf ihr kontrollierendes Terminal zugreift, die den Hintergrund-Prozeßgruppen verwehrt sind. Siehe Abschnitt *2.4 Allgemeine Terminalschnittstelle.*

Vordergrund-Prozeßgruppennummer (Foreground Process Group ID)

Die Prozeßgruppenummer einer Vordergrund-Prozeßgruppe.

Weitere Gruppennummer (Supplementary Group ID)

Ein Prozeß besitzt, zusätzlich zur effektiven Gruppennummer, bis zu {NGROUPS_MAX} weitere Gruppennummern, die verwendet werden, um die Zugriffsberechtigungen auf eine Datei zu entscheiden, Die weiteren Gruppennummern eines Prozesses werden bei dessen Erzeugung gleich den weiteren Gruppennummern des Vaterprozesses gesetzt. Ob die effektive Gruppennummer eines Prozesses mit in die Liste der weiteren Gruppennummern aufgenommen wird oder nicht, ist nicht festgelegt.

Zeichen (Character)

Eine Folge von einem oder mehreren Bytes, die ein einzelnes grafisches Symbol repräsentiert. Zeichen, die aus mehreren Bytes bestehen, heißen auch Multibyte-Zeichen.

Zeichenkette, leere (Empty String)

Die leere Zeichenkette ist ein char-Vektor, dessen erstes Element das Nullbyte ('\0') ist.

Zeichensatz (Character Set)

In der internationalen Umgebung für C werden die Zeichen gemäß den Regeln des 7-Bit US-ASCII-Zeichensatzes codiert. Dabei erhält jedes Zeichen des Zeichensatzes verschiedene Attribute zugeordnet, wie ein grafisches Symbol, mögliche Umwandlung in den entsprechenden Groß- oder Kleinbuchstaben, Zeichenklassen für die Klassifikation von Zeichen und eine Position innerhalb der Sortierreihenfolge. In internationalisierten Programmen sind hier beliebige landessprachliche Zeichensätze denkbar. Siehe auch *Abschnitt 2.5.*

Zeichensatz für portable Dateinamen (Portable Filename Character Set)

Damit ein Dateiname portabel ist für alle Implementierungen, die konform zum *IEEE Standard 1003.1-1988* sind, darf er nur aus folgenden Zeichen bestehen:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 . _ -

Die letzten drei Zeichen sind der Punkt, der Unterstrich und der Bindestrich.

Der Bindestrich sollte nicht als erstes Zeichen des portablen Dateinamens verwendet werden. Groß- und Kleinbuchstaben behalten unter allen konformen Implementierungen ihre eindeutigen Identitäten.

Ein portabler Dateiname über Implementierungen die konform zum *X/Open Portability Guide (Ausgabe 3)* sind, kann aus jedem beliebigen der *ISO 8859/1*-Zeichen bestehen:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 ! " # $ % & ' ( ) * + , - : ; < = > ? @ [ ] ^ _ { | } ~
¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ

Zombieprozeß (Zombie Process)

Ein inaktiver Prozeß, der zu einem späteren Zeitpunkt gelöscht werden wird, wenn sein Vaterprozeß eine der Funktionen *wait()* oder *waitpid()* ausführt.

Zugriffsberechtigungen für Dateien (File Access Permissions)

Der Standardmechanismus für die Zugriffskontrolle für Dateien benutzt die unten beschriebenen Schutzbits. Diese Bits werden bei der Erzeugung der Datei durch *open()*, *creat()*, *mkdir()* und *mkfifo()* gesetzt und durch *chmod()* geändert. Diese Bits werden von *stat()* oder *fstat()* gelesen.

Andere X/Open-kompatible Implementierungen können auch *zusätzliche* und/oder *alternative* Zugriffsmechanismen anbieten. Ein zusätzlicher Zugriffskontrollmechanismus für Dateien soll dabei nur die von den Schutzbits definierten Zugriffsberechtigungen weiter einschränken.

Ein alternativer Zugriffskontrollmechanismus für Dateien soll:

- Schutzbits für die Benutzerklassen Eigentümer, Gruppe, und übrige Benutzer festlegen, entsprechend den Zugriffsberechtigungen, die von *stat()* oder *fstat()* zurückgeliefert werden.
- Nur durch eine explizite Benutzeraktion auf einer je-Datei-Basis durch den Eigentümer der Datei oder einen Benutzer mit besonderen Rechten aktiviert werden.
- Deaktiviert werden, nachdem die Schutzbits durch *chmod()* geändert werden. Die Deaktivierung des alternativen Mechanismus muß nicht irgendwelche zusätzlichen, von der Implementierung definierten Mechanismen ebenfalls deaktivieren.

Sobald ein Prozeß für eine Datei die Zugriffsberechtigung zum lesen, schreiben oder oder ausführen/durchsuchen anfordert, wird der Zugriff wie folgt entschieden, sofern keine zusätzlichen Mechanismen den Zugriff verweigern:

- Wenn ein Prozeß besondere Rechte hat:
  - Wenn die Lese-, Schreib- oder Durchsucherlaubnis gefordert wird, dann wird der Zugriff gestattet.
  - Wenn die Ausführungserlaubnis gefordert wird, dann wird der Zugriff dann erlaubt, wenn die Ausführungserlaubnis zumindest einem Benutzer durch die Schutzbits oder einen anderen Zugriffskontroll-Mechanismus gewährt wird; andernfalls wird der Zugriff verweigert.
- Andernfalls:
  - Die Schutzbits einer Datei enthalten die Lese-, Schreib- und Ausführungs- bzw. Durchsucherlaubnis für die Benutzerklassen Eigentümer, Gruppe und übrige Benutzer. Der Zugriff wird gestattet, wenn ein alternativer Zugriffskontroll-Mechanismus nicht aktiviert ist und das Schutzbit für die geforderte Zugriffsberechtigung ist in der Benutzerklasse gesetzt, zu der der Prozeß gehört, oder wenn ein alternativer Zugriffskontroll-Mechanismus aktiviert ist und dieser den geforderten Zugriff erlaubt; andernfalls wird der Zugriff verweigert.

Zugriffsrecht (Access Mode)

Zugriffsrecht ist die spezielle Form des Zugriffs, die auf eine Datei gestattet wird.

[n, m] und [n, m)

Diese Notation bezeichnet einen mathematischen Bereich. Die eckigen Klammern [und] schließen die Grenzen jeweils mit ein, die runden Klammern (und) schließen diese aus. D.h. wenn x aus dem Bereich [0,1] ist, dann kann dies von 0 bis einschließlich 1 sein. Wenn aber x aus dem Bereich [0,1) ist, dann kann dies von 0 bis ausschließlich 1 sein.

1.3 Ratschläge

Dieses Kapitel gibt Ihnen einige Hinweise, die Ihnen bei der Erstellung von C Programmen hilfreich sein sollen, insbesondere, wenn Ihr Programm portabel sein, also auf verschiedenen Rechnern laufen soll.

1.3.1 Ein-/Ausgabe: C-Funktionen und Systemaufrufe nicht mischen

C-Funktionen zur Ein-/Ausgabe arbeiten mit einem eigenen Puffer, während die Systemaufrufe zur Ein-/Ausgabe ungepuffert lesen und schreiben. Wenn Sie beide Typen bei Ein-/Ausgabe-Operationen auf die gleiche (nur einmal geöffnete) Datei mischen, ist nicht gewährleistet, daß die Ein- und Ausgaben in der von Ihnen beabsichtigten Reihenfolge erfolgen. Es ist deshalb zu empfehlen, bei Ein-/Ausgabe-Operationen Systemaufrufe und C-Funktionen nicht zu mischen bzw. die Datei einmal für Ein-/Ausgabe mittels Systemaufrufen und einmal für Ein-/Ausgabe mittels C-Funktionen zu öffnen.

Sie erkennen die Systemaufrufe zur Ein- und Ausgabe daran, daß die betroffene Datei über eine Dateikennzahl angesprochen wird, während bei C-Funktionen ein Zeiger auf einen Datenstrom (Dateizeiger) angegeben werden muß.

1.3.2 Fehlerabfrage

Es gehört zu einem guten Programmierstil, nach jedem Aufruf einer Funktion abzufragen, ob ein Fehler vorliegt. Zum Beispiel liefert der Systemaufruf *read()*, wenn er erfolgreich ist, als Rückkehrwert die Zahl der gelesenen Bytes. Um einen Fehler anzuzeigen, liefert er einen Wert, der sonst nicht möglich ist, nämlich −1. Wenn Sie den Rückkehrwert einer Funktion nach jedem Aufruf prüfen, können Sie feststellen, ob ein Fehler eingetreten ist. Am Beispiel von *read()* sieht das so aus:

```
if (( zahl = read( datkz, puf, nbyte)) == -1)
{
   perror("read: ");
   exit(fehlercode);
}
else ...
```

Die Funktion *perror()* ist dabei eine hilfreiche Unterstützung. In den Beispielen der einzelnen Funktionsbeschreibungen wurden diese Abfragen häufig weggelassen, um die Beispiele kurz halten zu können.

1.3.3 Zeiger als Ergebnistyp

Funktionen, die einen Zeiger zurückliefern, schreiben ihr Ergebnis im allgemeinen in einen statischen Datenbereich, der bei jedem Aufruf überschrieben wird. Da dies eine häufige Fehlerquelle ist, wird bei der Beschreibung jeder Funktion darauf hingewiesen. Im folgenden sehen Sie zwei Beispiele für eine solche Funktion:

```
char *ctime(sekzg)

struct tm *localtime(sekzg)
```

1.3.4 Zeiger als Ergebnisparameter

Einige Funktionen haben einen Zeiger als Ergebnisparameter, d.h. sie tragen ihr Ergebnis z.B. in eine Struktur ein, wobei sie als Parameter einen Zeiger auf diese Struktur verlangen. Der Speicherplatz solcher Ergebnisparameter muß vor Aufruf der Funktion explizit bereitgestellt werden. Zum Beispiel:

```
struct stat statpuf          /* Strukturvariable deklarieren */
int ret;                     /* Platz fuer Returnwert */

ret=fstat(datkz, &statpuf);  /* Funktionsaufruf */
```

1.3.5 Konstante oder symbolische Konstante?

Sie sollten, um Ihr Programm lesbarer zu machen, symbolische Konstanten verwenden. Außerdem ist Ihr Programm, wenn sich der Wert einer Konstanten einmal ändern sollte (z.B. Mehrwertsteuer), dann leichter zu aktualisieren. Symbolische Konstanten sollten in Großbuchstaben angegeben werden, um sie von Variablen zu unterscheiden.

1.3.6 Include-Dateien berücksichtigen

Aus Portabilitätsgründen sollten Sie außerdem die Include-Dateien berücksichtigen, die in der Funktionsbeschreibung angegeben sind. Sie sollten prüfen, ob die Include Dateien auf Ihrem Rechner vorhanden sind und welchen Inhalt sie haben. Eine Beschreibung des Inhalts der SINIX Include-Dateien finden Sie in diesem Handbuch in Kapitel 4. Bei einigen Dateien aus */usr/include/sys* ist Vorsicht geboten, da sie i.d.R. maschinenabhängig sind.

Wenn Sie eine Funktion aus der mathematischen Bibliothek in einem Programm verwenden wollen, müssen Sie beim Übersetzen den Schalter *-lm* angeben:

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

2 Allgemeine Hinweise

Dieses Kapitel führt in einige Programmpakete und Bibliotheken ein, die im C-Entwicklungssystem unter SINIX V5.22 zur Verfügung stehen. Dabei handelt es sich um die folgendenden Themen:

- Dateikennzahlen und Datenströme,
- Fehlernummern und ihre Bedeutung,
- die Signale,
- die allgemeine Terminalschnittstelle,
- die NLS-Funktionen,
- die Funktionen zur Interprozeßkommunikation,
- die *curses*-Bibliothek,
- die *termcap*-Bibliothek.

Diese Themen werden in den folgenden Abschnitten behandelt. Bevor Sie sich zum erstenmal mit der Programmierung einer Anwendung aus einem dieser Bereiche befassen, sollten Sie die entsprechende Einführung zum jeweiligen Thema lesen. Sie finden dort allgemeine Hinweise zur Verwendung der Funktionen, die Ihnen für einen bestimmten Zweck zur Verfügung stehen, sowie einen allgemeinen Überblick über das jeweilige Thema.

2.1 Dateikennzahlen und Datenströme

Auf eine offene Dateibeschreibung kann über eine Dateikennzahl zugegriffen werden, die durch eine der Funktionen *open()* oder *pipe()* erzeugt worden ist, oder über einen Datenstrom, der durch eine der Funktionen *fopen()* oder *popen()* erzeugt worden ist. Sowohl eine Dateikennzahl als auch ein Datenstrom wird ein *Verweis* auf die offene Dateibeschreibung genannt (*Handle*); auf eine offene Dateibeschreibung können mehrere Verweise zeigen.

Verweise können durch konkrete Benutzer-Aktionen erzeugt oder gelöscht werden, ohne die zugrundeliegende offene Dateibeschreibung zu beeinflußen. Einige Funktionen, die solche Verweise erzeugen, sind die Funktionen *fcntl()*, *dup()*, *fdopen()*, *fileno()* und *fork()* ein. Diese Verweise können zumindest mit den Funktionen *fclose()*, *close()* und den *exec*-Funktionen wieder gelöscht werden.

Wird eine Dateikennzahl niemals in einer Operation verwendet, welche die Dateiposition beeinflußt (d.h. *read()*, *write()* oder *lseek()*), so gilt diese Dateikennzahl nicht als Verweis im Sinne dieses Abschnitts. Sie kann aber zu einem solchen werden, z.B. als Ergebnis von *fdopen()*, *dup()* oder *fork()*. Eine einem Datenstrom zugrundeliegende Dateikennzahl ist niemals eine solche Ausnahme, gleichgültig ob diese durch *fopen()* oder *fdopen()* erzeugt wurde, solange sie nicht direkt von der Anwendung benutzt wird, um die Dateiposition zu beeinflußen. Die Funktionen *read()* und *write()* beeinflußen die Dateiposition implizit; *lseek()* beeinflußt sie explizit.

Das Ergebnis von Funktionsaufrufen, die mit einem Verweis arbeiten (dem *aktiven Verweis*), kann jeweils in diesem Handbuch nachgelesen werden, aber wenn zwei oder mehr Verweise benutzt werden und einer davon ein Datenstrom ist, dann werden deren Aktionen so koordiniert, wie dies unten beschrieben wird. Wenn dies nicht geschieht, dann ist das Ergebnis undefiniert.

Ein Verweis, der ein Datenstrom ist, gilt dann als geschlossen, wenn entweder die Funktion *fclose()* oder die Funktion *freopen()* für diesen ausgeführt wird (das Ergebnis von *freopen()* ist ein neuer Datenstrom, der kein Verweis auf dieselbe offene Dateibeschreibung sein kann, auf die sein vorheriger Wert verwiesen hat), oder wenn der Prozeß, zu dem dieser Datenstrom gehört, mit *exit()* oder *abort()* beendet. Eine Dateikennzahl wird durch *close()*, *_exit()* oder eine der *exec*-Funktionen mit für diese Dateikennzahl gesetztem FD_CLOEXEC-Bit geschlossen.

Damit ein Verweis zum aktiven Verweis wird, müssen zwischen der letzten Verwendung des ersten, zur Zeit aktiven Verweises, und der ersten Verwendung des zweiten, zukünftig aktiven Verweises die unten beschriebenen Aktionen erfolgen. Dadurch wird der zweite Verweis zum aktiven Verweis. Alle die Dateiposition für den ersten Verweis beeinflußenden Aktivitäten der Anwendung müssen solange unterbunden werden, bis bis dieser wieder der aktive Verweis ist. Für eine Funktion zur Bearbeitung eines Datenstroms, die eine zugrundeliegende Funktion aufruft, die ihrerseits die Position in der Datei verändert, wird angenommen, daß die Funktion zur Bearbeitung des Datenstroms selbst die Dateiposition verändert. Die jeweils zugrundeliegenden Funktionen werden unten beschrieben.

Die Dateiverweise müssen nicht im selben Prozeß vorhanden sein, damit diese Regeln Anwendung finden.

Für den ersten Dateiverweis wird die erste zutreffende der unten angeführten Bedingungen angewendet. Wenn nach der Ausführung der unten notwendigen Aktionen dieser Verweis noch offen ist, dann kann ihn die Anwendung schließen.

- Wenn der erste Dateiverweis eine Dateikennzahl ist, dann ist keine Aktion notwendig.
- Wenn die einzige weitere auszuführende Aktion für einen Dateiverweis das Schließen ist, dann braucht keine Aktion mehr ausgeführt zu werden.
- Wenn es sich um einen ungepufferten Datenstrom handelt, dann braucht keine Aktion mehr ausgeführt zu werden.
- Wenn es sich um einen zeilengepufferten Datenstrom handelt und die letzte Aktion den selben Effekt auf die zugehörige Datei hat wie *fputs()*, dann braucht keine Aktion ausgeführt werden.
- Wenn es sich um einen, zum Schreiben oder Anfügen geöffneten Datenstrom handelt (der nicht gleichzeitig zum Lesen geöffnet ist), dann muß entweder ein *fflush()* ausgeführt, oder der Datenstrom geschlossen werden.
- Wenn der Datenstrom zum Lesen geöffnet und am Dateiende angelangt ist (d.h. *feof()* liefert wahr), dann muß keine Aktion ausgeführt werden.

- Wenn der Datenstrom auf eine Art geöffnet ist, die das Lesen gestattet und wenn die zugrundeliegende Dateibeschreibung auf ein Gerät verweist, das positionieren kann muß entweder ein *fflush()* ausgeführt, oder der Datenstrom geschlossen werden.
- Andernfalls ist das Ergebnis undefiniert.

Für den zweiten Dateiverweis:

- Wenn irgendein vorher aktiver Dateiverweis von einer Funktion verwendet wurde, welche die Dateiposition ausdrücklich veränderte, außer wie oben für den ersten Dateiverweis benötigt, dann muß die Anwendung, je nach Art des Verweises, eine der Funktionen *lseek()* oder *fseek()* ausführen, um an die entsprechende Position zu positionieren.

Wenn der aktive Verweis aufhört, zugreifbar zu sein, bevor die Anforderungen für den ersten Verweis erfüllt sind, dann geht die offene Dateibeschreibung in einen undefinierten Zustand über. Dies kann dann der Fall sein, wenn eine Funktion *fork()* oder *exit()* ausgeführt wird.

Die *exec*-Funktionen sorgen dafür, daß auf alle Datenströme, die zum Zeitpunkt ihres Aufrufs offen sind, nicht mehr zugegriffen werden kann, gleichgültig, welche Datenströme oder Dateikennzahlen für das Speicherabbild des neuen Prozesses zur Verfügung stehen.

SINIX stellt sicher, daß eine Anwendung, auch wenn sie aus mehreren Prozessen besteht, stets korrekte Ergebnisse liefert, d.h. daß beim Schreiben keine Daten verlorengehen oder doppelt geschrieben werden, daß alle Daten in der richtigen Reihenfolge geschrieben werden (außer bei einer entsprechenden Änderung durch das Positionieren), und daß beim sequentiellen Lesen alle Daten gefunden werden, sofern nach den oben angeführten Regeln vorgegangen wird. Dabei spielt es keine Rolle, in welcher Reihenfolge die Dateiverweise verwendet werden. Werden die oben aufgeführten Regeln nicht befolgt, dann ist das Ergebnis undefiniert.

2.2 Fehlernummern

Viele Funktionen legen eine Fehlernummer in der externen Variablen *errno* ab, die für eine Anwendung so definiert ist:

```
extern int errno;
```

Der Wert dieser Variablen ist nur nach dem Aufruf einer Funktion definiert, für die ausdrücklich angegeben wird, daß sie diese Variable besetzt, und bis zu ihrer Änderung durch einen nachfolgenden Funktionsaufruf. Die Variable *errno* sollte nur dann überprüft werden, wenn dies durch den Wert des Funktionsergebnisses angezeigt oder im Abschnitt HINWEIS für eine Funktion angegeben ist. Keine Funktion in diesem Handbuch setzt *errno* gleich 0, um einen Fehler anzuzeigen.

Wenn bei der Abarbeitung eines Funktionsaufrufs mehr als ein Fehler auftritt, dann kann ein beliebiger der möglichen Fehler zurückgeliefert werden, da die Reihenfolge ihrer Entdeckung undefiniert ist.

Der Abschnitt FEHLER für jeden Eintrag gibt an, ob ein Fehler unbedingt auftritt, oder ob er nur auftreten kann, wenn die entsprechenden Bedingungen erfüllt sind. Keine Funktion liefert zu einer in diesem Handbuch beschriebenen Fehlerbedingung eine andere Fehlernummer als in diesem Handbuch beschrieben.

Die folgenden symbolischen Namen kennzeichnen die möglichen Fehlernummern, die für die in diesem Handbuch beschriebenen Funktionen auftreten können. Die nachfolgenden allgemeinen Beschreibungen werden im Abschnitt FEHLER bei den jeweiligen Funktionen präziser angegeben, bei denen sie auftreten können. Nur diese symbolischen Namen sollten in Programmen verwendet werden, da die konkreten Werte für die Fehlernummern implementierungs-abhängig und somit nicht portabel sind. Alle in diesem Abschnitt angeführten Werte sind eindeutig. Die (implementierungs-abhängigen) Werte für alle diese Namen sind in der Include-Datei *<errno.h>* definiert.

[E2BIG] Argumentliste zu lang.
Die Summe der Bytes, die von der Argument- und Umgebungsliste des neuen Prozeßabbilds verwendet werden, ist größer als die systemabhängige Grenze {ARG_MAX}.

[EACCES] Zugriff verweigert.
Es wurde der Versuch unternommen, auf eine Art und Weise auf eine Datei zuzugreifen, die von deren Zugriffsberechtigungen verboten wird.

[EAGAIN] Betriebsmittel zeitweilig nicht verfügbar.
Dies ist eine zeitweilige Fehlerbedingung und spätere Aufrufe derselben Routine können normal beendet werden.

[EBADF] Ungültige Dateikennzahl.
Ein Argument für eine Dateikennzahl liegt außerhalb des zulässigen Bereichs, verweist nicht auf eine offene Datei oder ein Leseversuch wurde für eine nur zum Schreiben geöffnete Datei vorgenommen, bzw. umgekehrt.

[EBUSY] Betriebsmittel beschäftigt.
Es wurde der Versuch unternommen, ein Betriebsmittel zu verwenden, das zur Zeit nicht verfügbar ist, weil es von einem anderen Prozeß derart belegt ist, daß dies zu einem Konflikt mit der Anforderung des aktuellen Prozesses führen würde.

[ECHILD] Kein Sohnprozeß
Die Funktion *wait()* bzw. *waitpid()* wurde von einem Prozeß aufgerufen, der gar keine Sohnprozesse besitzt, oder der bereits auf alle Sohnprozesse gewartet hat.

[EDEADLK] Verklemmung würde eintreten.
Es wurde der Versuch unternommen, ein Betriebsmittel zu sperren und diese Sperre hätte eine Verklemmung verursacht.

[EDOM] Bereichsfehler.
Ein Eingabe-Argument liegt außerhalb des Definitionsbereichs einer mathematischen Funktion (definiert in *Draft ANSI X3.159 Programming Language C*).

[EEXIST] Datei existiert.
Eine existierende Datei wurde in einem ungültigen Zusammenhang verwendet, z.B. als Name eines neuen Verweises in der Funktion *link*().

[EFBIG] Datei zu groß.
Die Größe einer Datei würde die maximale Dateigröße überschreiten.

[EIDRM] Bezeichner wurde entfernt.
Dieser Fehler wird während der Interprozeßkommunikation geliefert, wenn ein Bezeichner aus dem System entfernt wurde.

[EINTR] Unterbrochener Funktionsaufruf.
Während der Ausführung einer unterbrechbaren Funktion wurde ein Signal durch den Prozeß abgefangen. Wenn die Signalbehandlungsfunktion ein normales *return* ausführt, dann kann die unterbrochene Funktion diese Bedingung liefern. (Siehe auch *<signal.h>*.)

[EINVAL] Ungültiges Argument.
Es wurde ein ungültiges Argument angegeben, z.B. ein undefiniertes Signal für die Funktion *signal()* oder für die Funktion *kill()*.

[EIO] Ein- oder Ausgabefehler.
Es ist ein physikalischer Ein- oder Ausgabefehler aufgetreten. Dieser Fehler kann auch von einer nachfolgenden Operation für dieselbe Dateikennzahl gemeldet werden. Jede andere, einen Fehler verursachende Operation kann dafür sorgen, daß die Fehleranzeige für [EIO] verloren geht.

[EISDIR] Ist ein Dateiverzeichnis.
Es wurde der Versuch unternommen, ein Dateiverzeichnis zum Schreiben zu öffnen.

[EMFILE] Zu viele offene Dateien.
Es wurde der Versuch unternommen, mehr als die maximale Anzahl von {OPEN_MAX} erlaubten Dateikennzahlen für diesen Prozeß zu öffnen.

[EMLINK] Zu viele Verweise.
Es wurde der Versuch unternommen, den Verweiszähler einer einzelnen Datei größer als {LINK_MAX} zu setzen.

[ENAMETOOLONG] Dateiname zu lang.
Die Länge eines Pfadnamens überschreitet {PATH_MAX} oder eine Pfadnamen-Komponente ist länger als {NAME_MAX} und {_POSIX_NO_TRUNC} ist für diese Datei aktiv.

[ENFILE] Zu viele Dateien im System offen.
Es sind zur Zeit zu viele Dateien im System offen. Das System hat eine vordefinierte Grenze für die gleichzeitig offenen Dateien erreicht und kann zeitweise keine weiteren Anforderungen zum Öffnen weiterer Dateien annehmen.

[ENODEV] Kein Gerät.
Es wurde der Versuch unternommen, eine ungültige Funktion für ein Gerät anzuwenden, zum Beispiel der Versuch, von einem Gerät zu lesen, auf das nur geschrieben werden kann, wie z.B. ein Drucker.

[ENOENT] Keine Datei oder Dateiverzeichnis.
Eine Komponente eines angegebenen Pfadnamens existiert nicht oder der Pfadname ist die leere Zeichenkette.

[ENOEXEC] Fehler im Ausführungsformat.
Es wurde versucht, eine Datei auszuführen, die zwar über die entsprechenden Zugriffrechte verfügt, aber nicht von dem Format ist, das bei ausführbaren Dateien benötigt wird.

[ENOLCK] Keine Sperren verfügbar.
Eine system-abhängige Grenze für die Anzahl der gleichzeitigen Datei- und Satzsperren wurde erreicht und es sind zur Zeit keine weiteren mehr verfügbar.

[ENOMEM] Nicht genügend Speicherplatz.
Das Speicherabbild des neuen Prozesses benötigt mehr Speicherplatz, als verfügbar ist.

[ENOMSG] Keine Nachricht des geforderten Typs.
Die Nachrichtenwarteschlange für die Interprozeßkommunikation enthält keine Nachricht des geforderten Typs.

[ENOSPC] Kein Platz mehr auf Gerät.
Während der Ausführung der Funktion *write()* für eine normale Datei oder bei der Erweiterung eines Dateiverzeichnisses ist kein weiterer Platz mehr auf dem Gerät verfügbar.

[ENOSYS] Funktion nicht implementiert.
Es wurde der Versuch unternommen, eine Funktion zu verwenden, die unter der aktuellen Implementierung nicht verfügbar ist. Dieser Fehler kann für die in diesem Handbuch beschriebenen Funktionen nur dann auftreten, wenn die Anwendung auf ein anderes X/Open-kompatibles System portiert werden soll.

[ENOTDIR] Kein Dateiverzeichnis.
Eine Komponente des angegebenen Pfadnamens existiert, ist aber kein Dateiverzeichnis, obwohl ein Dateiverzeichnis erwartet wurde.

[ENOTEMPTY] Dateiverzeichnis nicht leer.
Es wurde ein Dateiverzeichnis angegeben, das außer . und .. noch weitere Einträge enthält, obwohl ein leeres Dateiverzeichnis erwartet wurde.

[ENOTTY] Ungültige Ein-/Ausgabe-Kontrolloperation.
Eine Kontrollfunktion wurde für eine Datei oder eine Gerätedatei aufgerufen, für die diese Operation nicht erlaubt ist.

[ENXIO] Kein solches Gerät oder keine solche Adresse.
Die Ein- oder Ausgabe auf eine Gerätedatei erfolgt auf ein Gerät, das nicht existiert oder die Anforderung liegt außerhalb der Möglichkeiten dieses Geräts. Dieser Fehler kann zum Beispiel auch dann auftreten, wenn ein Bandlaufwerk nicht On-Line geschaltet ist.

[EPERM] Operation nicht erlaubt.
Es wurde der Versuch unternommen, eine Operation auszuführen, die Prozessen mit besonderen Rechten oder dem Eigentümer einer Datei oder eines anderen Betriebsmittels vorbehalten ist.

[EPIPE] Pipe abgebrochen.
Es wurde auf eine Pipe oder FIFO geschrieben, von der kein Prozeß Daten liest.

[ERANGE] Ergebnis zu groß.
Das Ergebnis der Funktion ist zu groß, um in den verfügbaren Platz zu passen (definiert in *Draft ANSI X3.159 Programming Language C*).

[EROFS] Dateisystem nur zum Lesen eingehängt.
Es wurde der Versuch unternommen, eine Datei oder ein Dateiverzeichnis in einem nur zum Lesen eingehängten Dateisystem zu ändern.

[ESPIPE] Positionierung nicht erlaubt.
Die Funktion *lseek*() wurde auf eine Pipe oder FIFO angewendet.

[ESRCH] Kein solcher Prozeß.
Zur angegebenen Prozeßnummer kann kein entsprechender Prozeß gefunden werden.

[ETXTBSY] Textdatei aktiv.
Es wurde der Versuch unternommmen, eine reine Prozedur auszuführen, die aktuell zum Schreiben geöffnet ist, oder der Versuch, eine reine Prozedur zu schreiben, die aktuell ausgeführt wird.

[EXDEV] Ungültiger Verweis.
Es wurde versucht, einen Verweis auf eine Datei in einem anderen Dateisystem einzurichten.

Die bisher aufgeführten Fehlernummern sind im X/Open-Standard definiert. Zusätzlich zu diesen Fehlernummern definiert SINIX noch die folgende Fehlernummer:

[EFAULT] Ungültige Adresse.
Dieser Fehler tritt auf, wenn ein Programm auf Daten außerhalb des zulässigen Adreßraums verweist. Es kann jedoch nicht garantiert werden, daß dieser Fehler immer zuverlässig erkannt wird. Wird dieser Fehler nicht erkannt, so kann dies dazu führen, daß ein Signal generiert und an den Prozeß gesendet wird, um die Adreß-Verletzung anzuzeigen.

2.3 Signale

In der Version V5.22 des C-Entwicklungssystems CES wurden eine Reihe neuer Funktionen zur Signalbehandlung aufgenommen. Diese Funktionen sind im X/Open-Standard (Ausgabe 3) definiert und bieten eine verbesserte Behandlung von Signalen gegenüber der bisher schon verfügbaren Funktion *signal()*.

Es handelt sich dabei um die folgenden Fuktionen:

sigaction()	*sigaddset()*	*sigdelset()*
sigemptyset()	*sgfillset()*	*sigismember()*
siglongjmp()	*sigpending()*	*sigprocmask()*
sigsetjmp()	*sigsuspend()*	

Diese Funktionen erlauben es, Signale abzufangen bzw. Signalbehandlungsfunktionen zu untersuchen, eine Signalmenge während der Signalbehandlung für ein Signal zu blockieren, anstehende Signale zu ermitteln und nichtlokale Sprünge im Zusammenhang mit einer Signalbehandlung auszuführen.

Anwendungen, die Signale behandeln, sollten die neue Funktion *sigaction()* anstelle der Funktion *signal()* verwenden, da sich durch diese Funktion zusammen mit den übrigen neuen Funktionen eine wesentlich verbesserte und komfortabler Signalbehandlung ergibt.

Aus Portierbarkeitsgründen sollten nur folgende Signale in Anwendungen abgefangen oder ignoriert werden:

Signal	Beschreibung
SIGHUP	Verbindung zu einer Datensichtstation ist unterbrochen
SIGINT	unterbrechen (durch Interrupt-Taste DEL)
SIGQUIT	abbrechen (durch Quit-Taste CTRL)
SIGILL	unzulässiger Befehl (wird nach dem Abfangen nicht zurückgesetzt)
SIGTRAP	Unterbrechung bei *ptrace* (wird nach dem Abfangen nicht zurückgesetzt)
SIGFPE	Fehler bei Gleitkommaoperation
SIGKILL	unbedingter Prozeßabbruch (kann weder abgefangen noch ignoriert werden)
SIGSYS	ungültiges Argument für einen Systemaufruf oder ungültiger Systemaufruf
SIGPIPE	Ausgabe auf eine Pipe, deren Leseseite geschlossen ist
SIGALRM	Alarmuhr abgelaufen
SIGTERM	Prozeßbeendigung bei *kill()*

SIGCHLD	Beenden eines Sohnprozesses. Defaultbehandlung ist Ignorieren des Signals (SIG_DFL)
SIGSEGV	Adreßfehler wegen unerlaubten Segmentzugriffs
SIGUSR1	vom Benutzer verwendbares Signal
SIGUSR2	vom Benutzer verwendbares Signal

2.4 Allgemeine Terminalschnittstelle

Der Abschnitt *Allgemeine Terminalschnittstelle* in diesem Handbuch ersetzt den Abschnitt *termio*(7) aus der Version V5.22 des CES. In der neuen Form wird durch den X/Open-Standard die Terminalschnittstelle jetzt über die Include-Datei *<termios.h>* und nicht mehr über *<sys/termio.h>* realisiert. Außerdem stehen eine Reihe von Funktionen zur Verfügung, die den *ioctl()*-Systemaufruf ablösen. Daher wird in diesem Abschnitt nicht mehr auf *ioctl()* eingegangen.

Alle diejenigen Teile der Beschreibung, die nach dem X/Open-Standard (Ausgabe 3) nicht mehr zur Terminalschnittstelle gehören, werden in diesem Handbuch nur noch kurz unter *ioctl()* bzw. unter *<sys/termio.h>* angesprochen.

Die in diesem Abschnitt verwendeten Begriffe werden im *Glossar* in Abschnitt 1.2 definiert, soweit diese Begriffe auch in den Funktionsbeschreibungen verwendet werden.

2.4.1 Schnittstellen-Eigenschaften

Beschreibung
Dieser Abschnitt beschreibt eine allgemeine Terminalschnittstelle, die zur Kontrolle serieller Kommunikations-Schnittstellen angeboten wird. Dies sind lokal angeschlossene, asynchrone Leitungen. SINIX unterstützt keine synchronen Leitungen; asynchrone Leitungen über Netze werden teilweise unterstützt (hardware-abhängig). In diesen Fällen werden die Schnittstellen nicht komplett wie beschrieben unterstützt.

Öffnen einer Gerätedatei für eine Datensichtstationen
Wenn eine Gerätedatei für eine Datensichtstation geöffnet wird, dann wartet der Prozeß normalerweise solange, bis eine Verbindung hergestellt wurde. (bei RS232 wird gewartet bis die Schnittstellensignale gesetzt sind; bei SS97 wird bei open() nicht gewartet) In der Praxis öffnen Anwendungen solche Dateien nur sehr selten; diese Dateien werden von speziellen Programmen geöffnet und werden dann zur Standardeingabe, Standardausgabe und Standardfehlerausgabe von Anwendungen.

Wie unter *open()* beschrieben, bewirkt das Öffnen einer Gerätedatei für eine Datensichtstation ohne gesetztes O_NONBLOCK-Bit, daß der Prozeß blockiert, bis die Datensichtstation bereit ist. Wenn der CLOCAL-Modus nicht eingeschaltet ist, dann bedeutet dies, daß gewartet wird, bis eine Verbindung aufgebaut ist. Wenn der CLOCAL-Modus für die Datensichtstation eingeschaltet oder das Bit O_NONBLOCK beim Aufruf von *open()* angegeben ist, dann liefert die Funktion *open()* eine Dateikennzahl, ohne auf den Aufbau einer Verbindung zu warten.

Prozeßgruppen

Einer Datensichtstation kann eine Vordergrund-Prozeßgruppe zugeordnet sein. Diese Vordergrund-Prozeßgruppe spielt eine besondere Rolle bei der Behandlung von Eingabezeichen, die Signale erzeugen, wie dies im Abschnitt *Sonderzeichen* weiter unten behandelt wird.

Die Vordergrund-Prozeßgruppe einer Datensichtstation kann von einem Prozeß gesetzt oder abgefragt werden, wenn die in diesem Abschnitt angegebenen Anforderungen hinsichtlich der Zugriffsrechte erfüllt sind; siehe auch *tcgetpgrp()* und *tcsetpgrp()*. Die Terminalschnittstelle hilft bei dieser Zuteilung, indem sie den Zugriff auf die Datensichtstation für solche Prozesse einschränkt, die nicht in der aktuellen Prozeßgruppe sind; siehe dazu auch den Abschnitt *Zugriffskontrolle für Datensichtstationen*.

Das kontrollierende Terminal

Eine Datensichtstation kann zu einem Prozeß als sein kontrollierendes Terminal gehören. Jeder Prozeß einer Sitzung, der ein kontrollierendes Terminal besitzt, besitzt dasselbe kontrollierende Terminal. Eine Datensichtstation kann für höchstens eine Sitzung das kontrollierende Terminal sein. Das kontrollierende Terminal wird vom Sitzungsführer reserviert. Wenn ein Sitzungsführer, der kein kontrollierendes Terminal besitzt, die Gerätedatei einer Datensichtstation ohne gesetztes O_NOCTTY-Bit öffnet, die noch nicht einer Sitzung zugeordnet ist (siehe auch *open()*), dann kann diese Datensichtstation das kontrollierende Terminal des Sitzungsführers werden. Wenn ein Prozeß, der kein Sitzungsführer ist, die Gerätedatei für eine Datensichtstation öffnet, oder wenn die Option O_NOCTTY beim Aufruf von *open()* verwendet wird, dann wird diese Datensichtstation nicht zum kontrollierenden Terminal für den Prozeß. Wenn ein kontrollierendes Terminal einer Sitzung zugeordnet wird, dann wird dessen Vordergrund-Prozeßgruppe gleich der Prozeßgruppe des Sitzungsführers gesetzt.

Das kontrollierende Terminal erbt einen Sohnprozeß von seinem Vaterprozeß. Ein Prozeß gibt sein kontrollierendes Terminal auf, wenn er eine neue Sitzung durch die Funktion *setsid()* erzeugt, oder wenn alle Dateikennzahlen, die dem kontrollierenden Terminal zugeordnet waren, geschlossen wurden.

Wenn ein kontrollierender Prozeß beendet wird, dann wird das kontrollierende Terminal von der aktuellen Sitzung gelöst, was einem neuen Sitzungsführer erlaubt, dieses für sich zu reservieren. Nachfolgende Zugriffe auf diese Datensichtstation durch andere Prozesse aus der früheren Sitzung können verweigert werden, wobei Versuche, auf die Datensichtstation zuzugreifen behandelt werden, als sei ein Verbindungsabbruch bei einem Modem festgestellt worden.

Zugriffskontrolle für Datensichtstationen

Wenn ein Prozeß in der Vordergrund-Prozeßgruppe seines kontrollierenden Terminals ist, dann ist ihm das Lesen von dieser Datensichtstation erlaubt, so wie dies unter *Eingabeverarbeitung und Lesen von Daten* beschrieben ist. Für die Implementierungen, die Auftragskontrolle unterstützen, verursacht jeder Versuch eines Prozesses aus einer Hintergrund-Prozeßgruppe, von seinem kontrollierenden Terminal zu lesen, daß seiner Prozeßgruppe das Signal SIGTTIN gesendet wird, sofern nicht einer der folgenden Fälle zutrifft:

- wenn der lesende Prozeß das Signal SIGTTIN ignoriert oder blockiert,
- oder wenn die Prozeßgruppe des lesenden Prozesses verwaist ist,

dann liefert die Funktion *read()* das Ergebnis –1, wobei *errno* gleich [EIO] gesetzt ist und kein Signal gesendet wird.

Die Standard-Aktion für die Behandlung des Signals SIGTTIN ist, den Prozeß anzuhalten, dem dieses Signal gesendet wird. Siehe auch *<signal.h>*.

Wenn ein Prozeß in der Vordergrund-Prozeßgruppe seines kontrollierenden Terminals ist, dann sind Schreiboperationen erlaubt, wie dies im Abschnitt *Schreiben von Daten und Ausgabeverarbeitung* beschrieben ist. Versuche eines Prozesses aus einer Hintergrund-Prozeßgruppe auf sein kontrollierendes Terminal zu schreiben, verursachen, daß der Prozeßgruppe das Signal SIGTTOU gesendet wird, sofern nicht einer der folgenden Spezialfälle gegeben ist:

- Wenn TOSTOP nicht gesetzt ist, oder wenn TOSTOP gesetzt ist und der Prozeß das Signal SIGTTOU ignoriert oder blockiert, dann darf der Prozeß auf die Datensichtstation schreiben und das Signal SIGTTOU wird nicht gesendet.
- Wenn TOSTOP gesetzt, die Prozeßgruppe des schreibenden Prozesses verwaist ist und der schreibende Prozeß das Signal SIGTTOU nicht blockiert, dann liefert die Funktion *write()* das Ergebnis –1, wobei *errno* gleich [EIO] gesetzt ist und kein Signal gesendet wird.

Bestimmte Aufrufe von Funktionen, die Parameter der Datensichtstation setzen, werden auf dieselbe Art behandelt wie Aufrufe von *write()*, außer daß TOSTOP ignoriert wird; d.h. deren Wirkung ist dieselbe wie die eines Schreibversuchs auf die Datensichtstation wenn TOSTOP gesetzt ist (siehe auch Abschnitt *Lokalmodi*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()* und *tcsetattr()*).

Eingabeverarbeitung und Lesen von Daten

Eine einer Terminal-Gerätedatei zugeordnete Datensichtstation kann im Vollduplexbetrieb arbeiten, so daß es jederzeit möglich ist, Zeichen einzugeben, auch bei laufender Ausgabe. Jeder Gerätedatei für eine Datensichtstation ist ein *Eingabepuffer* zugeordnet, in den die eingehenden Daten durch das System gespeichert werden, bevor sie vom Prozeß gelesen werden können.

Die Eingabe geht verloren, wenn die Eingabepuffer des Systems voll sind oder wenn eine Eingabezeile die zulässige Höchstzahl - z.Zt. {MAX_INPUT} - für die Eingabe von Zeichen überschreitet. Bei Erreichen dieser Grenze kann es passieren, daß alle für diesen Kanal gespeicherten Zeichen ohne Warnung weggeworfen werden.

Es sind zwei generelle Arten von Eingabeverarbeitung verfügbar, je nachdem, ob die Gerätedatei für die Datensichtstation im *Standard-Eingabemodus* oder im *besonderen Eingabemodus* arbeitet. Diese Modi sind in den Abschnitten *Standard-Eingabeverarbeitung* und *Besondere Eingabeverarbeitung* beschrieben. Zusätzlich werden Eingabezeichen entsprechend der Einstellung der Komponenten *c_iflag* (siehe auch *Eingabemodi*) und *c_lflag* (siehe auch *Lokalmodi*) verarbeitet. Diese Verarbeitung kann das lokale Echo einschließen. Dies bedeutet, daß Eingabezeichen sofort nach ihrem Empfang an die entsprechende Datensichtstation zurückgesendet werden. Dies ist besonders nützlich für Datensichtstationen, die im Vollduplexbetrieb arbeiten.

Wenn das Bit O_NONBLOCK nicht gesetzt ist, dann blockieren Leseanforderungen solange, bis Daten verfügbar sind, oder bis ein Signal eintrifft. Wenn das Bit O_NONBLOCK gesetzt ist, dann wird die Leseanforderung auf eine der nachfolgend beschriebenen drei Arten ohne Warten beendet:

- Sind genügend Daten verfügbar, um die konkrete Anforderung zu erfüllen, dann kehrt die Funktion *read()* erfolgreich zurück und liefert als Ergebnis die Anzahl der gelesenen Bytes.
- Wenn nicht genügend Daten verfügbar sind, um die konkrete Anforderung zu erfüllen, dann kehrt die Funktion *read()* erfolgreich zurück. Dabei hat sie so viele Daten wie möglich gelesen. Sie liefert als Ergebnis die Anzahl der tatsächlich gelesenen Bytes zurück.
- Sind keine Daten verfügbar, dann liefert die Funktion *read()* den Wert -1, wobei *errno* gleich [EAGAIN] gesetzt ist.

Wann Daten verfügbar sind, hängt davon ab, ob die Standard- oder die besondere Eingabeverarbeitung aktiv ist. Die folgenden Abschnitte *Standard-Eingabeverarbeitung* und *Besondere Eingabeverarbeitung* beschreiben jeden dieser Eingabeverarbeitungs-Modi.

Standard-Eingabeverarbeitung

Bei der Standard-Eingabeverarbeitung werden Eingaben von einer Datensichtstation zeilenweise bearbeitet. Eine Zeile wird begrenzt durch ein Neue-Zeile-Zeichen, (ASCII LF) einem Dateiende- oder Zeilenende-Zeichen. Für mehr Informationen zu EOF und EOL siehe auch den Abschnitt *Sonderzeichen*. Dies bedeutet, daß ein lesendes Programm so lange angehalten wird, bis eine vollständige Zeile eingegeben wurde. Ebenso besteht die Eingabe aus maximal einer Zeile, gleichgültig, wie viele Zeichen in dem Leseaufruf angefordert wurden. Es muß jedoch nicht notwendigerweise eine ganze Zeile auf einmal gelesen werden; es kann eine beliebige Anzahl Zeichen in einem Leseaufruf angefordert werden (auch nur 1 Zeichen), ohne daß Daten verloren gehen.

Wenn {MAX_CANON} definiert ist, dann entspricht dieser Wert der maximalen Anzahl von Bytes in einer Zeile. Wenn diese Grenze überschritten wird, dann ist das Verhalten des Systems undefiniert. Wenn {MAX_CANON} nicht definiert ist, dann gibt es keine solche Grenze (siehe auch *pathconf()*).

Die Verarbeitung von ERASE- und KILL-Zeichen geschieht dann, wenn eines der Sonderzeichen ERASE und KILL gelesen wird (siehe Abschnitt *Sonderzeichen*). Die Verarbeitung dieser Zeichen betrifft den Eingabepuffer der noch nicht durch ein Neue-Zeile-Zeichen, (ASCII LF) ein Dateiende- oder ein Zeilenende-Zeichen begrenzt wurde. Diese noch nicht begrenzten Daten bilden die aktuelle Zeile. Das Löschzeichen ERASE löscht das zuletzt eingegebene Zeichen der aktuellen Zeile, sofern ein solches nach dem Zeilenanfang vorhanden ist. Das Löschzeichen KILL entfernt die gesamte aktuelle Eingabezeile, sofern eine solche vorhanden ist. Dabei kann wahlweise die Ausgabe eines neuen Neue-Zeile-Zeichens erfolgen. Die Zeichen ERASE und KILL haben keine Wirkung, wenn sich keine Daten in der aktuellen Zeile befinden. Die Löschzeichen selbst werden nicht im Eingabepuffer abgelegt.

Beide Zeichen wirken unmittelbar bei Betätigen der entsprechenden Taste, unabhängig von eventuell eingegebenen Backspace- oder Tabulatorzeichen. Sie können auch direkt als Konstante eingegeben werden, indem man ihnen das Escape-Zeichen \ voranstellt. Das Escape-Zeichen selbst wird nicht gelesen. Die Löschzeichen können geändert werden.

Besondere Eingabeverarbeitung

Bei der besonderen Eingabeverarbeitung werden die Eingabezeichen nicht zu Zeilen zusammengefaßt und eine Verarbeitung von ERASE- und KILL-Zeichen findet nicht statt. Die Werte der Elemente MIN und TIME des Vektors *c_cc* werden verwendet, um zu entscheiden, wie der Prozeß die Zeichen erhalten soll.

MIN gibt die Mindestanzahl an Zeichen (maximal 255) an, die bei einer erfolgreich ausgeführten Funktion *read()* empfangen werden sollten (d.h. die dann dem Benutzer zurückgeliefert werden). TIME ist ein Timer (eine Zeitüberwachung) auf Zehntelsekunden-Basis für schubweise und geringe Datenübertragungen. Wenn MIN größer als {MAX_INPUT} ist, dann ist nicht festgelegt, wie die Anforderung behandelt wird. Die folgenden Absätze beschreiben die vier möglichen Kombinationen von MIN und TIME sowie ihre Wechselwirkung:

1. **Fall: MIN > 0, TIME > 0**
   In diesem Fall dient TIME als zeichenorientierter Timer und wird nach dem ersten empfangenen Zeichen aktiviert. Bei jedem neuen Zeichen wird TIME zurückgesetzt; sobald ein Zeichen empfangen wird, wird TIME gestartet. Werden MIN Zeichen empfangen, bevor der Timer TIME abgelaufen ist, so wird der Leseauftrag erfüllt. Läuft der Timer TIME ab, bevor MIN Zeichen empfangen wurden, so werden die bis zu diesem Zeitpunkt empfangenen Zeichen an den Benutzer übergeben. Es wird immer mindestens ein Zeichen zurückgeliefert, wenn TIME abläuft, da der Timer erst nach dem Empfang des ersten Zeichens aktiviert wird. In diesem Fall blockiert die Leseoperation solange, bis entweder der MIN- und TIME-Mechanismus durch den Empfang des ersten Bytes aktiviert wird, oder bis ein Signal eintrifft.

2. **Fall: MIN > 0, TIME = 0**
   Da TIME den Wert 0 hat, ist die Zeitüberwachung wirkungslos und nur MIN ist signifikant. In diesem Fall blockiert die Leseoperation solange, bis MIN Zeichen empfangen wurden oder bis ein Signal eintrifft. Ein Programm, das diesen Fall nutzt, um Datensätzen von einer Datensichtstation zu lesen, kann bei einer Leseoperation beliebig lange blockieren (d.h. auch unendlich lange).

3. **Fall: MIN = 0, TIME > 0**
   Da MIN = 0 ist, dient TIME nicht mehr als zeichenorientierter Timer, sondern als Zeitüberwachung für die gesamte Leseoperation, die bei der Bearbeitung des *read()*-Aufrufs aktiviert wird (Standardbehandlung). In diesem Fall wird eine Leseoperation ausgeführt, sobald entweder ein Zeichen empfangen wird oder der Timer TIME abläuft. Wenn innerhalb des Zeitraums von TIME * 0,1 Sekunden nach dem Aufruf von *read()* kein Byte empfangen wird, dann liefert die Funktion *read()* das Ergebnis 0 und hat keine Daten gelesen.

4. **Fall: MIN = 0, TIME = 0**
   In diesem Fall wird sofort die geforderte Anzahl von zu lesenden Zeichen zurückgeliefert, oder, wenn nicht so viele verfügbar sind, die Anzahl der aktuell verfügbaren Zeichen. Es wird nicht auf eine weitere Eingabe gewartet. Sind keine Eingabezeichen verfügbar, dann liefert die Funktion *read()* den Wert 0 als Ergebnis und hat keine Daten gelesen.

Schreiben von Daten und Ausgabeverarbeitung

Wenn ein Prozeß Bytes in eine Gerätedatei für eine Datensichtstation schreibt, dann werden diese Bytes gemäß den Einstellungen in *c_oflag* verarbeitet (siehe Abschnitt *Ausgabemodi*). Das System kann einen Puffer-Mechanismus bieten, der so arbeitet, daß alle Bytes, die ein Aufruf von *write()* geschrieben hat, nach dessen Beendigung zur Übertragung zum jeweiligen Gerät anstehen, aber noch nicht notwendigerweise auch schon vollständig übertragen wurden. Vgl. dazu *write()*, Auswirkungen von *write()* mit gesetztem O_NONBLOCK.

Sonderzeichen

Einigen Zeichen sind bei der Ein- und/oder Ausgabe bestimmte Sonderfunktionen zugeordnet, die nachfolgend beschrieben werden: In den Fällen, in denen die Zuordnung von Zeichen und Funktion nicht verändert werden darf, ist das entsprechende Zeichen in Klammern angegeben:

INTR
: Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ISIG gesetzt ist. Es erzeugt ein Unterbrechungssignal (interrupt) SIGINT, das an alle Prozesse in der Vordergrund-Prozeßgruppe der Datensichtstation abgesetzt wird. Wenn das Bit ISIG gesetzt ist, dann wird das Zeichen nach der Verarbeitung verworfen. Damit werden im Normalfall alle diese Prozesse abgebrochen, man kann jedoch Vorkehrungen treffen, daß das Signal ignoriert wird oder ein Sprung an eine vorher vereinbarte Adresse erfolgt (siehe *sigaction()* bzw. *signal()*).

QUIT
: Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ISIG gesetzt ist. Es erzeugt ein Signal zum Beenden (quit) SIGQUIT für alle Prozesse in der Vordergrund-Prozeßgruppe, die der Datensichtstation zugeordnet ist. Wenn ISIG gesetzt ist, dann wird das Zeichen QUIT nach der Verarbeitung verworfen. Es wird fast genauso behandelt wie das Unterbrechungssignal SIGINT, mit einer Ausnahme: hat der empfangende Prozeß keine anderen Vorkehrungen getroffen, so wird er nicht nur abgebrochen, sondern es wird auch ein Speicherabzug (core) erzeugt (siehe auch *sigaction()*).

ERASE
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist. Es löscht das vorhergehende Zeichen, allerdings nicht über den Zeilenanfang — d.h. ein NL-, EOF- oder EOL-Zeichen — hinaus (vgl. *Standard-Eingabeverarbeitung*). Wenn ICANON gesetzt ist, dann wird das Zeichen ERASE nach der Verarbeitung verworfen.

KILL
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist. Es löscht die gesamte Zeile vom letzten NL-, EOF- oder EOL-Zeichen ab. Wenn ICANON gesetzt ist, dann wird das Zeichen KILL nach der Verarbeitung verworfen.

EOF
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist. Beim Empfang von EOF werden alle noch nicht gelesenen Zeichen sofort an das Programm übergeben, ohne auf ein NL-Zeichen zu warten; das EOF-Zeichen wird verworfen. Sind keine Zeichen vorhanden, d.h. das EOF-Zeichen steht am Zeilenanfang, so liefert *read()* den Wert 0 zurück. Das Ergebnis 0 bei einer Leseoperation ist die Standardanzeige für das Dateiende. Wenn ICANON gesetzt ist, dann wird das Zeichen EOF nach der Verarbeitung verworfen.

NL (ASCII LF, Code 10)
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist. NL ist das normale Zeilen-Begrenzungszeichen '\n'. Es kann nicht geändert werden.

EOL
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist. EOL ist ein zusätzliches Zeilen-Begrenzungszeichen und hat dieselbe Funktion wie NL. Es wird normalerweise nicht verwendet.

STOP (ASCII DC3, Code 19, Ctrl-S)
Sonderzeichen für die Eingabe und für die Ausgabe, das erkannt wird, wenn eines der Bits IXON (für die Ausgabe) oder IXOFF (für die Eingabe) gesetzt ist. STOP kann dazu verwendet werden, eine Ausgabe vorübergehend anzuhalten. Damit kann man an Datensichtstationen verhindern, daß die Ausgabe vom Bildschirm verschwindet, bevor man sie lesen konnte. Wenn IXON gesetzt ist, dann wird das Zeichen STOP nach der Verarbeitung verworfen. Solange die Ausgabe angehalten wird, werden weitere STOP-Zeichen ignoriert und nicht gelesen. Das Zeichen STOP kann nicht geändert und nicht entwertet werden.

START (ASCII DC1, Code 17, Ctrl-Q)
Sonderzeichen für die Eingabe und für die Ausgabe, das erkannt wird, wenn eines der Bits IXON (für die Eingabe) oder IXOFF (für die Ausgabe) gesetzt ist. Das Zeichen START dient dazu, eine mit dem STOP-Zeichen angehaltene Ausgabe fortzusetzen. Solange die Ausgabe läuft, werden nachfolgende START-Zeichen ignoriert und nicht gelesen. Wenn IXON gesetzt ist, dann wird das Zeichen START nach der Verarbeitung verworfen. Das Zeichen START kann nicht geändert und nicht entwertet werden.

CR (ASCII CR, Code 13)
Sonderzeichen für die Eingabe, das erkannt wird, wenn das Bit ICANON gesetzt ist; es entspricht dem Zeichen '\r'. Wenn ICANON und ICRNL gesetzt sind und IGNCR nicht, dann wird dieses Zeichen in das Zeichen NL umgesetzt und hat dieselbe Wirkung wie das Zeichen NL. Das Zeichen CR kann nicht geändert werden.

SUSP
Wenn ein X/Open-kompatibles System Auftragskontrolle unterstützt (siehe auch Abschnitt *Steuerzeichen*), dann wird das Sonderzeichen SUSP bei der Eingabe erkannt. Wenn das Bit ISIG gesetzt ist, dann verursacht der Empfang des Zeichens SUSP, daß das Signal SIGTSTP an alle Prozesse in der Vordergrund-Prozeßgruppe, die der Datensichtstation zugeordnet ist, gesendet wird. Dann wird das Zeichen ebenfalls nach der Verarbeitung verworfen. Dieses Zeichen hat unter SINIX keine Wirkung, da hier die Auftragskontrolle nicht unterstützt wird.

Die Werte für INTR, QUIT, ERASE, KILL, EOF, EOL und SUSP (nur für Auftragskontrolle) können vom Benutzer geändert werden.

Wenn für die Gerätedatei der Datensichtstation {_POSIX_VDISABLE} aktiv ist, dann können die Sonderzeichen-Funktionen für die änderbaren Sonderzeichen einzeln ausgeschaltet werden.

Wenn zwei oder mehr Sonderzeichen denselben Wert haben, dann ist die Funktion undefiniert, die bei Empfang dieses Zeichens ausgeführt wird.

Die ERASE-, KILL- und EOF-Zeichen können durch ein vorangestelltes '\' (Escape-)Zeichen entwertet werden; in diesem Fall wird die ihnen zugeordnete Funktion nicht ausgeführt.

Verbindungsabbruch

Beim Verschwinden des Carrier-Signals (modem disconnect) an der Schnittstelle für ein kontrollierendes Terminal, wird, wenn in *c_cflag* CLOCAL nicht gesetzt ist (siehe *Steuermodi*), das Signal für den Verbindungsabbruch SIGHUP an den kontrollierenden Prozeß gesendet, der dieser Datensichtstation zugeordnet ist. Dadurch wird der kontrollierende Prozeß abgebrochen, sofern keine anderen Vorkehrungen getroffen wurden (siehe *exit()*). Alle nachfolgenden Leseoperationen von dieser Datensichtstation liefern dann die Anzeige für Dateiende. Damit können Prozesse, die Eingaben von einer Datensichtstation lesen und auf Dateiende prüfen, nach einem Verbindungsabbruch entsprechend beendet werden. Jede nachfolgende Schreiboperation mit *write()* auf diese Datensichtstation liefert das Ergebnis -1 und *errno* ist dann gleich [EIO] gesetzt, bis die Datei geschlossen wird.

Schließen einer Gerätedatei für eine Datensichtstation

Wenn der letzte Prozeß eine Gerätedatei für eine Datensichtstation schließt, dann werden alle noch anstehenden Ausgaben an dieses Gerät gesendet und alle noch nicht gelesenen Eingaben verworfen. Wenn HUPCL in den Steuermodi gesetzt ist und die Kommunikations-Schnittstelle eine Verbindungsabbruch-Funktion unterstützt, dann führt die Terminalschnittstelle einen Verbindungsabbruch aus.

2.4.2 Einstellbare Parameter

Die Struktur *termios*

Programme, die Ein- und Ausgabe-Kennzeichen für Datensichtstationen kontrollieren müssen, können dies über die Struktur *termios*, die in der Include-Datei *<termios.h>* definiert ist. Zu den Komponenten dieser Struktur gehören:

Komponenten-Typ	Vektor-Größe	Komponenten-Name	Beschreibung
tcflag_t		c_iflag	Eingabemodi
tcflag_t		c_oflag	Ausgabemodi
tcflag_t		c_cflag	Steuermodi
tcflag_t		c_lflag	Lokalmodi
cc_t	NCCS	c_cc[]	Sonderzeichen

Die Datentypen *tcflag_t* und *cc_t* sind in der Include-Datei *<termios.h>* definiert. Sie sind dort als *unsigned definiert.*

Eingabemodi

Die Komponente *c_iflag* beschreibt die grundlegende Eingabesteuerung der Datensichtstation:

Masken-Name	Beschreibung
BRKINT	sende Signal SIGINT bei "break"
ICRNL	CR bei Eingabe in NL umwandeln
IGNBRK	"break" ignorieren
IGNCR	CR ignorieren
IGNPAR	Zeichen mit Paritätsfehler ignorieren
INLCR	NL bei Eingabe in CR umwandeln
INPCK	Paritätsprüfung für Eingabe aktivieren
ISTRIP	8. Bit des Eingabezeichens löschen
IXOFF	START/STOP-Eingabesteuerung aktivieren
IXON	START/STOP-Ausgabesteuerung aktivieren
PARMRK	Paritätsfehler markieren
IUCLC	Bei Eingabe Groß- in Kleinbuchstaben umwandeln
IXANY	Fortsetzung der Ausgabe durch beliebiges Eingabezeichen

Im Zusammenhang mit der asynchronen Datenübertragung über eine serielle Schnittstelle ist ein "break" als eine Folge von 0-Bits definiert, die länger dauert, als für die Übertragung eines Bytes notwendig ist. Die gesamte Folge von 0-Bits wird als ein einziges "break" interpretiert, auch wenn es sich dabei um eine Folge handelt, die mehrere Bytes lang ist. In anderen Zusammenhängen als der asynchronen seriellen Datenübertragung ist die Bedeutung eines "break" nicht festgelegt.

Bei gesetztem IGNBRK wird ein in der Eingabe auftretendes "break" ignoriert, d.h. nicht in den Eingabepuffer eingetragen und deshalb von keinem Prozeß gelesen. Bei gesetztem BRKINT dagegen erzeugt ein "break" ein einzelnes Unterbrechungssignal SIGINT und sowohl die Ein- als auch die Ausgabepuffer werden gelöscht. Wenn weder IGNBRK noch BRKINT gesetzt ist, dann wird ein "break" als einzelnes Zeichen '\0' gelesen, wenn PARMRK gesetzt ist, dann als '\377', '\0', '\0'.

Ist IGNPAR gesetzt, dann wird jedes Byte mit einem Zeichen- oder Paritätsfehler ungleich einem "break" ignoriert.

Wenn PARMRK gesetzt und IGNPAR nicht gesetzt ist, dann wird jedes Byte mit einem Rahmen- oder Paritätsfehler, welches ungleich einem "break" ist, als eine Folge von drei Zeichen weitergegeben:'\377','\0' und *X*, wobei '\377' und '\0' ein zwei Byte langes Kennzeichen für jede dieser Sequenzen ist und *X* dem fehlerhaften Zeichen entspricht. Um Zweifelsfälle auszuschließen wird, wenn ISTRIP nicht gesetzt ist, ein gültiges Zeichen '\377' als '\377','\377' an die Anwendung ausgeliefert. Wenn weder PARMRK noch IGNPAR gesetzt ist, dann wird ein Rahmen- oder Paritätsfehler, der ungleich einem "break" ist, als ein einzelnes Zeichen '\0' an die Anwendung weitergegeben.

Bei gesetztem INPCK wird die Paritätsprüfung bei der Eingabe aktiviert. Bei nicht gesetztem INPCK wird die eingabeseitige Paritätsprüfung deaktiviert. Damit kann das Paritätsbit bei der Ausgabe ohne Berücksichtigung von eventuellen Paritätsfehlern bei der Eingabe erzeugt werden.

Hinweis
Ob die Paritätsprüfung bei der Eingabe aktiviert oder deaktiviert ist, hängt nicht davon ab, ob die Paritäts-Erkennung aktiviert oder deaktiviert ist (siehe auch Abschnitt *Steuermodi*). Wenn die Paritäts-Erkennung aktiviert, die Paritätsprüfung bei der Eingabe jedoch deaktiviert ist, dann erkennt zwar die Hardware, mit der die Datensichtstation verbunden ist, das Paritätsbit, aber die Gerätedatei der Datensichtstation überprüft nicht, ob dieses Bit korrekt gesetzt ist.

Bei gesetztem ISTRIP werden gültige Eingabezeichen zunächst auf 7 Bits verkürzt. Ist ISTRIP nicht gesetzt, so werden alle 8 Bits bearbeitet.

Bei gesetztem INLCR wird ein empfangenes NL-Zeichen (Zeilenvorschub) in ein CR-Zeichen (Wagenrücklauf) umgewandelt. Bei gesetztem IGNCR wird ein empfangenes CR-Zeichen ignoriert (nicht gelesen). Ist dagegen IGNCR nicht gesetzt und ICRNL gesetzt, so wird ein empfangenes CR-Zeichen in ein NL-Zeichen umgewandelt.

Bei gesetztem IXON wird die Ausgabesteuerung mit START/STOP aktiviert. Bei Empfang eines STOP-Zeichens wird die Ausgabe angehalten und bei Empfang eines START-Zeichens fortgesetzt. Die Steuerzeichen für START und STOP werden bei einer Leseoperation nicht gelesen, führen jedoch die Funktionen der Flußsteuerung aus, wenn IXON gesetzt ist. Ist IXON nicht gesetzt, dann werden START- und STOP-Zeichen gelesen. Bei gesetztem IXANY wird die angehaltene Ausgabe durch die Eingabe eines beliebigen Zeichens fortgesetzt.

Bei gesetztem IXOFF ist die Eingabe-Flußsteuerung aktiviert. Das System überträgt STOP-Zeichen, um die Datensichtstation zu veranlassen, keine weiteren Daten mehr zu übertragen, wenn dies notwendig ist, um ein Überlaufen des Eingabepuffers zu verhindern (nicht mehr als {MAX_INPUT} Bytes sind erlaubt). Es überträgt START-Zeichen, um die Datensichtstation zu veranlassen, die Übertragung von Daten wieder aufzunehmen, sobald dies wieder ohne Gefahr eines Überlaufs des Eingabepuffers möglich ist.

Bei gesetztem IUCLC wird ein empfangener Großbuchstabe in den entsprechenden Kleinbuchstaben umgewandelt.

Der Anfangswert für die Eingabemodi ist, daß kein Bit gesetzt ist.

Ausgabemodi

Die Komponente *c_oflag* gibt an, wie die Terminalschnittstelle Ausgaben behandelt:

Masken-Name	Beschreibung
OPOST	Ausgaben nachbearbeiten
OLCUC	Bei Ausgabe Klein- in Großbuchstaben umwandeln
ONLCR	Bei Ausgabe NL in CR-NL umwandeln
OCRNL	Bei Ausgabe CR in NL umwandeln
ONOCR	CR in Spalte 0 nicht ausgeben
ONLRET	NL übernimmt CR-Funktion
OFILL	Füllzeichen für Verzögerung verwenden
OFDEL	Das Füllzeichen ist DEL (sonst NUL)
NLDLY	Zeilenvorschub-(NL-)Verzögerungen auswählen:
NL0	NL-Zeichen Typ 0
NL1	NL-Zeichen Typ 1
CRDLY	Wagenrücklauf-(CR-)Verzögerungen auswählen:
CR0	CR-Verzögerung Typ 0
CR1	CR-Verzögerung Typ 1
CR2	CR-Verzögerung Typ 2
CR3	CR-Verzögerung Typ 3
TABDLY	Horizontaltabulator-Verzögerungen auswählen:
TAB0	Horizontaltabulator-Verzögerung Typ 0
TAB1	Horizontaltabulator-Verzögerung Typ 1
TAB2	Horizontaltabulator-Verzögerung Typ 2
TAB3	Tabulatorexpansion zu Leerzeichen
BSDLY	Backspace-Verzögerungen auswählen:
BS0	Backspace-Verzögerung Typ 0
BS1	Backspace-Verzögerung Typ 1
VTDLY	Vertikaltabulator-Verzögerungen auswählen:
VT0	Vertikaltabulator-Verzögerung Typ 0
VT1	Vertikaltabulator-Verzögerung Typ 1
FFDLY	Seitenvorschub-Verzögerungen auswählen:
FF0	Seitenvorschub-Verzögerung Typ 0
FF1	Seitenvorschub-Verzögerung Typ 1

Wenn OPOST gesetzt ist, dann werden Ausgabedaten gemäß den übrigen Bits von *c_oflag* nachbearbeitet, damit die Textzeilen so verändert werden, daß sie korrekt an der Datensichtstation erscheinen, andernfalls werden die Zeichen ohne Änderung übertragen.

Bei gesetztem OLCUC wird ein Kleinbuchstabe vor der Übertragung in den entsprechenden Großbuchstaben umgewandelt. Diese Funktion wird oft zusammen mit IUCLC bei den Eingabemodi verwendet.

Bei gesetztem ONLCR wird das NL-Zeichen (Zeilenvorschub) als das Zeichenpaar CR-NL (Wagenrücklauf-Zeilenvorschub) übertragen. Bei gesetztem OCRNL wird das CR-Zeichen als NL-Zeichen übertragen. Bei gesetztem ONOCR wird ein CR-Zeichen in Spalte 0 (erste Stelle in der Zeile) nicht übertragen. Bei gesetztem ONLRET wird angenommen, daß das NL-Zeichen die Wagenrücklauf-Funktion übernimmt; der Spaltenzeiger wird auf 0 gesetzt und die spezifischen Wagenrücklauf-Verzögerungen verwendet. Ist ONLRET nicht gesetzt, so wird angenommen, daß das NL-Zeichen nur die Zeilenvorschub-Funktion hat; der Spaltenzeiger bleibt dann unverändert. Der Spaltenzeiger wird ferner auf 0 gesetzt, wenn das CR-Zeichen übertragen wird.

Die Verzögerungs-Bits geben an, für wie lange die Übertragung angehalten wird, damit bestimmte mechanische oder sonstige Bewegungen bei der Übertragung bestimmter Zeichen an die Datensichtstation ausgeführt werden können. In allen Fällen bedeutet 0 'keine Verzögerung'. Bei gesetztem OFILL wird die zeitliche Verzögerung durch die Übertragung von Füllzeichen erreicht. Dies ist bei Datensichtstationen mit hoher Übertragungsgeschwindigkeit nützlich, die nur eine minimale Verzögerung benötigen. Bei gesetztem OFDEL wird DEL als Füllzeichen verwendet, sonst NUL.

Ist eine Seitenvorschub- oder Vertikaltabulator-Verzögerung angegeben, so dauert diese etwa 2 Sekunden.

Eine Zeilenvorschub-Verzögerung dauert etwa 0,10 Sekunden. Bei gesetztem ONLRET werden statt der Zeilenvorschub-Verzögerungen die Wagenrücklauf-Verzögerungen verwendet. Bei gesetztem OFILL werden zwei Füllzeichen übertragen.

Bei den Wagenrücklauf-Verzögerungen ist Typ 1 abhängig von der aktuellen Spaltenposition, Typ 2 dauert etwa 0,10 Sekunden, Typ 3 etwa 0,15 Sekunden. Bei gesetztem OFILL werden bei Typ 1 zwei Füllzeichen übertragen, bei Typ 2 vier.

Bei den Horizontaltabulatoren-Verzögerungen ist Typ 1 abhängig von der aktuellen Spaltenposition, Typ 2 dauert etwa 0,10 Sekunden, Typ 3 gibt an, daß Tabulatoren zu Leerzeichen expandiert werden sollen. Bei gesetztem OFILL werden für jede Verzögerung zwei Füllzeichen übertragen.

Die Backspace-Verzögerung dauert etwa 0,05 Sekunden. Bei gesetztem OFILL wird ein Füllzeichen übertragen.

Die tatsächlichen Verzögerungen hängen von der Leitungsgeschwindigkeit und der Systemauslastung ab.

Der Anfangswert für die Ausgabemodi (Wert von *c_oflag*) nach einem Aufruf von *open()* ist, daß kein Bit gesetzt ist.

Steuermodi

Die Komponente *c_cflag* beschreibt die hardwaremäßige Steuerung der Datensichtstation; nicht alle angegebenen Werte müssen unbedingt von der jeweils zugrundeliegenden Hardware unterstützt werden:

Masken-Name	Beschreibung
CLOCAL	Lokale Leitung (sonst Modemleitung)
CREAD	Empfänger aktivieren
CSIZE	Anzahl der Bits je Byte:
CS5	5 Bits
CS6	6 Bits
CS7	7 Bits
CS8	8 Bits
CSTOPB	2 Stopbits senden (sonst 1)
HUPCL	Bei letztem close() Verbindung abbauen
PARENB	Paritätserkennung und Paritätserzeugung aktivieren
PARODD	Ungerade Parität (sonst gerade)
LOBLK	Schicht-Ausgabe blockieren
CBAUD	Baudrate für Übertragung und Empfang

Zusätzlich werden die Ein- und Ausgabebaudraten in der Struktur *termios* abgespeichert. Die folgenden Werte werden unterstützt:

Name	Beschreibung
B0	Verbindung abbauen (Hang Up)
B50	50 Baud
B75	75 Baud
B110	110 Baud
B134	134.5 Baud
B150	150 Baud
B200	200 Baud
B300	300 Baud
B600	600 Baud
B1200	1200 Baud
B1800	1800 Baud
B2400	2400 Baud
B4800	4800 Baud
B9600	9600 Baud
B19200	19200 Baud
B38400	38400 Baud

Die folgenden Schnittstellen stehen für das Ermitteln und Setzen der Werte für Ein- und Ausgabebaudrate in der Struktur *termios* zur Verfügung: *cfgetispeed()*,*cfgetospeed()*, *cfsetispeed()* und *cfsetospeed()*.

Mit den CBAUD-Bits wird die Übertragungsgeschwindigkeit angegeben. Eine Übertragungsgeschwindigkeit von 0 Baud (B0) bewirkt einen Verbindungabbruch. Bei Angabe von B0 wird das Data-Terminal-Ready-Signal (DTR) in den Zustand AUS geschaltet. Dies führt normalerweise zum Verbindungsabbruch. Wird für eine bestimmte Hardware eine Übertragungsgeschwindigkeit angegeben die für diese Hardware nicht möglich ist, dann wird die angegebene Geschwindigkeit ignoriert.

Mit den CSIZE-Bits wird die Anzahl der Bits je Byte sowohl für die Übertragung als auch für den Empfang angegeben. Darin ist das Paritätsbit, sofern vorhanden, nicht enthalten. Bei gesetztem CSTOPB werden zwei Stopbits verwendet (sonst eins). Bei einer Übertragungsgeschwindigkeit von 110 Baud werden z.B. normalerweise zwei Stopbits verwendet.

Bei gesetztem CREAD wird der Empfänger aktiviert. Ist CREAD nicht gesetzt, werden keine Zeichen empfangen.

Bei gesetztem PARENB wird die Paritätserkennung und die Paritätserzeugung aktiviert, d.h. jedes Zeichen erhält ein Paritätsbit. In diesem Fall gibt das PARODD-Bit an, daß eine ungerade Parität verwendet wird (sonst wird eine gerade Parität verwendet).

Bei gesetztem HUPCL wird die Verbindung abgebaut, wenn der letzte Prozeß, der diese Leitung benutzt, die Verbindung schließt oder sich beendet. Das heißt, das Data-Terminal-Ready-Signal (DTR) wird zurückgesetzt. Dadurch wird die Verbindung abgebrochen.

Bei gesetztem CLOCAL wird angenommen, daß es sich bei der bestehenden Leitung um eine lokale, direkte Verbindung ohne Modemsteuerung handelt. Die Verbindung hängt dann nicht von den Leitungssignalen ab. Ansonsten wird eine Modemsteuerung angenommen und die Melde-Signale werden überwacht.

Unter normalen Umständen wartet ein Aufruf der Funktion *open()* auf das Ende des Verbindungsaufbaus. Wenn jedoch das Bit O_NONBLOCK beim Aufruf von *open()* angegeben wird, oder das Bit CLOCAL gesetzt ist, dann kehrt die Funktion *open()* sofort zurück, ohne auf die Verbindung zu warten.

Wenn das Objekt, für das die Steuermodi gesetzt sind, keine asynchrone serielle Verbindung ist, dann können einige der Modi ignoriert werden; wird z.B. der Versuch unternommen, die Baudrate für eine Netzverbindung zu einer Datensichtstation an einem anderen Rechner zu setzen, dann kann die Baudrate für die Verbindung zwischen der Datensichtstation und dem Rechner, mit dem sie direkt verbunden ist, gesetzt werden oder nicht.

Bei gesetztem LOBLK wird die Ausgabe einer Auftragskontrollschicht (Job-Control-Layer) blockiert, wenn sie nicht die aktuelle Schicht ist. Sonst (LOBLK nicht gesetzt) wird die Ausgabe dieser Schicht zusammen mit den Ausgaben der anderen Schichten an der Datensichtstation angezeigt.

Hinweis
Die Funktion LOBLK ist in SINIX V5.22 nicht implementiert.

Lokalmodi
Die Komponente *c_local* der Struktur wird verwendet, um verschiedene Funktionen zu kontrollieren:

Masken-Name	Beschreibung
ECHO	Echo-Funktion aktivieren
ECHOE	ERASE-Zeichen als BS-SP-BS ausgeben ("Echo") (korrigierender Backspace)
ECHOK	NL-Zeichen nach KILL-Zeichen ausgeben ("Echo")
ECHONL	NL-Zeichen ausgeben ("Echo")
ICANON	Standard-Eingabeverarbeitung aktivieren (zeilenorientierte Eingabe mit Behandlung von ERASE- und KILL-Zeichen)
IEXTEN	(Wird von SINIX nicht unterstützt) Zusätzliche (implementierungs-abhängige) Funktionen aktivieren
ISIG	Signalaktivierung
NOFLSH	Leeren der Ein- und Ausgabepuffer nach INTERRUPT oder QUIT von Tastatur deaktivieren
TOSTOP	Signal SIGTTOU bei Ausgabe für Hintergrund-Prozeßgruppe senden
XCASE	Standardmäßige Darstellung von Groß-/Kleinbuchstaben

Bei gesetztem ECHO werden eingegebene Zeichen so, wie sie empfangen wurden, wieder auf den Bildschirm ausgegeben. Ist ECHO nicht gesetzt, dann werden Eingabezeichen nicht angezeigt.

Sind ICANON und ECHOE gesetzt, so wird das ERASE-Zeichen als die Folge Backspace-Leerzeichen-Backspace zurückgeliefert, wodurch das letzte Zeichen, sofern vorhanden, auf dem Bildschirm gelöscht wird. Ist ECHOE gesetzt und ECHO nicht, so wird das ERASE-Zeichen als ASCII SP BS zurückgeliefert.

Bei gesetztem ECHOK und ICANON wird nach dem KILL-Zeichen ein NL-Zeichen auf den Bildschirm ausgegeben und damit angezeigt, daß die Zeile gelöscht wird oder die Zeile wird vom Bildschirm gelöscht.

Wenn ECHONL und ICANON gesetzt sind, dann wird ein NL-Zeichen auch dann ausgegeben, wenn ECHO nicht gesetzt ist. Dies ist bei Datensichtstationen im lokalen Echo-Modus (sog. Halbduplexbetrieb) nützlich. Ein EOF-Zeichen wird nur dann auf den Bildschirm ausgegeben, wenn es entwertet ist. Da das EOT-Zeichen ('Ende der Übertragung') standardmäßig als EOF-Zeichen verwendet wird, kann man auf diese Weise eine Verbindungsauflösung durch Datensichtstationen, die sich bei Empfang von EOT abmelden, verhindern.

Bei gesetztem ISIG wird bei jedem eingegebenen Zeichen geprüft, ob es sich um eins der Steuerzeichen INTR, QUIT oder SUSP (nur bei Auftragskontrolle) handelt. Ist dies der Fall, so wird die dazugehörige Funktion ausgeführt. Ist ISIG nicht gesetzt, so wird diese Prüfung nicht durchgeführt. D.h., diese Sonderfunktionen für die Eingabe können nur bei gesetztem ISIG durchgeführt werden. Sie können aber auch einzeln ausgeschaltet werden, indem man ihnen einen unwahrscheinlichen oder unmöglichen Wert als Steuerzeichen zuordnet (z.B. 0377).

Bei gesetztem ICANON wird die Standard Eingabeverarbeitung aktiviert. Dies aktiviert die Funktionen zur Behandlung von ERASE- und KILL-Zeichen. Die Eingabezeichen werden zeilenweise zusammengefaßt, das Ende einer Zeile wird mit NL, EOF oder EOL angegeben, so wie dies im Abschnitt *Standard-Eingabeverarbeitung* beschrieben wurde.

Ist ICANON nicht gesetzt, werden Leseaufträge direkt aus dem Eingabepuffer bedient. Dies geschieht erst dann, wenn mindestens MIN Zeichen empfangen wurden oder wenn der Timer TIME abgelaufen ist (siehe Abschnitt *Besondere Eingabeverarbeitung*). Die Angabe des TIME-Wertes erfolgt in Zehntelsekunden.

Bei gesetztem NOFLSH findet die normalerweise nach Empfang der Zeichen QUIT, INTR und SUSP (nur für Auftragskontrolle) durchgeführte Löschung der Ein- und Ausgabepuffer nicht statt.

Sind XCASE und ICANON gesetzt, so werden Großbuchstaben bei der Eingabe akzeptiert, sofern ihnen ein '\'-Zeichen vorangestellt ist; bei der Ausgabe wird ihnen ein '\'-Zeichen vorangestellt. In diesem Modus werden folgende Escape-Sequenzen bei der Ausgabe erzeugt und bei der Eingabe akzeptiert:

Bedeutung	Codierung
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

So werden z.B. für A die Zeichen \a eingegeben, für \n die Zeichen \\n und für \N die Zeichen \\\n.

Hinweis
Ein einem ERASE- oder KILL-Zeichen vorangehendes ESCAPE-Zeichen hebt die dem Zeichen zugeordnete Steuerfunktion auf.

Der Anfangswert für die Lokalmodi (Wert von *c_local*) nach einem Aufruf von *open()* ist, daß kein Bit gesetzt ist.

Steuerzeichen

Die Werte der Steuerzeichen werden durch den Vektor *c_cc*. Die Namen für die jeweiligen Indizes in diesem Vektor sowie die Beschreibungen jedes Vektorelements sowohl für die Standard-Eingabeverarbeitung als auch für die besondere Eingabeverarbeitung werden in der folgenden Tabelle aufgeführt:

Indexname im Standardmodus	Indexname im besonderen Modus	Beschreibung
VEOF	\	EOF-Zeichen
VEOL	\	EOL-Zeichen
VERASE	\	ERASE-Zeichen
VINTR	VINTR	INTR-Zeichen
VKILL	\	KILL-Zeichen
\	VMIN	Wert für MIN
VQUIT	VQUIT	QUIT-Zeichen
VSUSP	VSUSP	SUSP-Zeichen
\	VTIME	Wert für TIME
VSTART	VSTART	START-Zeichen
VSTOP	VSTOP	STOP-Zeichen

Die Indexnamen sind Konstanten, die den Index des jeweiligen Elements (Zeichens) im Vektor *c_cc* darstellen. So ist z.B. das Zeichen *c_cc[VSTOP]* sowohl im Standard-Eingabemodus als auch im besonderen Eingabemodus das STOP-Zeichen.

Die Indexnamen sind eindeutig, außer daß VMIN und VTIME jeweils die selben Werte wie VEOF und VEOL haben können.

Implementierungen wie SINIX, die Auftragskontrolle nicht unterstützen, können den Wert für das SUSP-Zeichen ignorieren, das im Vektor *c_cc* durch VSUSP indiziert wird.

Die Anzahl NCCS der Elemente im Vektor *c_cc* ist implementierungs-abhängig, ebenso wie die Anfangswerte für alle Steuerzeichen. Portable Anwendungen sollten daher keine Annahmen über diese Werte machen.

Wenn {_POSIX_VDISABLE} für die Gerätedatei der Datensichtstation definiert ist und der Wert eines der änderbaren Sonderzeichen gleich {_POSIX_VDISABLE} ist (vgl. Abschnitt *Sonderzeichen*), dann wird diese Funktion deaktiviert. Das heißt, kein Eingabezeichen wird als das deaktivierte Sonderzeichen erkannt. Wenn ICANON nicht gesetzt ist, dann hat der Wert von {_POSIX_VDISABLE} keine besondere Bedeutung für die Einträge mit den Indizes VMIN und VTIME im Vektor *c_cc*.

2.5 Internationalisierung in SINIX

Das Nationalsprachen-System (Native Language System - NLS) besteht aus einer Reihe von Schnittstellen, die die Entwicklung von Anwendungen ermöglichen, die in vielen verschiedenen Sprach- und Kulturumgebungen eingesetzt werden. Eine Reihe von Dienstprogrammen und Bibliotheken erlaubt es, verschiedene Sprachen, ihre Zeichensätze und Landeskonventionen zu unterstützen.
Waren die NLS-Funktionen bisher in einer eigenen Bibliothek zusammengefaßt, so sind diese in der Version V5.22 voll in die Standardfunktionen integriert. Im Kapitel 3 werden eine Reihe von Funktionen beschrieben, die auch schon bisher zum Standardumfang des C-Entwicklungssystems CES gehörten. In dieser Version sind diese Funktionen jedoch so erweitert worden, daß sie das Nationalsprachen-System NLS intern unterstützen.

NLS-Funktionen
Folgende Funktionen aus diesem Handbuch unterstützen das Nationalsprachen-System:

Funktionen für die Arbeit mit Meldungskatalogen:
catclose() catgets() catopen()

Funktionen zur Zeichenumwandlung:
toupper() _toupper() tolower()
_tolower() toascii()

Funktionen zur Zeichenklassifikation:
isalpha() isupper() islower()
isdigit() isxdigit() isalnum()
isspace() ispunct() isprint()
isgraph() iscntrl() isascii()
isfirst() nl_settype() nl_istype()

Funktionen für die Umwandlung von Gleitpunktzahlen in Zeichenketten:
ecvt() fcvt() gcvt()

Eine Funktion für die Bearbeitung von Aufrufargumenten:
getopt()

Eine Funktion für das Einlesen von Kennwörtern:
getpw()

Funktionen für die Verarbeitung von Zeichen, die aus mehreren Bytes bestehen:
mbtowc() wctomb()

Eine Funktion, die den aktuellen Zeichensatz analysiert:
nl_codesize()

Eine Funktion für Landessprachen-Information:
nl_langinfo()

Funktionen für die Ausgabe von Fehlermeldungen:
perror() strerror()

Funktionen für die formatierte Ausgabe:
printf() fprintf() sprintf()

Funktionen für die Bearbeitung regulärer Ausdrücke:
compile() step() advance()

Funktionen für die formatierte Eingabe:
scanf() fscanf() sscanf()

Eine Funktion für das Einstellen einer internationalen Umgebung für das Programm:
setlocale()

Eine Funktion für das Erstellen einer Datum/Zeit-Zeichenkette:
strftime()

Funktionen für die Zeichenkettenbearbeitung:

strcat()	strncat()	strcmp()
strncmp()	strcoll()	strxfrm()
strcpy()	strncpy()	strlen()
strchr()	strrchr()	strpbrk()
strspn()	strcspn()	strtok()

Funktionen für die Umwandlung von Zeichenketten in numerische Werte:

strtol()	atol()	atoi()
strtod()	atof()	

Alle diese Fuktionen erlauben es, landessprach-spezifische Eigenheiten in den Programmen zu berücksichtigen, so daß die mit Hilfe dieser Funktionen geschriebenen Programme unter beliebigen Landessprachen eingesetzt werden können.

Ein Programm, das keine bestimmten Annahmen über die Landessprache, den Zeichensatz und sonstigen Gegebenheiten seiner Umgebung macht, sondern diese Informationen über die oben angeführten Funktionen ermittelt oder verarbeitet, heißt ein *internationalisiertes Programm.*

Internationale Umgebung
Ein internationalisiertes Programm macht keine festen Annahmen über die Umgebung, in der es abläuft. Nichtsdestoweniger läuft es immer in einer ganz bestimmten internationalen Umgebung ab. Es wäre müßig, an dieser Stelle alle möglichen, internationalen Umgebungen zu beschreiben, in denen ein solches Programm ablaufen kann. Deshalb beschränken wir uns hier auf einige allgemeine Hinweise und verweisen ansonsten auf das eigene Handbuch *Internationalisation in SINIX* [5].

Die internationale Umgebung, unter der ein internationalisiertes Programm abläuft, ermittelt dieses aus den Umgebungsvariablen LANG und LC_xxx, bzw. für die Funktionen zur Bearbeitung von Meldungskatalogen durch die Umgebungsvariable NLSPATH. Diese Umgebungsvariablen werden nun erläutert:

LANG identifiziert die Benutzeranforderungen für Landessprache, länder- bzw. anwenderspezifische Eigenheiten und Zeichensatz in Gestalt einer ASCII-Zeichenkette der Form:

LANG=sprache[_gebiet[.zeichensatz]]

Besondere Sprach-Operationen werden zur Laufzeit durch den Aufruf der Funktion *setlocale()* initialisiert. Normalerweise werden die Sprach-Anforderungen des Benutzers, wie durch die Umgebungsvariable LANG angegeben, durch den nachfolgenden Aufruf von *setlocale()* zur internationalen Umgebung des Programms gebunden:

```
setlocale (LC_ALL, "");
```

Auf X/Open-Systemen ist diese Form eines Aufrufs von *setlocale()* definiert, um die internationale Umgebung des Programms aus den zugehörigen Umgebungsvariablen zu initialisieren. LC_ALL spricht die gesamte internationale Umgebung des Programms an und LANG stellt die notwendigen Voreinstellungen zur Verfügung, wenn eine oder mehrere der kategorie-spezifischen Variablen nicht gesetzt oder gleich der leeren Zeichenkette sind.

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME

Enthalten die Benutzeranforderungen für Sprache, Gebiet und Zeichensatz im Hinblick auf Sortierreihenfolge, Zeichenklassifikation und -umwandlung, Währungssymbol und Darstellung von Geldbeträgen, numerische Datendarstellung und Zeit-Formaten. Ist eine dieser Variablen nicht in der aktuellen Umgebung definiert, so stellt LANG die notwendigen Standardeinstellungen zur Verfügung.

Kategorie	Beschreibung
LC_COLLATE	beeinflußt das Verhalten von regulären Ausdrücken und der Funktionen zur Zeichenkettensortierung in *strcoll()* und *strxfrm()*.
LC_CTYPE	beeinflußt das Verhalten von regulären Ausdrücken und der Funktionen zur Zeichenbehandlung in *tolower()*, *toupper()*, *isalpha()* usw.
LC_MONETARY	bestimmt die Einflußfaktoren Sprache, Gebiet und Zeichensatz für die Darstellung von Geldbeträgen.
LC_NUMERIC	beeinflußt den Dezimalpunkt für die Funktionen zur formatierten Ein- und Ausgabe in *printf()* und *scanf()*, und die Funktionen zur Umwandlung von Zeichenketten in *strtod()*.
LC_TIME	beeinflußt das Verhalten der Zeitfunktionen in *strftime()*.

Das Verhalten der Sprachinformations-Funktion *nl_langinfo()* wird ebenfalls durch die Belegungen dieser Umgebungsvariablen beeinflußt (siehe auch *<langinfo.h>*).

LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC und LC_TIME sind so definiert, daß sie ein zusätzliches Feld "@modifikator" akzeptieren, welches es dem Benutzer erlaubt, einen besonderen Fall eines Umgebungsdatums innerhalb einer speziellen Kategorie auszuwählen (zum Beispiel, um das Wörterbuch entgegengesetzt zur Sortierreihenfolge der Zeichen zu definieren). Die Syntax dieser Umgebungsvariablen lautet daher:

[sprache[_gebiet[.zeichensatz]][@modifikator]]

Wenn zum Beispiel ein Benutzer mit dem System in französisch kommunizieren will, aber deutsche Textdateien sortieren muß, so könnten LANG und LC_COLLATE möglicherweise folgendermaßen definiert sein:

```
LANG=Fr_FR
LC_COLLATE=De_DE
```

Dies könnte noch erweitert werden, um zum Beispiel die Wörterbuch-Sortierung durch die Verwendung des Felds *@modifikator* auszuwählen:

LC_COLLATE=De_DE@dict

Zur Laufzeit werden diese Werte zur internationalen Umgebung des Programms gebunden, indem die Funktion *setlocale()* aufgerufen wird.

NLSPATH

Diese Variable enthält eine Folge von Schablonen, die von der Funktion *catopen()* verwendet werden, wenn diese versucht, Meldungskataloge zu finden. Jede Schablone besteht aus einem optionalen Vorspann, einem oder mehreren Ersetzungsfeldern, einem Dateinamen und einer optionalen Erweiterung.

Beispiel

NLSPATH="/system/nlslib/%N.cat"

definiert, daß *catopen()* alle Meldungskataloge im Dateiverzeichnis */system/nlslib* durchsuchen soll, wobei der Katalogname durch *%N* angegeben wird (der Parameter *name* der Funktion *catopen()*), sowie durch die Erweiterung *.cat*.

Ersetzungsfelder bestehen aus dem Zeichen '%', gefolgt von einem einbuchstabigen Schlüsselwort. Die folgenden Schlüsselworte sind derzeit definiert:

%N	Der Wert des Parameters *name* der Funktion *catopen*().
%A	Das Element *archivname* aus dem Argument *name* von *catopen()*. Dieses Schlüsselwort ist nur unter SINIX definiert.
%L	Der Wert von LANG.
%l	Das Element *sprache* aus LANG.
%t	Das Element *gebiet* aus LANG.
%c	Das Element *zeichensatz* aus LANG.
%%	Ein einzelnes Zeichen '%'.

Eine leere Zeichenkette wird ersetzt, wenn der angegebene Wert nicht aktuell definiert ist. Die Trennzeichen '_' und '.' werden bei Ersetzungen durch *%t* und *%c* nicht mit aufgenommen.

Schablonen, die in NLSPATH definiert werden, werden durch Doppelpunkt getrennt ':'. Ein führender oder zwei unmittelbar aufeinanderfolgende Doppelpunkte (::) sind äquivalent zur Angabe von *%N*.

Beispiel

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

gibt an, daß *catopen()* den geforderten Meldungskatalog in *name*, *name.cat* und */nlslib/$LANG/name.cat* suchen soll.

INTLINFO

Diese Variable enthält einen Pseudo-Pfadnamen, der von *setlocale()* verwendet wird, wenn diese Funktion versucht, die internationale *intlinfo*-Datenbank zu finden. Der Pfadname in dieser Variablen kann dieselben Schlüsselwörter enthalten, wie dies bei NLSPATH möglich ist.

Im folgenden Beispiel gibt INTLINFO an, daß die Funktion *setlocale()* die Datenbank De_CH.88591 im Dateiverzeichnis */usr/lib/intlinfo* suchen soll:

```
LANG=De_CH.88591
INTLINFO=/usr/lib/intlinfo/%L
```

Wenn die Datenbank nicht gefunden werden kann, dann finden die Regeln des 7-Bit US ASCII-Zeichensatzes Anwendung. Wenn INTLINFOnicht gesetzt oder leer ist, dann wird die Voreinstellung */usr/lib/nls/intlinfo/%L* verwendet. Der Name der Datenbank (*%L*) wird beim Öffnen der Datenbank durch *setlocale()* angegeben.

Die internationale Umgebung für die Sprache C

Wird bei einem Aufruf von *setlocale()* als erstes Argument *category* die Zeichenkette "C" angegeben oder ruft ein Programm die Funktion *setlocal()* nicht auf, so muß dennoch eine Minimal-Umgebung für dieses Programm zur Verfügng stehen. Diese Minimal-Umgebung ist die *internationale Umgebung für die Sprache C*. Diese Standardumgebung entspricht weitgehend der Voreinstellungen für die USA. So wird als Zeichensatz der US-ASCII Standardzeichensatz vorgegeben, der Dezimalpunkt ist das Zeichen '.' und Datum- und Zeitformat entsprechen den US-Normen.

Die folgenden Tabellen geben die Standardeinstellungen für diesen Zeichensatz wieder:

Zeichensatz

Die folgende Tabelle gibt für jedes zulässige Zeichen seinen ASCII-Code (dezimal, oktal und hexadezimal), seinen Namen oder seine Darstellung, seine Klasse (gemäß den Klassifikations-Routinen), eine ggf. vorhandene Umwndlungsmöglichkeit und die Sortierreihenfolge an:

dezi-mal	oktal	hexa-dez.		Klasse	anderer Fall	Sortier-folge
0	00	00	NUL	control		0/0
1	01	01	SOH	control		1/0
2	02	02	STX	control		2/0
3	03	03	ETX	control		3/0
4	04	04	EOT	control		4/0
5	05	05	ENQ	control		5/0
6	06	06	ACK	control		6/0
7	07	07	BEL	control		7/0
8	10	08	BS	control		8/0
9	11	09	HT	control space		9/0
10	12	0A	LF	control space		10/0
11	13	0B	VT	control space		11/0
12	14	0C	FF	control space		12/0
13	15	0D	CR	control space		13/0
14	16	0E	SO	control		14/0
15	17	0F	SI	control		15/0
16	20	10	DLE	control		16/0
17	21	11	DC1	control		17/0
18	22	12	DC2	control		18/0
19	23	13	DC3	control		19/0
20	24	14	DC4	control		20/0
21	25	15	NAK	control		21/0
22	26	16	SYN	control		22/0
23	27	17	ETB	control		23/0
24	30	18	CAN	control		24/0
25	31	19	EM	control		25/0
26	32	1A	SUB	control		26/0
27	33	1B	ESC	control		27/0
28	34	1C	FS	control		28/0
29	35	1D	GS	control		29/0
30	36	1E	RS	control		30/0
31	37	1F	US	control		31/0
32	40	20	SP	space		32/0
33	41	21	!	punct		33/0
34	42	22	"	punct		34/0
35	43	23	#	punct		35/0
36	44	24	$	punct		36/0
37	45	25	%	punct		37/0
38	46	26	&	punct		38/0

dezi-mal	oktal	hexa-dez.		Klasse	anderer Fall	Sortier-folge
39	47	27	'	punct		39/0
40	50	28	(	punct		40/0
41	51	29	)	punct		41/0
42	52	2A	*	punct		42/0
43	53	2B	+	punct		43/0
44	54	2C	'	punct		44/0
45	55	2D	-	punct		45/0
46	56	2E	.	punct		46/0
47	57	2F	/	punct		47/0
48	60	30	0	digit xdigit		48/0
49	61	31	1	digit xdigit		49/0
50	62	32	2	digit xdigit		50/0
51	63	33	3	digit xdigit		51/0
52	64	34	4	digit xdigit		52/0
53	65	35	5	digit xdigit		53/0
54	66	36	6	digit xdigit		54/0
55	67	37	7	digit xdigit		55/0
56	70	38	8	digit xdigit		56/0
57	71	39	9	digit xdigit		57/0
58	72	3A	:	punct		58/0
59	73	3B	;	punct		59/0
60	74	3C	<	punct		60/0
61	75	3D	=	punct		61/0
62	76	3E	>	punct		62/0
63	77	3F	?	punct		63/0
64	100	40	@	punct		64/0
65	101	41	A	upper xdigit	a	65/0
66	102	42	B	upper xdigit	b	66/0
67	103	43	C	upper xdigit	c	67/0
68	104	44	D	upper xdigit	d	68/0
69	105	45	E	upper xdigit	e	69/0
70	106	46	F	upper xdigit	f	70/0
71	107	47	G	upper	g	71/0
72	110	48	H	upper	h	72/0
73	111	49	I	upper	i	73/0
74	112	4A	J	upper	j	74/0
75	113	4B	K	upper	k	75/0
76	114	4C	L	upper	l	76/0
77	115	4D	M	upper	m	77/0
78	116	4E	N	upper	n	78/0
79	117	4F	O	upper	o	79/0
80	120	50	P	upper	p	80/0
81	121	51	Q	upper	q	81/0
82	122	52	R	upper	r	82/0
83	123	53	S	upper	s	83/0
84	124	54	T	upper	t	84/0

dezi-mal	oktal	hexa-dez.		Klasse	anderer Fall	Sortier-folge
85	125	55	U	upper	u	85/0
86	126	56	V	upper	v	86/0
87	127	57	W	upper	w	87/0
88	130	58	X	upper	x	88/0
89	131	59	Y	upper	y	89/0
90	132	5A	Z	upper	z	90/0
91	133	5B	[	punct		91/0
92	134	5C	\	punct		92/0
93	135	5D	]	punct		93/0
94	136	5E	^	punct		94/0
95	137	5F	-	punct		95/0
96	140	60	`	punct		96/0
97	141	61	a	lower xdigit	A	97/0
98	142	62	b	lower xdigit	B	98/0
99	143	63	c	lower xdigit	C	99/0
100	144	64	d	lower xdigit	D	100/0
101	145	65	e	lower xdigit	E	101/0
102	146	66	f	lower xdigit	F	102/0
103	147	67	g	lower	G	103/0
104	150	68	h	lower	H	104/0
105	151	69	i	lower	I	105/0
106	152	6A	j	lower	J	106/0
107	153	6B	k	lower	K	107/0
108	154	6C	l	lower	L	108/0
109	155	6D	m	lower	M	109/0
110	156	6E	n	lower	N	110/0
111	157	6F	o	lower	O	111/0
112	160	70	p	lower	P	112/0
113	161	71	q	lower	Q	113/0
114	162	72	r	lower	R	114/0
115	163	73	s	lower	S	115/0
116	164	74	t	lower	T	116/0
117	165	75	u	lower	U	117/0
118	166	76	v	lower	V	118/0
119	167	77	w	lower	W	119/0
120	170	78	x	lower	X	120/0
121	171	79	y	lower	Y	121/0
122	172	7A	z	lower	Z	122/0
123	173	7B	{	punct		123/0
124	174	7C	|	punct		124/0
125	175	7D	}	punct		125/0
126	176	7E	~	punct		126/0
127	177	7F	DEL	control		127/0
128	200	80				128/0
...	...	...				...
255	377	FF				255/0

Anwenderspezifische Umgebung
Die folgenden Daten der anwenderspezifischen Umgebung sind in der internationalen Umgebung der Sprache C definiert:

Name	Kategorie	Voreinstellung
D_T_FMT	LC_TIME	"%a %b %d %H:%M:%S %Y"
D_FMT	LC_TIME	"%m/%d/%y"
T_FMT	LC_TIME	"%H:%M:%S"
AM_STR	LC_TIME	"AM"
PM_STR	LC_TIME	"PM"
DAY_1	LC_TIME	"Sunday"
DAY_2	LC_TIME	"Monday"
DAY_3	LC_TIME	"Tuesday"
DAY_4	LC_TIME	"Wednesday"
DAY_5	LC_TIME	"Thursday"
DAY_6	LC_TIME	"Friday"
DAY_7	LC_TIME	"Saturday"
ABDAY_1	LC_TIME	"Sun"
ABDAY_2	LC_TIME	"Mon"
ABDAY_3	LC_TIME	"Tue"
ABDAY_4	LC_TIME	"Wed"
ABDAY_5	LC_TIME	"Thu"
ABDAY_6	LC_TIME	"Fri"
ABDAY_7	LC_TIME	"Sat"
MON_1	LC_TIME	"January"
MON_2	LC_TIME	"February"
MON_3	LC_TIME	"March"
MON_4	LC_TIME	"April"
MON_5	LC_TIME	"May"
MON_6	LC_TIME	"June"
MON_7	LC_TIME	"July"
MON_8	LC_TIME	"August"
MON_9	LC_TIME	"September"
MON_10	LC_TIME	"October"
MON_11	LC_TIME	"November"
MON_12	LC_TIME	"December"

Name	Kategorie	Voreinstellung
ABMON_1	LC_TIME	"Jan"
ABMON_2	LC_TIME	"Feb"
ABMON_3	LC_TIME	"Mar"
ABMON_4	LC_TIME	"Apr"
ABMON_5	LC_TIME	"May"
ABMON_6	LC_TIME	"Jun"
ABMON_7	LC_TIME	"Jul"
ABMON_8	LC_TIME	"Aug"
ABMON_9	LC_TIME	"Sep"
ABMON_10	LC_TIME	"Oct"
ABMON_11	LC_TIME	"Nov"
ABMON_12	LC_TIME	"Dec"
RADIXCHAR	LC_NUMERIC	"."
THOUSEP	LC_NUMERIC	""
YESSTR	LC_ALL	"yes"
NOSTR	LC_ALL	"no"
CRNCYSTR	LC_MONETARY	""

Ein Anwendungsbespiel:
Das folgende Beispiel zeigt, wie ein internationalisiertes Programm aussehen könnte:

```
/*   list - Datei seitenweise auf stdout ausgeben
 *
 *   Aufruf:  list datei
 */

#include <stdio.h>
#include <nl_types.h>

/* Dieses Programm bietet 2 Loesungsansaetze: */
/* #define yes yesmsg */  /* Mit einem Meldungskatalog */
#define yes yesdata  /* Ueber setlocale()/nl_langinfo() */

#define USAGE     catgets(_m_catd, 1, 1, "usage: list file\n")
#define NOTOPEN   catgets(_m_catd, 1, 2, "File %s cannot be opened\n")
#define CONT      catgets(-m_catd, 1, 3, "Continue ")
#define YES       catgets(-m_catd, 1, 4, "y")
#define NO        catgets(-m_catd, 1, 5, "n")
#define END       catgets(-m_catd, 1, 6, "End Of File (EOF)\n")

#define SCRLEN    22
#define LINELEN   256

nl_catd _m_catd = (nl_catd) -1, catopen();
char   *catgets();

FILE   *input;

int main(int argc, char *argv[])
{  char line[LINELEN];

   /* Meldungskatalog oeffnen: */
   _m_catd = catopen("list", 0);

   if (argc != 2)
   {  (void) fprintf(stderr, USAGE);
      return 1;
   }

   if ((input=fopen(argv[1],"r")) == NULL)
   {  (void) fprintf(stderr, NOTOPEN, argv[1]);
      return 1;
   }

   while (1)
   {  if (display(line) == NULL)
      {  (void) fprintf(stderr, END);
         break;
      }

      fprintf(stderr, CONT);
      /* Aufruf der Funktion yesmsg() oder yesdata(): */
      if (!yes())
         break;
   }
   (void) catclose(_m_catd);
   return 0;
}
```

```
/* Eine Bildschirmseite Text ausgeben: */
int display(char *line)
{  register int count=0;

   while (count<=SCRLEN && (line=fgets(line,LINELEN,input)) != NULL)
   {  (void) fprintf(stdout, "%s", line);
      count++;
   }

   (void) fflush(stdout);          /* Ausgabepuffer leeren */
   if (line == NULL)
      return 0;
   else
      return 1;
}

/* Die Funktion yesmsg() behandelt den interaktiven Teil unter
   Verwendung des Meldungskatalogs: */
int yesmsg()
{  int i, b;
   int c_yes;

   /* Ausgabe der Eingabe-Aufforderung: */
   (void) fprintf(stderr,"(%c/%c) ", c_yes=YES[0], NO[0]);
   i = b = getchar();          /* Nur erstes Zeichen der Eingabe
                                  ist relevant */
   while (b!='\n' && b>0)      /* Auf Zeichen '\n' warten */
      b = getchar();

   /* Ergebnis 1, wenn YES eingegeben wurde. */
   return (i==c_yes);
}

/* Die Funktion yesdata() behandelt den interaktiven Teil unter
   Verwendung von setlocale() und nl_langinfo(): */
#include <locale.h>

int yesdata()
{  int i, b;
   char *c_yes[2]; /* Vektor von Zeigern auf Zeichenketten */
   char *setlocale(), *nl_langinfo();

   /* Ja/Nein-Zeichenketten ermitteln: */
   (void) setlocale(LC_ALL, "");
   c_yes[0] = nl_langinfo(YESSTR);
   c_yes[1] = nl_langinfo(NOSTR);

   /* Ausgabe der Eingabe-Aufforderung: */
   (void) fprintf(stderr,"(%c/%c) ", c_yes[0], c_yes[1]);
   i = b = getchar();          /* Nur erstes Zeichen der Eingabe
                                  ist relevant */
   while (b!='\n' && b>0)      /* Auf Zeichen '\n' warten */
      b = getchar();

   /* Ergebnis 1, wenn Zeichenkette fuer YESSTR eingegeben wurde: */
   return (i==(int)*c_yes[0]);
}
```

2.6 Interprozeßkommunikation,

2.6.1 Auswirkungen auf die Systemschnittstellen

Die Funktionen der Interprozeßkommunikation beeinflussen andere Dienste. Die betroffenen Funktionen werden in der nachfolgenden Tabelle aufgeführt:

Beeinflußte Schnittstellen:

errno	*execve()*	*execl()*
execvp()	*execle()*	*exit()*
execlp()	*fork()*	*execv()*

2.6.2 Allgemeine Beschreibung

Das Paket Interprozeßkommunikation umfaßt drei Mechanismen:

- Nachrichten (*messages*) sind formatgebundene Datenströme, die von Prozessen an beliebige andere Prozesse gesendet werden können (dazu werden folgende Systemaufrufe verwendet: *msgget()*, *msgsnd()*, *msgrcv()*, *msgctl()*);
- Gemeinsame Speicherbereiche (*shared memory*) erlauben, daß Prozesse Teile ihres virtuellen Adreßraumes mit anderen Prozessen teilen (dazu werden folgende Systemaufrufe verwendet: *shmget()*, *shmat()*, *shmdt()*, *shmctl()*);
- Semaphoren ermöglichen die Synchronisation der Ausführung von Prozessen (dazu werden folgende Systemaufrufe verwendet: (*semget()*, *semop()*, *semctl()*)

Diejenigen Aspekte ihrer Funktionsweise, die die drei Mechanismen gemeinsam haben, werden nachfolgend beschrieben. Die Beschreibung gliedert sich in die Abschnitte:

- Einrichten eines Kommunikationselements (Nachrichten-Warteschlange, gemeinsamer Speicherbereich, Semaphore)
- Datenstrukturen
- Statusinformationen abfragen oder ändern

Dabei steht *xxx* jeweils für *msg*, *sem* oder *shm*.

Jedes Kommunikationselement (Nachrichten-Warteschlange, Gemeinsamer Speicherbereich, Semaphor) wird durch eine positive ganze Zahl identifiziert. Die Nummer wird beim Einrichten des Kommunikationselements *xxxget()* vom System vergeben. Der Benutzer kann zusätzlich einen Zahlenschlüssel als Namen eines von ihm erzeugten Kommunikationselements festlegen.

Zu jedem Mechanismus existiert eine Tabelle, deren Einträge alle Kommunikationselemente des jeweiligen Mechanismus enthalten. Dabei enthält jeder Eintrag einen vom Benutzer gewählten Zahlenschlüssel als Namen, durch den der Eintrag identifiziert wird.

Einrichten eines Kommunikationselements
xxxget()
Für jeden Mechanismus gibt es einen Systemaufruf *xxxget()*, mit dem ein neues Element erzeugt werden kann oder ein bereits existierendes Element für einen Prozeß verfügbar gemacht werden kann. Die Parameter der Systemaufrufe *xxxget()* sind ein vom Benutzer gewählter Zahlenschlüssel *key* als Benutzername und ein Schalter *xxxflg*.

key
Das Betriebssystem sucht in der zugehörigen Tabelle nach einem Eintrag, der durch den Schlüssel bezeichnet wird. Prozesse können den Systemaufruf *xxxget* mit dem Schlüssel IPC_PRIVATE aufrufen; damit wird sichergestellt, daß ein unbenutzter Eintrag zurückgegeben wird.

xxxflg
Der Schalter beeinflußt, ob und wie auf einen Eintrag zugegriffen werden kann, sowie gegebenenfalls die Zugriffsrechte. Wenn der Schalter IPC_CREAT gesetzt wird, wird ein neuer Eintrag erzeugt, falls noch keiner existiert. Die gewünschten Zugriffsrechte werden durch Bit-ODER mit IPC_CREAT kombiniert. Für den neuen Eintrag werden dann die neun rechten Bits des Schalters als Zugriffsrechte gesetzt. Die Anordnung der Bits entspricht der von *oflag* im Systemaufruf *open()*, wenngleich nur die Lese- und Schreibberechtigung von Bedeutung sind.

Falls bereits ein Eintrag mit dem angegebenen Schlüssel existiert, müssen die neun rechten Bits des Schalters eine Teilmenge der Zugriffsrechte des Eintrags sein, andernfalls scheitern die Systemaufrufe *xxxget()*. Es können also keine weitergehenden Rechte gefordert werden, als vorhanden sind. Zum Verändern der Zugriffsrechte muß der Systemaufruf *xxxctl()* abgesetzt werden (s.u.). Wenn der Schalter IPC_CREAT zusätzlich durch Bit-ODER mit dem Schalter IPC_EXCL kombiniert wird, kehrt *xxxget()* mit einem Fehler zurück, falls für den Schlüssel bereits ein Eintrag existiert.
Wenn der Schalter IPC_CREAT nicht gesetzt ist, muß bereits ein Eintrag existieren, andernfalls scheitern die Systemaufrufe *xxxget()*.

Die Systemaufrufe *xxxget()* liefern eine vom Betriebssystem ausgewählte eindeutige positive ganze Kennzahl (Systemkennzahl *xxxid*) die in den anderen, dem jeweiligen Mechanismus zugehörigen Systemaufrufen verwendet wird. Die Kennzahlen funktionieren vergleichbar wie die Dateikennzahlen - wie sie z. B. open(), dup() und pipe() liefern -, mit der Ausnahme, daß jeder Prozeß, der ihren Wert kennt, sie verwenden kann. D. h. sie müssen nicht vererbt werden, um gültig zu sein. Jeder gemeinsam benutzte Speicherbereich (shared memory), jede Nachrichtenwarteschlange und jede Semaphorenmenge wird so durch die sog. shared-memory-Speicherkennzahl (*shmid*), die Semaphorkennzahl (*semid*) bzw. die Warteschlangenkennzahl (*msqid*). identifiziert.

Datenstrukturen
Jeder Kennzahl ist eine Datenstruktur zugeordnet, die die operationsbezogenen Daten der durchzuführenden oder durchgeführten Operationen enthält. Diese Datenstrukturen (*msqid_ds*, *semid_ds*, *shmid_ds*) sind in *<sys/shm.h>*, *<sys/sem.h>* und *<sys/msg.h>* beschrieben. Unter anderem enthalten diese Datenstrukturen die Prozeßnummer des letzten Prozesses, der eine Operation durchgeführt hat (Nachricht senden oder empfangen, auf gemeinsamen Speicher zugreifen usw.) und die Zeit des letzten Zugriffs.

Alle Datenstrukturen enthalten Eigentümerinformationen und eine *ipc_perm*-Struktur (siehe *<sys/ipc.h>*), aufgrund der Prozessen, die IPC-Funktionen verwenden, Schreib-/Leseberechtigungen (bei Semaphoren Änder-/Leseberechtigungen) erteilt oder verweigert werden. Die *ipc_perm*-Struktur enthält die effektive Benutzer- und Gruppennummer des Prozesses, der den Eintrag erstellt hat (*xxx_perm.cuid* und *xxx_perm.cgid*), sowie eine Benutzer- und eine Gruppennummer (*xxx_perm.uid* und *xxx_perm.gid*), die auch durch den Systemaufruf *xxxctl()* gesetzt werden können. Dazu kommt ein Bitfeld von Zugriffsberechtigungen in der Komponente *mode* der *ipc_perm*-Struktur. Die Bits sind wie folgt belegt:

Bit	Bedeutung
0400	lesen (Eigentümer)
0200	schreiben (Eigentümer)
0040	lesen (Gruppe)
0020	schreiben (Gruppe)
0004	lesen (Andere)
0002	schreiben (Andere)

Die Struktur vom Typ *ipc_perm* hat den Namen *shm_perm*, *sem_perm* oder *msg_perm*, je nach verwendetem Mechanismus. Lese- und Schreib-/ Änderberechtigungen werden einem Prozeß erteilt, wenn eine oder mehrere der folgenden Bedingungen zutreffen.

- Die effektive Benutzernummer des Prozesses ist die des Systemverwalters.
- Die effektive Benutzernummer des Prozesses ist identisch mit *xxx_perm.cuid* oder *xxx_perm.uid* in der der IPC-Kennzahl zugeordneten Datenstruktur und das entsprechende Bit für *Eigentümer* in *xxx_perm.mode* ist gesetzt.
- Die effektive Benutzernummer des Prozesses ist nicht identisch mit *xxx_perm.cuid* oder *xxx_perm.uid*, aber die effektive Gruppennummer des Prozesses ist identisch mit *xxx_perm.cgid* oder *xxx_perm.gid* in der der IPC-Kennzahl zugeordneten Datenstruktur und das entsprechende Bit für *Gruppe* in *xxx_perm.mode* ist gesetzt.
- Die effektive Benutzernummer des Prozesses ist nicht identisch mit *xxx_perm.cuid* oder *xxx_perm.uid*, und die effektive Gruppennummer des Prozesses ist nicht identisch mit *xxx_perm.cgid* oder *xxx_perm.gid* in der der IPC-Kennzahl zugeordneten Datenstruktur, aber das entsprechende Bit für *Andere* in *xxx_perm.mode* ist gesetzt.

In allen anderen Fällen wird keine Berechtigung erteilt.

Statusinformationen abfragen oder ändern
xxxctl()
Für jeden Mechanismus gibt es einen Systemaufruf *xxxctl*(), mit dem der Status eines Eintrags abgefragt werden kann, Statusinformation gesetzt werden kann, oder ein Eintrag aus dem System entfernt werden kann.

Fragt ein Prozeß den Status eines Eintrags ab, so prüft das Betriebssystem, ob der Prozeß die Leseberechtigung hat, und kopiert danach Daten aus dem Tabelleneintrag in die vom Benutzer angegebene Struktur.

Will ein Prozeß die Parameter des Eintrags neu setzen, dann prüft das Betriebssystem, ob die effektive Benutzernummer des Prozesses mit der Benutzernummer des Eintrags oder mit der Benutzernummer des Erstellers des Eintrags übereinstimmt, bzw. daß die effektive Benutzernummer die des Systemverwalters ist. Um Parameter neu zu setzen ist Schreibberechtigung allein nicht ausreichend. Das Betriebssystem kopiert die vom Benutzer angegebenen Daten in den Tabelleneintrag, setzt dabei die Benutzernummer, die Gruppennummer, die Zugriffsrechte und andere von der Art des Mechanismus abhängige Felder. Nicht verändert werden die Felder mit der Benutzer- und Gruppennummer des Erstellers des Eintrags; dadurch verbleiben dem Ersteller des Eintrags immer Kontrollrechte.

Will ein Prozeß einen Eintrag entfernen, stellt das Betriebssystem sicher, daß die effektive Benutzernummer des Prozesses mit einer der Benutzernummern in der *ipc_perm*-Struktur übereinstimmt. Nachdem ein Eintrag entfernt wurde, ist es nicht mehr möglich, mit der alten Kennzahl auf den Eintrag zuzugreifen.

Hinweis
Die Verwendung der IPC-Mechanismen verlangt hohe Sorgfalt, da nicht benutzte oder benötigte IPC-Elemente vom Betriebssystem nicht in allen Fällen erkannt werden. Im Betriebssystem gibt es keine Aufzeichnungen darüber, welche Prozesse auf ein IPC-Element zugreifen - in der Tat kann jeder Prozeß auf ein IPC-Element zugreifen, dem die richtige Kennzahl bekannt ist und der zugriffsberechtigt ist, auch wenn er nie einen Systemaufruf *xxxget()*abgesetzt hat. Deshalb kann das Betriebssystem die IPC-Strukturen nicht implizit bereinigen (z.B. bei Prozeßende).

Die IPC-Mechanismen sollten nur bei extremen Performance-Anforderungen verwendet werden.

2.7 Bildschirmsteuerung mit curses

2.7.1 Überblick

curses ist eine Bildschirmschnittstellen-Definition, in der folgende C-Bibliotheksfunktionen zur Bildschirmbehandlung und-aktualisierung zusammengefaßt sind:

addch()
addstr()
attroff()
baudrate()
beep()
box()
cbreak()
clear()
clearok()
clrtobot()
clrtoeol()
def_prog_mode()
delay_output()
delch()
deleteln()
delwin()
echo()
endwin()
erase()
erasechar()
flushinp()
getch()
getstr()
getyx()
has_is()
has_il()
idlok()
inch()
initscr()
insch()
insertln()
intrflush()
keypad()
killchar()
leaveok()
longname()
move()
mvwin()
newpad()
newterm()
newwin()
nl()
nodelay()
overlay()
prefresh()
printw()
reset_prog_mode()
raw()
refresh()
resetty()
scanw()
scroll()
scrollok()
set_term()
setscrreg()
subwin()
touchwin()
typeahead()
unctrl()
wnoutrefresh()

Mit den *curses*-Bibliotheksfunktionen können Datenstrukturen, sog. Fenster, behandelt werden. Unter Fenster ist ein zweidimensionaler Vektor von Zeichen zu verstehen, das den ganzen Bildschirm einer Datensichtstation oder Teile davon darstellt. Die Fenster werden mit den Funktionen behandelt, die in diesem Abschnitt beschrieben werden. Das Programmpaket *curses* registriert und protokolliert die Zeichen, die auf dem physikalischen Bildschirm dargestellt werden. Die Manipulation erfolgt auf der untersten Ebene mit den Funktionen *move()* und *addch()*. Mit diesen Funktionen wird die Position innerhalb des Standard-Fensters *stdscr* festgelegt und Zeichen in das Standard-Fenster geschrieben; beim Standard-Fenster handelt es sich um den ganzen Bildschirm.

Eine Anwendung kann diese Funktionen benutzen, um Daten in der jeweils geeigneten Reihenfolge in das Fenster zu schreiben. Sobald alle Daten in das Fenster gesetzt wurden, muß die Funktion *refresh()* aufgerufen werden. *curses* stellt daraufhin fest, welche der vorgenommenen Änderungen den physikalischen Bildschirm betreffen und aktualisiert die Bildschirmanzeige entsprechend den momentan im Fenster enthaltenen Zeichen; dabei werden eine Reihe von Funktionen durchgeführt, die für die jeweils benutzte Datensichtstation optimal geeignet sind.

Auf einer höheren Ebene sind die durch *move()* und *addch()* durchgeführten Operationen in Funktionen zusammengefaßt, mit denen vollständige Zeichenketten auf den Bildschirm gesetzt und Formatumwandlungen entsprechend *printf*() vorgenommen werden können.

Weiterhin sind Schnittstellen definiert, mit denen der gesamte Bildschirm gelöscht und einzelnen Zeichen auf dem Bildschirm Merkmale wie z.B. inverse Darstellung, Unterstreichen und Blinken zugeordnet werden können.

Es besteht die Möglichkeit, neue Fenster zu öffnen, so daß eine Anwendung auf dem Bildschirm mehrere Bilder aufbauen und das jeweils gewünschte sehr schnell zur Darstellung bringen kann. Zur Eröffnung von neuen Fenstern wird die Funktion *newwin()* verwendet. Zu jeder Funktion zur Manipulation des Standard-Bildschirms *stdscr* gibt es eine äquivalente Funktion zur Behandlung eines bestimmten Fensters; eine solche Funktion setzt sich aus *w...()* und dem Namen der Standardfunktion zusammen (Beispiel: *move()* und *wmove()*). *move()* ist also beispielsweise funktionsgleich mit *wmove(stdscr,..)*, ähnlich wie *printf()* und *fprintf(stdout, ...)* beides Schnittstellen zur Ausgabe von Zeichen sind (vgl. *printf()*).

Ein Fenster muß nicht unbedingt dem ganzen Bildschirm entsprechen. Es besteht auch die Möglichkeit, kleinere Fenster zu erzeugen. Außerdem kann dafür gesorgt werden, daß ein Fenster nur teilweise auf dem Bildschirm zu sehen ist. Schließlich können große Fenster geöffnet werden, die als *Pad* bezeichnet werden; die Größe eines *Pad* kann die des Bildschirms übersteigen.

Sollen mit einer großen Anwendung mehrere Datensichtstationen gleichzeitig behandelt werden, so kann diese Anwendung mit der Funktion *newterm()* zusätzliche Datensichtstationen "eröffnen". Mit der Funktion *set_term()* kann die Datensichtstation ausgewählt werden, deren Bildschirm beim nächsten Aufruf von *refresh()* aktualisiert werden soll.

curses bietet auch Schnittstellen, mit denen Eingabezeichen manipuliert und Zeichen eine Reihe von Attributen zugeordnet und aufgehoben werden können: Bildschirmdarstellung (Echo) ein- und ausschalten, Eingabe von Einzelzeichen mit oder ohne Signalverarbeitung (CBREAK- und RAW-Modus), Wagenrücklauf mit oder ohne Zeilenvorschub, Durchführung des Bildlaufs möglich oder nicht, usw.

2.7.2 Datentypen und die Include-Datei curses.h

In diesem Abschnitt werden die Datentypen beschrieben, die vom Programm *curses* unterstützt werden.

Dabei wurde auf eine Beschreibung der Datenstrukturen selbst verzichtet, da die *curses*-Bibliotheksfunktionen Schnittstellen zu den Datentypen darstellen. Jegliche Datenmanipulation erfolgt in *curses* mit Hilfe der beschriebenen Funktionen.

Die Include-Datei <curses.h>

In der Include-Datei *<curses.h>* sind eine Reihe von Konstanten definiert; außerdem werden die Datentypen deklariert, die der betreffenden Anwendung zur Verfügung stehen.

Datentypen

Folgende Datentypen werden deklariert:

WINDOW*	Zeiger auf die Bildschirmdarstellung
SCREEN*	Zeiger auf Datensichtstationsbeschreibung
bool	Daten vom Typ Bool
chtype	ein Zeichen in einem Fenster

Die Variablen WINDOW und SCREEN werden zur Speicherung der Informationen benutzt; sie werden beim Aufrufen der entsprechenden Funktionen erzeugt und über einen Zeiger zugänglich gemacht. Über diesen Zeiger erfolgen alle Datenmanipulationen.

Konstanten

Folgende Konstanten werden definiert:

Allgemeine Konstanten

COLS	Zahl der Spalten auf dem Bildschirm der Datensichtstation
ERR	Wert, der bei Auftreten eines Fehlers ausgegeben wird
FALSE	Der logische Wert FALSCH
LINES	Zahl der Zeilen auf dem Bildschirm der Datensichtstation
OK	Wert, der nach erfolgreicher Durchführung einer Funktion ausgegeben wird
NULL	Nullzeiger
TRUE	Der logische Wert WAHR

Bildschirmattribute

Die folgenden Konstanten können an die Funktionen *attron()*, *attroff()* und *attrset()* übergeben werden:

A_BLINK	Blinken
A_BOLD	Hell oder Fett
A_DIM	Halbhell
A_REVERSE	Umkehranzeige
A_STANDOUT	Hervorhebung
A_UNDERLINE	Unterstreichen
A_ATTRIBUTES	Bitmaske zur Merkmalentnahme
A_CHARTEXT	Bitmaske zur Zeichenentnahme

Merkmale sind normalerweise fest mit den Zeichen verknüpft.

Eingabewerte

Folgende Konstanten können durch *getch()* zurückgeliefert werden, vorausgesetzt, *keypad()* ist zuvor aktiviert worden. Zu beachten ist, daß ein Teil dieser Werte nicht auf allen Datensichtstationen unterstützt wird. Dies kann der Fall sein, wenn eine Datensichtstation nach Betätigung einer Taste keinen eindeutigen Code übermittelt oder die Tastenbelegung nicht in der Tastaturtabelle der Datensichtstation enthalten ist.

KEY_BREAK	Break
KEY_HOME	Bildschirmanfang (Pfeil nach links oben)
KEY_BACKSPACE	Backspace
KEY_F0	Funktionstasten; Platz für 64 Tasten reserviert
KEY_F(n)	(KEY_F0+(n))
KEY_DL	Zeile löschen
KEY_IL	Zeile einfügen
KEY_DC	Zeichen löschen
KEY_IC	Zeichen einfügen oder Einfügemodus einschalten
KEY_EIC	Einfügemodus ausschalten
KEY_CLEAR	Bildschirm löschen
KEY_EOS	Bis Bildschirmende löschen
KEY_EOL	Bis Zeilenende löschen
KEY_SF	1 Zeile vorwärtsblättern
KEY_SR	1 Zeile rückwärts blättern
KEY_NPAGE	1 Seite vorwärts
KEY_PPAGE	1 Seite zurück
KEY_STAB	Tabulator setzen
KEY_CTAB	Tabulator löschen
KEY_CATAB	Alle Tabulatoren löschen
KEY_ENTER	Enter oder senden
KEY_SRESET	teilweise ("soft") zurücksetzen
KEY_RESET	("hard") zurücksetzen
KEY_PRINT	drucken oder kopieren
KEY_LL	Bildschirmanfang oder unterer Bildschirmrand (Pfeil nach links unten)

KEY_A1	links oben virtueller Funktionstastenblock
KEY_A3	rechts oben virtueller Funktionstastenblock
KEY_B2	Mitte virtueller Funktionstastenblock
KEY_C1	links unten virtueller Funktionstastenblock
KEY_C3	rechts unten virtueller Funktionstastenblock

Die Tasten auf dem virtuellen Funktionstastenblock sind folgendermaßen angeordnet:

```
A1          nach oben     A3
nach links  B2            nach rechts
C1          nach unten    C3
```

2.7.3 Funktionen

Die Funktionstypen

Die Funktionen können in folgende Gruppen eingeteilt werden:

Allgemeine Bildschirmbehandlung
Behandlung von Fenstern und *Pad*
Ausgabe
Eingabe
Setzen von wahlweisen Ausgabefunktionen
Setzen von wahlweisen Eingabefunktionen
Informationen über die Umgebung
Sonstiges

Benennung der Funktionen

Von einer Reihe von Funktionen gibt es zwei oder mehr Varianten. Bei den Funktionen, die mit einem w beginnen, ist ein Fenster- oder *Pad*-Argument erforderlich. Dies gilt jedoch nicht für *wnoutrefresh()* und *wrefresh*(win), bei denen ausschließlich Fenster-Argumente zulässig sind; zur Aktualisierung eines *Pad* muß dann eine der Funktionen *prefresh()* oder *pnoutrefresh()* benutzt werden. Bei den Funktionen, die mit einem *p* beginnen, ist ein *Pad*-Argument erforderlich. Funktionen, die weder mit einem *w* noch mit einem *p* beginnen, werden i.a. zur Behandlung des Standard-Bildschirms *stdscr* benutzt.

Bei Funktionen, die mit dem Präfix *mv* versehen sind, werden *x* und *y* Koordinaten benötigt, auf die vor der Durchführung der Operation positioniert wird. Bei einer mit *mv* beginnenden Funktion wird in jedem Fall zunächst *move()*, dann erst die auf *mv* folgende Funktion aufgerufen. Die obere linke Ecke hat in jedem Fall die Koordinaten (0,0), nicht (1,1). Werden bei der Funktion *move(y,x)* also die Koordinaten $y = 1$ und $x = 0$ angegeben, so wird die Schreibmarke in die erste Spalte der zweiten Zeile des Fensters bewegt.

Beginnt eine Funktion mit *mvw*, so ist sowohl ein Fenster- oder *Pad*-Argument als auch die Angabe der Koordinaten *x* und *y* erforderlich. Das Fenster-Argument muß in jedem Fall den Koordinaten vorangestellt werden.

Beispiel

Das folgende Programmbeispiel zeigt die Anwendung des Paketes curses. Mit diesem Programm wird zunächst an einer beliebigen Stelle auf dem Bildschirm ein Sternchen (*) dargestellt; dann wartet es auf die Eingabe eines Leerzeichens und startet eine Schleife. Die Eingabe wird Zeichen für Zeichen eingelesen; die Zeichen werden dabei nicht auf dem Bildschirm ausgegeben.

```
/*
**stars.c
**       Demonstrationsprogramm für das Paket curses
*/

#include<curses.h>
#incluce<signal.h>

extern void srand();
extern void exit();

/*
**trap()
**       Wird bei Empfang eines Signals aufgerufen:
**       Betriebsarten der Datensichtstation zurücksetzen
**       und Programm beenden
*/
trap(sig)
int sig;
{
      (void) endwin();
      exit(sig);
}

main()
{
            int x, y;
            struct sigaction act;

            /* Unterbrechungssignale intitialisieren
               und abfangen */
            act.sa_handler = trap;
            (void) sigemptyset($act.sa_mask);
            act.sa_flags = 0;
            (void) sigaction(SIGINT, &act,
                        (struct sigaction*) NULL);

            /* Datensichtstation initialisieren */
            (void) initscr();
            (void) noecho();
            (void) cbreak();
            (void) clear();

            /* Zufallszahlengenerator initialisieren */
            srand((unsigned) getpid());
```

```
            /* Schleife */
            for(;;) {

            /* Zufallskoordinaten generieren */
            y = rand() % LINES;
            x = rand() % COLS;
            /* '*'in Fenster schreiben */
            (void) move(y,x);
            (void) addch('*');
            /* und darstellen */
            (void) refresh();
            /* auf Eingabe eines Leerzeichens warten */
            while (getch() != ' ')
            ;

            }
      /* nicht durchlaufen */
}
```

2.7.4 Synchrone und vernetzte asynchrone Datensichtstationen

In diesem Kapitel sind Hinweise für den Ersteller von Anwendungsprogrammen enthalten, die bei der Entwicklung von Treibern für synchrone, vernetzte asynchrone (NWA) oder lokal angeschlossene, asynchrone Nichtstandard-Datensichtstationen beachtet werden sollten.

Derartige Datensichtstationen werden häufig von einem Großrechner aus gesteuert; die Zeichen werden zwischen dem Hauptrechner und den Datensichtstationen in vielen Fällen blockweise übertragen. Das heißt, die Zeichen, die der Benutzer an der Datensichtstation eingibt, werden erst bei Betätigung einer bestimmten Taste an den Hauptrechner übermittelt.

Jedoch ist es - selbst wenn die Möglichkeit zur Übertragung von Blöcken beliebiger Größe besteht - nicht immer möglich oder erwünscht, daß ein Zeichen bei Betätigung einer einzigen Taste an den Hauptrechner übermittelt wird.

Dies kann bei einem Anwendungsprogramm, bei dem die Eingabe von Einzelzeichen möglich sein soll, zu großen Probleme führen; siehe Eingabe, unten.

Ausgabe

Bei Funktionen im Zusammenhang mit der Zeichenausgabe an eine Datensichtstation kann das Programm *curses* ohne Einschränkungen benutzt werden; dies gilt jedoch nicht für alle Datensichtstationen, bei denen zu einer Aktualisierung mit *refresh()* in jedem Fall ein vollständig neuer Aufbau der Bildschirmanzeige gehört.

Ist vor einer derartigen Funktion jedesmal zusätzlich das Löschen des Bildschirms erforderlich, so kann es zu unakzeptablen Wartezeiten kommen. Nach dem aktuellen Kenntnisstand gibt es mit *curses* aber auf derartigen Datensichtstationen keine Probleme.

Eingabe

Auf einigen synchronen (blockorientiert arbeitenden) und asynchronen, vernetzten Datensichtstationen werden die *curses*-Eingabefunktionen konzeptionsbedingt nur teilweise oder überhaupt nicht unterstützt. Folgende Punkte sind besonders zu beachten:

- Die Eingabe von Einzelzeichen ist teilweise nicht möglich. In manchen Fällen muß die Übertragung aller an einer Datensichtstation eingegebenen Zeichen durch Betätigung einer bestimmten Taste ausgelöst werden.
- Die Zeichendarstellung (Echo) kann nicht in jedem Fall ausgeschaltet werden. Die Darstellung der Zeichen wird möglicherweise direkt von der Datensichtstation durchgeführt. Bei der Erstellung einer *curses*-Anwendung für derartige Datensichtstationen sollte beachtet werden, daß alle eingegebenen Zeichen an der jeweils aktuellen Position der Schreibmarke auf dem Bildschirm dargestellt werden. Diese muß nicht unbedingt mit der Position der Schreibmarke im Fenster übereinstimmen.

2.8 Bildschirmsteuerung mit termcap/terminfo

Das CES C-Entwicklungssystem unter SINIX stellt neben den im vorangegangenen Abschnitt 2.6 beschriebenen *curses*-Funktionen, die eine komfortable Möglichkeit der Bildschirmsteuerung bieten, noch eine weitere Schnittstelle zur Bildschirmsteuerung zur Verfügung: *termcap*.

Die *termcap*-Schnittstelle bietet eine Reihe von Funktionen, die auf einer niedrigen Ebene die Bildschirmsteuerung beliebiger Datensichtstationen zur Verfügung stellt.

In der vorliegenden Version V5.22 wird die früher weitverbreitete *termcap*-Schnittstelle über die neuere *terminfo*-Datenbank realisiert. Daher werden in diesem Abschnitt beide Begriffe gemeinsam erläutert. Die *terminfo*-Datenbank stellt keine eigenen Funktionen zur Verfügung, über die Terminal-Eigenschaften realisiert werden können. Die gesamte Bildschirmsteuerung durch *terminfo* erfolgt implizit durch die Funktionen des *curses*-Pakets (siehe auch Abschnitt 2.7).

Da die für die *termcap*-Schnittstelle benötigten Informationen bereits zu einem großen Teil bei der *terminfokBeschreibung zu finden sind, erläutert der Abschnitt 2.8.1 die terminfo*-Datenbank und der daran anschließende Abschnitt 2.8.2 die eigentliche *termcap*-Schnittstelle.

2.8.1 Die terminfo-Datenbank

terminfo ist eine Datenbank, in der Datensichtstationen beschrieben werden und die u.a. von *vi* und *curses* verwendet wird. Die Beschreibung der Datensichtstationen in *terminfo* besteht aus der Auflistung ihrer Eigenschaften und einer Beschreibung, wie Operationen durchgeführt werden. Auch eventuell erforderliche Füllzeichen und Initialisierungssequenzen sind in *terminfo* aufgeführt.

Die Einträge in *terminfo* bestehen aus einer Reihe durch Komma voneinander getrennter Felder. Ein Zwischenraum nach dem Komma wird ignoriert. Der erste Eintrag für eine Datensichtstation gibt die der Station zugeordneten Namen an, durch "|"-Zeichen voneinander getrennt. Der erste Name ist die am häufigsten für diese Station verwendete Abkürzung; der letzte Name sollte ein vollständiger Name sein, mit dem die Station eindeutig identifiziert ist; alle anderen Namen sind als Synonyme des Datensichtstationsnamens zu verstehen. Mit Ausnahme des letzten Namens sollten alle Namen nur Kleinbuchstaben und keine Leerzeichen enthalten; der letzte Name kann aus Gründen der besseren Lesbarkeit sowohl Großbuchstaben als auch Leerzeichen enthalten.

Mit Ausnahme des letzten, ausführlichen Namens sollten die Datensichtstationsnamen den folgenden Konventionen entsprechen. Der Hardware der Datensichtstation sollte ein Stamm-Name zugeordnet werden, z.B. *hp2621*. Dieser Name sollte keine Bindestriche enthalten; Synonyme, die mit keinem anderen Namen kollidieren, sind allerdings möglich. Verschiedene mögliche Betriebsarten der Hardware oder Benutzerpräferenzen sollten angegeben werden, indem man an den Namen einen Bindestrich, gefolgt von dem Code für die Betriebsart, anhängt. Ein *vt100* in 132-Spaltenmodus würde z.B. mit *vt100-w* angegeben. Soweit möglich, sollten folgende Suffixe verwendet werden:

Suffix	Bedeutung	Beispiel
-w	Breitformat (über 80 Spalten)	vt100-w
-am	mit automatischem Rand (Standard)	vt100-am
-nam	ohne automatischen Rand	vt100-nam
-*n*	Anzahl Bildschirmzeilen	aaa-60
-na	keine Pfeiltasten (Lokalmodus)	c100-na
-*n*p	Anzahl Seiten im Speicher	c100-4p
-rv	Inversdarstellung	c100-rv

Eigenschaften von Datensichtstationen
In der folgenden Liste bedeuten:

Variable: der Name, mit dem der Programmierer die Eigenschaft (auf terminfo-Ebene) angibt.

Cap-Name: der in der Datenbank verwendete Kurzname, der z.B. bei der Aktualisierung der Datenbank verwendet wird.

I.-Code: der aus zwei Buchstaben bestehende, in der übersetzten Datenbank verwendete interne Code; dieser entspricht dem früheren *termcap*-Namen. Diese Namen werden für die Verwendung der *termcap*-Funktionen benötigt.

Es gibt keine strengen Einschränkungen bezüglich der Länge der Cap-Namen; es hat sich jedoch eine informelle Begrenzung auf 5 Zeichen eingebürgert, um eine gewisse Kürze beizubehalten und eine Ausrichtung der Tabulatoren in der Quelldatei *caps* zu ermöglichen. Wo möglich, werden dieselben oder ähnliche Namen wie im ANSI X3.64-1979-Standard verwendet. Auch die Bedeutung ist nach Möglichkeit dieselbe.

(P)
bedeutet, daß Verzögerungen angegeben werden können.

(G)
bedeutet, daß die Zeichenkette von *tparm* mit den angegebenen Parametern bearbeitet wird (#i).

(*)
bedeutet, daß die Anzahl Füllzeichen pro betroffene Zeile angegeben werden kann.

($\#_i$)
bezeichnet den *i*ten Parameter.

Variable	Cap-Name	I.-Code	Bedeutung
Boolesche Angaben:			
auto_left_margin	bw	bw	cub1 in Spalte 0 bewirkt Sprung zur letzten Spalte
auto_right_margin	am	am	Automat. Randeinstellung vorhanden
beehive_glitch	xsb	xb	f1 = Escape, f2 = Control-C (speziell für Beehive-Terminals)
ceol_standout_glitch	xhp	xs	Hervorhebung wird durch Überschreiben nicht gelöscht (hp)
eat_newline_glitch	xenl	xn	Neue-Zeile nach Spalte 80 wird ignoriert (Concept)
erase_overstrike	eo	eo	Übereinander geschriebene Zeichen können mit einem Leerzeichen gelöscht werden
generic_type	gn	gn	Generischer Leitungstyp (z.B. Wählleitung)
hard_copy	hc	hc	Hardcopy-Gerät
has_meta_key	km	km	Meta-Taste vorhanden (Umschalten u. Setzen des Paritätsbits)
has_status_line	hs	hs	Gesonderte "Statuszeile" vorhanden
insert_null_glitch	in	in	Einfügemodus erkennt Nullzeichen
memory_above	da	da	Bildschirminhalt kann über den Bildschirm hinaus nach oben verschoben u. festgehalten werden
memory_below	db	db	Bildschirminhalt kann über den Bildschirm hinaus nach unten verschoben u. festgehalten werden
move_insert_mode	mir	mi	Positionieren im Einfügemodus möglich
move_standout_mode	msgr	ms	Positionieren im Hervorhebungsmodus möglich
over_strike	os	os	Übereinanderschreiben möglich
status_line_esc_ok	eslok	es	Escape kann in Statuszeile verwendet werden
teleray_glitch	xt	xt	Löschende Tabulatoren, Hervorhebungsmodus wird mit "magischem Zeichen" aktiviert (Teleray 1061)
tilde_glitch	hz	hz	Tilde nicht abdruckbar (Hazeltine)
transparent_underline	ul	ul	Unterstreichen möglich
xon_xoff	xon	xo	xon/xoff-Quittungsbetrieb
Numerische Angaben:			
columns	cols	co	Anzahl Spalten pro Zeile
init_tabs	it	it	Tabulatoren bei Start auf jeder #-ten Position
lines	lines	li	Anzahl Zeilen pro Bildschirm- o. Speicherseite
lines_of_memory	lm	lm	Zeilen im Speicher wenn > 'lines'; 0 bedeutet: variabel
magic_cookie_glitch	xmc	sg	Anzahl Leerzeichen nach smso oder rmso (Hervorhebungsmodus ein/aus)
padding_baud_rate	pb	pb	Niedrigste Übertragungsgeschwindigkeit, ab der Verzögerungen für CR/NL erforderlich sind
virtual_terminal	vt	vt	Log. Datensichtstationsnummer (UNIX-System)
width_status_line	wsl	ws	Anzahl Spalten in Statuszeile

Steuerzeichenfolgen:

back_tab	cbt	bt	Rückwärts-Tabulator-Zeichen (P)
bell	bel	bl	Akustisches Signal (Klingel) (P)
carriage_return	cr	cr	Wagenrücklauf (P*)
change_scroll_region	csr	cs	Scrollbereich wechseln (Zeilen #1 bis #2 (vt100) (PG)
clear_all_tabs	tbc	ct	Tabulatoren löschen (P)
clear_screen	clear	cl	Bildschirm löschen und Schreibmarke auf Bildschirmanfang positionieren (P*)
clr_eol	el	ce	Löschen bis Zeilenende (P)
clr_eos	ed	cd	Löschen bis Bildschirmende (P*)
column_address	hpa	ch	Schreibmarke auf Spalte positionieren (PG)
command_character	cmdch	CC	Modifizierbares Befehlszeichen (Prototyp)
cursor_address	cup	cm	Schreibmarke positionieren (bildschirm-bezogen, Zeile #1, Spalte #2) (PG)
cursor_down	cud1	do	Schreibmarke eine Zeile nach unten
cursor_home	home	ho	Schreibmarke an Bildschirmanfang (falls cup nicht vorhanden)
cursor_invisible	civis	vi	Schreibmarke auf nicht sichtbar setzen
cursor_left	cub1	le	Schreibmarke eine Stelle nach links
cursor_mem_address	mrcup	CM	Schreibmarke positionieren (speicher-bezogen)
cursor_normal	cnorm	ve	Schreibmarke auf Normalzustand setzen (vs/vi rücksetzen)
cursor_right	cuf1	nd	Schreibmarke nach rechts ohne Überschreiben/Löschen
cursor_to_ll	ll	ll	Schreibmarke auf letzte Zeile, erste Spalte (falls cup nicht vorhanden)
cursor_up	cuu1	up	Schreibmarke eine Zeile nach oben
cursor_visible	cvvis	vs	Schreibmarke hervorgehoben darstellen
delete_character	dch1	dc	Zeichen ausfügen (P*)
delete_line	dl1	dl	Zeile ausfügen (P*)
dis_status_line	dsl	ds	Statuszeile ausschalten
down_half_line	hd	hd	Halbe Zeile nach unten (halber Zeilenvorschub)
enter_alt_charset_mode	smacs	as	Alternativen Zeichensatz einschalten (P)
enter_blink_mode	blink	mb	Blinken einschalten
enter_bold_mode	bold	md	Hell- oder Fettmodus einschalten
enter_ca_mode	smcup	ti	Steuerzeichenfolge, die vor Benutzung von cup ausgegeben werden muß
enter_delete_mode	smdc	dm	Ausfügemodus einschalten
enter_dim_mode	dim	mh	Halbhellmodus einschalten
enter_insert_mode	smir	im	Einfügemodus einschalten
enter_protected_mode	prot	mp	Geschützten Modus einschalten
enter_reverse_mode	rev	mr	Inversdarstellung einschalten
enter_secure_mode	invis	mk	Dunkelmodus einschalten (Zeichen unsichtbar)
enter_standout_mode	smso	so	Hervorhebungsmodus einschalten
enter_underline_mode	smul	us	Unterstreichungsmodus einschalten
erase_chars	ech	ec	#1 Zeichen löschen (mit Leerzeichen überschreiben) (PG)
exit_alt_charset_mode	rmacs	ae	Alternativen Zeichensatz ausschalten (P)
exit_attribute_mode	sgr0	me	Alle Darstellungsarten ausschalten
exit_ca_mode	rmcup	te	Beendet Programme, die cup verwenden
exit_delete_mode	rmdc	ed	Ausfügemodus ausschalten
exit_insert_mode	rmir	ei	Einfügemodus ausschalten
exit_standout_mode	rmso	se	Hervorhebungsmodus ausschalten
exit_underline_mode	rmul	ue	Unterstreichungsmodus ausschalten
flash_screen	flash	vb	Optisches Signal (ohne Schreibmarken-bewegung)
form_feed	ff	ff	Seitenvorschub bei Hardcopy-Gerät (P*)
from_status_line	fs1	fs	Rücksprung von Statuszeile
init_1string	is1	i1	Datensichtstationsinitialisierung
init_2string	is2	i2	Datensichtstationsinitialisierung
init_3string	is3	i3	Datensichtstationsinitialisierung
init_file	if	if	Name der Datei, die *ls* enthält
insert_character	ich1	ic	Zeichen einfügen (P)
insert_line	il1	al	Leerzeile einfügen (P*)
insert_padding	ip	ip	Einfügen von Füllzeichen nach ic (P*)

key_backspace	kbs	kb	Code der Backspace-Taste
key_catab	ktbc	ka	Code der Tabulatoren-löschen-Taste
key_clear	kclr	kC	Code der Bildschirm-löschen- oder Lösch-Taste
key_ctab	kctab	kt	Code der Tabulator-löschen-Taste
key_dc	kdch1	kD	Code der Zeichen-ausfügen-Taste
key_dl	kdl1	kL	Code der Zeile-ausfügen-Taste
key_down	kcud1	kd	Code der Pfeil-nach-unten-Taste
key_eic	krmir	kM	Code von *rmir* oder *smir* (Einfügemodus aus/ein)
key_eol	kel	kE	Code der Löschen-bis-Zeilenende-Taste
key_eos	ked	kS	Code der Löschen-bis-Bildschirmende-Taste
key_f0	kf0	k0	Code der Funktionstaste f0
key_f1	kf1	k1	Code der Funktionstaste f1
key_f10	kf10	ka	Code der Funktionstaste f10
key_f2	kf2	k2	Code der Funktionstaste f2
key_f3	kf3	k3	Code der Funktionstaste f3
key_f4	kf4	k4	Code der Funktionstaste f4
key_f5	kf5	k5	Code der Funktionstaste f5
key_f6	kf6	k6	Code der Funktionstaste f6
key_f7	kf7	k7	Code der Funktionstaste f7
key_f8	kf8	k8	Code der Funktionstaste f8
key_f9	kf9	k9	Code der Funktionstaste f9
key_home	khome	kh	Code der HOME-Taste (Bildschirmanfang)
key_ic	kich1	kI	Code der Zeichen-einfügen- bzw. Einfügemodus-einschalten-Taste
key_il	kil1	kA	Code der Zeile-einfügen-Taste
key_left	kcub1	kl	Code der Pfeil-nach-links-Taste
key_ll	kll	kH	Code der Letzte-Zeile-erste-Spalte-Taste
key_npage	knp	kN	Code der Nächste-Seite-Taste
key_ppage	kpp	kP	Code der Seite-zurück-Taste
key_right	kcuf1	kr	Code der Pfeil-nach-rechts-Taste
key_sf	kind	kF	Code der Vorwärts/nach-unten-Scrollen-Taste
key_sr	kri	kR	Code der Rückwärts/nach-oben-Scrollen-Taste
key_stab	khts	kT	Code der Tabulator-setzen-Taste
key_up	kcuu1	ku	Code der Pfeil-nach-oben-Taste
keypad_local	rmkx	ke	Tastenübertragungs-Modus ausschalten
keypad_xmit	smkx	ks	Tastenübertragungsmodus einschalten
lab_f0	lf0	10	Beschriftung von Funktionstaste f0 wenn ungleich "f0"
lab_f1	lf1	11	Beschriftung von Funktionstaste f1 wenn ungleich "f1"
lab_f10	lf10	1a	Beschriftung von Funktionstaste f10 wenn ungleich "f10"
lab_f2	lf2	12	Beschriftung von Funktionstaste f2 wenn ungleich "f2"
lab_f3	lf3	13	Beschriftung von Funktionstaste f3 wenn ungleich "f3"
lab_f4	lf4	14	Beschriftung von Funktionstaste f4 wenn ungleich "f4"
lab_f5	lf5	15	Beschriftung von Funktionstaste f5 wenn ungleich "f5"
lab_f6	lf6	16	Beschriftung von Funktionstaste f6 wenn ungleich "f6"
lab_f7	lf7	17	Beschriftung von Funktionstaste f7 wenn ungleich "f7"
lab_f8	lf8	18	Beschriftung von Funktionstaste f8 wenn ungleich "f8"
lab_f9	lf9	19	Beschriftung von Funktionstaste f9 wenn ungleich "f9"
meta_on	smm	mm	Meta-Modus einschalten (8. Bit)
meta_off	rmm	mo	Meta-Modus ausschalten
newline	nel	nw	Neue-Zeile; wie *cr* plus *lf*
pad_char	pad	pc	Füllzeichen (ungleich Null)
parm_char	dch	DC	1 Zeichen ausfügen (PG*)
parm_delete_line	dl	DL	1 Zeile ausfügen (PG*)
parm_down_cursor	cud	DO	Schreibmarke #1 Zeilen nach unten(PG*)
parm_ich	ich	IC	#1 Leerzeichen einfügen (PG*)
parm_index	indn	SF	#1 Zeilen vorwärts scrollen (PG)
parm_insert_line	il	AL	#1 Leerzeilen einfügen (PG*)

parm_left_cursor	cub	LE	Schreibmarke #1 Stellen nach links (PG)
parm_right_cursor	cuf	RI	Schreibmarke #1 Stellen nach rechts (PG*)
parm_rindex	rin	SR	#1 Zeilen rückwärts scrollen (PG)
parm_up_cursor	cuu	UP	Schreibmarke #1 Zeilen nach oben (PG*)
pkey_key	pfkey	pk	Belegen der Funktionstaste #1: Zeichenkette #2 schreiben
pkey_local	pfloc	pl	Belegen der Funktionstaste #1: Zeichenkette #2 ausführen
pkey_xmit	pfx	px	Belegen der Funktionstaste #1: Zeichenkette #2 übertragen
print_screen	mc0	ps	Bildschirminhalt ausdrucken
prtr_off	mc4	pf	Drucker ausschalten
prtr_on	mc5	po	Drucker einschalten
repeat_char	rep	rp	Zeichen #1 #2-Mal wiederholen (PG*)
reset_1string	rs1	r1	Datensichtstation in Normalzustand rücksetzen
reset_2string	rs2	r2	Datensichtstation in Normalzustand rücksetzen
reset_3string	rs3	r3	Datensichtstation in Normalzustand rücksetzen
reset_file	rf	rf	Name der Datei, die die Reset-Zeichenketten enthält
restore_cursor	rc	rc	Schreibmarke rücksetzen auf Position des letzten *sc*
row_address	vpa	cv	Absolute vertikale Position (auf Zeile) (PG)
save_cursor	sc	sc	Schreibmarkenposition sichern (P)
scroll_forward	ind	sf	Scrollen nach oben (P)
scroll_reverse	ri	sr	Scrollen nach unten (P)
set_attributes	sgr	sa	Darstellungsarten definieren (PG9)
set_tab	hts	st	Tabulator in allen Zeilen in der aktuellen Spalte setzen
set_window	wind	wi	Aktuelles Fenster: Zeilen #1-#2, Spalten #3-#4
tab	ht	ta	Tabulator auf die nächste Hardware-Tabulatorposition (Spalte 1,9,17,25 usw.)
to_status_line	tsl	ts	Sprung in Statuszeile, Spalte #1
underline_char	uc	uc	1 Zeichen unterstreichen und dann 1 Stelle nach rechts
up_half_line	hu	hu	Halbe Zeile nach oben
init_prog	iprog	iP	Pfadname des Initialisierungsprogramms
key_a1	ka1	K1	Links oben virtueller Funktionstasten-block
key_a3	ka3	K3	Rechts oben virtueller Funktionstasten-block
key_b2	Kb2	K2	Mitte virtueller Funktionstastenblock
key_c1	kc1	K4	Links unten virtueller Funktionstasten-block
key_c3	kc3	K5	Rechts unten virtueller Funktionstasten-block
prtr_non	mc5p	pO	Drucker für #1 Bytes einschalten

Beispiel

Das folgende Beispiel zeigt einen der komplexeren Einträge in der *terminfo*-Datei; er beschreibt ein *Concept-100*-Gerät.

```
concept100|c100|concept|c104|c100-4p|concept 100,
     am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
     cols#80, cr=^M$<9>, cub1=^H, cud1=^J,cuf1=\E=,
     cup=\Ea%p1%' '%+%c%p2%' '%+%c,
     cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
     ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\Ek$<20>\EK, ht=\t$<8>,
     il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
     is2=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200\Eo\47\E,
     kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
     kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
     lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
     rev=\ED, rmcup=\Ev$<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
     rmso=\Ed\Ee, rmul=\Eg, sgr0=\EN\200,
     smcup=\EU\Ev 8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
     smul=\EG, tabs, ul, vt#8, xenl,
```

Bei Einträgen, die sich über mehrere Zeilen erstrecken, müssen alle Zeilen mit Ausnahme der ersten mit Leerzeichen oder Tabulatoren beginnen. Kommentarzeilen müssen mit "#" beginnen. Es gibt drei Arten von Feldern in *terminfo*, die die Eigenschaften von Datenstatonen beschreiben:

- die Booleschen Felder zeigen das Vorhandensein bestimmter Funktionen in der Datensichtstation an
- die numerischen Felder geben die Bildschirmgröße oder die Dauer bestimmter Verzögerungen an
- die Felder mit Steuerzeichenfolgen für bestimmte Bildschirmoperationen

Typen von Datensichtstations-Eigenschaften

Allen Datensichtstations-Eigenschaften sind bestimmte Namen zugeordnet. So bedeutet *am* im oben aufgeführten Beispiel, daß das *Concept*-Gerät über eine automatische Randeinstellung verfügt, d.h., am Zeilenende erfolgt automatisch ein Wagenrücklauf mit Zeilenvorschub. Numerische Eigenschaften werden mit anschließendem "#" und danach dem entsprechenden Wert angegeben. Im obigen Beispiel wird mit *cols* angegeben, daß der Bildschirm des *Concept*-Geräts 80 Spalten hat.

Eigenschaften, die mit Steuerzeichenfolgen angegeben werden, z.B. *el* (Löschen bis Zeilenende), werden in folgender Form angegeben: der 2 Zeichen lange Code, dann '=', danach eine mit ',' endende Zeichenkette. Eine Verzögerung (in Millisekunden) kann an beliebiger Stelle in der Form $< > angegeben werden (z.B. *el=\EK$<3>*); die für die Verzögerung erforderlichen Füllzeichen werden von *tputs* geliefert. Eine Verzögerung kann entweder als Zahl, z.B. '20', oder als Zahl mit anschließendem '*', z.B. '3*', angegeben werden wobei '*' bedeutet, daß die erforderliche Anzahl Füllzeichen proportional zur Anzahl der von der Operation betroffenen Zeilen ist, und die angegebene Menge zu verstehen ist als Anzahl Füllzeichen pro betroffene Einheit. Auch beim Einfügen von Zeichen gibt der Faktor die Anzahl der betroffenen Zeilen an und ist, sofern die Datensichtstation nicht über die Eigenschaft *xenl* verfügt und diese auch von der Software verwendet wird, immer 1. Bei Angabe von '*' ist es manchmal nützlich, eine Verzögerung in der Form '3.5' anzugeben, d.h. eine Verzögerung pro Einheit auf Zehntel-Millisekunden genau anzugeben. Dabei darf nicht mehr als eine Dezimalstelle angegeben werden.

Für die einfache Kodierung von Steuerzeichenfolgen stehen mehrere Escape-Sequenzen zur Verfügung. Ein ESCAPE-Zeichen kann sowohl mit *\E* als auch mit *\e* angegeben werden, *^x* entspricht Control-x, wobei *x* für einen beliebigen Control-Wert steht, und die Sequenzen *\n \l \r \t \b \f \s* erzeugen Neue-Zeile, Zeilenvorschub, Wagenrücklauf, Tabulator, Backspace, Seitenvorschub und Leerzeichen.
Ferner ist '\ ' der Code für '^'; '\\' ist der Code für '\'; '\,' ist der Code für Komma; '\:' der für Doppelpunkt und '\0' ist der Code für Null ('\0' erzeugt '\200', was bei den meisten Datensichtstationen als Ziffer Null interpretiert wird und nicht als Schlußzeichen einer Zeichenkette). Generell können Zeichen mit '\', gefolgt von drei Oktalziffern, angegeben werden.

Manchmal ist es notwendig eine bestimmte Eigenschaft zu deaktivieren. Dies wird durch einen Punkt vor dem Namen der betreffenden Eigenschaft realisiert (s. das zweite *ind* im obigen Beispiel).

Erstellen von Datensichtstations-Beschreibungen

Im Folgenden wird erläutert, wie man die Beschreibungen der Datensichtstationen erstellt. Die effektivste Methode ist, die Beschreibung einer ähnlichen Datensichtstation in der *terminfo* zu kopieren, die neue schrittweise aufzubauen und Teilbeschreibungen mit *vi* auf Fehler hin zu überprüfen. Bei einer sehr ungewöhnlichen Datensichtstation kann sich jedoch herausstellen, daß sie mit der *terminfo*-Datei nur unzulänglich beschrieben werden kann oder das in *vi* Fehler auftreten. Eine neue Datensichtstationsbeschreibung läßt sich auf einfache Art testen, indem Sie mit der Umgebungsvariablen *TERMINFO* den Pfadnamen eines Dateiverzeichnisses angeben, das die übersetzte Beschreibung, an der Sie arbeiten, enthält; dann greifen die Programme auf die Variable statt auf */usr/lib/terminfo* zu. Ein "Härtetest", um die korrekte Verzögerung für das Einfügen von Zeilen zu ermitteln (sofern sie nicht vom Hersteller angegeben ist), sieht folgendermaßen aus: Eine Datei mit ca. 100 Zeilen mit 9600 Baud editieren, etwa 16 Zeilen in der Mitte des Bildschirms ausfügen, dann mehrmals hintereinander schnell die 'u'-Taste drücken. Wenn dies den Bildschirm durcheinander bringt, heißt das, daß eine größere Verzögerung benötigt wird. Einen ähnlichen Test kann man für die Funktion zum Einfügen von Zeichen durchführen.

Grundlegende Datensichtstations-Eigenschaften

Mit der numerischen Eigenschaft *cols* gibt man die Anzahl Spalten pro Zeile für die Datensichtstation an. Handelt es sich um eine Datensichtstation, so wird die Anzahl der Bildschirmzeilen mit *lines* angegeben. Springt die Schreibmarke bei Erreichen des rechten Randes an den Anfang der nächsten Zeile, sollte *am* angegeben werden. Steht die Schreibmarke nach dem Löschen des Bildschirms am Bildschirmanfang, so ist *clear* anzugeben. Können mehrere Zeichen übereinandergeschrieben werden (ohne daß das erste Zeichen durch das zweite ersetzt wird), so ist *os* anzugeben. Für eine Druckerstation ohne Bildschirmaufzeichnung ist sowohl *hc* als auch *os* anzugeben. (*os* kann sowohl für Datensichtstationen mit Bildschirmspeicher als auch für Hardcopy-Geräte und *APL*-Stationen angegeben werden.) Gibt es einen Code für die Positionierung der Schreibmarke auf die erste Spalte der aktuellen Zeile, so ist dieser mit *cr* anzugeben. (Normalerweise ist das der Wagenrücklauf, d.h. ^M.) Gibt es einen Code, der ein akustisches Signal erzeugt (Klingel, Piepston usw.), so ist dieser mit *bel* anzugeben.

Ein eventuell vorhandener Code für die Positionierung der Schreibmarke um eine Stelle nach links (z.B. Backspace) ist mit *cubl* anzugeben. Entsprechend sind Codes für die Positionierung der Schreibmarke nach rechts, oben bzw. unten mit *cufl*, *cuul* bzw. *cudl* anzugeben. Dabei sollte der Text, über den die Schreibmarke bei der Positionierung hinweg läuft, unverändert bleiben. So wird man normalerweise nicht '*cufl*= ' angeben, da dadurch die Zeichen, die die Schreibmarke passiert, mit Leerzeichen überschrieben würden.

In diesem Zusammenhang ist auf folgenden wichtigen Punkt hinzuweisen: die in *terminfo* kodierten lokalen Schreibmarkenbewegungen sind hinsichtlich des linken und des oberen Bildschirmrandes undefiniert. Deshalb sollte ein Programm nie einen Rückwärtsschritt über den linken Bildschirmrand hinaus versuchen (es sei denn, *bw* ist angegeben) oder nach oben über den oberen Bildschirmrand hinaus. Ein Scrollen nach oben wird dadurch erreicht, daß im Programm erst auf die Bildschirmecke links unten positioniert wird und dann die Zeichenkette *ind* (Index) abgesetzt wird.

Ein Scrollen nach unten wird so programmiert, daß erst auf die linke obere Bildschirmecke positioniert wird und dann die Zeichenkette *ri* (Index invers) abgesetzt wird. *ind* und *ri* sind undefiniert, wenn nicht vorher auf die entsprechende Bildschirmecke positioniert wird.

Zu diesen Scroll-Sequenzen existieren auch parametrisierte Varianten, nämlich *indn* und *rin*. Diese haben dieselbe Bedeutung wie *ind* bzw. *ri*, außer daß sie eine bestimmte Anzahl Zeilen entsprechend der Parameterangabe nach oben bzw. unten blättern. Auch sie sind undefiniert, wenn nicht vorher auf die entsprechende Bildschirmecke positioniert wird.

Mit *am* wird angegeben, ob die Bildschirmmarke bei der Ausgabe am rechten Bildschirmrand stehen bleibt; diese Angabe gilt jedoch nicht notwendigerweise auch für ein *cufl* in der letzten Spalte. Die einzige in Bezug auf den linken Rand definierte lokale Schreibmarkenbewegung ist die mit *bw* angegebene. Diese Angabe bewirkt, daß ein *cubl* am linken Rand an den rechten Rand der vorausgehenden Zeile springt. Ohne die Angabe von *bw* ist diese Positionierung undefiniert.

Diese Funktion kann z.B. gut dazu verwendet werden, einen Rahmen am Bildschirmrand entlang zu erzeugen. Verfügt die Datensichtstation über umschaltbare automatische Randeinstellungen, so wird in der *terminfo*-Datei normalerweise angenommen, daß diese eingeschaltet sind (Angabe von *am*). Gibt es ein Kommando, das auf die erste Spalte der nächsten Zeile positioniert, so kann dies mit *nel* (Neue-Zeile) angegeben werden. Dabei ist unwichtig, ob das Kommando den Rest der aktuellen Zeile löscht. Auf diese Weise kann eine brauchbare Positionierung auf den Anfang der nächsten Zeile auch bei einer Datensichtstation, die kein *cr* oder *lf* hat, erreicht werden.

Mit diesen Angaben können die Eigenschaften von Hardcopygeräten und "dummen" Datensichtstationen ausreichend beschrieben werden.

Parametrisierte Steuerzeichenfolgen

Schreibmarken-Positionierungen und andere Funktionen, die mit datensichtstations-spezifischen Parametern versorgt werden müssen, werden mit parametrisierten Steuerzeichenfolgen beschrieben, wobei wie in *printf()* Escape-Sequenzen des Formats %x verwendet werden. So verwendet z.B. die Schreibmarken-Positionierung mit *cup* zwei Parameter: die Zeile und die Spalte, auf die positioniert werden soll. (Zeilen und Spalten werden beginnend mit Null durchnummeriert; die Angaben beziehen sich auf den für den Benutzer sichtbaren Bildschirm und nicht etwa auf einen unsichtbaren Speicherbereich.) Erlaubt die Datensichtstation eine speicherbezogene Schreibmarken-Positionierung, so ist diese mit *mrcup* anzugeben.

Die Parameterversorgung erfolgt mittels eines Kellers (stack) und spezieller %-Codes, wobei üblicherweise jeweils einer der Parameter im Keller eingetragen und dann im entsprechenden Format ausgedruckt wird. In vielen Fällen sind jedoch kompliziertere Operationen erforderlich.

In der nachfolgenden Tabelle werden zwei Operationen *push()* und *pop()* verwendet, um die Operationen auf dem Keller darzustellen.

Diese Operationen haben die folgende Bedeutung:

push(x):
Schiebt das Argument *x* in den Keller. *x* ist danach das oberste Kellerelement. Alle bisher im Keller befindlichen Elemente werden um eine Position nach "unten" verschoben.

pop():
Liefert das oberste Kellerelement als Ergebnis zurück und entfernt dieses aus dem Keller. Alle übrigen Kellerelemente werden durch diese Operation um eine Position nach "oben" verschoben.

Die %-Codes haben folgende Bedeutung:

% %	'%' ausgeben
%d	oberstes Kellerelement ausgeben wie in printf()
%2d	oberstes Kellerelement ausgeben wie %2d
%3d	oberstes Kellerelement ausgeben wie %3d
%02d	wie in printf()
%03d	wie in printf()
%c	oberstes Kellerelement ausgeben als %c
%s	oberstes Kellerelement ausgeben als %s
%p[1-9]	*i*ten Parameter in Keller eintragen
%P[a-z]	Variable [a-z] mit pop()-Element versorgen
%g[a-z]	Variable [a-z] im Keller eintragen
%'c'	Zeichenkonstante c
%{nn}	ganzzahlige Konstante nn
%+ %- %* %/ %m	arithmetische Operatoren (%m = modulo)
%& %| %^	Bitoperatoren
%= %> %<	logische Operatoren Die mit diesen Operatoren durchgeführten Operationen folgen alle dem Schema: push(pop() op pop())
%! %~	unäre Operatoren Operationen mit unären Operatoren folgen dem Schema: push(op pop())
%i	zu den ersten beiden Parametern 1 addieren (für ANSI-Datensichtstationen)
%? *expr* %t *thenpart* %e *elsepart* %;	if-then else, %e *elsepart* ist wahlweise; else-if's können auch wie in Algol68 geschrieben werden: %? c_1 %t b_1 %e c_2 %t b_2 %t b_3 %e c_4 %t b_4 %e %; c_i sind Bedingungen, b_i sind Rümpfe.

Binäroperationen werden im Postfix-Format ausgeführt, wobei die Operanden in der üblichen Reihenfolge auftreten. D.h., um das Ergebnis x-5 zu erhalten, verwendet man "%gx%{5}%-".

Als Beispiel sei hier die *HP2645* betrachtet: Um die Schreibmarke auf Zeile 3, Spalte 12 zu positionieren, muß \E&a12c03Y plus eine Verzögerung von 6 Millisekunden abgesetzt werden. Man beachte, daß hier Zeile und Spalte in umgekehrter Reihenfolge angegeben werden müssen und beide als zwei Ziffern ausgegeben werden. Bei *cup* ist daher anzugeben cup=6\E&%p2%03dc%p1%2dY.

Bei Datensichtstationen, die %c verwenden, müssen *cubl* (Schreibmarke um eine Stelle nach links) und *cuul* (Schreibmarke eine Bildschirmzeile nach oben) möglich sein, da die Übertragung von \n^D und \r nicht immer zuverlässig ist (sie könnten vom System geändert oder weggeworfen werden). (Die von *terminfo* verwendeten Bibliotheksfunktionen setzen den Datensichtstationsmodus so, daß Tabulatorexpansion in keinem Fall durchgeführt wird; die Übertragung von \t ist daher sicher.)

Zum Schluß sei die *LSI ADM-3a* betrachtet: Hier werden Zeilen und Spaltenangaben durch ein Leerzeichen voneinander getrennt; demzufolge ist bei *cup* anzugeben
cup=\E=%p1%' '%+%c%p2%' '%+%c.
Nachdem '\E=' abgesetzt ist, wird der erste Parameter im Keller eingetragen, dann der ASCII-Code für Leerzeichen (32), dann werden beide addiert (wobei die Summe anstelle der beiden Werte im Keller eingetragen wird) und schließlich die Summe als Zeichen ausgegeben. Danach werden dieselben Operationen für den zweiten Parameter durchgeführt. Kompliziertere arithmetische Operationen sind mit dem Keller ebenfalls möglich.

Erlaubt die Datensichtstation eine absolute Zeilen- bzw. Spaltenpositionierung, so kann dies mit den nur einen Parameter erfordernden Steuerzeichenfolgen *hpa* (horizontale Positionierung absolut) und *vpa* (vertikale Positionierung absolut) angegeben werden. Diese Angaben sind manchmal kürzer als die allgemeineren Sequenzen mit zwei Parametern (s. Beispiel zu *HP2645*) und ihre Verwendung ist der von *cup* vorzuziehen. Erlaubt die Datensichtstation lokale Schreibmarken-Positionierungen (z.B. *n* Stellen nach rechts), so sind diese mit *cud*, *cub*, *cuf* und *cuu* anzugeben, wobei der einzige Parameter die Anzahl Stellen angibt, um die die Schreibmarke bewegt werden soll. Diese Funktionen sind insbesondere bei Datensichtstationen nützlich, die nicht über die *cup*-Funktion verfügen.

Schreibmarken-Positionierungen

Verfügt die Datensichtstation über eine Einrichtung, die die Schreibmarke schnell an den Bildschirmanfang positioniert, so kann diese mit *home* angegeben werden; eine entsprechende Einrichtung zur Positionierung der Schreibmarke auf die untere linke Bildschirmecke wird mit *ll* angegeben. Dabei kann diese Positionierung auch durch ein *cuul* ab Bildschirmanfang erreicht werden; dies sollte jedoch nie direkt in einem Programm erfolgen (es sei denn, *ll* impliziert *cuul*), da in einem Programm keine Annahmen bezüglich der Auswirkung einer Positionierung vom Bildschirmanfang nach oben getroffen werden können.

Hinweis
home positioniert auf dieselbe Stelle wie die Angabe (0,0), nämlich auf die obere linke Ecke des Bildschirms, nicht im Speicher.

Bereich löschen

Erlaubt die Datensichtstation das Löschen ab aktueller Position bis Zeilenende, wobei die Schreibmarke an der aktuellen Position bleibt, sollte dies mit *el* angegeben werden. Ist das Löschen ab aktueller Position bis Bildschirmende möglich, so ist dies mit *ed* anzugeben. *ed* ist so definiert, daß es nur ab der ersten Spalte einer Zeile wirkt. D.h. man kann diese Funktion auch simulieren, indem man ein Kommando zum Löschen einer großen Anzahl Zeilen eingibt, wenn *ed* selbst nicht vorhanden ist.

Zeile ein-/ausfügen

Eine Funktion, mit der eine Leerzeile oberhalb der aktuellen Zeile eingefügt wird, sollte mit *il1* angegeben werden. Die entsprechende Steuerzeichenfolge ist so definiert, daß sie nur ab der ersten Spalte einer Zeile wirkt. Die Schreibmarke sollte dann in der neuen (leeren) Zeile stehen. Eine Funktion, mit der die aktuelle Zeile ausgefügt wird, sollte mit *dl1* angegeben werden. Auch hier ist die entsprechende Steuerzeichenfolge so definiert, daß sie nur ab der ersten Spalte der zu löschenden Zeile wirkt. Entsprechende Funktionen, bei denen mit einem Parameter die Anzahl ein- bzw. auszufügender Zeilen angegeben wird, können mit *il* und *dl* angegeben werden. Kann bei der Datensichtstation ein bestimmter Scroll-Bereich gesetzt werden, so wird das entsprechende Kommando mit *csr* beschrieben; dabei werden zwei Parameter verwendet, die die erste und die letzte Zeile des Scroll-Bereichs angeben.

Leider ist die Position der Schreibmarke nach diesem Kommando undefiniert. Dasselbe Kommando kann auch dazu verwendet werden, eine Zeile ein- oder auszufügen; ferner sind *sc* und *rc* (Schreibmarkenposition sichern bzw. rücksetzen) in diesem Zusammenhang nützlich. Bei vielen Datensichtstationen, die keine spezielle Zeilenein-/ausfüge-Funktion haben, können am oberen oder unteren Bildschirmrand Zeilen auch mit *ri* bzw. *ind* eingefügt werden; dieses Verfahren ist oft auch auf Stationen mit Zeilenein-/ausfüge-Funktionen schneller.

Kann an einer Datensichtstation ein Fenster als Speicherbereich definiert werden, auf den sich alle Kommandos auswirken, so sollte dies mit der parametrisierten Steuerzeichenfolge *wind* angegeben werden. Die vier hier benötigten Parameter bezeichnen die erste und die letzte Zeile und die erste und letzte Spalte im Speicherbereich (in dieser Reihenfolge).

Kann der Bildschirm-Inhalt über den Bildschirm hinaus nach oben verschoben und festgehalten werden, so ist *da* anzugeben. Kann der Bildschirm-Inhalt über den Bildschirm hinaus nach unten verschoben und festgehalten werden, so ist *db* anzugeben. Dies bedeutet, daß beim Ausfügen einer Zeile oder beim Vorwärts-Scrollen beschriebene Zeilen von unten - bzw. beim Rückwärts-Scrollen mit *ri* von oben - auf den Bildschirm ausgegeben werden können.

Zeichen ein-/ausfügen

Mittels *terminfo* können zwei Grundtypen von Datensichtstationen beschrieben werden, die hinsichtlich Zeichenein-/ausfüge-Funktionen programmierbar sind. Die gebräuchlichsten Zeichenein-/ausfüge-Operationen wirken nur auf die Zeichen in der aktuellen Zeile und verschieben die Zeichen am Ende der Zeile so, daß sie verloren gehen. Bei anderen Datensichtstationen wird zwischen über die Tastatur eingegebenen Leerzeichen und nicht belegten Leerstellen auf dem Bildschirm in der Weise unterschieden, daß beim Ein- bzw. Ausfügen nur eine nicht belegte Leerstelle überschrieben bzw. verdoppelt wird. Zu welchem Typ Ihre Datensichtstation gehört, können Sie dadurch feststellen, daß Sie zunächst den Bildschirm löschen und dann Text, unterbrochen durch Schreibmarken-Positionierungen, eingeben. Geben Sie

abc → → def

ein, wobei Sie zwischen 'abc' und 'def' nicht die Leerzeichentaste, sondern die Pfeil-nach-rechts-Taste betätigen. Dann bewegen Sie die Schreibmarke vor 'abc' und schalten den Einfügemodus ein.

Wenn Sie jetzt weiteren Text eingeben und dadurch der Rest der Zeile entsprechend verschoben wird, so daß am Zeilenende Zeichen verloren gehen, dann kann Ihre Datensichtstation nicht zwischen eingegebenen Leerzeichen und nicht belegten Leerstellen unterscheiden. Wenn 'abc' an 'def' heran verschoben und dann durch weitere Eingaben beide zusammen über das Ende der aktuellen Zeile hinaus auf die nächste Zeile verschoben werden, so gehört Ihre Datensichtstation zum zweiten Typ; d.h., Sie sollten die Eigenschaft *in* ('insert null') eingeben. Es handelt sich hier zwar um zwei von der Logik her verschiedene Eigenschaften (einzeiliger bzw. mehrzeiliger Einfügemodus und Sonderbehandlung von nicht belegten Leerstellen), aber bisher ist uns keine Datensichtstation bekannt, deren Einfügemodus nicht mit einem einzigen Merkmal beschrieben werden konnte.

Mit *terminfo* lassen sich sowohl Datensichtstationen beschreiben, die einen eigenen Einfügemodus haben, beschreiben, als auch solche, die nur eine einfache Sequenz zum Einfügen einer Leerstelle in der aktuellen Zeile absetzen. Mit *smir* gibt man die Sequenz zum Einschalten des Einfügemodus an, mit *rmir* die Sequenz zum Ausschalten des Einfügemodus und mit *ich1* eine beliebige Sequenz, die unmittelbar vor dem einzufügenden Zeichen abgesetzt werden muß. Für die meisten Datensichtstationen mit einem eigenen Einfügemodus ist *ich1* nicht anzugeben, wohl aber für solche, die eine Sequenz zum Einfügen einer Leerstelle auf dem Bildschirm absetzen. Erlaubt Ihre Datensichtstation beide Funktionen, so ist der Einfügemodus *ich1* vorzuziehen. Beides sollte nur dann angegeben werden, wenn die Datensichtstation auch wirklich beides zusammen benötigt. Eventuell nach dem Einfügen erforderliche Verzögerungen sind - in Millisekunden - mit *ip* anzugeben. Hiermit kann man auch andere Zeichenketten angeben, die ggf. nach dem Einfügen eines Zeichens abgesetzt werden müssen. Muß an Ihrer Datensichtstation sowohl der Einfügemodus eingeschaltet als auch ein spezieller Code vor jedem einzufügenden Zeichen abgesetzt werden, dann können Sie sowohl *smir/rmir* als auch *ich1* angeben und beides wird ausgewertet. *ich* mit einem Parameter *n* wiederholt die Wirkung von *ich1 n* Mal.

Manchmal ist es erforderlich, daß man im Einfügemodus an anderer Stelle in derselben Zeile Zeichen ausfügt (z.B. wenn nach der Einfügestelle ein Tabulator gesetzt ist). Erlaubt Ihre Datensichtstation Positionieren im Einfügemodus, so können Sie das Einfügen durch die Angabe von *mir* beschleunigen. Geben Sie *mir* nicht an, so wirkt sich dies nur auf die Geschwindigkeit, mit der Sie Einfügungen vornehmen können, aus. Bei einigen Datensichtstation kann *mir* jedoch aufgrund der Funktionsweise ihres Einfügemodus nicht angegeben werden.

Schließlich kann man mit *dch1* die Funktion 'Zeichen ausfügen', mit *dch* plus Parameter *n* die Funktion '*n* Zeichen ausfügen' angeben und mit *smdc* bzw. *rmdc* die Kommandos zum Ein- bzw. Ausschalten des Ausfügemodus (Modus, der eingeschaltet werden muß, damit *dch1* funktioniert).

Ein Kommando zum Löschen von *n* Zeichen (sprich Überschreiben mit *n* Leerzeichen, ohne Veränderung der Schreibmarkenposition) kann man mit *ech* plus 1 Parameter angeben.

Hervorhebungen, Unterstreichen und optische Signale

Verfügt Ihre Datensichtstation über eine oder mehrere Darstellungsarten, so können diese auf verschiedene Art und Weise angegeben werden. Eine Darstellungsart sollte als *STANDOUT* (Hervorhebungsmodus) definiert werden - ein klares, kontrastreiches Anzeigeformat, das angenehm für die Augen ist und bei dem Fehlermeldungen und andere wichtige Anzeigen besonders hervorgehoben werden. Wenn Sie zwischen verschiedenen Möglichkeiten wählen können, empfehlen wir Inversdarstellung plus halbhell oder nur Inversdarstellung. Die Steuerzeichenfolgen zum Ein- und Ausschalten des Hervorhebungsmodus werden mit *smso* bzw. *rmso* angegeben. Bleiben nach dem Absetzen des Codes zum Ein- bzw. Ausschalten des Hervorhebungsmodus ein oder sogar zwei Leerstellen auf dem Bildschirm, so kann man sich durch die Angabe von *xmc* über die Anzahl vorhandener Leerstellen informieren.

Codes zum Ein- und Ausschalten des Unterstreichungsmodus können mit *smul* bzw. *rmul* angegeben werden. Besitzt die Datensichtstation einen Code zum Unterstreichen des aktuellen Zeichens und anschließender Schreibmarkenpositionierung um eine Stelle nach rechts, so ist dieser mit *uc* anzugeben.

Weitere Hervorhebungsarten und die entsprechenden Steuerzeichenfolgen sind *blink* (Blinken), *bold* (fett oder hell), *dim* (halbhell), *invis* (Bildschirm dunkel bzw. Text unsichtbar), *prot* (geschützt), *rev* (invers), *sgr0* (alle Darstellungsarten ausgeschaltet), *smacs* (alternativer Zeichensatz eingeschaltet) und *rmacs* (alternativer Zeichensatz ausgeschaltet). Das Einschalten einer dieser Darstellungsarten kann (aber muß nicht) dazu führen, daß andere ausgeschaltet werden.

Ist eine Sequenz vorhanden, mit der beliebige Kombinationen von Darstellungsarten gesetzt werden, so sollte diese mit *sgr* (Merkmale setzen) angegeben werden. Die Funktion arbeitet mit 9 Parametern, wobei jeder Parameter entweder den Wert 0 oder den Wert 1 annimmt, je nachdem, ob die entsprechende Darstellungsart aus- oder eingeschaltet ist. Die 9 Parameter beziehen sich (in dieser Reihenfolge) auf:

Hervorhebung, Unterstreichen, invers, Blinken, halbhell, fett/hell, dunkel, geschützt, alternativer Zeichensatz.

sgr muß jedoch nur für diejenigen Darstellungsarten angegeben werden, die mit gesonderten Kommandos angesprochen werden.

Datensichtstationen mit der Eigenschaft *xmc* ('magic cookie glitch') legen bei Empfang von Steuersequenzen für bestimmte Darstellungsarten spezielle Zeichen zur Auswahl des Anzeige-Algorithmus (sog. "Magic Cookies") ab (statt etwa für jedes einzelne Zeichen ein Extrabit zu setzen). Bei manchen Datensichtstationen wird der Hervorhebungsmodus bei der Eingabe von 'Neue-Zeile' oder einer Schreibmarkenbewegung automatisch verlassen. In Programmen, die den Hervorhebungsmodus benutzen, sollte dieser vor Schreibmarken-Positionierungen oder 'Neue-Zeile' ausgeschaltet werden, es sei denn, die Eigenschaft *msgr* ist vorhanden; diese gibt an, daß Positionierungen im Hervorhebungsmodus möglich sind.

Verfügt die Datensichtstation über eine Möglichkeit, den gesamten Bildschirm blinken zu lassen, um einen Fehler anzuzeigen (als Ersatz für ein akustisches Signal), so kann dies mit *flash* angegeben werden; die Schreibmarke darf durch das Blinken nicht bewegt werden.

Soll die Schreibmarke mehr als normal auffallen, wenn sie sich nicht in der untersten Zeile befindet (z.B. damit ein nicht blinkender Unterstrich in einen leichter zu findenden Block oder in einen blinkenden Unterstrich umgewandelt wird), so ist die entsprechende Steuerzeichenfolge mit *cvvis* anzugeben. Kann die Schreibmarke am Bildschirm vollständig unterdrückt werden, so ist dies mit *civis* anzugeben. Mit *cnorm* werden beide Funktionen wieder ausgeschaltet.

Muß sich eine Datensichtstation beim Ablaufen eines Programms, das diese Eigenschaften verwendet, in einem speziellen Modus befinden, so sind die Codes zum Ein- und Ausschalten dieses Modus mit *smcup* bzw. *rmcup* anzugeben. Dies ist erforderlich, wenn mehrere Speicherseiten bearbeitet werden. Verwendet die Datensichtstation nur speicherbezogene und keine bildschirmbezogene Schreibmarken-Positionierung, so muß ständig ein Fenster in Bildschirmgröße vorhanden sein, damit die Schreibmarken-Positionierung funktioniert.

Wenn Ihre Datensichtstation ohne spezielle Codes Zeichen mit Unterstreichung erzeugen kann, obwohl nicht mehrere Zeichen übereinander geschrieben werden können, so müssen Sie *ul* angeben. Können übereinandergeschriebene Zeichen mit einem Leerzeichen gelöscht werden, so ist *eo* anzugeben.

Funktionstastenblock

Wenn die Datensichtstation mit einem Funktionstastenblock ausgestattet ist, dessen Tasten Codes übertragen, so kann diese Tatsache in *terminfo* angegeben werden. Sie sollten wissen, daß Datensichtstationen, deren Funktionstastenblock nur in Lokalbetrieb arbeitet, nicht behandelt werden können. Kann der Tastenblock zwischen Übertragen bzw. Nicht-Übertragen umgeschaltet werden, so sind die entsprechenden Codes mit *smkx* und *rmkx* anzugeben. Ist ein Umschalten nicht möglich, so wird angenommen, daß sich der Funktionstastenblock immer im Übertragungsmodus befindet. Die von den Pfeiltasten 'nach links', 'nach rechts', 'nach oben', 'nach unten' und 'Bildschirmanfang' abgesetzten Codes können mit *kcubl*, *kcufl*, *kcuul*, *kcudl* bzw. *khome* angegeben werden. Ist die Datensichtstation mit Funktionstasten vom Typ f0, f1, ..., f10 ausgestattet, so sind die damit abgesetzten Codes mit *kf0*, *kf1*, ..., *kf10* anzugeben. Sind die Funktionstasten anders als mit f0 bis f10 (Standard) beschriftet, so sind ihre Beschriftungen mit *lf0*, *lf1*, ..., *lf10* anzugeben.

Andere Sondertasten bzw. die von ihnen abgesetzten Codes können wie folgt angegeben werden: *kll* (linke untere Bildschirmecke), *kbs* (Backspace), *ktbc* (alle Tabulatoren löschen), *kctab* (Tabulator in dieser Spalte löschen), *kclr* ((Bildschirm) löschen), *kdch1* (Zeichen ausfügen), *kdl1* (Zeile ausfügen), *krmir* (Einfügemodus ausschalten), *kel* (löschen bis Zeilenende), *ked* (löschen bis Bildschirmende), *kich1* (Zeichen einfügen oder Einfügemodus einschalten), *kil1* (Zeile einfügen), *knp* (nächste Seite), *kpp* (vorhergehende Seite), *kind* (vorwärts/nach unten scrollen), *kri* (rückwärts/nach oben scrollen), *khts* (Tabulator in dieser Spalte setzen). Ferner können bei einem Funktionstastenblock mit einem 3-mal-3 Tastenfeld, zu dem u.a. die vier Pfeiltasten gehören, die übrigen fünf Tasten mit *ka1*, *ka3*, *kb2*, *kc1* und *kc3* angegeben werden. Diese Tasten sind in solchen Fällen nützlich, in denen die Funktionen eines 3-mal-3 Pfeiltastenfeldes benötigt werden.

Tabulatoren und Initialisierung

Bei Datensichtstationen, die mit hardwaremäßigen Tabulatoren ausgestattet sind, kann man das Kommando, das ein Vorrücken zum nächsten Tabulator bewirkt, mit *ht* (üblicherweise Control-I) angeben. Ein "Tabulator-rückwärts"-Kommando ('back_tab'), das einen Rücksprung zum letzten Tabulator bewirkt, kann mit *cbt* angegeben werden. Konventionsgemäß sollte in Programmen, die mit Datensichtstationen arbeiten, bei denen die Tabulatorexpansion rechnerintern erfolgt und nicht an die Datensichtstation übergeben wird, weder *ht* noch *cbt* verwendet werden, selbst wenn diese Eigenschaften vorhanden sind, weil die Tabulatoren u.U. vom Benutzer nicht korrekt gesetzt wurden. Verfügt die Datensichtstation über Hardware-Tabulatoren, die automatisch in jeder *n*-ten Spalte gesetzt werden, sobald die Station eingeschaltet wird, so ist mit der numerischen Funktion *it* die Position der Tabulatoren anzugeben. Von dieser Information ist es normalerweise abhängig, ob das *tset*-Kommando den Modus 'Hardware-Tabulatorexpansion' oder den Modus 'Tabulatoren setzen' einschaltet. Können die Tabulatoren im nicht-flüchtigen Speicher gesichert werden, so kann in der *terminfo*-Beschreibung davon ausgegangen werden, daß sie korrekt gesetzt sind.

Mit *is1*, *is2* und *is3* (Datensichtstations-Initialisierung), *iprog* (Pfadname eines Programms, das bei der Initialisierung abläuft) und *if* (Name einer Datei, die die mit *is* angegebenen Zeichenketten für die Initialisierung enthält) werden die Steuerzeichenfolgen angegeben, mit denen die Datensichtstation so initialisiert werden soll, daß der Rest der *terminfo*-Beschreibung zutrifft. Sie können mit dem *tset*-Programm übergeben werden. Die Übergabe erfolgt in der Reihenfolge: *is1*; *is2*; Setzen der Tabulatoren mit *tbc* und *hts*; *if*; Ablauf des Programms *iprog*; und schließlich *is3*. Der größte Teil der Initialisierung wird mit *is2* durchgeführt. Eine Wiederholung von Angaben beim Setzen spezieller Betriebsarten läßt sich dadurch vermeiden, daß man die Sequenzen für den normalen Betrieb in *is2* und diejenigen für Sonderfälle in *is1* and *is3* ablegt. Sequenzen, die ein Rücksetzen eines völlig unbekannten Zustandes analog zu *is2* und *if* veranlassen, können entsprechend mit *rs1*, *rs2*, *rf* und *rs3* angegeben werden. Sie werden vom *reset*-Programm abgesetzt, das im Falle von "Verklemmungen" der Datensichtstation aktiviert wird. Normalerweise werden Kommandos nur dann in *rs2* und *rf* angegeben, wenn sie unerwünschte Auswirkungen auf den Bildschirm haben und beim Login nicht benötigt werden.

Kommandos zum Setzen und Löschen von Tabulatoren können mit *tbc* (alle Tabulatoren löschen) und *hts* (Tabulator in allen Zeilen in der aktuellen Spalte setzen) angegeben werden. Ist für auf diese Weise beschreibbare Funktionen eine komplexere Sequenz erforderlich, so kann sie in *is2* oder *if* abgelegt werden.

Verschiedenes

Falls die Datensichtstation ein anderes Füllzeichen als das Nullzeichen erfordert, so kann dies mit *pad* angegeben werden. Dabei wird von einer mit *pad* angegebenen Zeichenkette nur das erste Zeichen verwendet.

Verfügt die Datensichtstation über eine gesonderte "Statuszeile", die normalerweise von Programmen nicht verwendet wird, so kann dies mit *hs* angegeben werden; die Statuszeile wird dann als Extrazeile unter der letzten Zeile verstanden, die mit normalen Schreibmarkenbewegungen ansprechbar ist. Mit *tsl* und *fsl* gibt man spezielle Steuerzeichenfolgen an, die auf den Anfang der Statuszeile positionieren bzw. aus der Statuszeile zurückspringen. Nach *fsl* muß die Schreibmarke an derselben Stelle stehen wie vor *tsl*.

Falls erforderlich, können zu diesem Zweck *sc* und *rc* in *tsl* und *fsl* integriert werden. *tsl* hat einen Parameter, der die Spalte in der Statuszeile angibt, auf die die Schreibmarke positioniert werden soll. Kann man auch in der Statuszeile Escape-Sequenzen und andere spezielle Kommandos, z.B. Tabulatoren, verwenden, so ist *eslok* anzugeben. Eine Zeichenkette, die die Statuszeile ausschaltet (oder auf andere Art löscht), sollte mit *dsl* angegeben werden. Spezielle Kommandos zum Sichern bzw. Wiederherstellen der Schreibmarkenposition sollten mit *sc* bzw. *rc* angegeben werden. Normalerweise wird davon ausgegangen, daß die Statuszeile dieselbe Länge wie die übrigen Bildschirmzeilen hat, z.B. *cols*. Ist dies nicht der Fall (weil z.B. die Datensichtstation keine ganze Zeile laden kann), so kann man die Länge der Statuszeile mit der numerischen Funktion *wsl* angeben.

Sind an der Datensichtstation Positionierungen um eine halbe Zeile nach oben oder unten möglich, so ist dies mit *hu* (halbe Zeile nach oben) und *hd* (halbe Zeile nach unten) anzugeben. Dies ist vor allem an Hardcopy-Geräten für das Drucken von hoch- und tiefgestellten Zeichen nützlich. Verfügt ein Hardcopy-Gerät über eine Seitenvorschub-Einrichtung, so ist diese mit *ff* (gewöhnlich Control-L) anzugeben.

Ein Kommando, das die mehrmalige Wiederholung eines Zeichens bewirkt (wodurch man Zeit bei der Übertragung einer großen Anzahl identischer Zeichen spart), wird mit der parametrisierten Zeichenkette *rep* angegeben. Dabei ist der erste Parameter das zu wiederholende Zeichen, der zweite gibt an, wie oft dieses Zeichen wiederholt werden soll. So ist tparm(repeat_char,'x',10) entsprechend zu 'xxxxxxxxxx'.

Arbeitet die Datensichtstation mit einem modifizierbaren Befehlszeichen, so wird dieses mit *cmdch* angegeben. Damit definiert man den Prototyp eines Befehlszeichens, das dann in allen Steuerzeichenfolgen verwendet wird.

terminfo-Beschreibungen, die sich nicht auf einen bestimmten Datensichtstationstyp beziehen - z.B. *switch*, *dialup*, *patch* und *network* - sollten die Angabe *gn* (generischer Typ) enthalten. Dies erlaubt es Programmen, zu signalisieren, daß sie die Datensichtstation nicht ansprechen können. Dies gilt nicht für die Beschreibungen von logischen Datensichtstationen, deren Escape-Sequenzen bekannt sind.

Bei Datensichtstationen, die für die Flußsteuerung den XON/XOFF-Quittungsbetrieb verwenden, ist *XON* anzugeben. Eventuelle Verzögerungen sollten auch hier angegeben werden, um die Kostenermittlung durch Routinen zu erleichtern, aber es werden keine Füllzeichen übertragen.

Mit *km* gibt man eine eventuell vorhandene "Meta-Taste" an, mit der auf den Modus umgeschaltet wird, in dem alle Zeichen einschließlich ihres 8. Bits übertragen werden. Sonst wird das 8. Bit als Paritätsbit interpretiert und üblicherweise entfernt. Steuerzeichenfolgen, mit denen dieser "Meta-Modus" an- und ausgeschaltet werden kann, können mit *smm* bzw. *rmm* angegeben werden.

Ist die Anzahl Zeilen im Speicher der Datensichtstation größer als die Anzahl Zeilen, die auf ein Mal auf den Bildschirm ausgegeben werden können, so ist die Anzahl Zeilen im Speicher mit *lm* anzugeben. Hat *lm* den Wert 0, so bedeutet dies, daß die Zeilenzahl im Speicher die Bildschirmkapazität übersteigt, jedoch variabel ist.

Bei Datensichtstationen, die das UNIX Virtual Terminal Protocol unterstützen, kann die Datensichtstationsnummer mit *vt* angegeben werden.

Zeichenketten für die Steuerung eines an die Datensichtstation angeschlossenen Druckers können mit *mc0* (Bildschirminhalt ausdrucken), *mc4* (Drucker ausschalten) und *mc5* (Drucker einschalten), angegeben werden. Bei eingeschaltetem Drucker geht jeder an die Datensichtstation übertragene Text auch an den Drucker. Ob Text bei eingeschaltetem Drucker auch auf den Bildschirm ausgegeben wird, ist undefiniert. Bei der *mc5*-Variante *mc5p* (mit einem Parameter) bleibt der Drucker eingeschaltet, bis die mit dem Parameter angegebene Anzahl Zeichen gedruckt ist und wird dann ausgeschaltet. Dabei sollte der Parameter den Wert 255 nicht überschreiten. Solange *mc5p* gilt, wird der gesamte Text einschließlich *mc4* in transparenter Form an den Drucker übertragen.

Steuerzeichenfolgen zur Programmierung von Funktionstasten können mit *pfkey*, *pfloc* und *pfx* angegeben werden. Jede dieser Steuerzeichenfolgen hat zwei Parameter: die Nummer der zu programmierenden Funktionstaste (von 0 bis 10) und die Steuerzeichenfolge, mit der die Funktionstaste belegt werden soll. Gibt man eine Tastennummer außerhalb des Wertebereichs 0 - 10 an, so kann man damit eine datenstationsabhängige Programmierung undefinierter Tasten erreichen. Die oben aufgeführten Angaben haben unterschiedliche Auswirkungen: bei *pfkey* hat die Betätigung der angegebenen Taste dieselbe Wirkung wie die Eingabe der angegebenen Zeichenkette über die Tastatur; bei *ploc* wird die angegebene Zeichenkette lokal an der Datensichtstation ausgeführt und bei *pfx* wird sie an den Rechner übertragen.

Fehlfunktionen

Für Datensichtstationen, bei denen ein unmittelbar auf einen *am*-Sprung folgender Zeilenvorschub ignoriert wird, sollte *xenl* angegeben werden.

Kann eine Hervorhebung ('standout') nur durch die Eingabe von *el* gelöscht werden (und nicht durch einfaches Überschreiben mit normalem Text), so ist *xhp* anzugeben.

Für Teleray-Stationen, bei denen bei der Positionierung auf einen Tabulator alle passierten Zeichen gelöscht (mit Leerzeichen überschrieben) werden, ist *xt* (löschende Tabulatoren) anzugeben. Eine weitere Bedeutung von *xt* ist, daß die Schreibmarke nicht auf ein "Magic Cookie" positioniert werden kann und der Hervorhebungsmodus daher nur durch das Aus-/einfügen von Zeilen rückgesetzt werden kann.

Andere spezielle Probleme Ihrer Datensichtstation können Sie dadurch korrigieren, daß Sie entsprechende Angaben in der Form **x***x* machen.

Ähnliche Datensichtstationen

Zwei einander sehr ähnliche Datensichtstationen können - mit gewissen Einschränkungen - auf dieselbe Art und Weise beschrieben werden: Mit *use* gibt man in der einen Beschreibung den Namen der anderen Datensichtstation an. Dabei werden die Eigenschaften der mit *use* aufgerufenen Datensichtstation durch diejenigen ersetzt, die vor *use* angegeben sind. Setzt man *xx@* der Definition mit *use* voran — wobei *xx* der Code einer Eigenschaft ist, — so wird diese Eigenschaft dadurch außer Kraft gesetzt. Zum Beispiel definiert die Angabe

2621-nl,smkx@,rmkx@,use=2621,

eine *2621-nl* ohne die Funktionen *smkx* und *rmkx*, d.h., bei dieser Station werden im Visual-Modus die Funktionstasten nicht auf Lokalbetrieb gesetzt. Diese Möglichkeit ist nützlich, wenn eine Datensichtstation in verschiedenen Betriebsarten arbeiten kann oder wenn bestimmte Benutzerpräferenzen zu berücksichtigen sind.

2.8.2 Die termcap-Schnittstelle

termcap ist eine Programmierhilfe, mit der bildschirmorientierte Anwendungen wie Emulatoren, Editoren, Menüsysteme, usw. unabhängig von dem jeweiligen Bildschirm programmiert werden können. Diese Konzeption macht ein Programm unabhängig von dem jeweiligen Typ der Bedieneinheit und dadurch leichter portierbar.

Die *termcap*-Schnittstelle besteht aus den Umgebungs-Variablen TERM und TERMCAP sowie aus den Funktionen:

tgetent()	*tgetflag()*	*tgetnum()*
tgetstr()	*tgoto()*	*tputs()*

Umgebungsvariablen
Wenn ein Programm startet, steht ihm vom aufrufenden Prgramm ein Vektor (*char **environ*) zur Verfügung, in dem gewisse Umgebungsvariablen definiert sind (siehe *exec*). Zu diesen Umgebungsvariablen gehören die Variablen TERM und TERMCAP, die von den *termcap*-Funktionen benutzt werden:

TERM Typ des Bildschirms, mit dem *termcap*-Anwendungen arbeiten sollen, z.B. 97801, wird automatisch von der Shell gesetzt, wenn ein Benutzer sich anmeldet.

TERMCAP
Quelle, in der die Bildschirmbeschreibung steht:
- beginnt der Inhalt der Variablen mit '/' wird er als Pfadname für die termcap-Datei interpretiert und die Datei eröffnet,
- stimmt der Inhalt von TERM mit dem Anfang des Inhalts dieser Variablen überein, wird die nachfolgende Zeichenkette als *termcap*-Beschreibung interpretiert,
- ist der Inhalt nicht definiert, wird standardmäßig aus der Datei */etc/termcap* gelesen. In der SINIX-Version V5.22 wird dieses Lesen mit Hilfe eines *terminfo*-Eintrags simuliert.

Die Datei /etc/termcap
Da die *termcap*-Funktionen das Vorhandensein dieser Datei simulieren, ist es notwendig, hier noch einmal auf das Format dieser Datei einzugehen. Die Unterschiede zwischen *termcap* und *terminfo* wurden bereits im Abschnitt 2.8.1 beschrieben.

Die Datei */etc/termcap* enthielt Einträge für verschiedene Bildschirmtypen. Ein Eintrag war gekennzeichnet durch eine Folge von Namen und beschrieb die einzelnen Funktionen und Möglichkeiten des Bildschirms in Form von Feldern, formal:

<kennzeichnung>:<feld>...

Ein Eintrag für einen Bildschirm durfte aus höchstens 1024 Zeichen bestehen. Ging ein Eintrag über mehrere Zeilen, so mußte das Zeilenende jeweils durch '\' entwertet werden.

Als Beispiel eines solchen Eintrages sehen Sie nachfolgend die */etc/termcap*-Beschreibung für eine 97801-Datensichtstation:

```
standard|97801:\
      :co#80:li#24:am:bs:bt=\E[Z:cm=\E[%i%d;%dH:nd=\E[C:up=\E[A:\
      :ce=\E[0K:cd=\E[0J:cl=\E[H\E[2J:dl=\E[M:al=\E[L:sr=\E[T:sf=\E[S:\
      :ae=\E[2m:as=\E[m:so=\E[7m:se=\E[m:ti=\E)w:\
      :ic=\E[@:dc=\E[P:us=\E[4m:ue=\E[m:ta=^I:\
      :ku=\E[A:kd=\E[B:kr=\E[C:kl=\E[D:kh=\E[H:\
      :k0=\E[@:k1=\E[P:k2=\Eo:k3=\Ep:k4=\E[L:k5=\E[M:\
      :k6=\E\072:k7=\E9:k8=\E[T:k9=\E[S:10=\E>:11=\Em:12=^D:\
      :F1=\E :F2=\E;:F3=\E":F4=\E#:F5=\E$:F6=\E%:F7=\E&:F8=\E':F9=\E<:Fa=\E=:\
      :P1=\E@:P2=\EA:P3=\EB:P4=\EC:P5=\ED:P6=\EF:P7=\EG:P8=\EH:P9=\EI:Pa=\EJ:\
      :Pb=\EK:Pc=\EL:Pd=\EM:Pe=\EN:Pf=\EO:Pg=\EP:Ph=\E0:Pi=\E_:Pj=\Ed:Pk=\ET:\
      :y0=^NB^O:y1=^NC^O:y2=^ND^O:y3=^NE^O:y4=^NA^O:y5=^N@^O:y6=\E[2;7m:\
      :y7=\E[m:ya=\012:yb=\177:yc=\015:yd=^R:ye=^X:yf=^H:P1=\Eg:GS=^N:\
      :GE=^O:GV=`:GH=A:
```

Jeder *termcap*-Eintrag begann mit mehreren Namen, getrennt durch '|'. In SINIX bezeichnete der erste Name den gesamten Eintrag (im Beispiel: *standard*) im Hinblick auf seine häufigste Anwendung. Der zweite Name war die Typbezeichnung des Bildschirms (im Beispiel: *97801*). Es konnten noch weitere Namen angegeben werden.

Ein Feld beschrieb eine Funktion oder Möglichkeit des Bildschirms. Jedes Feld begann mit einem zwei Zeichen langen Feldnamen. Ein Feldeintrag wurde mit ':'abgeschlossen.

Es gab drei Arten von Feldern:

Typ 1: Boolsche Felder
Format: <feldname>
Sie zeigten das Vorhandensein bestimmter Funktionen an, z.B. *bs* bedeutete, daß auf dem Bildschirm durch das Backspace-Zeichen (in C: '\b') eine Position nach links gegangen werden konnte.

Typ 2: Numerische Felder
Format: <feldname> # <zahl>
Sie gaben die Größe eines Bereiches an, z.B. li#24 bedeutete daß der Bildschirm 24 Zeilen hatte.

Typ 3: Zuweisungsfelder
Format: <feldname>=[wartezeit]<steuerzeichenfolge>
Sie definierten Steuerzeichenfolgen für die Bildschirmsteuerung, und für spezielle Tastenfunktionen, z.B. "dl=\E[M" bedeutete, daß "\E[M" die Steuerzeichenfolge für das Löschen einer Zeile war.

Im Abschnitt 2.8.1 sind die möglichen Feldbezeichnungen und deren Bedeutung zusammen mit den entsprechenden terminfo-Bezeichnungen in einer Tabelle aufgeführt.

Wartezeiten Bei der Verwendung von Steuerzeichenfolgen konnte nach dem '=' optional eine Wartezeit von Millisekunden abgegeben werden, die durch das Aussenden von Füllzeichen im Anschluß an die Zeichenkette erreicht wurde. Eine Wartezeit-Angabe hatte die Form:

<zahl>,z.B. 30 oder
<zahl>.<zahl>*, z.B. 4.5*

für Anzahl der Millisekunden je betroffener Zeile. Die Datensichtstationen für SINIX-PC's benötigen keine Wartezeit nach komplexen Funktionen.

Wenn in einer Steuerzeichenfolge nicht druckbare Zeichen vorkamen, mußten diese kodiert werden.

Einige nicht druckbare Zeichen hatten einen symbolischen Code:

Code	Bedeutung
\E	Escape-Zeichen
^x	Control-X
\n	Neue Zeile
\r	Wagenrücklauf
\t	Tabulator
\b	Backspace
\f	Seitenvorschub

Wegen ihrer besonderen Bedeutung im *termcap*-Eintrag, mußten die Zeichen ':' (Trenner), '^' (Control), '\' (Kodierung von Sonderzeichen) und '\0' wie folgt kodiert werden:

Code	Bedeutung
\^	^
\\	\
\072	:
\200	binäre Null

Außer den oben aufgeführten Spezialkodierungen, konnte jedes Zeichen durch seinen ASCII-Oktalcode angegeben werden, in der Form \ddd, wobei d eine Oktalziffer war.

Steuerzeichenfolgen können auf der Tastatur durch gleichzeitiges Drücken der Taste CTRL und einer weiteren Taste erzeugt werden. Sie können ein Steuerzeichen dementsprechend auch durch ^*x* darstellen, z.B. ^D für die Taste END.

Formate für die Cursor-Positionierung Steuerzeichenfolgen für die absolute Cursorpositionierung müssen in der Regel mit aktuellen Werten für Zeile und Spalt versorgt werden, bevor sie an den Bildschirm geschickt werden. Im *termcap*-Eintrag standen anstelle der aktuellen Werte Formatparameter, die später in einer Anwendung mit Hilfe der Funktion *tgoto()* ersetzt werden konnten. Die Formatangabe war ähnlich wie das Format in der Standardausgabefunktion printf. Im Einzelnen gab es folgende Möglichkeiten:

Format	Bedeutung
%%	für %
%d	wie in printf (ganze Zahl)
%2	wie %2d in printf (zweistellige ganze Zahl)
%3	wie %3d in printf
%.	wie %c in printf (ein Zeichen)
%+*x*	addiert *x* zum aktuellen Wert, dann wie %

Die folgenden Formate konnten zusätzlich zu den o.g. eingetragen werden, um die Umwandlung zu steuern, bewirkten aber keine Ausgabe:

Format	Bedeutung
%>*xy*	falls der aktuelle Wert größer als *x*, addiere *y*
%r	vertausche die Reihenfolge von Zeilen- und Spalten-angaben
%i	erhöhe den aktuellen Zeilen- und Spaltenwert um 1
%n	exklusiv-ODER vom aktuellen Zeilen- und Spaltenwert mit 0140
%B	BCD-Ausgabe (16*(*x*/10)) + (*x*%10)
%D	BCD-Umgekehrte Codierung (*x* - 2*(*x*%16))

Prinzipieller Ablauf eines *termcap*-Programms
Das folgende Beispiel zeigt, wie ein *termcap*-Programm prinzipiell aufgebaut ist.

Generell wird zunächst ein *termcap*-Eintrag mit der Funktion *tgetent()* eingelesen. Danach werden die entsprechenden booleschen Felder, numerischen Werte und Steuersequenzen mit den Funktionen *tgetflag()*, *tgetnum()* und *tgetent()* eingelesen. Sollen Steuersequenzen ausgegeben werden, so können diese, nach einer möglicherweise erforderlichen Aufbereitung mit *tgoto()* von der Funktion *tputs()* ausgegeben werden.

Das folgende Beispiel ermittelt die Steuerseqenzen für die Cursor-Positionierung ("cm") und für das Bildschirmlöschen ("cl"):

```
#include <stdio.h>

/* Hilfsfunktion fuer tputs() zur Ausgabe eines Zeichens */
int outc(c)
int c;
{  putchar(c);
}

main()
{     char entry[1024], *terminal, *tgetent();
      char *cm, *cl,          /* Adressen der Steuerseqenzen   */
           buffer[1024],      /* Speicher fuer Steuersequenzen */
          *bufptr=buffer,     /* Zeiger auf buffer             */
          *tgetstr();

      if ((terminal = getenv("TERM")) == NULL)
      {  fprintf(stderr, "Umgebungsvariable TERM existiert nicht!\n");
         exit(1);
      }

      if (tgetent(entry, terminal) != 1)
      {  fprintf(stderr, "Eintrag fuer '%s' nicht gefunden!\n", terminal);
         exit(1);
      }

      if ((cm=tgetstr("cm",&bufptr)) == NULL )
      {  fprintf("termcap: Cursorpositionierung unmoeglich!\n");
         exit(1);
      }

      if ((cl=tgetstr("cl",&bufptr)) == NULL )
      {  fprintf("termcap: Bildschirmloeschen unmoeglich!\n");
         exit(1);
      }

      /* Bildschirmloeschen: */
      tputs("cl", 0, outc);

      /* Cursor auf Zeile 5, Spalte 35: */
      tputs(tgoto("cl",34,4), 0, outc);
}
```

3 System-Schnittstellen

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

3.1 Funktionsübersicht

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

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

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

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

Dateibearbeitung

Dateizugriff

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

Dateiverwaltung

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

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

Datei-Information

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

Ein/Ausgabe

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

Prozesse

Prozeßverwaltung

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

Interprozeßkommunikation

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

Pipes

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

Zusammenwirken von Prozessen

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

Steuerung von Programmabläufen

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

Programmtest

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

Prozeßuniversum

Name	Kurzbeschreibung
universe()	Prozeßuniversum abfragen/wechseln

Speicherverwaltung und -operationen

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

Systemorganisation

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

Zeichen und Zeichenketten

Einzelne Zeichen bearbeiten

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

Konvertierung von Größen

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

Zeichenketten bearbeiten

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

Reguläre Ausdrücke

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

Fehlermeldungen

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

Zeitfunktionen

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

Mathematische Funktionen

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

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

Zufallszahlen

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

Such- und Sortierverfahren

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

Manipulation einer seriellen Schnittstelle

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

Bildschirmsteuerung

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

noch Bildschirmsteuerung

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

noch Bildschirmsteuerung

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

noch Bildschirmsteuerung

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

Funktionen zur C-Schnittstelle des Druckspools

Druckerbeschreibung

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

Durchsuchen der Auftragsdateien

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

3.2 Funktionen a - i

NAME

a64l, l64a - Ascii (base 64) to long
Ganze Zahl (Typ long) in ASCII-Zeichenkette (Basis 64) umwandeln und umgekehrt

DEFINITION

```
long a64l (s)
char *s;

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

BESCHREIBUNG

Mit diesen Funktionen können Zahlen, die im ASCII-Code (Basis 64) gespeichert werden, übernommen werden. Mit diesem Code können große ganzzahlige Werte (Typ long) bis maximal 6 Zeichen dargestellt werden, wobei jedes Zeichen eine "Ziffer" aus einer Basis-64-Notation darstellt.

Dabei ergeben sich folgende Entsprechungen:

.	entspricht	0
/	entspricht	1
0 bis 9	entsprechen	2-11
A bis Z	entsprechen	12-37
a bis z	entsprechen	38-63

Bei *a64l()* verweist ein Zeiger auf eine mit Null abgeschlossene Zeichenkette in Basis-64-Notation; als Ergebnis wird der entsprechende Wert vom Typ long zurückgeliefert. Wenn die Zeichenkette, auf die der Zeiger *s* verweist, länger als 6 Zeichen ist, werden nur die ersten 6 Zeichen von *a64l()* ausgewertet.

Die Zeichen werden von rechts nach links umgesetzt, d.h. das höchstwertige Zeichen steht rechts. Die Zeichenkette in Basis-64-Notation kann den positiven Maximalwert einer ganzen Zahl vom Typ long überschreiten; die Werte werden dann negativ.

l64a() liest ein Argument vom Typ long und liefert als Ergebnis einen Zeiger auf die entsprechende Basis-64-Darstellung. Ist das Argument 0, so liefert *l64a()* als Ergebnis einen Zeiger auf eine leere Zeichenkette.

ERGEBNIS

Das von *l64a()* gelieferte Ergebnis ist ein Zeiger auf einen statischen Puffer, dessen Inhalt bei jedem Aufruf überschrieben wird.

HINWEIS

Da das Ergebnis von *l64a()* in einem statischen Puffer abgelegt wird, muß eine Anwendung, die mehrere solche Zeichenketten verwenden will, diese in eigene Puffer umspeichern.

PORTABILITÄT

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

NAME

abort - generate an abnormal process abort
anormaler Prozeßabbruch

DEFINITION

```
#include <stdlib.h>

void abort()
```

BESCHREIBUNG

Die Funktion *abort()* führt zuerst passende Aufräumarbeiten durch, die zumindest das Schließen aller offenen Dateien, Ströme, Dateiverzeichnisströme und Meldungskatalog-Kennzahlen einschließt, sofern dies möglich ist. Danach sorgt sie dafür, daß das Signal SIGABRT an den Prozeß gesendet wird. Dies setzt Aktionen für dem anormalen Prozeßabbruch in Gang, unter SINIX wird ein *Kernspeicherabzug* erzeugt, andere X/Open-kompatible Implementierungen können hier weitere Aktionen vorsehen.

ERGEBNIS

Die Funktion *abort()* kehrt nicht zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Signal SIGABRT nicht abgefangen werden.

PORTABILITÄT

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

SIEHE AUCH

exit(), *kill()*, *signal()*, *<stdlib.h>*.

NAME

abs - return integer absolute value
Absolutbetrag einer ganzen Zahl berechnen

DEFINITION

```
#include <stdlib.h>

int abs (i)
int i;
```

BESCHREIBUNG

Die Funktion *abs()* berechnet den Absolutbetrag ihres ganzzahligen Arguments *i*. Wenn das Resultat nicht dargestellt werden kann, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *abs()* liefert den Absolutbetrag ihres ganzzahligen Arguments.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

In der Zweierkomplement-Darstellung ist der Absolutbetrag der größtmöglichen negativen Ganzzahl {INT_MIN} nicht definiert.

PORTABILITÄT

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

BEISPIEL

Gibt zu einem einzulesenden Wert den entsprechenden Absolutbetrag aus:

```
#include <stdio.h>

main()
{
   int i;
   if (scanf("%d", &i) == 1)
      printf("i = %d : |i| = %d\n",i,abs(i));
}
```

SIEHE AUCH

fabs(), *<stdlib.h>*.

NAME

access - determine accessibility of a file
Zugriffsrechte für eine Datei prüfen

DEFINITION

```
#include <unistd.h>

int access (path, amode)
char *path;
int amode;
```

BESCHREIBUNG

Die Funktion *access()* prüft die Zugriffsrechte der durch das Argument *path* angegebenen Datei gemäß dem Bitmuster, das in *amode* enthalten ist, und verwendet dabei die reale anstelle der effektiven Benutzernummer und die reale anstelle der effektiven Gruppennummer.

Der Wert von *amode* ist entweder das bitweise Inklusiv-ODER der zu prüfenden Zugriffsrechte (R_OK,W_OK,X_OK), oder der Test auf Existenz F_OK. Siehe auch *<unistd.h>*.

Wenn Zugriffsberechtigungen geprüft werden sollen, dann wird jede einzelne Zugriffsberechigung für sich geprüft, so wie dies unter *Zugriffrechte für Dateien* im Glossar beschrieben wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *access()* den Wert 0. Andernfalls wird der Wert -1 zurückgegeben und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] Die Schutzbits der Datei erlauben den geforderten Zugriff nicht, oder für eine Komponente des Pfadnamen-Anfangs von *path* existiert keine Durchsuchberechtigung.

[EINVAL] Der Wert des Arguments *amode* ist ungültig. Dieser Fehler muß bei anderen X/Open-kompatiblen Implementierungen nicht auftreten.

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

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

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

[EROFS] Es wird der Schreibzugriff für eine Datei gefordert, die sich in einem nur zum Lesen eingehängten Dateisystem befindet.

[ETXTBSY]
Der Schreibzugriff für eine reine Prozedur (gemeinsamer Text) wird angefordert, die gerade ausgeführt wird. Unter anderen X/Open-kompatiblen Implementierungen als SINIX wird dieser Fehler nicht unbedingt angezeigt.

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

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

HINWEIS

Die Werte für *amode* sollten aus Portabilitätsgründen unbedingt als symbolische Konstanten und nicht mehr als Zahlenwerte angegeben werden.

PORTABILITÄT

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

Portabilitäts-Hinweis:
Wenn der Prozeß die entsprechenden Zugriffsrechte hat, dann können andere Implementierungen den Erfolg für X_OK anzeigen, auch wenn keines des Schutzbits für Ausführung gesetzt ist.

BEISPIEL

Mit dem folgenden Programm wird überprüft, ob eine Datei ausführbar ist.

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

int main(argc, argv)
int argc;
char *argv[ ];

{
   if(access(argv[1], X_OK))==0)
   {
      printf("Datei %s ausfuehrbar\n", argv[1]);
      exit (0);
   }
   else
   {
       perror ("error");
       exit (1);
   }
}
```

SIEHE AUCH

chmod(), *stat()*, *<unistd.h>*.

NAME

acos - arc cosine function
Arcus Cosinus

DEFINITION

```
#include <math.h>

double acos (x)
double x;
```

BESCHREIBUNG

Die Funktion *acos()* berechnet den Stammwert des Arcus Cosinus von x. Der Wert von x sollte im Intervall [-1,1] liegen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *acos()* den Arcus Cosinus von x im Bogenmaß im Intervall [0, π].

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

Andernfalls wird der Wert 0 zurückgegeben und *errno* wird gesetzt um den Fehler anzuzeigen, oder NaN wird zurückgegeben und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

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

[EDOM] Der Wert von x liegt nicht im Intervall [-1,1].

HINWEIS

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

Unter SINIX und einigen anderen Implementierungen wird bei Auftreten eines Fehlers eine Fehlermeldung nach *stderr* geschrieben. In diesem Fall kann die Fehlerbehandlung durch Bereitstellen einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel bei *asin()*.

SIEHE AUCH

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

NAME

advance - pattern match given a compiled regular expression
Zeichenkette mit regulärem Ausdruck vergleichen

DEFINITION

```
int advance (string, expbuf)
char *string, *expbuf;
```

BESCHREIBUNG

Siehe unter *regexp*.

PORTABILITÄT

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

NAME

alarm - schedule an alarm signal
Alarmsignal festsetzen

DEFINITION

```
unsigned int alarm (seconds)
unsigned int seconds;
```

BESCHREIBUNG

Die Funktion *alarm()* veranlaßt das System dem aufrufenden Prozeß ein SIGALRM-Signal zu senden, nachdem die Anzahl *seconds* von Echtzeit-Sekunden vergangen ist. Vergabeverzögerungen für den Prozessor können vehindern, daß der Prozeß das Signal sofort nach seiner Generierung behandelt.

Wenn *seconds* gleich 0 ist, dann wird eine vorangegangene Alarmanforderung, falls vorhanden, gelöscht.

Alarmanforderungen werden nicht gekellert, nur eine Generierung von SIGALRM kann auf diese Art festgesetzt werden; wenn das Signal SIGALRM noch nicht generiert wurde, dann verursacht der Aufruf eine Neufestsetzung der Zeit, zu der das Signal SIGALRM generiert wird.

ERGEBNIS

Die Funktion *alarm()* liefert den Rest der Zeit in Sekunden, die noch übrig ist, bevor das System die Generierung des Signals SIGALRM aus einem vorangegangenen Aufruf von *alarm()* vornimmt, oder 0, wenn es keinen vorangegangenen Aufruf von *alarm()* gegeben hat.

FEHLER

Die Funktion *alarm()* ist immer erfolgreich und es ist kein Ergebniswert zur Anzeige eines Fehlers reserviert.

HINWEIS

Die Funktion *fork()* löscht anstehende Alarme des Sohnprozesses. Ein Prozeß, der durch eine der *exec*-Funktionen aufgerufen wurde, behält die bis zu einem Alarmsignal verbleibende Zeit aus dem ursprünglichen Prozeß.

PORTABILITÄT

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

BEISPIEL

Circa alle zwei Sekunden ertönt am Bildschirm der Piepston:

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

main()
{  int catch();

   signal(SIGALRM, catch);
   alarm(2);
   for(;;)
      /* printf("*") */
      ;
}

int catch()                    /* Signalbehandlung fuer SIGALRM  */
{
   putchar(0x7);               /* Klingel                        */
   putchar(0xA);               /* Zeilenvorschub                 */
   fflush(stdout);             /* bei einigen Systemen notwendig */
   signal(SIGALRM, catch);     /* Signalroutinen neu aufsetzen,  */
                               /*  damit der Prozess nicht       */
                               /*  abgebrochen wird              */

   alarm(2);                   /* Alarmuhr neu aufziehen */
 }
```

SIEHE AUCH

exec, fork(), pause(), sigaction(), <signal.h>.

NAME

asctime - convert date and time to string
Datum und Zeit in Zeichenkette umwandeln

DEFINITION

```
#include <time.h>

char *asctime (timeptr)
struct tm *timeptr;
```

BESCHREIBUNG

Die Funktion *asctime()* wandelt eine, in der Struktur aufgeschlüsselte Zeitangabe, auf die *timeptr* zeigt, in eine Zeichenkette mit folgendem Format um:

Sun Sep 16 01:03:52 1973\n\0

Dabei verwendet Sie in etwa den folgenden Argorithmus:

```
char *asctime(timeptr)
struct tm *timeptr; {
    static char wday_name[7][3] = {
            "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static char mon_name[12][3] = {
            "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
            "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];

    sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\en",
            wday_name[timeptr->tm_wday],
            mon_name[timeptr->tm_mon],
            timeptr->tm_mday, timeptr->tm_hour, timeptr-
            >tm_min, timeptr->tm_sec, 1900 + timeptr-
            >tm_year);
    return result;
}
```

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *asctime()* einen Zeiger auf die Zeichenkette zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Ergebnis kann auf einen statischen Datenbereich zeigen, der bei jedem Aufruf neu überschrieben wird.

Werte für die aufgeschlüsselte Zeit in der Struktur können durch Aufrufe von *gmtime()* oder *localtime()* ermittelt werden.

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *localtime()*

SIEHE AUCH

gmtime(), *localtime()*, *sprintf()*, *<time.h>*.

NAME

asin - arc sine function
Arcus Sinus

DEFINITION

```
#include <math.h>

double asin(x)
double x;
```

BESCHREIBUNG

Die Funktion *asin()* berechnet den Stammwert des Arcus Sinus von *x*. Der Wert von *x* sollte im Intervall [-1,1] liegen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *asin()* den Arcus Sinus von *x* im Bogenmaß aus dem Intervall [- π /2, π /2].

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

Andernfalls wird 0 zurückgeliefert und *errno* wird gesetzt, um den Fehler anzuzeigen, oder NaN wird zurückgeliefert und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

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

[EDOM] Der Wert von *x* liegt nicht im Intervall [-1,1].

HINWEIS

Eine Anwendung, die Fehlersituationen portabel prüfen möchte, sollte *errno* gleich 0 setzen und dann *asin()* aufrufen. Wenn *errno* nach der Rückkehr gesetzt, oder das Ergebnis gleich NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

Unter SINIX und einigen anderen Implementierungen wird unter Fehlerbedingungen eine Fehlermeldung auf *stderr* ausgegeben. In diesen Fall ist es möglich, die Fehlerbehandlung durch das Bereitstellen einer *matherr()*-Funktion zu ändern (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Drucke für die Werte 0.0, 0.1, ..., 1.0 die entsprechenden Arcus Sinus Werte:

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

main()
{
   double x;
   for (x = 0.0; x< 1.1; x = x + 0.1)
       printf("x = %g: asin(%g) = %g \n", x, x, asin(x));
}
```

SIEHE AUCH

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

NAME

assert - insert program diagnostics
Diagnoseteile in Programme einfügen

DEFINITION

```
#include <assert.h>

void assert (expression)
int expression;
```

BESCHREIBUNG

Das Makro *assert()* fügt Diagnoseteile in Programme ein. Wenn es ausgeführt wird und wenn *expression* falsch ist (d.h. gleich 0 ist), dann schreibt *assert()* Informationen über den speziellen Aufruf, der fehlgeschlagen ist, auf *stderr*, einschließlich des Argumenttextes, des Namens der Quelldatei und der Zeilenummer (letztere sind die Werte der Präprozessor-Makros __FILE__ und __LINE__). Danach ruft es die Funktion *abort()* auf.

ERGEBNIS

Das Makro *assert()* liefert kein Ergebnis zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Eine Definition des Namens *NDEBUG* (entweder aus der Aufrufzeile oder durch die Präprozessoranweisung #*define NDEBUG*), vor der Anweisung #*include* <*assert.h*> verhindert, daß mit *assert()* definierte Annahmen in das Programm eingefügt werden.

PORTABILITÄT

Das Makro *assert()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

abort(), *stderr*, <*assert.h*>.

NAME

atan - arc tangent function
Arcus Tangens

DEFINITION

```
#include <math.h>

double atan(x)
double x;
```

BESCHREIBUNG

Die Funktion *atan()* berechnet den Stammwert des Arcus Tangens von *x*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *atan()* den Arcus Tangens von *x* im Bogenmaß aus dem Intervall $[\pi/2, \pi/2]$.

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

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

FEHLER

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

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

HINWEIS

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

Unter SINIX und einigen anderen Implementierungen wird unter Fehlerbedingungen eine Fehlermeldung auf *stderr* ausgegeben. In diesem Fall ist es möglich, die Fehlerbehandlung durch Bereitstellen einer *matherr()*-Funktion zu ändern (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Folgendes Programm druckt zu einem eingelesenen Wert den entsprechenden Arcus Tangens:

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

main()
{
   double x;
   if (scanf("%lf", &x) == 1)
      printf("x = %g: atan(%g) = %g \n", x, x, atan(x));
}
```

SIEHE AUCH

atan2(), *isnan()*, *matherr()*, *tan()*, *<math.h>*.

NAME

atan2 - arc tangent function
Arcus Tangens

DEFINITION

```
#include <math.h>

double atan2(y, x)
double y, x;
```

BESCHREIBUNG

Die Funktion *atan2()* berechnet den Stammwert des Arcus Tangens von *y/x*, wobei sie die Vorzeichen beider Argumente verwendet, um den Quadranten des Ergebnisses zu bestimmen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *atan2()* den Arcus Tangens von *y/x* im Bereich [- π , π] im Bogenmaß.

Wenn *x* oder *y* ein NaN ist, dann wird NaN zurückgegeben.

Andernfalls wird entweder 0 zurückgegeben und *errno* gesetzt, um den Fehler anzuzeigen, oder NaN wird zurückgegeben und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

Die Funktion *atan2()* schlägt fehl, wenn gilt:

[EDOM] Beide Argumente sind gleich 0.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel prüfen will, sollte *errno* vor dem Aufruf von *atan2()* gleich 0 setzen. Wenn *errno* nach der Rückkehr gesetzt, oder das Ergebnis ein NaN ist (siehe auch *isnan()*), dann ist ein Fehler aufgetreten.

SINIX und einige andere Implementierungen schreiben unter Fehlerbedingungen eine Fehlermeldung nach *stderr*. Diese Fehlerbehandlung kann durch Bereitstellen einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

Die Funktion *atan2()* ist im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

atan(), *isnan()*, *matherr()*, *tan()*, *<math.h>*.

NAME

atof - convert string to double-precision number
Zeichenkette in double-Wert umwandeln

DEFINITION

```
#include <stdlib.h>

double atof (str)
char *str;
```

BESCHREIBUNG

Die Funktion *atof()* wandelt die Zeichenkette, auf die *str* zeigt, bis zum ersten, mit dem Format einer Gleitpunktzahl inkonsitenten Zeichen, in ein *double* um, wobei führende Leerzeichen ignoriert werden. Der Aufruf von *atof(str)* ist äquivalent zu *strtod(str,(char**) NULL)*, mit Ausnahme der Fehlerbehandlung. Wenn der Wert nicht dargestellt werden kann, ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *atof()* liefert den umgewandelten Wert zurück.

FEHLER

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

[ERANGE] Der korrekte Wert des Ergebnisse würde einen Über- oder Unterlauf verursachen.

PORTABILITÄT

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

BEISPIEL

Folgendes Programm wandelt eine beim Aufruf übergebene Zeichenkette in die entsprechende Gleitkommazahl um:

```
#include <stdio.h>
double atof();

main(argc,argv)
int argc;
char *argv[ ];        /* Zahlen werden als Zeichenketten uebergeben.
                         Eine Umwandlung ist erforderlich,
                         falls der Zahlenwert benoetigt wird */
{  if (argc > 1)
      printf("float %f\n", atof(argv[1]));
}
```

SIEHE AUCH

strtod(), *<stdlib.h>*.

NAME

atoi - convert string to integer
Zeichenkette in ganze Zahl umwandeln

DEFINITION

```
#include <stdlib.h>

int atoi (str)
char *str;
```

BESCHREIBUNG

Die Funktion *atoi()* wandelt die Zeichenkette, auf die *str* zeigt, bis zum ersten Zeichen, das nicht mehr zum Format einer ganzen Zahl (in Dezimaldarstellung) paßt, in ein *int* um, wobei führende Leerzeichen (Blank, Tab, NL) ignoriert werden. Der Aufruf *atoi(str)* iat äquivalent zu *(int) strtol(str,(char **)NULL, 10)*, außer in Bezug auf die Fehlerbehandlung. Wenn der Wert nicht dargestellt werden kann, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *atoi()* liefert den umgewandelten Wert zurück.

FEHLER

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

[ERANGE] Der korrekte Wert des Ergebisses würde einen Überlauf verursachen.

PORTABILITÄT

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

BEISPIEL

Folgendes Programm wandelt eine beim Aufruf übergebene Zeichenkette in den entsprechenden ganzzahligen Wert um:

```
#include <stdio.h>

main(argc,argv)     /* Zahlen werden als Zeichenkette uebergeben. */
int argc;           /*  Eine Umwandlung ist erforderlich,         */
char *argv[ ];      /*  falls der Zahlenwert benoetigt wird       */
{  if (argc>1)
   printf("int: %d\n", atoi(argv[1]));
}
```

SIEHE AUCH

strtol(), *<stdlib.h>*.

NAME

atol - convert string to long integer
Zeichenkette in long int umwandeln

DEFINITION

```
#include <stdlib.h>

long atol (str)
char *str;
```

BESCHREIBUNG

Die Funktion *atol()* wandelt die Zeichenkette, auf die *str* zeigt, bis zu ersten Zeichen, das inkonsistent zum Format einer dezimalen Ganzzahl ist, in ein *long* um, wobei führende Leerzeichen usw. ignoriert werden. Der Aufruf *atol(str)* ist äquivalent zu *strtol(str,(char **) NULL,10)*, mit Ausnahme der Fehlerbehandlung. Wenn der Wert nicht dargestellt werden kann, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *atol()* liefert den umgewandelten Wert zurück.

FEHLER

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

[ERANGE] Der korrekte Wert würde einen Überlauf verursachen.

PORTABILITÄT

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

SIEHE AUCH

strtol(), *<stdlib.h>*

NAME

j0, j1, jn, y0, y1, yn - bessel functions
Besselfunktionen

DEFINITION

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

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

double y0 (x)
double x;

double y1 (x)
double x;

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

BESCHREIBUNG

Die Funktionen *j0()*, *j1()* und *jn()* werden in diesem Handbuch unter *j0()* beschrieben; die Funktionen *y0()*, *y1()* und *yn()* unter *y0()*. Nähere Einzelheiten siehe unter *j0()* und *y0()*.

PORTABILITÄT

Die Besselfunktionen sind im X/Open-Standard (Ausgabe 3) definiert.

NAME

brk, sbrk - change data segment space allocation
Größe des Datensegments ändern

DEFINITION

```
int brk (endds)
char *endds;

char *sbrk (inkr)
int inkr;
```

BESCHREIBUNG

brk() und *sbrk()* werden dazu verwendet, den Speicherplatz für das Datensegment eines Prozesses dynamisch zu ändern (s. *exec*).

Bei Beginn der Programmausführung mit *exec* wird der "break" automatisch hinter die größte Adresse gesetzt, das ist die Summe aus dem Platzbedarf von Text- und Datenbereichen. Daher müssen nur die Programme, deren Datensegmente während der Ausführung (dynamisch) wachsen, selbständig die Speicherverwaltung organisieren.

Mit *brk()* und *sbrk()* wird der "break"-Wert des Prozesses neu gesetzt und der gewünschte Speicherplatz zur Verfügung gestellt

brk() setzt die niedrigste vom Programm nicht belegte Adresse des Datensegments (sog. "break") auf *endds* (aufgerundet auf das nächste Vielfache einer von der Hardware abhängigen Zahl von Bytes), wodurch der verfügbare Speicherplatz verändert wird. Er kann danach größer oder kleiner sein als vorher, je nach dem Wert von *endds*.

sbrk() ändert den "break"-Wert um *inkr* Bytes und damit den zugewiesenen Speicherplatz entsprechend. Wird für *inkr* ein negativer Wert angegeben, so wird der Speicherplatz verringert. Wird *sbrk()* mit *inkr* = 0 aufgerufen, so ist der gelieferte Wert die aktuelle Basis des Speicherplatzes für das Datensegment. So läßt sich der aktuelle "break" über *sbrk(0)* feststellen.

Die Daten im neu zugewiesenen Speicher sind undefiniert.

ERGEBNIS

Bei erfolgreicher Ausführung liefert *brk()* den Wert 0 und *sbrk()* den alten "break"-Wert. In allen anderen Fällen liefert *brk()* den Wert –1 und *sbrk()* den Wert *(char *) -1*; der Fehler wird in *errno* angezeigt.

FEHLER

brk() und *sbrk()* werden nicht ausgeführt, d.h. der zugewiesene Speicherplatz wird nicht verändert, wenn der folgende Fehler auftritt.

[ENOMEM]
Die gewünschte Veränderung fordert mehr Speicherplatz an, als das System maximal zuläßt (s. *ulimit()*). oder der momentane Speicherplatz (Hauptspeicher, *swap*-Bereich) ist fast vollständig belegt.

HINWEIS

– Der "break"-Wert kennzeichnet das Ende des für den Benutzer zugänglichen Datensegments und ist die erste *nicht* belegte Adresse. Der daran anschließende Bereich gehört nicht mehr zum Adreßraum des Prozesses. Der Versuch in diesen Bereich zu adressieren führt zu einem Speicherfehler.

– *brk()* kann mit jedem Wert aufgerufen werden, der im Bereich der Adressen liegt, die von einem *sbrk()*-Aufruf geliefert wurden. Die einzig sinnvolle Verwendung von *brk()* ist die Freigabe eines großen, zuvor mit *sbrk()* zugewiesenen Speicherbereichs. Wenn Sie mit *brk()* Speicherplatz außerhalb dieses Bereichs zuweisen oder freigeben, kann dies unerwünschte Folgen haben. Wird ein Bereich freigegeben und anschließend wieder zugewiesen, bleibt der alte Inhalt nicht notwendig erhalten.

  Es wird empfohlen, zusätzlichen Speicherplatz mit *malloc()* anzufordern.

Vorsicht
Auf keinen Fall sollte in einem Programm sowohl *malloc()* (bzw. *<stdio.h>*) als auch *brk()* (*sbrk()*) vorkommen!
Begründung:
Bei jeder Bibliotheksfunktion, muß damit gerechnet werden, daß sie *malloc()* verwendet. *malloc()* selbst verwendet seinerseits *brk()*. So kann es leicht zum Chaos kommen.

PORTABILITÄT

Die Funktionen *brk()* und *sbrk()* sind im X/OPEN-Standard nicht enthalten. Anwendungen sollten statt dessen immer *malloc()* verwenden.

BEISPIEL

Folgendes Programm zeigt die Wirkung von *brk()* und *sbrk()* und die Veränderung des Kellers (stacks) beim Ablauf einer rekursiven Funktion.
Wenn Sie das Programm übersetzt haben, können Sie es in der Form:
`a.out [-badresse] [-sinkr] [-rn]`
aufrufen. Dabei bedeutet:

-badresse	setze den "break" auf *adresse*
-sinkr	verändere den "break" um *inkr*
-rn	berechne n-Fakultät

```
#include <stdio.h>

extern etext();
extern edata;
extern end;
char gebrauch[ ] ="gebrauch: a.out -badr -sinkr -rn \n";

anzeige()
{
                                /* Variable, mit der die Veraenderung  */
                                /* des Kellers gezeigt wird            */
   int mark = 0;
   printf("%u: %u: %u: %u:\n", &edata, &end, sbrk(0), &mark);
}

int fak(n)
                                /* Fakultaetsfunktion */
int n;
{
   printf("%d!\n",n);
   anzeige();
   if (n)
      return(n*fak(n-1));
   return 1;
}

main(argc, argv)
int argc;
char **argv;
```

```
{
   register int num;
   printf("edata: end: break: stack\n");
   anzeige();

   while (--argc && **++argv == '-')
   {
      printf("hier\n");

      num =atoi(*argv +2);
      switch (*(*argv+1))
      {
       case 'b':  printf("brk(%d) = %d\n",num,brk(num));
                  break;
       case 's':  printf("sbrk(%d) = %d\n",num,sbrk(num));
                  break;
       case 'r':  printf("%d! = %d\n",num, fak(num));
                  break;
       default:  fputs(gebrauch,stderr);
                  exit(1);
      }
   anzeige();
   }
}
```

SIEHE AUCH

exec, *ulimit()*, *malloc()*.

NAME

bsearch - binary search a sorted table
binäre Suche in sortiertem Vektor

DEFINITION

```
#include <stdlib.h>

void *bsearch (key, base, nel, width, compar)
void *key, *base;
size_t nel, width;
int (*compar)();
```

BESCHREIBUNG

Die Funktion *bsearch()* ist eine Routine für die binäre Suche. Sie liefert einen Zeiger in einen Vektor, der angibt, wo ein Element gefunden werden kann. Der Vektor muß vorher bezüglich einer angegebenen Vergleichsfunktion *compar* aufsteigend sortiert worden sein. Das Argument *key* zeigt auf ein Datenelement, nach dem im Vektor gesucht werden soll. Das Argument *base* zeigt auf das Element am Anfang des Vektors. Das Argument *nel* ist gleich der Anzahl der Elemente des Vektors. Das Argument *width* entspricht der Größe eines Elements in Bytes. Das Argument *compar* ist der Name einer Vergleichsfunktion, die mit zwei Argumenten aufgerufen wird, die ihrerseits auf die Elemente zeigen, die verglichen werden sollen. Die Funktion muß eine ganze Zahl liefert, die kleiner, gleich oder größer 0 ist, je nachdem, ob das erste Argument als kleiner als, gleich dem oder größer als das zweite gelten soll.

ERGEBNIS

Die Funktion *bsearch()* liefert einen Zeiger auf ein passendes Element des Vektors oder den NULL-Zeiger, falls kein solches gefunden wird. Wenn zwei oder mehr Elemente gleich sind, dann ist nicht festgelegt, welches Element geliefert wird.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Das nachfolgende Beispiel durchsucht einen Vektor, der Zeiger auf Strukturen enthält, die aus einer Zeichenkette und deren Länge bestehen. Die Tabelle ist alphabetisch nach den Zeichenketten in den Strukturen sortiert, auf die die einzelnen Elemente zeigen.

Dieses Programmstück liest Zeichenketten ein. Wenn es danach die passende Struktur findet, dann wird die Zeichenkette und deren Länge ausgegeben, sonst eine Fehlermeldung.

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

#define TABSIZE    1000

struct node {                        /* Typ der Tabelleneinträge */
   char *string;
   int length;
};
struct node table[TABSIZE];          /* Suchtabelle */
   .
   .
   .
{
   struct node *node_ptr, node;
   int node_compare( );  /* Funktion für den Vergleichzweier Einträge */
   char str_space[20];   /* Platz für eingelesene Zeichenketten */
   .
   .
   .
   node.string = str_space;
   while (scanf("%s", node.string) != EOF) {
           node_ptr = (struct node *)bsearch((void *)(&node),
                      (void *)table, TABSIZE,
                      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("nicht gefunden: %s\n", node.string);
           }
   }
}
/*
   Diese Funktion vergleicht zwei Strukturen im Hinblick
   auf eine alphabetische Ordnung
*/
int node_compare(node1, node2)
void *node1, *node2;
{
   return strcmp((struct node *)node1->string,
           (struct node *)node2->string);
}
```

HINWEIS

Die Zeiger auf das zu suchende Datenelement und das Element am Anfang des Vektors sollten vom Typ *Zeiger auf Element* sein und mit dem Cast-Operator in den Typ *Zeiger auf void* umgewandelt werden.

Die Vergleichsfunktion muß nicht jedes Byte vergleichen, so daß zusätzlich zu den zu vergleichenden noch beliebige sonstige Daten in den Elementen vorkommen können.

Obwohl als *Zeiger auf void* vereinbart, sollte der Ergebniswert mit dem Cast-Operator in *Zeiger auf Element* umgewandelt werden.

Aus Gründen der Rückwärtskompatibilität mit Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<search.h>* anstelle von *<stdlib.h>*. Die Verwendung von *<search.h>* wird nicht empfohlen, da diese Funktionalität in Zukunft entfallen wird.

PORTABILITÄT

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

SIEHE AUCH

hsearch(), *lsearch()*, *qsort()*, *tsearch()*, *<stdlib.h>*.

NAME

calloc - memory allocator
Speicherplatzreservierung

DEFINITION

```
#include <stdlib.h>

void *calloc (nelem, elsize)
size_t nelem, elsize;
```

BESCHREIBUNG

Die Funktion *calloc()* reserviert ungenutzten Speicher für einen Vektor der Größe *nelem* Elemente, von denen jedes eine Größe von *elsize* Bytes besitzt. Im reservierten Speicher werden alle Bits auf 0 gesetzt.

Der Platz ist passend eingerichtet für die Speicherung eines beliebigen Typs von Objekt (nach einer möglichen Zeigeranpassung).

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *calloc()* einen Zeiger auf den reservierten Speicher. Andernfalls wird der NULL-Zeiger geliefert und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[ENOMEM] Es ist nicht genug Speicher verfügbar.

HINWEIS

Aus Gründen der Rückwärtskompatibilität zu Ausgabe 2 unterstützen X/Open-kompatible Systeme auch die Einbindung von *<malloc.h>* anstelle von *<stdlib.h>*. Die Verwendung von *<malloc.h>* ist nicht empfehlenswert, da diese Funktionalität in Zukunft entfallen wird.

PORTABILITÄT

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

SIEHE AUCH

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

NAME

catclose - close a message catalogue descriptor
Meldungskatalog-Deskriptor schließen

DEFINITION

```
#include <nl_types.h>

int catclose (catd)
nl_catd catd;
```

BESCHREIBUNG

Die Funktion *catclose()* schließt den Meldungskatalog, der durch *catd* identifiziert wird. Wenn eine Dateikennzahl verwendet wird, um den Typ *nl_catd* zu implementieren, dann wird diese Dateikennzahl geschlossen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

BEISPIEL

Siehe Beispiel im *Abschnitt 2.5*.

SIEHE AUCH

catopen(), *<nl_types.h>*, *Abschnitt 2.5*.

NAME

catgets - read a program message
Programmeldung lesen

DEFINITION

```
#include <nl_types.h>

char *catgets (catd, set_id, msg_id, s)
nl_catd catd;
int set_id, msg_id;
char *s;
```

BESCHREIBUNG

Die Funktion *catgets()* versucht, die Meldung *msg_id* in der Menge *set_id* aus dem Meldungskatalog, der durch *catd* identifiziert wird, zu lesen. Das Argument *catd* ist ein Meldungskatalog-Deskriptor, der durch einen vorausgegangenen Aufruf von *catopen()* gewonnen wurde. Das Argument *s* zeigt auf eine Standard-Meldungszeichenkette, die von *catgets()* geliefert wird, wenn die Funktion die angegebene Meldung nicht lesen kann.

ERGEBNIS

Wenn die angegebene Meldung erfolgreich gelesen werden konnte, dann liefert *catgets()* einen Zeiger auf einen internen Pufferbereich, der die mit einem Nullbyte abgeschlossene Medungszeichenkette enthält. Wenn der Aufruf aus igendeinem Grund fehlschlägt, dann wird *s* zurückgegeben.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Siehe das Beispiel im *Abschnitt 2.5*.

PORTABILITÄT

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

SIEHE AUCH

catopen(), *<nl_types.h>*, *Abschnitt 2.5*.

NAME

catopen - open a message catalogue
Meldungskatalog öffnen

DEFINITION

```
#include <nl_types.h>

nl_catd catopen (name, oflag)
char *name;
int oflag;
```

BESCHREIBUNG

Die Funktion *catopen()* öffnet einen Meldungskatalog und liefert einen Meldungskatalog-Deskriptor zurück. Das Argument *name* gibt den Namen des zu öffnenden Meldungskatalogs an. Wenn *name* einen '/' enthält, dann gibt *name* einen vollständigen Namen für den Meldungskatalog an. Anderfalls wird die Umgebungsvariable *NLSPATH* verwendet, wobei *name* für *%N* eingesetzt wird. (siehe auch *Abschnitt 2.5,*). Wenn *NLSPATH* in der Umgebung nicht existiert, oder wenn ein Meldungskatalog in irgeneiner der in *NLSPATH* angegebenen Komponenten nicht geöffnet werden kann, dann wird als Voreinstellung */usr/lib/nls/msg/%l/%N.cat* angenommen.

Weil für die Implementierung von Meldungskatalog-Deskriptoren Dateikennzahlen verwendet werden, wird das Bit FD_CLOEXEC gesetzt; siehe auch *<fcntl.h>*.

Wenn die besondere Syntax *programmname@archivname* für das Argument *name* verwendet wird, dann werden von *catopen()* nur *Archive* geöffnet. In diesem Fall wird in der Variablen NLSPATH das Schlüsselwort *%A* durch den Teil *archivname* von *name* ersetzt. Nur die Teile von NLSPATH, die ein *%A* enthalten, werden ausgewertet. Wenn das Archiv erfolgreich geöffnet werden konnte, dann wird der Eintrag *programmname* in diesem Archiv als Meldungskatalog verwendet.

Das Argument *oflag* ist für eine zukünftige Verwendung reserviert und sollten gleich 0 gesetzt werden. Das Ergebnis einer anderen Besetzung dieses Arguments ist undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *catopen()* einen Meldungskatalog-Deskriptor für die Verwendung in nachfolgenden Aufrufen von *catgets()* und *catclose()*. Andernfalls liefert *catopen()* den Wert *(nl_catd)* -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

HINWEIS

SINIX und einige andere Implementierungen von *catopen()* verwenden *malloc()* um Speicherplatz für die internen Pufferbereiche zu reservieren. Die Funktion *catopen()* kann fehlschlagen, wenn nicht genügend Speicherplatz für die Unterbringung dieser Puffer verfügbar ist.

PORTABILITÄT

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

Portabilitätshinweis:
Die spezielle Syntax *programmname@archivname* für das Argument *name* ist nur unter SINIX definiert. Andere X/Open-kompatible Implementierungen müssen diese Syntax nicht unterstützen.

BEISPIEL

Siehe das Beispiel im *Abschnitt 2.5*.

SIEHE AUCH

catclose(), *catgets()*, *<fcntl.h>*, *<nl_types.h>*, *Abschnitt 2.5*.

NAME

ceil - ceiling value function
Aufrundungsfunktion

DEFINITION

```
#include <math.h>

double ceil (x)
double x;
```

BESCHREIBUNG

Die Funktion *ceil()* berechnet den kleinsten ganzzahligen Wert, der nicht größer als *x* ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ceil()* den kleinsten ganzzahligen Wert, der nicht größer als *x* ist, dargestellt als ein Wert vom Typ *double*.

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

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

FEHLER

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

[ERANGE] Das Ergebnis würde einen Überlauf verursachen.

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

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

HINWEIS

Die von *ceil()* als *double* zurückgelieferte ganze Zahl kann unter Umständen nicht als *int* oder *long* dargestellt werden. Das Ergebnis sollte vor einer Zuweisung an einen ganzzahligen Datentyp überprüft werden, um undefinierte Ergebnisse von ganzzahligen Überläufen zu vermeiden.

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

SINIX und einige Implementierungen schreiben unter Fehlerbedingungen eine Fehlermeldung auf *stderr*. Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Das folgende Beispiel liest eine Gleitpunktzahl ein und rundet diese auf:

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

main()
{
   double x;
   printf("\nGeben Sie eine Zahl vom typ double ein: ");
   if (scanf("%lf",&x) == 1)
      printf("\nDie Zahl %g wird aufgerundet zu %f\n",x, ceil(x));
}
```

SIEHE AUCH

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

NAME

cfgetispeed - get input baud rate
Eingabe-Baudrate ermitteln

DEFINITION

```
#include <termios.h>

speed_t cfgetispeed (termios_p)
struct termios *termios_p;
```

BESCHREIBUNG

Die Funktion *cfgetispeed()* ermittelt die Eingabe-Baudrate aus der *termios*-Struktur auf die das Argument *termios_p* zeigt.

Falls der Wert in der Struktur *termios* nicht durch einen erfolgreichen Aufruf von *tcgetattr()* ermittelt wurde, ist das Verhalten der Funktion undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung, liefert die Funktion *cfgetispeed()* einen Wert von Typ *speed_t* zurück, der die Eingabe-Baudrate repräsentiert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

cfgetospeed(), cfsetispeed(), cfsetospeed(), tcgetattr(), <termios.h>, Abschnitt 2.4, Allgemeine Terminalschnittstelle.

NAME

cfgetospeed - get output baud rate
Ausgabe-Baudrate ermitteln

DEFINITION

```
#include <termios.h>

speed_t cfgetospeed (termios_p)
struct termios *termios_p;
```

BESCHREIBUNG

Die Funktion *cfgetospeed()* ermittelt die Ausgabe-Baudrate aus der *termios*-Struktur auf die das Argument *termios_p* zeigt.

Falls der Wert in der Struktur *termios* nicht durch einen erfolgreiche Aufruf von *tcgetattr()* ermittelt wurde, ist das Verhalten der Funktion undefiniert.

ERGEBNIS

Bei erfolgreicher Beendigung, gibt die Funktion *cfgetospeed() einen Wert von Typ speed_t* zurück, der die Ausgabe-Baudrate repräsentiert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

cfgetispeed(), cfsetispeed(), cfsetospeed(), tcgetattr(), <termios.h>, Abschnitt 2.6, Allgemeine Terminalschnittstelle.

NAME

cfsetispeed - set input baud rate
Eingabe-Baudrate festlegen

DEFINITION

```
#include <termios.h>

int cfsetispeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;
```

BESCHREIBUNG

Die Funktion *cfsetispeed()* setzt die Eingabe-Baudrate in der Struktur auf die *termios_p* zeigt, auf den Wert von *speed.*

Für jede beliebige Hardware werden nichtunterstützte Baudratenwechsel ignoriert. Dies gilt sowohl für den Wechsel zu Baudraten, die nicht von Hardware unterstützt werden, als auch für Wechsel, die Eingabe- und Ausgabe-Baudrate auf unterschiedliche Werte setzen, wenn die Hardware dies nicht unterstützt.

Dies hat keinen Einfluß auf die Hardware-Baudraten, solange nicht ein nachfolgender erfolgreicher Aufruf von *tcsetattr()* mit derselben Struktur vom Typ *termios* erfolgt ist.

ERGEBNIS

Die Funktion *cfsetispeed()* gibt bei Erfolg den Wert 0 zurück. Andernfalls wird das Ergebnis -1 geliefert und die Variable *errno* wird gesetzt um den Fehler anzuzeigen.

FEHLER

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

[EINVAL] Die Variable *speed* entspricht keiner gültigen Baudrate.

PORTABILITÄT

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

SIEHE AUCH

cfgetispeed(), cfgetospeed(), cfsetospeed(), tcsetattr(), <termios.h>, Abschnitt 2.6, Allgemeine Terminalschnittstelle.

NAME

cfsetospeed - set output baud rate
Ausgabe-Baudrate festlegen

DEFINITION

```
#include <termios.h>

int cfsetospeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;
```

BESCHREIBUNG

Die Funktion *cfsetospeed()* setzt die Ausgabe-Baudrate in der Struktur auf die *termios_p* zeigt, auf den Wert von *speed*. Die Null-Baudrate B0 wird benutzt, um die Verbindung zu beenden. Falls B0 angegeben wird, werden die Kontroll-Leitungen des Modems nicht länger angesprochen, wodurch normalerweise die Verbindung beendet wird.

Für jede beliebige Hardware werden nichtunterstützte Baudratenwechsel ignoriert. Dies gilt sowohl für den Wechsel zu Baudraten, die nicht von Hardware unterstützt werden, als auch für Wechsel, die Eingabe- und Ausgabe-Baudrate auf unterschiedliche Werte setzen, wenn die Hardware dies nicht unterstützt.

Dies hat keinen Einfluß auf die Hardware-Baudraten, solange nicht ein nachfolgender erfolgreicher Aufruf von *tcsetattr()* mit derselben Struktur vom Typ *termios* erfolgt.

ERGEBNIS

Die Funktion *cfsetospeed()* gibt bei Erfolg den Wert 0 zurück. Andernfalls wird das Ergebnis -1 geliefert und die Variable *errno* wird gesetzt um den Fehler anzuzeigen.

FEHLER

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

[EINVAL] Die Variable *speed* entspricht keiner gültigen Baudrate

PORTABILITÄT

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

SIEHE AUCH

cfgetispeed(), cfgetospeed(), cfsetispeed(), tcsetattr(), <termios.h>, Abschnitt 2.4, Allgemeine Terminalschnittstelle.

NAME

chdir - change working directory
aktuelles Dateiverzeichnis wechseln

DEFINITION

```
int chdir (path)
char *path;
```

BESCHREIBUNG

Die Funktion *chdir()* sorgt dafür, daß das durch das Argument *path* angegebene Dateiverzeichnis zum aktuellen Dateiverzeichnis wird. Dieses ist der Anfangspunkt für alle Pfadnamen-Suchen, die nicht mit dem Schrägstrich (/) beginnen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird der Wert -1 zurückgeliefert, das aktuelle Dateiverzeichnis bleibt unverändert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] Für irgendeine Komponente des Pfadnamens wird die Durchsucherlaubnis nicht erteilt.

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

[ENOENT] Das angegebene Dateiverzeichnis existiert nicht, oder *path* ist eine leere Zeichekette.

[ENOTDIR]
Eine Komponente das Pfadnamens ist kein Dateiverzeichnis.

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

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

PORTABILITÄT

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

BEISPIEL

Sie möchten das beim Programmaufruf übergebene Argument zum aktuellen Dateiverzeichnis machen und dessen Inhaltsverzeichnis ausdrucken.
Wenn Sie das Programm übersetzt haben, rufen sie es in der Form auf:

```
a.out pfad
```

Der Wechsel des Dateiverzeichnisses gilt nur für diesen Prozess. Andere Prozesse (außer Sohnprozesse) bleiben davon unberührt.

```
int main(argc,argv)
int argc;
char **argv;
{
   if (chdir(*++argv))
   {   perror("chdir");
       exit(1);
   }
   else system("ls -l");
      /* system() erzeugt einen Sohnprozess. An diesen */
      /* wird das aktuelle DVZ vererbt.                */
}
```

SIEHE AUCH

chroot(), *getcwd()*.

NAME

chmod - change mode of file
Dateizugriffsrechte änden

DEFINITION

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

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

BESCHREIBUNG

Die Funtkion *chmod()* ändert S_ISUID, S_ISGID und die Schutzbits der Datei, die durch das Argument *path* angesprochen wird, in die entsprechenden Bits des Arguments *mode* um. Die effektive Benutzernummer des Prozesses muß zum Eigentümer der Datei passen oder besondere Rechte besitzen, um dies zu tun.

S_ISUID, S_ISGID und die Schutzbits einer Datei werden in *<sys/stat.h>* beschrieben.

Wenn der aufrufende Prozß keine besonderen Rechte besitzt und die Gruppennummer der Datei nicht zur effektiven oder eine der weiteren Gruppennummern paßt und die Datei eine normale Datei ist, dann wird das Bit S_ISGID (Setze Gruppennummer bei Ausführung) in den Zugriffsrechten der Datei bei einer erfolgreichen Rückkehr von *chmod()* gelöscht.

Die Wirkung der Funktion *chmod()* auf Dateikennzahlen von Dateien, die zu diesem Zeitpunkt offen sind, ist implementierungsabhängig. Unter SINIX wird *chmod()* auch für offene Dateien ausgeführt.

Bei erfolgreicher Beendigung markiert die Funktion *chmod()* das Feld *st_ctime* der Datei zum Ändern.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Anderfalls wird -1 zurückgegeben und *errno* wird gesetzt, um den Fehler anzuzeigen. Wenn -1 zurückgegeben wird, dann findet keine Änderung der Dateizugriffsrechte statt.

FEHLER

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

[EACCES] Für eine Komponente des Pfadnamen-Anfangs existiert keine Durchsuchberechtigung.

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

[ENOTDIR] Eine Komponente des Anfangs von *path* ist kein Dateiverzeichnis.

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

[EPERM] Die effektive Benutzernummer entspricht nicht dem Eigentümer der Datei und der Prozeß besitzt auch keine besonderen Rechte.

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

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

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

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

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

HINWEIS

Um sicherzustellen, daß die Bits S_ISUID und S_ISGID gesetzt sind, sollte eine Anwendung, die dies fordert, nach einem erfolgreichen Aufruf von *chmod()* die Funktion *stat()* aufrufen, um dies zu überprüfen.

Dateikennzahlen, die zu diesem Zeitpunkt von einem Prozeß offen gehalten sind, können durch die Änderung der Dateizugriffsrechte ungültig werden, wenn diese so geändert werden, daß dieser Wert den Zugriff für den Prozeß verweigern würde. Eine Situation, in der dies passieren könnte, ist ein zustandsloses Dateisystem.

PORTABILITÄT

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

BEISPIEL

Das Programm ändert die Zugriffsrechte der als Argument übergebenen Datei so, daß Eigentümer und Gruppe lesen, schreiben und ausführen dürfen:

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

int main(argc,argv)
int argc;
char **argv;
{
   if (argv[1] != NULL)
      return chmod(argv[1],00770);
}
```

Das übersetzte Programm rufen Sie folgendermaßen auf:

```
a.out dateiname
```

SIEHE AUCH

chown(), *mkdir()*, *mkfifo()*, *open()*, *stat()*, *<sys/stat.h>*, *<sys/types.h>*.

NAME

chown - change owner and group of a file
Eigentümer und Gruppe einer Datei ändern

DEFINITION

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

int chown (path, owner, group)
char *path;
uid_t owner;
gid_t group;
```

BESCHREIBUNG

Das Argument *path* zeigt auf einen Pfadnamen, der eine Datei bezeichnet. Die Benutzer- und Gruppennummer der benannten Datei werden auf die numerischen Werte gesetzt, die in *owner* und *group* enthalten sind.

Nur Prozesses, deren effektive Benutzernummer gleich der Benutzernummer der Datei ist, oder die besondere Rechte haben, können die Benutzer- oder Gruppennummer einer Datei ändern. Wenn {_POSIX_CHOWN_RESTRICTED} für *path* aktiv ist (d.h. unter SINIX für alle Dateien), dann gilt:

- Die Änderung der Benutzernummer ist auf Prozesse mit besonderen Rechten beschränkt.
- Die Änderung der Gruppennummer ist einem Prozeß, dessen effektive Benutzernummer gleich der Benutzernummer der Datei ist, der aber keine besonderen Rechte hat, dann und nur dann erlaubt, wenn *owner* gleich der Benutzernummer der Datei und *group* entweder gleich der effektiven Benutzernummer des Prozesses oder aber gleich einer seiner weiteren Gruppennummern ist.

Wenn das Argument *path* eine nomale Datei bezeichnet, dann werden die Bits S_ISUID und S_ISGID der Datei bei erfolgreicher Rückkehr von *chown()* gelöscht, solange der Aufruf nicht von einem Prozeß mit besonderen Rechten erfolgte. In diesem Fall ist es implementierungs-abhängig, ob diese Bits geändert werden. Unter SINIX werden diese Bits nicht geändert. Wenn die Funktion *chown()* erfolgreich für eine Datei aufgerufen wird, die keine normale Datei ist, dann können diese Bits gelöscht werden. Diese Bits sind in *<sys/stat.h>* definiert.

Wenn *owner* oder *group* als *(uid_t) -1 oder (gid_t)* -1 angegeben werden, dann wird die entsprechende Nummer der Datei nicht geändert.

Bei erfolgreicher Beendigung markiert die Funktion *chown()* das Feld *st_ctime* dieser Datei zum Ändern.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und *errno* wird gesetzt, um den Fehler anzuzeigen. Wenn -1 zurückgeliefert wird, dann werden Eigentümer und Gruppe der Datei nicht geändert.

FEHLER

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

[EACCES] Für eine Komponente des Anfangs von *path* existiert keine Durchsuchberechtigung.

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

[ENOTDIR] Eine Komponente des Anfangs von *path* ist kein Dateiverzeichnis.

[ENOENT] Das Argument *path* zeigt auf eine Datei, die nicht existiert, oder auf eine leere Zeichenkette.

[EPERM] Die effektive Benutzernummer paßt nicht zum Eigentümer der Datei oder der aufrufende Prozeß besitzt keine besonderen Rechte obwohl {_POSIX_CHOWN_RESTRICTED} anzeigt, daß solche besonderen Rechte gefordert werden.

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

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

[EINVAL] Der Wert der angegebenen Benutzer- oder Gruppennummer wird nicht unterstützt, z.B. der Wert ist kleiner als 0.

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

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

PORTABILITÄT

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

SIEHE AUCH

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

NAME

chroot - change root directory
Root-Dateiverzeichnis wechseln

DEFINITION

```
int chroot (path)
char *path;
```

BESCHREIBUNG

Das Argument *path* zeigt auf einen Pfadnamen, der ein Dateiverzeichnis bezeichnet. Die Funktion *chroot()* sorgt dafür, daß das bezeichnete Dateiverzeichnis zum Wurzel-Dateiverzeichnis wird, dem Anfangspunkt für alle Pfadnamen, die mit dem Zeichen '/' beginnen. Das aktuelle Dateiverzeichnis des Benutzers wird durch die Funktion *chroot*() nicht beeinflußt.

Der Prozeß muß besondere Rechte haben, um das Wurzel-Dateiverzeichnis zu wechseln.

Der Eintrag .. im Wurzel-Dateiverzeichnis wird so interpretiert, daß er das Wurzel-Dateiverzeichnis selbst bedeutet. Daher kann .. nicht dazu verwendet werden, auf Dateien außerhalb des beim Wurzel-Dateiverzeichnis beginnenden Unterbaums zuzugreifen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion den Wert 0. Andernfalls wird der Wert -1 zurückgegeben und *errno* wird besetzt, um den Fehler anzuzeigen. Wenn -1 zurückgegeben wird, dann ist das Wurzel-Dateiverzeichnis nicht geändert worden.

FEHLER

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

[EACCES] Für eine Komponente von *path* existiert keine Durchsuchberechtigung.

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

[ENOENT] Das Argument *path* zeigt auf den Namen eines Dateiverzeichnisses, das nicht existiert, oder auf die leere Zeichenkette.

[ENOTDIR]
Eine Komponente des Pfadnamens *path* ist kein Dateiverzeichnis.

[EPERM] Die effektive Benutzernummer ist nicht die eines Prozesses mit besonderen Rechten.

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

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

PORTABILITÄT

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

SIEHE AUCH

chdir().

NAME

clearerr - clear indicators on a stream
Fehleranzeigen für einen Strom löschen

DEFINITION

```
#include <stdio.h>

void clearerr (stream)
FILE *stream;
```

BESCHREIBUNG

Die Funktion *clearerr()* löscht die Anzeigen für das Dateiende und Fehler für den Strom, auf den *stream* zeigt.

ERGEBNIS

Die Funktion *clearerr()* hat kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

<stdio.h>

NAME

clock - report CPU time used
Verbrauchte Rechenzeit angeben

DEFINITION

```
clock_t clock()
```

BESCHREIBUNG

Die Funktion *clock()* gibt den Betrag der verbrauchten Rechenzeit an. Die angegebene zeit ist die Summe der Rechenzeit des aufrufenden Prozesses und seiner beendeten Sohn-Prozesse, für die dieser *wait*(), *system*() oder *pclose*() ausgeführt hat.

ERGEBNIS

Die Funktion *clock()* liefert den Betrag der Rechenzeit (in Mikrosekunden) seit dem ersten Aufruf von *clock()*.

PORTABILITÄT

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

HINWEIS

Der von *clock()* zurückgelieferte Wert ist in Mikrosekunden definiert, um die Kompatibilität zwischen Systemen herzustellen, die Systemuhren mit verschiedenen Auflösungen besitzen. Die Auflösung auf einem speziellen Systems muß auch nicht der Genauigkeit einer Mikrosekunde entsprechen.

Der von *clock()* zurückgelieferte Wert kann auf einigen Systemen in 0 überlaufen. Auf einem System mit *long*-Werten von 32 Bit geschieht dies nach 2147 Sekunden bzw. 36 Minuten.

SIEHE AUCH

system(), *times*(), *wait*().

NAME

close - close a file descriptor
Dateikennzahl schließen

DEFINITION

```
int close (fildes)
int fildes;
```

BESCHREIBUNG

Die Funktion *close()* gibt die durch *fildes* Dateikennzahl frei (d.h. macht sie als Rückgabe für nachfolgende Aufrufen von *open()* durch diesen Prozeß verfügbar). Alle bestehenden Sperren für Dateibereiche auf der Datei, die dem Prozeß gehören, die dieser Dateikennzahl zugeordnet ist, werden entfernt (d.h. entsperrt).

Wenn die Funktion *close()* durch ein Signal unterbrochen wird, das abgefangen werden soll, dann liefert Sie -1 und setzt *errno* auf [EINTR]; der Zustand von *fildes* ist danach unbestimmt.

Sobald alle Dateikennzahlen, die einer Pipe oder einer FIFO-Gerätedatei zugeordnet waren, geschlossen worden sind, sind alle Daten, die noch in dieser Pipe oder FIFO enthalten sind, verworfen.

Sobald alle Dateikennzahlen, die einer offenen Dateibeschreibung zugeordnet waren, geschlossen worden sind, wird die offene Dateibeschreibung freigegeben.

Wenn der Verweiszähler der Datei gleich 0 ist und wenn alle Dateikennzahlen zu dieser Datei geschlossen worden sind, dann wird der von dieser Datei belegte Platz freigegeben und es kann nicht länger darauf zugegriffen werden.

ERGEBNIS

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

FEHLER

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

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

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

HINWEIS

Eine Anwendung, die zum Öffnen einer Datei die *stdio*-Routine *fopen()* verwendet hat, sollte die entsprechende Routine *fclose()* zum Schließen verwenden, anstelle von *close()*.

PORTABILITÄT

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

SIEHE AUCH

fclose(), *fopen()*, *open()*.

NAME

closedir - close a directory stream
Dateiverzeichnis-Strom schließen

DEFINITION

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

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

BESCHREIBUNG

Die Funktion *closedir()* schließt den Dateiverzeichnis-Strom, der durch das Argument *dirp* angegeben wird. Nach der Rückkehr zeigt der Wert von *dirp* nicht länger auf ein zugreifbares Objekt des Typs *DIR*. Wenn, wie unter SINIX, eine Dateikennzahl verwendet wird, um den Typ *DIR* zu implementieren, dann wird diese Dateikennzahl geschlossen.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *closedir()* den Wert 0 zurück. Andernfalls wird der Wert -1 zurückgegeben und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EBADF] Das Argument *dirp* bezieht sich nicht auf einen offenen Dateiverzeichnis-Strom

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

[EINTR] Die Funktion *closedir()* wurde von einem Signal unterbrochen.

PORTABILITÄT

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

SIEHE AUCH

opendir(), *<dirent.h>*, *<sys/types.h>*.

NAME

compile - produce compiled regular expression
regulären Ausdruck übersetzen

DEFINITION

```
char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;
```

BESCHREIBUNG

Siehe unter *regexp()*.

PORTABILITÄT

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

NAME

cos - cosine function
Cosinus

DEFINITION

```
#include <math.h>

double cos (x)
double x;
```

BESCHREIBUNG

Die Funktion *cos()* berechnet den Cosinus von *x*, das im Bogenmaß angegeben ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *cos()* den Cosinus von *x*.

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

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

FEHLER

Die Funkton *cos()* kann fehlschlagen, wenn gilt:

[EDOM] *x* ist -HUGE_VAL bzw. +HUGE_VAL.

[ERANGE] Die Größe von *x* ist derart, daß ein vollständiger oder teilweiser Verlust an Signifikanz daraus resultiert.

HINWEIS

Die Funktion *cos()* kann an Genauigkeit einbüßen, wenn ihr Argument sehr von 0 verschieden ist.

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

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

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

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

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *sin()*.

SIEHE AUCH

acos(), *isnan()*, *matherr()*, *sin()*, *tan()*, *<math.h>*.

NAME

cosh - hyperbolic cosine function
Cosinus Hyperbolicus

DEFINITION

```
#include <math.h>

double cosh (x)
double x;
```

BESCHREIBUNG

Die Funktion *cosh()* berechnet den Cosinus Hyperbolicus von *x*.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *cosh()* den Cosinus Hyperbolicus von *x*.

Wenn das Ergebnis einen Überlauf verursachen würde, dann wird HUGE_VAL zurückgegeben und *errno* kann gleich [ERANGE] gesetzt sein.

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

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

FEHLER

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

[ERANGE] Das Ergebnis würde einen Überlauf verursachen.

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

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

HINWEIS

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

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

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

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

PORTABILITÄT

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

SIEHE AUCH

isnan(), *matherr()*, *sinh()*, *tanh()*, *<math.h>*.

NAME

creat - create a new file or rewrite an existing one
Neue Datei anlegen oder vorhandene überschreiben

DEFINITION

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

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

BESCHREIBUNG

Der Funktionsaufruf

creat(path, mode)

ist äquivalent zu

open(path, O_WRONLY | O_CREAT | O_TRUNC, mode)

ERGEBNIS

Siehe unter *open()*.

FEHLER

Siehe unter *open()*.

PORTABILITÄT

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

BEISPIEL

Anlegen der Datei *neu* mit der Schutzbitbelegung: -rwsr--r--

```
#include <sys/types.h>
#include <sys/stat.h>
#include <femtl.h>
#define MODE 04744

main()
{
   int dk;          /* Prozessmaske auf 0 setzen,   */
                    /*  d.h. keine Einschraenkungen */
   umask(000);
   dk = creat("neu",MODE);
   printf("%d\n",dk);
}
```

SIEHE AUCH

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

NAME

crypt - string encoding function
Zeichenkette verschlüsseln

DEFINITION

```
char *crypt (key, salt)
char *key, *salt;
```

BESCHREIBUNG

Die Funktion *crypt()* ist eine Zeichenketten-Verschlüsselungs-Funktion. Der verwendete Algorithmus ist implementierungsabhängig, daher sollte sich eine Anwendung nicht darauf verlassen, daß die Ergebnisse dieser Funktion unter allen X/Open-kompatiblen Systemen, die diese Funktion unterstützen, gleich sind.

Das Argument *key* zeigt auf die zu verschlüsselnde Zeichenkette. Das Argument *salt* ist eine Zeichenkette, die aus den Zeichen [a-zA-Z0-9./] gebildet wird; die ersten beiden Zeichen dieser Zeichenkette können dazu verwendet werden, den Verschlüsselungsalgorithmus weiter zu verändern.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *crypt()* einen Zeiger auf die verschlüsselte Zeichenkette. Die ersten beiden Zeichen des zurückgelieferten Wertes sind die aus dem Argument *salt*.

Andernfalls liefert sie NULL und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[ENOSYS]
Die Funktion *crypt()* wird unter dieser Implementierung nicht unterstützt.

HINWEIS

Das Ergebnis von *crypt()* zeigt auf statische Daten, die bei jedem Aufruf überschrieben werden.

Die von dieser Funktion gelieferten Ergebnisse müssen unter verschiedenen *X/Open*-kompatiblen Systemen nicht kompatibel sein.

PORTABILITÄT

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

SIEHE AUCH

encrypt(), *setkey()*.

NAME

ctermid - generate pathname for controlling terminal
Pfadname für kontrollierendes Terminal erzeugen

DEFINITION

```
#include <stdio.h>

char *ctermid (s)
char *s;
```

BESCHREIBUNG

Die Funktion *ctermid()* erzeugt eine Zeichenkette, die auf das aktuelle kontrollierende Terminal des aktuellen Prozesses verweist, wenn sie als Pfadname verwendet wird. Die Zeichenkette wird dort abgelegt, wohin *s* zeigt. *s* muß ein char-Vektor mit wenigstens {L_ctermid} Elementen sein. Wenn *s* der Nullzeiger ist, dann kann die Zeichenkette in einem internen, statischen Bereich abgelegt werden, dessen Inhalt durch den nächsten Aufruf von *ctermid()* überschrieben werden kann.

Wenn die Funktion *ctermid()* einen Pfadnamen zurückliefert, dann ist ein Zugriff auf diese Datei nicht garantiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ctermid()* die Adresse der erzeugten Zeichenkette.

Die Funktion *ctermid()* liefert eine leere Zeichenkette, wenn der Pfadname für das kontrollierende Terminal nicht ermittelt werden kann oder wenn ein Fehler auftritt.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Der Unterschied zwischen *ctermid()* und *ttyname()* ist der, daß *ttyname()* als Argument eine Dateikennzahl benötigt und den Pfadnamen der Datensichtstation liefert, die dieser Dateikennzahl zugeordnet ist. Demgegenüber liefert *ctermid()* eine Zeichenkette (wie z.B. */dev/tty*), die auf das aktuelle kontrollierende Terminal verweist, wenn sie als Pfadname benutzt wird.

PORTABILITÄT

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

SIEHE AUCH

ttyname(), *<stdio.h>*.

NAME

ctime - convert time value to date and time string
Datum und Zeit in Zeichenkette umwandeln

DEFINITION

```
#include <time.h>

char *ctime (clock)
time_t *clock;
```

BESCHREIBUNG

Die Funktion *ctime()* wandelt die Zeit, auf die *clock* zeigt, und welche die Zeit in Sekunden seit dem Epochenwert repräsentiert, in die örtliche Zeit in Form einer Zeichenkette um. Dies ist äquivalent zu

asctime (localtime (clock))

ERGEBNIS

Die Funktion *ctime()* liefert den Zeiger, der von der Funktion *asctime()* mit genau der gleichen, aufgeschlüsselten Zeit als Argument geliefert wird.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Das Ergebnis kann auf einen statischen Bereich zeigen, dessen Inhalt bei jedem Aufruf überschrieben wird.

PORTABILITÄT

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

BEISPIEL

Das Programm wandelt einen Wert in Ortszeit um und gibt das Ergebnis in Form einer englischen Datum-mit-Uhrzeit-Angabe aus:

```
#include <time.h>

main()
{
   long sek,time();
   char *ctime();

   sek = time(0L);
   printf("%s",ctime(&sek));
}
```

SIEHE AUCH

asctime(), *getenv()*, *localtime()*, *time()*, *<time.h>*.

NAME

isalnum, isalpha, isascii, iscntrl,
isdigit, isgraph, islower, isspace,
isprint, ispunct, isupper, isxdigit - character classification
Zeichenklassifizierung

DEFINITION

```
#include <ctype.h>

int isalpha (c)
int c;
.
.
.
```

BESCHREIBUNG

Die Funktionen zur Zeichenklassifizierung werden ab dieser Ausgabe jeweils unter ihrem Namen beschrieben. Eine ausführliche Beschreibung finden Sie unter dem jeweiligen Namen der Funktion (siehe Abschnitt SIEHE AUCH).

PORTABILITÄT

Die Funktionen zur Zeichenklassifizierung sind im X/Open-Standard (Ausgabe 3) definiert.

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *nl_istype()*, *nl_settype()*, *setlocale()*, *<ctype.h>*.

NAME

addch, waddch, mvaddch, mvwaddch - add character to window
Zeichen in Fenster ausgeben

DEFINITION

```
#include <curses.h>

int addch(ch)
chtype ch;

int waddch(win,ch)
WINDOW *win;
chtype ch;

int mvaddch(y, x, ch)
int y, x;
chtype ch;

int mvwaddch(win, y, x, ch)
WINDOW *win;
int y, x;
chtype ch;
```

BESCHREIBUNG

Das Zeichen *ch* wird, ähnlich wie bei der Funktion *putchar()*, an der aktuellen Position der Schreibmarke im Fenster ausgegeben, und die Schreibmarke wird um eine Position nach rechts bewegt. Ist die Schreibmarke am rechten Rand des Fensters angelangt, so wird automatisch ein Zeilenvorschub durchgeführt. Ist die Schreibmarke am Ende des Bildlaufbereichs angelangt, so wird - falls *scrollock()* aktiv ist - der Bildlaufbereich um eine Zeile nach oben verschoben.

Bei den Funktionen *mvaddch()* und *mvwaddch()* wird die Schreibmarke vor der Ausgabe an die Position (*x*,*y*) des Fensters bewegt. Die Funktionen *addch()* und *mvaddch()* geben das Zeichen in das Standardfenster *stdscr* aus, bei den Funktionen *waddch()* und *mvwaddch()* gibt das Argument *win* an, in welches Fenster ausgegeben werden soll.

Ist *ch* ein Tabulator-, Neue Zeile- oder Rückschritt-Zeichen, so wird die Schreibmarke der Funktion des Zeichens entsprechend im Fenster bewegt. Bei einem Neue Zeile-Zeichen wird vor der Durchführung des Zeilenvorschubs die Funktion *clrtoeol()* ausgeführt. Bei einem Tabulatorzeichen wird die Schreibmarke zur nächsten Tabulatorposition im Fenster bewegt. Ist *ch* ein anderes Steuerzeichen, so wird es in der Notation ^X dargestellt.

Wird nach der Eingabe eines Steuerzeichens die Funktion *winch()* aufgerufen, so wird nicht das Steuerzeichen, sondern diese Darstellung für das Steuerzeichen zurückgegeben.

Ein Bildschirmmerkmal kann mit einem Zeichen durch logische ODER-Verknüpfung kombiniert werden. Dies bewirkt, daß auch diese Merkmale aktiviert werden (diese Möglichkeit wurde geschaffen, damit Text samt Merkmalen mit den Funktionen *inch()* und *addch()* an eine andere Stelle kopiert werden kann); vgl. Eintrag *curses: standout()*.

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

addch(), *mvaddch()* und *mvwaddch()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

addstr, waddstr, mvaddstr, mvwaddstr - add string to window
Zeichenkette auf Fenster schreiben

DEFINITION

```
#include <curses.h>

int addstr(str)
char *str;

int waddstr(win, str)
WINDOW *win;
char *str;

int mvaddstr(y, x, str)
int y, x;
char *str;

int mvwaddstr(win, y, x, str)
WINDOW *win;
int x, y;
char *str;
```

BESCHREIBUNG

Mit diesen Funktionen werden alle Zeichen der mit dem Null-Zeichen abgeschlossenen Zeichenkette *str* in das angegebene Fenster geschrieben. Hierfür könnte ebenso *waddch*() für jedes Zeichen in der Zeichenkette aufgerufen werden.

Bei den Funktionen *mvaddstr()* und *mvwaddstr()* wird die Schreibmarke vor der Ausgabe an die Position (*x*,*y*) bewegt. Die Funktionen *addstr()* und *mvaddstr()* geben die Zeichenkette in das Standardfenster *stdscr* aus, bei den Funktionen *waddstr()* und *mvwaddstr()* gibt das Argument *win* an, in welches Fenster ausgegeben werden soll.

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

addstr(), *mvaddstr()* und *mvwaddstr()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

attroff, attron, attrset, standend, standout, wstandend, wstandout, wattroff, wattron, wattrset - attribute manipulation
Fenster-Attribute behandeln

DEFINITION

```
#include <curses.h>

int attroff(attrs)
int attrs;

int wattroff(win, attrs)
WINDOW *win;
int attrs;

int attron(attrs)
int attrs;

int wattron(win, attrs)
WINDOW *win;
int attrs;

int attrset(attrs)
int attrs;

int wattrset(win, attrs)
WINDOW *win;
int attrs;

int standend()

int wstandend(win)
WINDOW *win;

int standout()

int wstandout(win)
WINDOW *win
```

BESCHREIBUNG

Mit diesen Funktionen werden die aktuellen Attribute des angegebenen Fensters behandelt. Bei den Attributen kann es sich um eine beliebige Kombination aus A_STANDOUT, A_REVERSE, A_BOLD, A_DIM, A_BLINK und A_UNDERLINE handeln. Diese Konstanten werden in der Include-Datei *<curses.h>* definiert; sie können mit dem C-Operator | (bitweise ODER) miteinander verknüpft werden.

Die aktuellen Attribute eines Fensters werden allen Zeichen zugeordnet, die mit der Funktion *waddch()* in ein Fenster geschrieben werden. Die Attribute sind fest mit den Zeichen verknüpft; sie gehen auch dann nicht verloren, wenn der Text im Fenster verschoben wird und Zeichen bzw. Zeilen eingefügt bzw. gelöscht werden. Sofern dies auf der betreffenden Datensichtstation möglich ist, werden die Attribute der Zeichen auf dem Bildschirm sichtbar gemacht.

Mit der Funktion *attrset(attrs)* werden die aktuellen Attribute des Fensters auf *attrs* gesetzt. Mit *attroff(attrs)* werden die angegebenen Attribute aufgehoben; die übrigen Attribute bleiben unverändert erhalten. Mit *attron(attrs)* werden die angegebenen Attribute aktiviert; an den übrigen Attributen ändert sich nichts. *standout()* ist funktionsgleich mit attron(A_STANDOUT), das heißt mit dieser Funktion werden alle nachfolgenden Ausgaben besonders hervorgehoben. *standend()* ist äquivalent zu *attrset*(0), das heißt, mit dieser Funktion werden alle Attribute aufgehoben. Alle diese Funktionen arbeiten mit dem Standardfenster (*stdscr*).

Die Funktionen *wattroff()*, *wattron()*, *wattrset()*, *wstandend()* und *wstandout()* haben dieselbe Wirkung wie ihre bereits beschriebenen Entsprechungen. Diese Funktionen arbeiten jedoch mit dem als Argument *win* angegebenen Fenster.

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

attroff(), *attron()* und *attrset()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

attron - attribute manipulation
Fenster-Attribute behandeln

DEFINITION

```
#include <curses.h>

int attron(attrs)
int attrs;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

attrset - attribute manipulation
Fenster-Attribute behandeln

DEFINITION

```
#include <curses.h>

int attrset(attrs)
int attrs;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

baudrate - return terminal baud rate
Übertragungsgeschwindigkeit der Datensichtstation

DEFINITION

```
int baudrate()
```

BESCHREIBUNG

Die Funktion *baudrate()* liefert die Ausgabebaudrate der Datensichtstation. Die zurückgelieferte Zahl vom Typ int liefert dafür den Wert in Bits pro Sekunde, z.B. 9600 (Bits pro Sekunde).

ERGEBNIS

Die Funktion liefert die Ausgabebaudrate der Datensichtstation in Bits pro Sekunde.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

beep, flash - generate audio-visual alarm
audiovisuelle Signale abgeben

DEFINITION

```
#include <curses.h>

int beep()

int flash()
```

BESCHREIBUNG

Diese Funktionen bewirken die Ausgabe von akustischen oder optischen Signalen. Bei *beep()* wird ein akustisches Signal ausgegeben; ist dies nicht möglich, so wird *flash()* durchgeführt (falls möglich). Bei *flash()* blinkt der Bildschirm; ist dies nicht möglich, so wird ersatzweise *beep()* durchgeführt (falls möglich). Kann die Datensichtstation weder akustische noch optische Signale ausgeben, so haben diese Funktionen keine Auswirkungen. Die Möglichkeit eines akustischen Signals (Piepston oder Klingel) ist bei fast allen Datensichtstationen gegeben; optische Signale dagegen sind nur auf wenigen Datensichtstationen möglich.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

box - draw box
Rahmen zeichnen

DEFINITION

```
#include <curses.h>

int box(win, vert, hor)
WINDOW *win;
chtype vert, hor;
```

BESCHREIBUNG

Um das Fenster wird ein Rahmen gezogen. Mit *vert* und *hor* werden die zu benutzenden Zeichen angegeben. Sind *vert* und *hor* gleich 0, so werden die entsprechenden Standardzeichen verwendet.

ERGEBNIS

Die Funktion *box()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Das nachfolgende Beispiel zeichnet einen Rahmen um den gesamten Bildschirm (Funktionsergebnisse werden ignoriert und eine Signalbehandlung findet nicht statt):

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

main()
{  /* curses initialisieren und RAW-Modus einschalten */
   if ( initscr() == NULL )
      exit(1);
   (void) raw();

   /* Bildschirm loeschen */
   (void) clear(); (void) refresh();
   /* Rahmen mit Standardzeichen zeichnen */
   (void) box(stdscr, 0, 0); (void) refresh();

   /* Auf Eingabe warten */
   (void) getch();

   /* curses beenden */
   (void) endwin();
}
```

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

cbreak, nocbreak - set/clear cbreak mode
BREAK-Modus ein/ausschalten

DEFINITION

```
int cbreak()

int nocbreak()
```

BESCHREIBUNG

Mit diesen beiden Funktionen kann auf einer Datensichtstation der CBREAK-Modus ein- und ausgeschaltet werden. Ist der CBREAK-Modus eingeschaltet (Aufruf von *cbreak()*), so werden die vom Benutzer eingegebenen Zeichen sofort an das Programm weitergeleitet; die ERASE- und KILL-Zeichen werden gegebenenfalls ignoriert. Ist dieser Modus ausgeschaltet (Aufruf von nocbreak()), so werden die vom Benutzer eingegebenen Zeichen bis zum Empfang eines Neue-Zeile- oder Wagenrücklauf-Zeichens in einem Bildschirmpuffer zwischengespeichert. Unterbrechungs- und Datenflußsteuerungszeichen werden jedoch weiterhin verarbeitet. Die Einstellung dieses Modus zu Beginn einer Arbeitssitzung hängt von seiner Einstellung bei der vorhergehenden Arbeitssitzung ab.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

clear, wclear - clear window
Fenster löschen

DEFINITION

```
#include <curses.h>

int clear()

int wclear(win)
WINDOW *win
```

BESCHREIBUNG

Die Funktion *clear()* löscht das Standardfenster, die Funktion *wclear() löscht das durch das Argument win* angegebene Fenster.

Diese Funktionen leisten dasselbe wie *erase*() und *werase*(); nur wird zusätzlich *clearok*() aufgerufen, so daß der Bildschirm beim nächsten Aufruf von *wrefresh*() für dieses Fenster vollständig gelöscht und vollkommen neu aufgebaut wird.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

clear() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

clearok - enable screen clearing
Bildschirmlöschen aktivieren

DEFINITION

```
#include <curses.h>

int clearok(win, bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Ist *bf* gleich TRUE, so wird mit dem nächsten Aufruf von *wrefresh*() für dieses Fenster der Bildschirm vollständig gelöscht und das Bild auf dem Bildschirm vollkommen neu aufgebaut. Dies ist vor allem hilfreich, wenn der Inhalt des Bildschirms unbestimmt ist; teilweise wird diese Funktion auch einfach aus optischen Gründen verwendet.

ERGEBNIS

Die Funktion *clearok()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

clrtobot, wclrtobot - clear to end of screen
Löschen bis zum Ende des Bildschirms

DEFINITION

```
#include <curses.h>

int clrtobot()

int wclrtobot(win)
WINDOW *win;
```

BESCHREIBUNG

Mit der Funktion *clrtobot* werden alle Zeilen im Standardfenster unterhalb der aktuellen Position der Schreibmarke gelöscht. Dazu gehören auch die Zeichen, die sich zur Rechten der Schreibmarke und an der Position der Schreibmarke im Standardfenster befinden.

Die Funktion *wclrtobot()* verhält sich ebenso wie die Funktion *clrtobot()*, nur daß hier zusätzlich mit dem Argument *win* ein spezielles Fenster angegeben werden kann.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

clrtobot() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

clrtoeol, wclrtoeol - clear to end of line
Löschen bis zum Zeilenende

DEFINITION

```
#include <curses.h>

int clrtoeol()

int wclrtoeol(win)
WINDOW *win
```

BESCHREIBUNG

Die Funktion *clrtoeol()* löscht Zeichen zur Rechten und an der aktuellen Position der Schreibmarke im Standardfenster. Bei der Funktion *wclrtoeol()* kann zusätzlich mit dem Argument *win* ein Fenster angegeben werden.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

clrtoeol() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

def_prog_mode, def_shell_mode - save terminal modes
aktuellen Datensichtstationsmodus speichern

DEFINITION

```
int def_prog_mode()

int def_shell_mode()
```

BESCHREIBUNG

Der aktuelle Zustand der Datensichtstation wird für die Funktionen *reset_prog_mod*() und *reset_shell_mode*() als "Programm-" (in *curses*) oder "Shell-" (nicht in *curses*) Zustand abgespeichert. Bei *initscr*() werden diese Funktionen automatisch durchgeführt.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

reset_prog_mode(), *reset_shell_mode()*.

NAME

delay_output - cause short delay
Verzögerte Ausgabe

DEFINITION

```
int delay_output(ms)
int ms;
```

BESCHREIBUNG

Mit *ms* wird angegeben, um wieviele Millisekunden die Ausgabe verzögert werden soll. Diese Funktion wird nicht von allen Systemen unterstützt.

ERGEBNIS

Die Funktion *delay_output()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wdelch, mvdelch, mvwdelch, wdelch - remove character from window
Zeichen im Fenster löschen

DEFINITION

```
#include <curses.h>

int delch()

int wdelch(win)
WINDOW *win;

int mvdelch(y,x)
int y, x;

int mvwdelch(win, y, x)
WINDOW *win;
int y, x;
```

BESCHREIBUNG

Die Funktion *delch()* löscht das Zeichen unter der Schreibmarke im Standardfenster. Alle Zeichen auf derselben Zeile, die sich rechts des gelöschten Zeichens befinden, werden um eine Position nach links verschoben; das letzte Zeichen der Zeile wird mit einem Leerzeichen aufgefüllt.

Bei den Funktionen *mvdelch()* und *mvwdelch()* wird die Schreibmarke vor dem Löschen an die Position (x,y) bewegt. Die Funktionen *delch()* und *mvdelch()* löschen das Zeichen im Standardfenster *stdscr*, bei den Funktionen *wdelch()* und *mvwdelch()* gibt das Argument *win* an, in welchem Fenster gelöscht werden soll.

Die Position der Schreibmarke wird durch das Löschen nicht verändert. Bei den Funktionen *delch()* und *wdelch()* entspricht sie der Position vor dem Aufruf der Funktion. Bei den Funktionen *mvdelch()* und *mvwdelch()* ist sie nach dem Ende der Funktion gleich (x, y).

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

delch(), *mvdelch()* und *mvwdelch()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

deleteln, wdeleteln - remove line from window
Zeile im Fenster löschen

DEFINITION

```
#include <curses.h>

int deleteln()

int wdeleteln(win)
WINDOW *win;
```

BESCHREIBUNG

Die Funktion *deleteln()* löscht die Zeile im Standardfenster, auf der sich die Schreibmarke befindet. Alle Zeilen unterhalb der aktuellen Zeile schließen auf. Die unterste Zeile im Fenster wird gelöscht. An der Position der Schreibmarke ändert sich nichts.

Bei der Funktion *wdeleteln()* kann mit dem Argument *win* zusätzlich ein bestimmtes Fenster angegeben werden.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

deleteln() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

delwin - delete window
Fenster löschen

DEFINITION

```
#include <curses.h>

delwin(win)
WINDOW *win;
```

BESCHREIBUNG

Mit dieser Funktion wird das angegebene Fenster gelöscht; der Speicherplatz, der durch den Inhalt dieses Fensters beansprucht wurde, wird freigemacht. Gibt es für dieses Fenster ein oder mehrere Unterfenster, so sollten diese vor dem Löschen des Hauptfensters gelöscht werden.

ERGEBNIS

Die Funktion *delwin()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses:subwin(), Abschnitt 2.7 .

NAME

doupdate - do efficient refresh
Effiziente Aktualisierung

DEFINITION

```
#include <curses.h>

int doupdate()
```

BESCHREIBUNG

Diese Funktion dient dazu, mehrfache Aktualisierungen effizient durchzuführen. Eine ausführliche Beschreibung finden Sie unter *curses: wnoutrefresh()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

echo, noecho - enable/disable terminal echo
Ausgabe unterdrücken

DEFINITION

```
int echo()

int noecho()
```

BESCHREIBUNG

Mit diesen Funktionen kann festgelegt werden, ob die vom Benutzer eingegebenen Zeichen durch die Eingabefunktion auf den Bildschirm geschrieben werden sollen. Standardmäßig werden die Zeichen ausgegeben. Bei interaktiven Programmen, bei denen das Paket *curses* zur Anwendung kommt, wird jedoch häufig einer kontrollierten Zeichenausgabe der Vorzug gegeben; daher wird bei derartigen Programmen die Zeichenausgabe häufig ausgeschaltet. Aufeinanderfolgende Aufrufe von *echo*() und *noecho*() führen nicht zum Löschen des Zeichenpuffers.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

endwin - restore initial terminal environment
Parameter der Datensichtstation zurücksetzen

DEFINITION

```
int endwin()
```

BESCHREIBUNG

Die Funktion *endwin*() sollte in einem Programm auf jeden Fall einer Funktion zur Beendigung des Programms oder zum vorübergehenden Verlassen von *curses* vorangestellt werden. Mit dieser Funktion wird die Datensichtstation wieder in den Anfangszustand versetzt, die Schreibmarke in die linke untere Ecke des Bildschirms bewegt und die Datensichtstation in den geeigneten normalen Betriebszustand zurückgesetzt. Mit *refresh*() und *doupdate*() kann das vorübergehend verlassene *curses* wieder aktiviert werden.

ERGEBNIS

Die Funktion *endwin()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

erase, werase - copy blanks into window
Fenster mit Leerzeichen füllen

DEFINITION

```
#include <curses.h>

int erase()

int werase(win)
WINDOW *win;
```

BESCHREIBUNG

Mit diesen Funktionen wird an jede Zeichenposition im Fenster ein Leerzeichen gesetzt, d.h. das Fenster wird gelöscht.

Die Funktion *erase()* löscht das Standardfenster *stdscr*, die Funktion *werase()* das Fenster, das mit dem Argument *win* angegeben wird.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

erase() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

erasechar - return current ERASE character
Ermitteln des ERASE-Zeichens

DEFINITION

```
char erasechar()
```

BESCHREIBUNG

Die Funktion *erasechar()* liefert das aktuelle ERASE-Zeichen das Benutzers.

ERGEBNIS

Siehe unter BESCHREIBUNG.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

flash - generate audio-visual alarm
audiovisuelle Signale abgeben

DEFINITION

```
#include <curses.h>

int flash()
```

BESCHREIBUNG

Diese Funktion bewirkt die Ausgabe von akustischen oder optischen Signalen. Eine ausführliche Beschreibung von *flash()* finden Sie unter *curses: beep()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

flushinp - discard type-ahead
Zeichenpuffer löschen

DEFINITION

```
#include <curses.h>

int flushinp()
```

BESCHREIBUNG

Mit dieser Funktion können diejenigen Zeichen aus dem Puffer gelöscht werden, die vom Benutzer eingegeben, jedoch noch nicht vom Programm eingelesen worden sind.

ERGEBNIS

Die Funktion *flushinp()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

getch, wgetch, mvwgetch, wgetch - read character
Zeichen einlesen

DEFINITION

```
#include <curses.h>

int getch()

int wgetch(win)
WINDOW *win;

int mvgetch(y,x)
int y,x;

int mvwgetch(win,y,x)
WINDOW *win;
int y,x;
```

BESCHREIBUNG

Von der Datensichtstation, die dem Fenster zugeordnet ist, wird ein Zeichen eingelesen. Wenn die Betriebsart *nodelay()* aktiv ist und nicht gewartet wird, bis Eingaben zur Verfügung stehen, so wird der Wert ERR zurückgegeben. Wenn *delay()* aktiv ist, so wird das Programm blockiert, bis vom System Text an das Programm übergeben wird. Je nach dem Status von *cbreak*() kommt es hierzu nach einem einzelnen Zeichen oder nach dem ersten Neue Zeile-Zeichen. Das Zeichen wird zusätzlich im angegebenen Fenster ausgegeben, es sei denn, *noecho*() ist aktiv.

Wenn *keypad*() aktiv ist und eine Funktionstaste betätigt wird, so werden nicht die unbearbeiteten Zeichen ausgegeben (raw-Modus), sondern sondern ein Wert, der die Funktionstaste darstellt (siehe auch BEISPIEL). Den Funktionstasten sind in der Include-Datei *<curses.h>* Konstanten zugeordnet; die Namen dieser Funktionstasten-Konstanten beginnen mit KEY_. Wird ein Zeichen eingelesen, bei dem es sich um den Beginn einer Funktionstaste handeln könnte (z.B. das Escapezeichen), so schaltet *curses* einen Zeitgeber ein. Folgen die übrigen Zeichen nicht innerhalb einer bestimmten Frist, so wird das Zeichen unbearbeitet übergeben; andernfalls wird der Wert ausgegeben, der der Funktionstaste zugeordnet ist. Aus diesem Grund wird die Betätigung der Esc-Taste durch den Benutzer mit etwas Verzögerung an das Programm weitergegeben (es wird davon abgeraten, die Escape-Taste zur Abfrage eines Einzelzeichens zu benutzen).

Bei den Funktionen *mvgetch()* und *mvwgetch()* wird die Schreibmarke vor dem Einlesen an die Position (*x*,*y*) bewegt. Die Funktionen *getch()* und *mvgetch()* lesen das Zeichen im Standardfenster *stdscr* ein, bei den Funktionen *wgetch()* und *mvwgetch()* gibt das Argument *win* an, in welchem Fenster eingelesen werden soll.

ERGEBNIS

Alle Funktionen liefern den Wert des eingelesenen Zeichens bei Erfolg und ERR, wenn kein Zeichen verfügbar und die Betriebsart *nodelay()* aktiv ist.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

getch(), *mvgetch()* und *mvwgetch()* sind Makros.

BEISPIEL

Das folgende Beispiel demonstriert eine einfache Steuerung für die Schreibmarke auf dem Bildschirm. Aus Platzgründen erfolgt weder eine Überprüfung der Funktionsergebnisse noch eine Signalbehandlung:

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

main()
{  int y=13, x=40, key;

   if (initscr()==NULL)
      exit(1);
   (void) raw(); (void) keypad(stdscr,TRUE);

   (void) clear(); (void) refresh();

   while ( (key=mvgetch(y, x)) != 'q' )
      switch (key)
      { case KEY_UP:    y--;           break;
        case KEY_DOWN:  y++;           break;
        case KEY_LEFT:  x--;           break;
        case KEY_RIGHT: x++;           break;
        case KEY_HOME:  x = y = 1;     break;
        default:        (void) beep(); break;
      }

   (void) endwin();
}
```

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

getstr, mvgetstr, mvwgetstr, wgetstr - read string
Zeichenkette einlesen

DEFINITION

```
#include <curses.h>

int getstr(str)
char *str;

int wgetstr(win, str)
WINDOW *win
char *str;

int mvgetstr(y, x, str)
int y, x;
char *str;

int mvwgetstr(win, y, x, str)
WINDOW *win;
int y, x;
char *str;
```

BESCHREIBUNG

Mit diesen Funktionen werden Zeichen bis zum Empfang eines Neue-Zeile-Zeichens oder eines Wagenrücklaufs eingelesen. Das Ergebnis wird in den Bereich gesetzt, auf den der char-Zeiger *str* zeigt. Die vom Benutzer gegebenenfalls eingegebenen ERASE und KILL Zeichen werden interpretiert.

Bei den Funktionen *mvgetstr()* und *mvwgetstr()* wird die Schreibmarke vor dem Einlesen an die Position (*x*,*y*) bewegt. Die Funktionen *getstr()* und *mvgetstr()* lesen die Zeichenkette im Standardfenster *stdscr* ein, bei den Funktionen *wgetstr()* und *mvwgetstr()* gibt das Argument *win* an, in welchem Fenster eingelesen werden soll.

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

getstr(), *mvgetstr()* und *mvwgetstr()* sind Makros.

Bei einigen Implementierungen von *curses* wird die Betätigung von Funktionstasten ähnlich wie bei *getch* umgesetzt. Für die ordungsgemäße Funktion in Anwendungsprogrammen kann jedoch keine Gewähr übernommen werden.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

getyx - get cursor position
Position der Schreibmarke ermitteln

DEFINITION

```
#include <curses.h>

int getyx(win,y,x)
WINDOW *win;
int y,x;
```

BESCHREIBUNG

Die beiden ganzzahligen Variablen *y* und *x* werden mit der aktuellen Position der Schreibmarke im Fenster belegt. Da diese Funktion als Makro realisiert ist, muß den Variablen kein Adreßoperator (&) vorangestellt werden.

ERGEBNIS

Für die Funktion *getyx()* ist kein Ergebnis definiert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

has_ic - determine whether insert/delete character available
Überprüfung auf Möglichkeit zum Einfügen und Löschen von Zeichen

DEFINITION

```
#include <curses.h>

bool has_ic()
```

BESCHREIBUNG

Diese Funktion liefert den Wert TRUE, wenn die Datensichtstation über Möglichkeiten zum Einfügen und Löschen von Zeichen verfügt.

ERGEBNIS

Diese Funktion liefert den Wert TRUE, wenn die Datensichtstation über Möglichkeiten zum Einfügen und Löschen von Zeichen verfügt. Andernfalls liefert sie den Wert FALSE.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Im Paket *curses* sind die Funktionen *insch*() und *delch*() enthalten und somit jederzeit verfügbar.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

has_il - determine whether insert/delete line is available
Überprüfung auf Möglichkeit zum Einfügen und Löschen von Zeilen

DEFINITION

```
#include <curses.h>

bool has_il()
```

BESCHREIBUNG

Diese Funktion liefert den Wert TRUE, wenn die Datensichtstation über Möglichkeiten zum Einfügen und Löschen von Zeilen verfügt bzw. diese Funktionen durch bereichsweises Blättern simulieren kann. Mit dieser Funktion kann z.B. überprüft werden, ob die Aktivierung eines physikalischen Blätterns mit *scrollok*() sinnvoll ist.

ERGEBNIS

Die Funktion liefert den Wert TRUE, wenn die Datensichtstation die überprüften Fähigkeiten besitzt; andernfalls liefert sie den Wert FALSE.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Im Paket *curses* sind die Funktionen *insertln*() und *deleteln*() enthalten und somit jederzeit verfügbar.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

idlok - enable use of insert/delete line
Hardwaremäßiges Einfügen/Löschen von Zeichen aktivieren

DEFINITION

```
#include <curses.h>

int idlok(win, bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Wird diese Funktion aktiviert (d.h. *bf* ist TRUE), so führt *curses* - eine geeignete Ausrüstung der Datensichtstation vorausgesetzt - ein hardwaremäßiges Löschen/Einfügen von Zeilen durch. Andernfalls werden diese Funktionen von *curses* nicht benutzt (die Einrichtung zum Löschen und Einfügen von Zeichen wird auf jeden Fall verwendet). Diese Funktion sollte nur benutzt werden, wenn das Anwendungsprogramm (z.B. ein bildschirmorientierter Editor) Zeilen löschen und einfügen muß. Das Löschen und Einfügen von Zeilen kann sich auf das Arbeiten am Bildschirm störend auswirken. Daher ist diese Funktion standardmäßig nicht aktiv; sie sollte nur eingeschaltet werden, wenn sie wirklich notwendig ist. Ist kein hardwaremäßiges Einfügen/Löschen von Zeilen möglich, so erstellt *curses* die geänderten Abschnitte aller Zeilen neu.

ERGEBNIS

Die Funktion liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

inch, mvinch, mvwinch, winch - return character from window
Zeichen aus Fenster liefern

DEFINITION

```
#include <curses.h>

chtype inch()

chtype winch(win)
WINDOW *win;

chtype mvinch(y, x)
int y,x;

chtype mvwinch(win, y, x)
WINDOW *win
int y,x;
```

BESCHREIBUNG

Die Funktion *inch()* liefert den Wert des Zeichens an der aktuellen Position der Schreibmarke im Standardfenster als einen Wert vom Typ *chtype* zurück. Sind für diese Zeichenposition Attribute gesetzt, so werden ihre Werte mit dem zurückgegebenen Wert durch bitweises ODER verknüpft. Die vordefinierten Konstanten A_CHARTEXT und A_ATTRIBUTES, deren Definition in der Include-Datei *<curses.h>* enthalten sind, können zusammen mit dem Bit-Operator & (UND) benutzt werden, um entweder nur das Zeichen oder nur die Merkmale zurückzugeben.

Bei den Funktionen *mvinch()* und *mvwinch()* wird die Schreibmarke vor der Ermittlung des Zeichens an die Position (x,y) bewegt. Die Funktionen *inch()* und *mvinch()* ermitteln das Zeichen im Standardfenster *stdscr*, bei den Funktionen *winch()* und *mvwinch()* gibt das Argument *win* an, welches Fenster gewählt werden soll.

ERGEBNIS

Bei erfolgreicher Beendigung liefern *inch()*, *mvinch()*, *mvwinch()* und *winch()* das Zeichen an der ausgewählten Position, einschließlich vorhandener Attribute. Andernfalls liefern die Funktionen *mvinch()* und *mvwinch()* den Wert ERR zurück.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

inch(), *winch()*, *mvinch()* und *mvwinch()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

initscr - initialise terminal environment
Umgebung der Datensichtstation initialisieren

DEFINITION

```
#include <curses.h>

WINDOW *initscr()
```

BESCHREIBUNG

Die Funktion *initscr()* sollte fast in jedem Fall die erste Funktion nach der Signalbehandlung sein, die in einem curses-Programm aufgerufen wird. Diese Funktion dient zur Bestimmung des Typs der Datensichtstation sowie zur Initialisierung der *curses*-Datenstrukturen. Außerdem sorgt *initscr*() dafür, daß beim ersten Aufruf von *refresh*() der Bildschirm gelöscht wird. Tritt ein Fehler auf, so schreibt *initscr*() eine geeignete Fehlermeldung auf die Standard-Fehlerausgabe und bricht das Programm ab. Sollen im Programm Informationen über die Fehlerursache bearbeitet werden, so sollte nicht *initscr*(), sondern *newterm*() benutzt werden.

ERGEBNIS

Die Funktion *initscr()* liefert den Wert *stdscr* bei Erfolg und ruft im Fehlerfall die Funktion *exit()* auf.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

BEISPIEL

Siehe das Beispiel in Abschnitt 2.7

SIEHE AUCH

newterm(), Abschnitt 2.7 .

NAME

insch, mvinsch, mvwinsch, winsch - insert character
Zeichen einfügen

DEFINITION

```
#include <curses.h>

int insch(ch)
chtype ch;

int winsch(win, ch)
WINDOW *win;
chtype ch;

int mvinsch(y, x, ch)
int y,x;
chtype ch;

int mvwinsch(win, y, x, ch)
WINDOW *win;
int y,x;
chtype ch;
```

BESCHREIBUNG

Die Funktion *insch()* fügt das Zeichen *ch* vor dem Zeichen ein, das sich unter der aktuellen Position der Schreibmarke im Standardfenster *stdscr* befindet. Alle Zeichen rechts des eingefügten Zeichens werden um eine Position nach rechts verschoben; dabei geht das äußerst rechte Zeichen gegebenenfalls verloren.

Bei den Funktionen *mvinsch()* und *mvwinsch()* wird die Schreibmarke vor der Ausgabe an die Position (x,y) bewegt. Die Funktionen *insch()* und *mvinsch()* geben das Zeichen in das Standardfenster *stdscr* aus, bei den Funktionen *winsch()* und *mvwinsch()* gibt das Argument *win* an, in welches Fenster ausgegeben werden soll.

An der Position der Schreibmarke ändert sich nichts (nachdem sie gegebenenfalls mit *move*() an die Position bewegt worden ist, die über die Koordinaten *y, x* angegeben wurde).

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

insch(), *mvinsch()* und *mvwinsch()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

insertln, winsertln - insert line
Zeile in Fenster einfügen

DEFINITION

```
#include <curses.h>

int insertln()

int winsertln(win)
WINDOW *win;
```

BESCHREIBUNG

Unterhalb der aktuellen Zeile wird eine Leerzeile eingefügt; die unterste Zeile auf dem Bildschirm geht verloren.

Die Funktion *insertln()* fügt die Leerzeile im Standardfenster *stdscr* ein; bei der Funktion *winsertln()* erfolgt das Einfügen in dem Fenster, das durch das Argument *win* angegeben wird.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

insertln() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

intrflush - enable flush on interrupt
Warteschlange bei Unterbrechungssignal löschen

DEFINITION

```
#include <curses.h>

int intrflush(win,bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Wird diese Funktion aktiviert (d.h. *bf* ist TRUE), dann bewirkt die Betätigung einer Unterbrechungs-Taste (INTR, BREAK, QUIT), daß alle Daten im Ausgabepuffer des Bildschirmtreibers gelöscht werden. Dies hat den Vorteil, daß bei einem Unterbrechungssignal eine schnellere Reaktion erfolgt; andererseits werden an *curses* Informationen über den Inhalt des Bildschirms geliefert, die nicht den Tatsachen entsprechen. Ist die Funktion *intrflush*() ausgeschaltet (durch einen Aufruf mit *bf* gleich FALSE), so wird die Warteschlange nicht gelöscht. Die Einstellung dieser Funktion entspricht standardmäßig der Einstellung des Bildschirmtreibers. Das Argument *win* wird ignoriert.

ERGEBNIS

Die Funktion *intrflush()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

keypad - enable keypad
Funktionstastenblock aktivieren

DEFINITION

```
#include <curses.h>

int keypad(win, bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Mit dieser Funktion kann der Funktionstastenblock auf der Datensichtstation des Benutzers aktiviert werden. Ist diese Funktion aktiviert (d.h. wurde *keypad()* mit dem Argument *bf* gleich TRUE aufgerufen), so wird bei Betätigung einer Funktionstaste durch den Benutzer (z.B. einer Pfeiltaste) von der Funktion *getch*() oder ihren verwandten Funktionen ein einzelner Wert zurückgegeben, der der Funktionstaste zugeordnet ist, wie z.B. KEY_LEFT (siehe Abschnitt *Funktionstasten* in der *curses*-Einführung in Abschnitt 2.7). Ist diese Funktion ausgeschaltet (*bf* gleich FALSE), so werden Funktionstasten von *curses* nicht besonders behandelt; das Programm muß gelieferte Escape-Sequenzen dann selbst interpretieren. Kann der Tastenblock der Datensichtstation ein- (Übertragungsbetrieb) und ausgeschaltet (Lokalbetrieb) werden, so wird bei einer Aktivierung dieser Funktion der Tastenblock der Datensichtstation vor dem Beginn der Eingabe ein-(d.h. auf Übertragungsbetrieb) geschaltet.

ERGEBNIS

Die Funktion liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

In Bezug auf die Anordnung der Tasten auf dem Funktionstastenblock kann es zwischen Datensichtstationen große Unterschiede geben; bei einigen Datensichtstationen ist eventuell sogar überhaupt kein derartiger Tastenblock vorhanden.

BEISPIEL

Siehe das Beispiel unter *curses: getch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

killchar - return current KILL character
KILL-Zeichen ermitteln

DEFINITION

```
#include <curses.h>

char killchar()
```

BESCHREIBUNG

Das aktuelle KILL-Zeichen des Benutzers wird ermittelt und zurückgegeben. Mit diesem Zeichen kann der Benutzer eine teilweise eingegebene Zeile vollständig verwerfen.

ERGEBNIS

Die Funktion *killchar()* liefert den Wert des aktuell vom Benutzer definierten KILL-Zeichens.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

leaveok - enable non-tracking cursor
Position der Schreibmarke unverändert lassen

DEFINITION

```
#include <curses.h>

int leaveok(win,bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Nach *wrefresh*() steht die (physikalische) Schreibmarke auf dem Bildschirm normalerweise an der aktuellen Schreibmarkenposition dieses Fensters, auch wenn sie zuletzt in einem anderen Fenster stand. Ruft man für das aktuelle Fenster *leaveok*() auf (mit *bf* gleich TRUE), so kann man diese Bewegung unterdrücken, sofern man seit dem letzten *wrefresh*() auf dieses Fenster keine Änderungen in diesem Fenster vorgenommen hat. Die Schreibmarke bleibt dann an der Position stehen, auf der sie in dem dem *wrefresh*-Aufruf vorhergehenden Fenster stand. Standardmäßig ist *leaveok*() nicht aktiv.

ERGEBNIS

Die Funktion *leaveok()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

longname - return full terminal type name
Vollständigen Namen der Datensichtstation ermitteln

DEFINITION

```
char *longname()
```

BESCHREIBUNG

Die Funktion *longname()* liefert einen Zeiger auf einen statischen Bereich, der eine ausführliche Beschreibung der aktuellen Datensichtstation enthält. Die Beschreibung besteht aus maximal 128 Zeichen. Erst nach einem Aufruf von *newterm()* oder *initscr()* ist diese Beschreibung definiert, daher darf die Funktion *longname()* erst nach einem Aufruf einer dieser Funktionen aufgerufen werden.

Der Bereich wird bei jedem Aufruf von *newterm*() überschrieben; bei einem Aufruf von *set_term*() wird sein alter Inhalt nicht wiederhergestellt. Daher sollte sein Inhalt zwischen verschiedenen Aufrufen von *newterm()* abgespeichert werden, wenn *longname*() für mehrere Datensichtstationen benutzt werden soll.

ERGEBNIS

Die Funktion *longname()* liefert einen Zeiger auf eine vollständige Beschreibung der aktuellen Datensichtstation oder den Nullzeiger im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

move, wmove - move cursor in window
Schreibmarke im Fenster verschieben

DEFINITION

```
#include <curses.h>

int move(y, x)
int y,x;

int wmove(win, y, x)
WINDOW *win;
int y,x;
```

BESCHREIBUNG

Mit der Funktion *move()* wird die Schreibmarke im Standardfenster an eine bestimmte Stelle bewegt. Die Position der physikalischen Schreibmarke der Datensichtstation ändert sich erst, wenn die Funktion *refresh*() aufgerufen wird. Die neue Position wird relativ zur linken oberen Ecke des Fensters (Koordinaten 0,0) angegeben.

Die Funktion *wmove()* bewegt die Schreibmarke in dem Fenster, das durch das Argument *win* angegeben wird.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

move() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

mvaddch - add character to window
Zeichen in Fenster ausgeben

DEFINITION

```
#include <curses.h>

int mvaddch(y, x, ch)
int y, x;
chtype ch;
```

BESCHREIBUNG

Die Funktion *mvaddch()* dient dazu, ein Zeichen in ein Fenster auszugeben. Eine ausführliche Beschreibung finden Sie unter *curses: addch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvaddstr - add string to window
Zeichenkette auf Fenster schreiben

DEFINITION

```
#include <curses.h>

int mvaddstr(y, x, str)
int y, x;
char *str;
```

BESCHREIBUNG

Diese Funktion gibt eine Zeichenkette in ein Fenster aus. Eine ausführliche Beschreibung finden Sie unter *curses: addstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvdelch - remove character from window
Zeichen im Fenster löschen

DEFINITION

```
#include <curses.h>

int mvdelch(y,x)
int y, x;
```

BESCHREIBUNG

Diese Funktion löscht ein Zeichen im Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: delch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvgetch - read character
Zeichen einlesen

DEFINITION

```
#include <curses.h>

int mvgetch(y,x)
int y,x;
```

BESCHREIBUNG

Diese Funktion liest ein Zeichen aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvgetstr - read string
Zeichenkette einlesen

DEFINITION

```
#include <curses.h>

int mvgetstr(y, x, str)
int y, x;
char *str;
```

BESCHREIBUNG

Diese Funktion liest eine Zeichenkette aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvinch - return character from window
Zeichen aus Fenster liefern

DEFINITION

```
#include <curses.h>

chtype mvinch(y, x)
int y,x;
```

BESCHREIBUNG

Die Funktion *inch()* liefert den Wert des Zeichens in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: inch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvinsch - insert character
Zeichen einfügen

DEFINITION

```
#include <curses.h>

int mvinsch(y, x, ch)
int y,x;
chtype ch;
```

BESCHREIBUNG

Diese Funktion fügt das Zeichen *ch* vor dem Zeichen an der Position (x,y) im Standardfenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: insch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvprintw - formatted write to a window
Formatierte Ausgabe

DEFINITION

```
#include <curses.h>

int mvprintw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Ausgabe in ein Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: printw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvscanw - formatted read from window
Formatiertes Lesen

DEFINITION

```
#include <curses.h>

int mvscanw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Eingabe in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: scanw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwaddch - add character to window
Zeichen in Fenster ausgeben

DEFINITION

```
#include <curses.h>

int mvwaddch(win, y, x, ch)
WINDOW *win;
int y, x;
chtype ch;
```

BESCHREIBUNG

Die Funktion *mvwaddch()* dient dazu, ein Zeichen in ein Fenster auszugeben. Eine ausführliche Beschreibung finden Sie unter *curses: addch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwaddstr - add string to window
Zeichenkette auf Fenster schreiben

DEFINITION

```
#include <curses.h>

int mvwaddstr(win, y, x, str)
WINDOW *win;
int x, y;
char *str;
```

BESCHREIBUNG

Diese Funktion gibt eine Zeichenkette in ein Fenster aus. Eine ausführliche Beschreibung finden Sie unter *curses: addstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwdelch - remove character from window
Zeichen im Fenster löschen

DEFINITION

```
#include <curses.h>

int mvwdelch(win, y, x)
WINDOW *win;
int y, x;
```

BESCHREIBUNG

Diese Funktion löscht ein Zeichen im Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: delch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwgetch - read character
Zeichen einlesen

DEFINITION

```
#include <curses.h>

int mvwgetch(win,y,x)
WINDOW *win;
int y,x;
```

BESCHREIBUNG

Diese Funktion liest ein Zeichen aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwgetstr - read string
Zeichenkette einlesen

DEFINITION

```
#include <curses.h>

int mvwgetstr(win, y, x, str)
WINDOW *win;
int y, x;
char *str;
```

BESCHREIBUNG

Diese Funktion liest eine Zeichenkette aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwin - move window
Fenster verschieben

DEFINITION

```
#include <curses.h>

int mvwin(win, y, x)
WINDOW *win;
int y, x;
```

BESCHREIBUNG

Mit dieser Funktion wird das Fenster so verschoben, daß seine linke obere Ecke sich an der Position (*y*, *x*) befindet. Geben die Koordinaten eine Position außerhalb des Bildschirmbereichs an, so wird dieser Aufruf ignoriert, d.h., das Fenster wird nicht verschoben.

ERGEBNIS

Die Funktion *mvwin()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

mvwinch - return character from window
Zeichen aus Fenster liefern

DEFINITION

```
#include <curses.h>

chtype mvwinch(win, y, x)
WINDOW *win
int y,x;
```

BESCHREIBUNG

Diese Funktion liefert den Wert des Zeichens in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: inch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwinsch - insert character
Zeichen einfügen

DEFINITION

```
#include <curses.h>

int mvwinsch(win, y, x, ch)
WINDOW *win;
int y,x;
chtype ch;
```

BESCHREIBUNG

Diese Funktion fügt das Zeichen *ch* vor dem Zeichen an der Position (x,y) im Fenster *win* ein. Eine ausführliche Beschreibung finden Sie unter *curses: insch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwprintw - formatted write to a window
Formatierte Ausgabe

DEFINITION

```
#include <curses.h>

int mvwprintw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Ausgabe in ein Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: printw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

mvwscanw - formatted read from a window
Formatiertes Lesen

DEFINITION

```
#include <curses.h>

int mvwscanw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Eingabe in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: scanw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

newpad - create new pad
neuen *Pad* einrichten

DEFINITION

```
#include <curses.h>

WINDOW *newpad(nlines, ncols)
int nlines, ncols;
```

BESCHREIBUNG

Mit dieser Funktion wird für einen neuen *Pad* eine Datenstruktur erzeugt. Ein *Pad* ist im Prinzip ein Fenster, nur ist er weder auf den Bildschirmbereich beschränkt, noch muß er unbedingt einem bestimmten Teil des Bildschirms zugeordnet sein. Ein *Pad* wird immer dann eingerichtet, wenn ein großes Fenster benötigt wird, das jedoch nie vollständig auf dem Bildschirm sichtbar sein muß. Bei *Pad* erfolgt keine automatische Aktualisierung (z.B. beim Blättern oder wenn die Funktion *echo*() aktiv ist). Das Aufrufen von *refresh*() mit einem *Pad*-Argument ist nicht zulässig; hierfür muß eine der Funktionen *prefresh*() oder *pnoutrefresh*() benutzt werden. Zu beachten ist, daß bei diesen Funktionen zusätzliche Parameter erforderlich sind, mit denen der darzustellende Teil des *Pad* und der für die Zeichendarstellung zu benutzende Teil des Bildschirms angegeben wird.

ERGEBNIS

Bei Erfolg liefert die Funktion *newpad()* einen Zeiger auf die neue WINDOW-Struktur für den Pad. Im Fehlerfall liefert sie den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: prefresh(), *curses: pnoutfresh()*, Abschnitt 2.7 .

NAME

newterm - open new terminal
neue Datensichtstation eröffnen

DEFINITION

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

SCREEN *newterm(type, outfd, infd)
char *type;
FILE *outfd, *infd;
```

BESCHREIBUNG

In einem Programm, das an zwei oder mehr Datensichtstationen Daten ausgibt, sollte für die einzelnen Datensichtstationen *newterm*() und nicht *initscr*() angegeben werden. Die Funktion *newterm*() sollte für jede Datensichtstation einmal aufgerufen werden. Als Ergebnis liefert sie einen Zeiger auf eine Struktur SCREEN, der als Verweis auf diese Datensichtstation gespeichert werden sollte. Als Argumente werden der Datensichtstationstyp (der anstelle der Umgebungsvariablen TERM verwendet werden soll), ein Dateizeiger für die Ausgabe auf die Station und ein weiterer für Eingaben von der Station angegeben. Das Programm muß auch vor seiner Beendigung für jede benutzte Datensichtstation die Funktion *endwin*() aufrufen.

ERGEBNIS

Bei Erfolg liefert die Funktion *newterm()* einen Zeiger auf die neue Struktur vom Typ SCREEN. Im Fehlerfall liefert sie den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Diese Funktion zeigt Fehlerbedingungen an. Daher ist ist besonders für solche Programme nützlich, die an Datensichtstationen ablaufen, die keine bildschirmorientierte Betriebsart erlauben. Solche Programme benötigen die Anzeige von Fehlerbedingungen, um in der zeilenorientierten Betriebsart zu bleiben.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: initscr(), Abschnitt 2.7 .

NAME

newwin - create new window
neues Fenster einrichten

DEFINITION

```
#include <curses.h>

WINDOW *newwin(nlines, ncols, begin_y, begin_x)
int nlines, ncols, begin_y, begin_x;
```

BESCHREIBUNG

Mit dieser Funktion kann eine neues Fenster mit *nlines* Zeilen und *ncols* Spalten eingerichtet werden. Mit *begin_y* und *begin_x* wird angegeben, in welche Zeile und Spalte die linke obere Ecke dieses Fensters gesetzt werden soll. Wird entweder für *nlines* oder für *ncols* der Wert 0 angegeben, so wird der entsprechende Parameter auf LINES - *begin_y* bzw. COLS - *begin_x* gesetzt. Ein Fenster, das den gesamten Bildschirm ausfüllt, kann mit der Funktion *newwin(0,0,0,0)* eingerichtet werden.

ERGEBNIS

Bei Erfolg liefert die Funktion *newwin()* einen Zeiger auf die neue Struktur WINDOW. Im Fehlerfall liefert sie den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um in einem Fenster ein Unterfenster zu erzeugen, steht die Funktion *subwin()* zur Verfügung.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: subwin(), Abschnitt 2.7 .

NAME

nl, nonl - enable/disable newline control
Umsetzung des Neue-Zeile-Zeichens (NL) steuern

DEFINITION

```
#include <curses.h>
int nl()
int nonl()
```

BESCHREIBUNG

Mit diesen Funktionen kann angegeben werden, ob ein Neue-Zeile-Zeichen bei der Ausgabe in einen Wagenrücklauf und einen Zeilenvorschub oder nur in einen Zeilenvorschub umgesetzt werden soll. Standardmäßig findet erstere Umsetzung statt. Durch das Ausschalten dieser Umsetzung sind schnellere Bewegungen der Schreibmarke möglich, da *curses* die Einrichtung zur Durchführung des Zeilenvorschubs besser nutzen kann.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

nl() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

nocbreak - disable creak mode
CBREAK-Modus ausschalten

DEFINITION

```
int nocbreak()
```

BESCHREIBUNG

Mit dieser Funktion kann auf einer Datensichtstation der CBREAK-Modus ausgeschaltet werden. Eine ausführliche Beschreibung finden Sie unter *curses: cbreak()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

nodelay - disable block during read
Blockierung beim Einlesen von Zeichen ausschalten

DEFINITION

```
#include <curses.h>

int nodelay(win,bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Diese Funktion steuert die Blockierung beim Einlesen von Zeichen. Ist sie aktiviert, d.h. wurde die Funktion *nodelay()* mit dem Argument *bf* gleich TRUE aufgerufen, dann blockiert *getch*() auch dann nicht, wenn keine Eingabedaten zur Verfügung stehen; *getch*() gibt dann den Wert ERR zurück. Ist *nodelay()* nicht aktiv (*bf* gleich FALSE oder Standard), so blockiert *getch*() solange, bis Eingabedaten zur Verfügung steht.

ERGEBNIS

Die Funktion *nodelay()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

noecho - disable terminal echo
Ausgabe unterdrücken

DEFINITION

```
int noecho()
```

BESCHREIBUNG

Diese Funktion unterdrückt das Echo am Bildschirm. Eine ausführliche Beschreibung finden Sie unter *curses: echo()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

nonl - disable newline control
Umsetzung des Neue-Zeile-Zeichens ausschalten

DEFINITION

```
#include <curses.h>

int nonl()
```

BESCHREIBUNG

Mit dieser Funktion wird die Umsetzung eines Neue-Zeile-Zeichens bei der Ausgabe in einen Wagenrücklauf und einen Zeilenvorschub ausgeschaltet. Eine ausführliche Beschreibung finden Sie unter *curses: nl()*. *nl()* ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

noraw - disable raw mode
Raw-Modus ausschalten

DEFINITION

```
int noraw()
```

BESCHREIBUNG

Mit der Funktion *noraw()* wird für die aktuelle Datensichtstation der Raw-Modus ausgeschaltet. Eine ausführliche Beschreibung finden Sie unter *curses: raw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

overlay, overwrite - overlay windows
Fenster überlagern

DEFINITION

```
#include <curses.h>

int overlay(srcwin, dstwin)
WINDOW *srcwin, *dstwin;

int overwrite(srcwin, dstwin)
WINDOW *srcwin, *dstwin;
```

BESCHREIBUNG

Mit diesen Funktionen wird *dstwin* durch *srcwin* überlagert; das heißt, der in *scrwin* enthaltene Text wird nach *dstwin* kopiert. *srcwin* und *dstwin* müssen nicht unbedingt dieselbe Größe haben. Mit dem Kopieren wird in jedem Fenster bei der Koordinate (0,0) begonnen. Der Unterschied zwischen den beiden Funktionen ist, daß bei *overlay*() der Inhalt des Zielfensters erhalten bleibt (es werden keine Leerzeichen kopiert), während er bei *overwrite*() überschrieben wird.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

prefresh, pnoutrefresh - refresh pad
Pad aktualisieren

DEFINITION

```
#include <curses.h>

int prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol);

WINDOW *pad;
int pminrow, pmincol, sminrow,

int pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol);

WINDOW *pad;
int pminrow, pmincol, sminrow,
    smincol, smaxrow, smaxcol;
```

BESCHREIBUNG

Diese Funktionen sind funktionsgleich mit *wrefresh*() und *wnoutrefresh*(), nur werden keine Fenster, sondern *Pads* behandelt. Die zusätzlichen Parameter werden für die Angabe der zu behandelnden Ausschnitte aus dem *Pad* bzw. Bildschirm benötigt. Mit *pminrow* und *pmincol* wird die obere linke Ecke des *Pad* angegeben, der auf dem Bildschirm ausgegeben werden soll; mit *sminrow*, *smincol*, *smaxrow* und *smaxcol* werden die Seiten des Bildschirmausschnitts angegeben, über den der Ausschnitt des *Pads* gelegt werden soll. Die rechte untere Ecke des auszugebenden *Pad*ausschnitts wird aus den Bildschirmkoordinaten errechnet, da beide Ausschnitte gleich groß sein müssen. Beide Ausschnitte müssen in der jeweils entsprechenden Struktur vollständig enthalten sein.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: newpad(), Abschnitt 2.7 .

NAME

printw, mvprintw, mvwprintw, wprintw - formatted write to a window
Formatierte Ausgabe

DEFINITION

```
#include <curses.h>

int printw(fmt [, arg] ...)
char *fmt;

int wprintw(win, fmt [, arg] ...)
WINDOW *win
char *fmt;

int mvprintw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;

int mvwprintw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktionen sind funktionsgleich mit *printf*; nur werden die Zeichen hier mit *waddstr*() in das Fenster geschrieben.

Bei den Funktionen *mvprintw()* und *mvwprintw()* wird die Schreibmarke vor der Ausgabe an die Position (*x*,*y*) bewegt. Die Funktionen *printw()* und *mvprintw()* geben das Zeichen in das Standardfenster *stdscr* aus, bei den Funktionen *wprintw()* und *mvwprintw()* gibt das Argument *win* an, in welches Fenster ausgegeben werden soll.

ERGEBNIS

Alle Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

printf(), Abschnitt 2.7 .

NAME

raw, noraw - enable/disable raw mode
Raw-Modus ein/ausschalten

DEFINITION

```
int raw()
int noraw()
```

BESCHREIBUNG

Mit der Funktion *raw()* wird für die betreffende Datensichtstation der Raw-Modus ein- bzw. mit der Funktion *noraw()* wieder ausgeschaltet.

Die Betriebsart Raw-Modus ähnelt insofern dem CBREAK-Modus, als Eingaben von der Datensichtstation dem Benutzerprogamm sofort übergeben werden. Im Gegensatz zu CBREAK werden im Raw-Modus die INTR-, QUIT- und STOP-Zeichen nicht interpretiert, d.h. sie werden übergeben, ohne ein Signal auszulösen. Die Wirkung der BREAK-Taste hängt von anderen, nicht von *curses* gesetzten Parametern der Terminalschnittstelle ab (siehe auch Abschnitt 2.4, *Allgemeine Terminalschnittstelle*).

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.4, *Allgemeine Terminalschnittstelle*

NAME

refresh, wrefresh - refresh window
Fenster aktualisieren

DEFINITION

```
#include <curses.h>

int refresh()

int wrefresh(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktionen müssen immer dann aufgerufen werden, wenn irgendwelche Ausgaben tatsächlich auf eine Datensichtstation geschrieben werden sollen, da die übrigen Funktionen nur mit Datenstrukturen arbeiten. Mit *wrefresh*() wird das angegebene Fenster auf den physikalischen Bildschirm übertragen, wobei Vorhandenes aus Optimierungsgründen berücksichtigt wird. *refresh*() hat dieselbe Funktion, nur wird *stdscr* als Standardfenster benutzt. Die Schreibmarke der physikalischen Datensichtstation bleibt an der Position der Fenster-Schreibmarke, es sei denn, *leaveok*() ist zuvor aktiviert worden.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

refresh() ist ein Makro.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

reset_prog_mode, reset_shell_mode - restore terminal mode
Betriebsart der Datensichtstation zurücksetzen

DEFINITION

```
int reset_prog_mode()

int reset_shell_mode()
```

BESCHREIBUNG

Mit diesen Funktionen wird die Datensichtstation wieder in den "Programm"-Zustand (in *curses*) oder den "Shell"-Zustand (nicht in *curses*) zurückgesetzt. Dies erfolgt automatisch durch *endwin*() und *doupdate*(); die *reset*-Funktionen werden also normalerweise nicht vor diesen Funktionen benutzt.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: def_prog_mode(), def_shell_mode().

NAME

resetty - restore terminal modes
Betriebsart der Datensichtstation

DEFINITION

```
int resetty()
```

BESCHREIBUNG

Mit diesen Funktionen werden die Betriebsarten der Datensichtstation abgespeichert bzw. zurückgesetzt. Mit *savetty*() wird der aktuelle Status in einem Puffer abgespeichert; mit *resetty*() wird der Status wiederhergestellt, der beim letzten Aufruf von *savetty*() gültig war.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

savetty - save terminal modes
Betriebsart der Datensichtstation abspeichern

DEFINITION

```
int savetty()
```

BESCHREIBUNG

Mit dieser Funktion wird die Betriebsarten der Datensichtstation abgespeichert. Eine ausführliche Beschreibung finden Sie unter *curses: resetty()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

scanw, mvscanw, mvwscanw, wscanw - formatted read from window
Formatiertes Lesen

DEFINITION

```
#include <curses.h>

int scanw(fmt [, arg] ...)
char *fmt;

int wscanw(win, fmt [, arg] ...)
WINDOW *win;
char *fmt;

int mvscanw(y, x, fmt [, arg] ...)
int y, x;
char *fmt;

int mvwscanw(win, y, x, fmt [, arg] ...)
WINDOW *win;
int y, x;
char *fmt;
```

BESCHREIBUNG

Diese Funktionen arbeiten genauso wie *scanf()*. Zunächst wird *wgetstr*() für das Fenster aufgerufen; das Ergebnis (eine Zeile) dient als Eingabe für die Leseoperation.

Bei den Funktionen *mvscanw()* und *mvwscanw()* wird die Schreibmarke vor dem Einlesen an die Position (*x*,*y*) bewegt. Die Funktionen *scanw()* und *mvscanw()* lesen im Standardfenster *stdscr* ein, bei den Funktionen *wscanw()* und *mvwscanw()* gibt das Argument *win* an, in welchem Fenster eingelesen werden soll.

ERGEBNIS

Bei erfolgreicher Beendigung liefern alle Funktionen die Anzahl der korrekt eingelesenen und umgewandelten Formatelemente. Bei Dateiende liefern die Funktionen den Wert EOF; ansonsten wird der Wert ERR zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

scanf(), Abschnitt 2.7 .

NAME

scroll - scroll window
Bildlauf durchführen

DEFINITION

```
#include <curses.h>

int scroll(win)
WINDOW *win;
```

BESCHREIBUNG

Mit dieser Funktion wird im Fenster um eine Zeile weitergeblättert. Dadurch werden die Zeilen in der Datenstruktur des Fensters verschoben.

ERGEBNIS

Die Funktion *scroll()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

scrollok - enable screen scrolling
"Blättern" einschalten

DEFINITION

```
#include <curses.h>

int scrollok(win, bf)
WINDOW *win;
bool bf;
```

BESCHREIBUNG

Von der Einstellung durch diese Funktion hängt es ab, was geschieht, wenn die Schreibmarke eines Fensters über den Rand eines Fensters oder eines Bildlaufbereichs hinaus bewegt wird, sei es, weil in der untersten Zeile ein Neue Zeile-Zeichen eingegeben wurde, oder weil in der letzten Zeile alle Zeichenpositionen besetzt sind. Ist diese Funktion aktiv (d.h. wurde scrollok() mit dem Argument *bf* gleich TRUE aufgerufen), so bleibt die Schreibmarke in der untersten Zeile. Ist diese Funktion nicht aktiv, d.h. Aufruf mit *bf* gleich FALSE, dann wird die Anzeige im Fenster um eine Zeile nach oben verschoben und *refresh*() aufgerufen.

ERGEBNIS

Die Funktion *scrollok()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

set_term - switch between terminals
Umschalten zwischen Datensichtstationen

DEFINITION

```
#include <curses.h>

SCREEN *set_term(new)
SCREEN *new;
```

BESCHREIBUNG

Mit diesen Funktionen kann zwischen verschiedenen Datensichtstationen umgeschaltet werden. Mit dem Argument *new* wird die neue Datensichtstation angegeben. Der Wert von *new* wird durch einen vorangegangenen Aufruf der Funktion *newterm() geliefert. set_term()* ist die einzige Funktion, die mit SCREEN-Zeigern arbeitet; alle anderen Funktionen beziehen sich nur auf die aktuelle Datensichtstation.

ERGEBNIS

Die Funktion *set_term()* liefert einen Zeiger auf die SCREEN-Struktur der bisher gültigen Datensichtstation zurück. Im Fehlerfall liefert Sie den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: newterm(), Abschnitt 2.7 .

NAME

setscrreg, wsetscrreg - set scrolling region
Bildlaufbereich einstellen

DEFINITION

```
#include <curses.h>

int setscrreg(top, bot)
int top, bot;

int wsetscrreg(win, top, bot)
WINDOW *win;
int top, bot;
```

BESCHREIBUNG

Mit diesen Funktionen kann der Benutzer angeben, welche Zeilen eines Fensters softwaremäßig weitergeblättert werden sollen. Mit *top* und *bot* wird die oberste und unterste Zeile des Bildlaufbereichs angegeben (die oberste Zeile im Fenster hat die Nummer 0). Ist sowohl diese Funktion als auch *scrollok*() aktiv, so werden beim Versuch, die Schreibmarke über die unterste Zeile des Bildlaufbereichs hinaus zu bewegen, alle Zeilen innerhalb des Bildlaufbereich um eine Zeile nach oben verschoben. Dabei wird innerhalb des Fensters nur der Text verschoben; an der Position der Schreibmarke ändert sich nichts.

Die Funktion *setscrreg()* setzt den Bildlaufbereich für das Standardfenster *stdscr*; bei der Funktion *wsetscrreg()* gibt das Argument *win* an, für welches Fenster der Bildlaufbereich gesetzt werden soll.

ERGEBNIS

Für diese Funktionen ist kein bestimmtes Ergebnis vorgesehen.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: scrollok(), Abschnitt 2.7 .

NAME

standend - Hervorheben ausschalten
switch off highlight mode

DEFINITION

```
#include <curses.h>

int standend()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie schaltet den Hervorhebe-Modus aus. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

standout - switch on highlight mode
Hervorheben einschalten

DEFINITION

```
#include <curses.h>

int standout()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie schaltet den Hervorhebe-Modus ein. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

subwin - creat subwindow
Fenster eröffnen

DEFINITION

```
#include <curses.h>

WINDOW *subwin(orig, nlines, ncols, begin_y, begin_x)
WINDOW *orig;
int nlines, ncols, begin_y, begin_x;
```

BESCHREIBUNG

Mit dieser Funktion wird ein neues Fenster eröffnet. Mit *nlines* wird die Zahl der Zeilen, mit *ncols* die Zahl der Spalten in diesem neuen Fenster angegeben. Die obere linke Ecke des Fensters wird mit *begin_y* und *begin_x* angegeben; diese Angaben werden relativ zum gesamten Bildschirm, nicht zum Fenster *orig* gemacht). Das Fenster wird innerhalb des Fensters *orig* eröffnet, so daß Änderungen an einem Fensters sich auf beide Fenster auswirken. Wurde mit dieser Funktion ein neues Fenster eröffnet, so ist vor dem Aufrufen der Funktion *wrefresh*() in vielen Fällen die Funktion *touchwin*() erforderlich.

ERGEBNIS

Bei Erfolg liefert die Funktion *subwin()* einen Zeiger auf die neue WINDOW-Struktur zurück. Im Fehlerfall liefert die den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Bevor ein Fenster, für das Unterfenster eröffnet wurden sind, mit der Funktion *delwin()* geschlossen wird, sollten zuerst alle Unterfenster geschlossen werden.

Ein neues, von anderen Fenstern unabhängiges Fenster kann mit der Funktion *newwin()* erzeugt werden.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

curses: delwin(), *curses: newwin()*, Abschnitt 2.7 .

NAME

touchwin - touch window
Fensteränderungs-Information löschen

DEFINITION

```
#include <curses.h>

int touchwin(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktion bewirkt, daß alle Informationen darüber, welche Teile des Fensters verändert wurden, verworfen werden; es wird davon ausgegangen, daß die Änderungen sich auf das ganze Fenster beziehen. Dies ist in manchen Fällen bei sich überlappenden Fenstern notwendig, da Änderungen im einen Fenster auch für das andere Fenster gelten, diese Änderungen jedoch in den Informations-Sätzen, die den geänderten Zeilen im anderen Fenster entsprechen, nicht wiedergegeben werden.

ERGEBNIS

Die Funktion *touchwin()* liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

typeahead - check for type ahead
Puffer prüfen

DEFINITION

```
int typeahead(datkz)
int datkz;
```

BESCHREIBUNG

curses sorgt für einen optimierten Bildschirmaufbau, indem es in regelmäßigen Abständen im Rahmen der Bildschirmaktualisierungen den Pufferinhalt überprüft. Sind im Puffer Daten enthalten, so wird die fällige Aktualisierung verschoben, bis erneut *wrefresh*() oder *doupdate*() aufgerufen wird. Dies beschleunigt die Verarbeitung von im voraus eingegebenen Kommandos. Normalerweise wird für die Pufferüberprüfung der an *newterm*() übergebene Dateizeiger für die Eingabedatei oder - für *initscr*() - der Dateizeiger *stdin* verwendet. Durch den Aufruf von *typeahead*() wird festgelegt, daß stattdessen die Dateikennzahl (*datkz*) verwendet werden soll. Ist *datkz* gleich –1, so wird keine Überprüfung des Puffers durchgeführt.

ERGEBNIS

Für die Funktion *typeahead()* ist kein bestimmtes Ergebnis reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

unctrl - convert character to printable form
Zeichen in darstellbares Format bringen

DEFINITION

```
#include <curses.h>

char * unctrl(c)
chtype c;
```

BESCHREIBUNG

Mit diesem Makro kann das Zeichen *c* in das entsprechende darstellbare Zeichen umgewandelt. Steuerzeichen werden im Format ^X ausgegeben. Darstellbare Zeichen werden unverändert ausgegeben.

ERGEBNIS

Die Funktion *unctrl()* liefert einen Zeiger auf die umgewandelte Darstellung des Zeichens als Zeichenkette.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

unctrl() ist ein Makro, das in der Datei <*unctrl.h*> definiert ist.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

waddch - add character to window
Zeichen in Fenster ausgeben

DEFINITION

```
#include <curses.h>

int waddch(win,ch)
WINDOW *win;
chtype ch;
```

BESCHREIBUNG

Die Funktion *waddch()* dient dazu, ein Zeichen in ein Fenster auszugeben. Eine ausführliche Beschreibung finden Sie unter *curses: addch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

waddstr - add string to window
Zeichenkette auf Fenster schreiben

DEFINITION

```
#include <curses.h>

int waddstr(win, str)
WINDOW *win;
char *str;
```

BESCHREIBUNG

Diese Funktion gibt eine Zeichenkette in ein Fenster aus. Eine ausführliche Beschreibung finden Sie unter *curses: addstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wattroff - attribute manipulation
Fenster-Attribute ausschalten

DEFINITION

```
#include <curses.h>

int wattroff(win, attrs)
WINDOW *win;
int attrs;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie schaltet die im Argument *attrs* angegebenen Attribute aus. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wattron - attribute manipulation
Fenster-Attribute einschalten

DEFINITION

```
#include <curses.h>

int wattron(win, attrs)
WINDOW *win;
int attrs;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie schaltet die im Argument *attrs* angegebenen Attribute ein. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*. *attroff()*, *attron()* und *attrset()* sind Makros.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wattrset - attribute manipulation
Fenster-Attribute setzen

DEFINITION

```
#include <curses.h>

int wattrset(win, attrs)
WINDOW *win;
int attrs;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie setzt die im Argument *attrs* angegebenen Attribute. Eine ausführliche Beschreibung der Funktion finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wclear - clear window
Fenster löschen

DEFINITION

```
#include <curses.h>

int wclear(win)
WINDOW *win
```

BESCHREIBUNG

Die Funktion *wclear()* löscht das durch das Argument *win* angegebene Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: clear()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wclrtobot - clear to end of screen
Löschen bis zum Ende des Bildschirms

DEFINITION

```
#include <curses.h>

int wclrtobot(win)
WINDOW *win;
```

BESCHREIBUNG

Mit der Funktion *wclrtobot* werden alle Zeilen im Standardfenster unterhalb der aktuellen Position der Schreibmarke im Fenster win gelöscht. Eine ausführliche Beschreibung dieser Funktion finden Sie unter *curses: clrtobot()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wclrtoeol - clear to end of line
Löschen bis zum Zeilenende

DEFINITION

```
#include <curses.h>

int wclrtoeol(win)
WINDOW *win
```

BESCHREIBUNG

Die Funktion *wclrtoeol()* löscht Zeichen zur Rechten und an der aktuellen Position der Schreibmarke im Fenster ***win***. Eine ausführliche Beschreibung dieser Funktion finden Sie unter *curses: clrtoeol()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wdelch - remove character from window
Zeichen im Fenster löschen

DEFINITION

```
#include <curses.h>

int wdelch(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktion löscht ein Zeichen im Fenster *win*. Eine ausführliche Beschreibung finden Sie unter *curses: delch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wdeleteln - remove line from window
Zeile im Fenster löschen

DEFINITION

```
#include <curses.h>

int wdeleteln(win)
WINDOW *win;
```

BESCHREIBUNG

Die Funktion *wdeleteln()* löscht die Zeile im Fenster *win*. Eine ausführliche Beschreibung finden Sie unter *curses: deleteln()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

werase - copy blanks into window
Fenster mit Leerzeichen füllen

DEFINITION

```
#include <curses.h>

int werase(win)
WINDOW *win;
```

BESCHREIBUNG

Die Funktion *werase()* setzt an jede Zeichenposition im Fenster *win* ein Leerzeichen, d.h. das Fenster wird gelöscht.

ERGEBNIS

Die Funktion liefert den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

wgetch - read character
Zeichen einlesen

DEFINITION

```
#include <curses.h>

int wgetch(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktion liest ein Zeichen aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wgetstr - read string
Zeichenkette einlesen

DEFINITION

```
#include <curses.h>

int wgetstr(win, str)
WINDOW *win
char *str;
```

BESCHREIBUNG

Diese Funktion liest eine Zeichenkette aus einem Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: getstr()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

winch - return character from window
Zeichen aus Fenster liefern

DEFINITION

```
#include <curses.h>

chtype winch(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktion liefert den Wert des Zeichens in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: inch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

winsch - insert character
Zeichen einfügen

DEFINITION

```
#include <curses.h>

int winsch(win, ch)
WINDOW *win;
chtype ch;
```

BESCHREIBUNG

Diese Funktion fügt das Zeichen *ch* vor dem aktuellen Zeichen im Fenster ein. Eine ausführliche Beschreibung finden Sie unter *curses: insch()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

winsertln - insert line
Zeile in Fenster einfügen

DEFINITION

```
#include <curses.h>

int winsertln(win)
WINDOW *win;
```

BESCHREIBUNG

Unterhalb der aktuellen Zeile wird eine Leerzeile im Fenster *win* eingefügt. Eine ausführliche Beschreibung finden Sie unter *curses: insertln()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wmove - move cursor in window
Schreibmarke im Fenster verschieben

DEFINITION

```
#include <curses.h>

int wmove(win, y, x)
WINDOW *win;
int y,x;
```

BESCHREIBUNG

Mit der Funktion *wmove()* wird die Schreibmarke im Fenster *win* an eine bestimmte Stelle bewegt. Eine ausführliche Beschreibung finden Sie unter *curses: move()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

doupdate, wnoutrefresh - do efficient refresh
Effiziente Aktualisierung

DEFINITION

```
#include <curses.h>

int wnoutrefresh()
WINDOW *win;

int doupdate()
```

BESCHREIBUNG

Mit diesen beiden Funktionen können mehrfache Aktualisierungen effizienter durchgeführt werden als nur mit *wrefresh*(). Zusätzlich zu den Fenster-Datenstrukturen bietet *curses* zwei Datenstrukturen, die den Bildschirm der Datensichtstation darstellen: die *physical*-Struktur, die den tatsächlichen Bildschirminhalt beschreibt, und die *virtual* Struktur, die den programmierten Bildschirminhalt beschreibt. Die *wrefresh*()-Funktion ruft zunächst *wnoutrefresh*() auf, wodurch das angegebene Fenster auf den virtuellen Bildschirm übertragen wird, und dann *doupdate*(); diese Funktion vergleicht den virtuellen Bildschirm mit dem physischen und führt die tatsächliche Änderung durch. Sollen mehrere Fenster gleichzeitig ausgegeben werden, so ist eine Folge von *wrefresh*()-Aufrufen anzugeben. Jeder *wrefresh*()-Aufruf wird in je einen *wnoutrefresh*()-Aufruf und einen *doupdate*()-Aufruf umgesetzt; diese bewirken eine Serie von Ausgaben auf dem Bildschirm. Gibt man allerdings als erstes je einen *wnoutrefresh*() pro Fenster an, so braucht *doupdate*() nur ein Mal aufgerufen zu werden. In diesem Fall erfolgt nur eine einzige Ausgabe, d.h. es werden wahrscheinlich insgesamt weniger Zeichen übertragen und mit Sicherheit weniger Rechenzeit verbraucht.

ERGEBNIS

Beide Funktionen liefern den Wert OK bei Erfolg und ERR im Fehlerfall.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

SIEHE AUCH

Abschnitt 2.7 .

NAME

wprintw - formatted write to a window
Formatierte Ausgabe

DEFINITION

```
#include <curses.h>

int wprintw(win, fmt [, arg] ...)
WINDOW *win
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Ausgabe in ein Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: printw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wrefresh - refresh window
Fenster aktualisieren

DEFINITION

```
#include <curses.h>

int wrefresh(win)
WINDOW *win;
```

BESCHREIBUNG

Diese Funktion dient der Aktualisierung eines Fensters. Eine ausführliche Beschreibung finden Sie unter *curses: refresh()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wscanw - formatted read from window
Formatiertes Lesen

DEFINITION

```
#include <curses.h>

int wscanw(win, fmt [, arg] ...)
WINDOW *win;
char *fmt;
```

BESCHREIBUNG

Diese Funktion erlaubt die formatierte Eingabe in einem Fenster. Eine ausführliche Beschreibung finden Sie unter *curses: scanw()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wsetscrreg - set scrolling region
Bildlaufbereich einstellen

DEFINITION

```
#include <curses.h>

int wsetscrreg(win, top, bot)
WINDOW *win;
int top, bot;
```

BESCHREIBUNG

Mit dieser Funktion kann der Benutzer angeben, welche Zeilen des Fensters *win* softwaremäßig weitergeblättert werden sollen. Eine ausführliche Beschreibung finden Sie unter *curses: setscrreg()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

wstandend, wstandout - switch highlight mode on/off
Fenster-Attribut Hervorheben behandeln

DEFINITION

```
#include <curses.h>

int wstandend(win)
WINDOW *win;

int wstandout(win)
WINDOW *win
```

BESCHREIBUNG

Diese Funktionen gehören zu einer Gruppe von *curses*-Funktionen, die Fensterattribute manipulieren. Sie schaltet den Hervorhebe-Modus für das Fenster *win* ein (*wstandout()*), bzw. aus (*wstandend()*). Eine ausführliche Beschreibung der Funktionen finden Sie unter *curses: attroff()*.

PORTABILITÄT

Alle curses-Funktionen sind im X/Open-Standard (Ausgabe 3) als optional definiert. Dies bedeutet, daß diese Funktionen nicht von jeder X/Open-kompatiblen Implementierung unterstützt werden müssen.

NAME

cuserid - character login name of the user
Zeichenkette mit Benutzernamen erzeugen

DEFINITION

```
#include <stdio.h>

char *cuserid (s)
char *s;
```

BESCHREIBUNG

Die Funktion *cuserid()* erzeugt eine Zeichenkettendarstellung des Namens, der der effektiven Benutzernummer des Prozesses zugeordnet ist.

Wenn *s* der Nullzeuger ist, dann wird diese Darstellung in einem Bereich erzeugt, der statisch sein kann (und daher durch nachfolgende Aufrufe von *cuserid()* überschrieben werden kann). Dessen Adresse wird zurückgeliefert. Wenn *s* nicht der Nullzeiger ist, dann wird vorausgesetzt, daß *s* auf einen Vektor von mindestens {L_cuserid} Bytes zeigt; die Darstellung wird in diesem Vektor abgelegt. Die symbolische Konstante {L_cuserid} ist in *<stdio.h>* definiert und besitzt einen Wert größer als 0;

ERGEBNIS

Wenn *s* nicht der Nullzeiger ist, dann wird *s* zurückgeliefert. Wenn *s* nicht der Nullzeiger ist, die Benutzerkennung jedoch nicht gefunden werden kann, dann wird das Nullbyte '\0' unter **s* abgelegt. Wenn *s* der Nullzeiger ist und die Benutzerkennung nicht gefunden werden kann, dann liefert *cuserid()* den Nullzeiger.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Funktion *cuserid()* kann *getpwuid()* verwenden. Daher können die Ergebnisse eines Aufrufs von *getpwuid()* oder *getpwnam()* durch einen nachfolgenden Aufruf von *cuserid()* gelöscht werden.

Drei, zum aktuellen Prozeß gehörende Namen können bestimmt werden: *cuserid()* liefert den Namen, der der effektiven Benutzernummer des Prozesses zugeordnet ist; *getlogin()* liefert den Namen, der den aktuellen Anmeldungs-Aktivitäten zugeordnet ist und *getpwuid* (*getuid()*) liefert den Namen, der zur realen Benutzernummer des Prozesses gehört.

PORTABILITÄT

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

SIEHE AUCH

getlogin(), *getpwnam()*, *getpwuid()*, *<stdio.h>*.

NAME

daylight - daylight savings time flag
Sommerzeit-Kennzeichen

DEFINITION

```
#include <time.h>

extern int daylight;
```

BESCHREIBUNG

Siehe unter *tzset()*.

PORTABILITÄT

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

NAME

dial - establish outgoing terminal line connection
Kommunikationsleitung einrichten

DEFINITION

```
nclude <dial.h>

int dial(call)
CALL call;
```

BESCHREIBUNG

Die Funktion *dial()* richtet eine Kommunikationsleitung zu einer externen Datensichtstation ein. Sie liefert bei Erfolg eine Dateikennzahl zurück, die zum Lesen und Schreiben von bzw. auf diese Kommunikationsleitung verwendet werden kann.

Das Argument *call* der Funktion *dial()* ist eine Struktur des Typs *CALL*, der in der Datei *<dial.h>* wie folgt definiert ist:

```
typedef struct {
   struct termio  *attr; /* Zeiger auf eine Attributstruktur
                            des Typs termio                */
   int     baud;         /* Baudrate fuer Datenuebertragung */
   int     speed;        /* 21A-Modem: low=300, high=1200   */
   char   *line;         /* Name der Geraetedatei für die
                            Kommunikationsleitung          */
   char   *telno;        /* Zeiger auf eine Telefonnummer   */
   int     modem;        /* Modem-Steuerung fuer direkte
                            Leitung                        */
   char   *device;       /* nicht benutzt                   */
   int     dev_len;      /* nicht benutzt                   */
} CALL;
```

Die Komponente *attr* ist ein Zeiger auf eine Struktur des Typs *struct termio*, so wie diese in der Datei *<sys/termio.h>* definiert ist. Wird für diesen Zeiger der Nullzeiger angegeben, so werden keine Eigenschaften für diese Leitung festgelegt. Wird hier ein anderer Zeiger als der Nullzeiger angegeben, so werden die Eigenschaften der Leitung so gesetzt, wie dies in der *termio*-Struktur angegeben wird.

Die Komponente *speed* der *CALL*-Struktur ist nur für die Verwendung im Zusammenhang mit abgehenden Telefonanrufen gedacht. In diesem Fall sollte ihr Wert entweder gleich 300 oder gleich 1200 sein, um das 113A-Modem zu kennzeichnen oder um die Geschwindigkeitseinstellung des 212A-Modems zu bestimmen. Die Geschwindigkeit 300 für das 113A-Modem bzw. für die niedrige Geschwindigkeit des 212A-Modems erlaubt es, Daten in jeder beliebigen Geschwindigkeit zwischen 0 und 300 Baud zu übertragen.

Die Einstellung 1200 für die hohe Geschwindigkeit des 212A-Modems erlaubt nur eine Übertragung mit 1200 Baud.

Die Komponente *baud* der *CALL*-Struktur gibt an, mit welcher Baudrate die Daten übertragen werden sollen. Hier kann zum Beispiel der Wert 110 angegeben werden, wenn *speed* gleich 300 gesetzt ist. Wenn *speed* gleich 1200 gesetzt ist, dann muß auch *baud* gleich 1200 gesetzt sein.

Wenn die geforderte Leitung eine direkte Verbindung ist, dann sollte ein Zeiger auf den Namen der zugehörigen Gerätedatei in die Komponente *line* der Struktur eingetragen werden. Zulässige Namen für diese Gerätedateinamen sind in der Datei */usr/lib/uucp/Devices* eingetragen. Wird ein solcher Gerätedateiname angegeben, dann sollte die Komponente *baud* mit dem Wert –1 besetzt werden. Dadurch ermittelt die Funktion *dial()* den benötigten Wert in der Datei */usr/lib/uucp/Devices*.

Die Komponente *telno* kann, im Fall von abgehenden Telefonanrufen, einen Zeiger auf die zu wählende Telefonnummer enthalten. Die Telefonnummer ist dann eine Zeichenkette. Die einzelnen Zeichen in dieser Zeichenkette haben dabei folgende Bedeutungen:

'0' ... '9'	eine der Ziffern 0 .. 9 wählen
'*'	die Wähltaste '*' wählen
'#'	die Wähltaste '#' wählen
'='	auf einen Sekundär-Wählton warten
'-'	Wählpause von ca. 4 Sekunden

Die Komponente *modem* gibt die Modem-Kontrolle für direkte Leitungen an. Wenn Modem-Kontrolle benötigt wird, dann sollte diese Komponente ungleich 0 sein. Die Komponenten *device* und *dev_len* werden nicht mehr benutzt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *dial()* eine Dateikennzahl für die geöffnete Verbindung zur externen Datensichtstation. Im Fehlerfall wird ein negativer Wert geliefert.

In der Datei *<dial.h>* sind folgende Konstanten für mögliche Fehler reserviert:

Name	Wert	Bedeutung
INTRPT	-1	Es wurde ein Signal empfangen
D_HUNG	-2	Wähleinheit blockiert (keine Rückkehr von *write()*)
NO_ANS	-3	Keine Antwort innerhalb von 10 Sekunden
ILL_BD	-4	Es wurde eine illegale Baudrate angegeben
A_PROB	-5	*open()* fehlgeschlagen (acu-Problem)
L_PROB	-6	*open()* fehlgeschlagen (Leitungs-Problem)
NO_Ldv	-7	Datei *Devices* kann nicht geöffnet werden
DV_NT_A	-8	Gerätedatei nicht verfügbar
DV_NT_K	-9	Gerätedatei nicht vorhanden
NO_BD_A	-10	Gerät mit geforderter Baudrate nicht verfügbar
NO_BD_K	-11	Gerät mit geforderter Baudrate nicht vorhanden
DV_NT_E	-12	Angeforderte Baudrate paßt nicht
BAD_SYS	-13	Rechner nicht in Datei */usr/lib/uucp/Systems*

FEHLER

Es sind keine Fehler definiert.

HINWEIS

- Wenn die Verbindung zu der externen Datensichtstation abgebrochen wurde, dann muß das aufrufende Programm die Funktion *undial()* aufrufen, um das Semaphor freizugeben, das von der Funktion *dial()* intern reserviert wurde.
- Die Datei *<dial.h>* bindet die Datei *<sys/termio.h>* intern automatisch durch #*include* mit ein.
- Die Funktion *dial()* ruft intern die Funktion *alarm()* mit dem Wert 3600 auf und fängt das Signal SIGALRM auch ab. Wenn das Benutzerprogramm länger als eine Stunde mit der Leitung arbeitet, dann kann das SIGALRM-Signal auftreten, während eine der Funktionen *read()* oder *write()* abgearbeitet wird. In solchen Fällen kehrt dann die *read()*- oder *write()*-Funktion ggf. mit einem Fehler zurück (errno == [EINTR]). Daher sollte eine Anwendung solche Funktionsaufrufe überprüfen und bei Auftreten dieses Fehlers noch einmal aufrufen.

PORTABILITÄT

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

SIEHE AUCH

alarm(), *read()*, *undial()*, *write()*, *<sys/termio.h>*.

NAME

closedir, opendir, readdir,
rewinddir, seekdir, telldir - directory operations
Operationen auf Dateiverzeichnissen

DEFINITION

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

void closedir(dvz_zg)
DIR *dvz_zg;

DIR *opendir(dateiname)
char *dateiname;

struct direct *readdir(dvz_zg)
DIR *dvz_zg;

void rewinddir(dvz_zg)
DIR *dvz_zg;

void seekdir(dvz_zg, adr)
DIR *dvz_zg;
long adr;

long telldir(dvz_zg)
DIR *dvz_zg;
```

BESCHREIBUNG

Diese Funktionen stellen eine systemunabhängige Schnittstelle für den Zugriff auf Dateiverzeichnisse dar.

Ab dieser Ausgabe des Handbuchs werden die im Abschnitt DEFINITION genannten Funktionen unter ihrem jeweiligen Namen beschrieben. Dort finden Sie eine ausführliche Beschreibung jeder einzelnen Funktion.

PORTABILITÄT

Die Funktionen zur Bearbeitung von Dateiverzeichnissen sind im X/Open-Standard (Ausgabe 3) definiert.

NAME

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 - generate uniformly distributed pseudo-random numbers
Gleichverteilte Pseudo-Zufallszahlen erzeugen

DEFINITION

```
double drand48 ( )

double erand48 (xsubi)
unsigned short xsubi[3];

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

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

long lrand48 ( )

long mrand48 ( )

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

unsigned short *seed48 (seed16v)
unsigned short seed16v[3];

void srand48 (seedval)
long seedval;
```

BESCHREIBUNG

Diese Familie von Funktionen erzeugt Pseudo-Zufallszahlen unter Verwendung des Algorithmus der linearen Kongruenz und von ganzzahliger 48-Bit-Arithmetik.

Die Funktionen *drand48()* und *erand48()* liefern nichtnegative Gleitpunktzahlen doppelter Genauigkeit, die gleichmäßig über das Intervall [0.0 , 1.0) verteilt sind.

Die Funktionen *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 vorzeichenbehaftete Ganzzahlen vom Typ long, die gleichmäßig über das Intervall interval [-2^{31},2^{31}) verteilt sind.

Die Funktionen *srand48()*, *seed48()* und *lcong48()* sind Initialisierungsprozeduren, von denen eine vor dem Aufruf von entweder *drand48()*, *lrand48()* oder *mrand48()* aufgerufen werden sollte. (Obwohl es nicht empfohlene Praxis ist, werden bei Aufruf von *drand48()*, *lrand48()* oder *mrand48()* ohne vorhergehenden Aufruf zur Initialisierung automatisch voreingestellte Anfangswerte zur Verfügung gestellt.) Die Funktionen *erand48()*, *nrand48()* und *jrand48()* benötigen keinen vorausgehenden Initialisierungsaufruf.

Alle Prozeduren arbeiten mit einer Sequenz von ganzzahligen 48-bit
Werten X_i, die der linear kongruenten Formel entsprechend gebildet werden:

$$X_{n+1} = (aX_n + c)_{mod\ m} \quad n \geq 0$$

Für den Parameter m gilt: $m = 2^{48}$; es wird also 48-bit Ganzzahlarithmetik betrieben. Sofern nicht *lcong48()* aufgerufen wurde, ist der Wert des Faktors a und der additiven Konstante c gegeben durch:

$$a = 5DEECE66D_{16} = 273673163155_8$$
$$c = B_{16} = 13_8.$$

Jedes Ergebnis einer der Funktionen *drand48, erand48, lrand48()*, *nrand48, mrand48()* oder *jrand48()* entsteht folgendermaßen. Zuerst wird das nächste 48-bit Reihenelement X_i berechnet. Dann werden, je nach Datentyp der Rückgabevariablen, entsprechend viele Bits aus dem höchstwertigen Anteil von X_i (äußerst linke Bits) kopiert und in das Ergebnis umgewandelt.

Die Funktionen *drand48, lrand48()* und *mrand48()* speichern den zuletzt erzeugten 48-bit Wert X_i in einem internen Puffer; dies ist auch der Grund, weshalb sie vor dem ersten Aufruf initialisiert werden müssen. Die Funktionen *erand48, nrand48()* und *jrand48()* erwarten vom aufrufenden Programm die Bereitstellung von Speicherplatz für die sukzessiven Werte X_i in Form eines Vektors, der beim Aufruf als Parameter übergeben wird. Deswegen brauchen diese Funktionen nicht initialisiert zu werden; das aufrufende Programm braucht lediglich den benötigten Anfangswert von X_i in dem Vektor abzulegen und diesen als Argument zu übergeben.

Durch die Verwendung verschiedener Argumente erlauben es die Funktionen *erand48()*, *nrand48()* und *jrand48()*, in getrennten Modulen größerer Programme mehrere *unabhängige* Folgen von Pseudo-Zufallszahlen zu generieren, d.h. die Reihenfolge der Zahlen in einer Folge ist *nicht* abhängig davon, wie oft die Routinen für die Generierung von Zahlen in anderen Folgen aufgerufen wurden.

Die Initialisierungsfunktion *srand48()* setzt die höherwertigen 32 Bit von X_i auf den Wert der {LONG_BIT} Bits ihres Arguments. Die niederwertigen 16 Bit von X_i werden mit dem willkürlichen Wert $330E_{16}$ belegt.

Die Initialisierungsfunktion *seed48()* setzt den Wert von X_i auf den 48-bit Wert, der im übergebenen Vektor angegeben wird. Zusätzlich wird der frühere Wert von X_i in einem internen 48-bit Puffer abgespeichert, der nur von *seed48()* verwendet wird und dessen Adresse von *seed48()* zurückgegeben wird. Dieser zurückgegebene Zeiger kann ignoriert werden, wenn er nicht benötigt wird. Er ist jedoch nützlich, falls ein Programm zu irgendeinem späteren Zeitpunkt von einer gegebenen Stelle aus neu gestartet werden soll. Es kann den Zeiger dazu benutzen, den letzten Wert von X_i zu ermitteln und abzuspeichern, um dann durch *seed48()* diesen Wert für eine Reinitialisierung beim Neustart zu verwenden.

Die Initialisierungsfunktion *lcong48()* erlaubt dem Benutzer, die Voreinstellungswerte von X_i, des Faktors a und der additiven Konstante c festzulegen. Die Vektorelemente *param[0]* bis *param[2]* legen X_i fest, *param[3]* bis *param[5]* legen den Faktor a und *param[6]* legt die 16-bit Additionskonstante c fest. Nach dem Aufruf von *lcong48()* wird ein nachfolgender Aufruf von entweder *srand48()* oder *seed48()* diesen "Standard"-Faktor a und die additive Komponente c wiederherstellen, wie oben angegeben.

ERGEBNIS

Wie oben im Abschnitt BESCHREIBUNG angegeben.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

rand().

NAME

dup, dup2 - duplicate an open file descriptor
Offene Dateikennzahl duplizieren

DEFINITION

```
int dup (fildes)
int fildes;

int dup2 (fildes, fildes2)
int fildes, fildes2;
```

BESCHREIBUNG

Die Funktionen *dup()* und *dup2()* bieten eine alternative Schnittstelle zu dem Dienst, der durch die Funktion *fcntl()* mit dem Kommando F_DUPFD bereitgestellt wird. Der Aufruf:

```
fid = dup (fildes);
```

ist äquivalent zu:

```
fid = fcntl (fildes, F_DUPFD, 0);
```

Der Aufruf:

```
fid = dup2 (fildes, fildes2);
```

ist äquivalent zu:

```
close (fildes2);
fid = fcntl (fildes, F_DUPFD, fildes2);
```

mit Ausnahme der folgenden Punkte:

- Wenn *fildes2* kleiner als 0 oder größer oder gleich {OPEN_MAX} ist, dann liefert *dup2()* den Wert -1, wobei *errno* gleich [EBADF] gesetzt wird.
- Wenn *fildes* eine gültige Dateikennzahl und gleich *fildes2* ist, dann liefert die Funktion *dup2()* *fildes2* ohne diese zu schließen.
- Wenn *fildes* keine gültige Dateikennzahl ist, dann liefert *dup2()* den Wert -1 und schließt *fildes2* nicht.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion eine nichtnegative ganze Zahl, die Dateikennzahl. Andernfalls wird der Wert -1 geliefert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Die Funktionen *dup()* und *dup2()* schlagen fehl, wenn gilt:

[EBADF] Das Argument *fildes* ist keine gültige offene Dateikennzahl oder *fildes2* ist außerhalb des erlaubten Bereichs.

[EINTR] Die Funktion *dup()* oder *dup2()* wurde vorzeitig durch ein Signal abgebrochen.

[EMFILE] Die Anzahl der durch den Prozeß verwendeten Dateikennzahlen würde {OPEN_MAX} überschreiten, oder es sind keine Dateikennzahlen größer oder gleich *fildes2* verfügbar.

HINWEIS

Die Funktion *dup2()* kann erfolgreich zurückkehren, ohne die erwartete Dateikennzahl zu liefern. Dies könnte zum Beispiel dann der Fall sein, wenn ein Signal zwischen den Aufrufen von *close()* und *fcntl()* eintrifft und die Signalbehandlung eine andere Datei öffnet.

PORTABILITÄT

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

BEISPIEL

Dateikennzahl 0 für Standardeingabe wird mit *datei* verbunden:

```
#include <stdio.h>

main()
{
   int dk, c;
   /* datei zum Lesen oeffnen: */
   if ((dk = open("datei",0)) ==-1)
      printf("Datei kann nicht geoeffnet werden\n");

   /* Dateikennzahl 0 fuer Standardeingabe freigeben: */
   close(0);

   /* dk verdoppeln; niedrigste freie          */
   /* Dateikennzahl (jetzt 0) wird zugewiesen  */
   dup(dk);

   /* Dateikennzahl 0 weist jetzt auf datei. Dateikennzahl
      dk wird nicht mehr benoetigt: */
   close(dk);

   /* Einlesen von der "neuen" Standardeingabe datei: */
   while ((c=getchar()) != EOF)
       putchar(c);
}
```

SIEHE AUCH

close(), *fcntl()*, *open()*.

NAME

ecvt - convert floating-point number to string
Gleitkommazahl in Zeichenkette umwandeln

DEFINITION

```
char *ecvt (wert, anz, dezpkt, vz)
double wert;
int anz, *dezpkt, *vz;
```

BESCHREIBUNG

ecvt() konvertiert *wert* in einer mit dem Nullbyte abgeschlossene Zeichenkette von *anz* Ziffern und gibt einen Zeiger darauf zurück. Die höchstwertige Ziffer ist ungleich Null, es sei denn, der Wert von *wert* ist Null (0). Die niedrigstwertige Ziffer ist gerundet.

Die Position des Dezimalpunkts relativ zum Anfang der Zeichenreihe wird indirekt über *dezpkt* zurückgegeben (ein negativer Wert bedeutet links vom ersten Zeichen der zurückgegebenen Zeichenreihe). Der Dezimalpunkt selbst ist nicht in der zurückgegebenen Zeichenkette enthalten.

Das Zeichen für den Dezimalpunkt wird aus der internationalen Umgebung des Programms ermittelt (Kategorie LC_NUMERIC), wenn zuvor ein erfolgreicher Aufruf von *setlocale()* erfolgt ist.

Bei negativem Vorzeichen ist der durch *vz* referenzierte Wert ungleich Null, andernfalls ist er Null.

HINWEIS

Die Rückgabewerte von *ecvt()* und *fcvt()* verweisen auf denselben statischen Datenbereich, dessen Inhalt bei jedem Aufruf einer dieser Funktionen überschrieben wird.

PORTABILITÄT

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

BEISPIEL

Das Programm liest einen Gleitkommawert ein, wandelt ihn nach der Angabe in *anz* um und gibt ihn als ASCII-Zeichenkette wieder aus. Zusätzlich wird das berechnete Vorzeichen ausgegeben:

```
#include <stdio.h>
char *ecvt();

main()
{
   double wert;
   int anz, dez_pkt,vorzeichen;
   printf("Bitte Gleitkommazahl eingeben: ");
   if (scanf("%lf",&wert) == 1)
   {
      printf("Wieviel signifikante Stellen. ");
      if (scanf("%d",&anz) == 1)
      {
       printf("Die Zahl lautet umgewandelt : %s \n",
              ecvt(wert,anz,&dez_pkt,&vorzeichen));
       printf("Das Vorzeichen ist %s\n",
             (vorzeichen == 0 ? "positiv" : "negativ"));
      }
   }
}
```

SIEHE AUCH

fcvt(), *gcvt()*, *printf()*.

NAME

edata - lowest address beyond initialised data
Erste Adresse hinter dem initialisierten Datenbereich

DEFINITION

```
extern char *edata;
```

BESCHREIBUNG

Die Adresse *edata* ist der erste Speicherplatz hinter dem initialisierten Datenbereich.

Weitere Adressen sind:

etext	erste Adresse hinter dem Programmcode (s. *etext*)
end	erste Adresse hinter dem nicht initialisierten Datenbereich (s. *end*)

PORTABILITÄT

Die Variable *edata* ist im X/Open-Standard nicht mehr enthalten.

SIEHE AUCH

brk(), *end*, *etext*.

NAME

encrypt - encoding function
Verschlüsselungsfunktion

DEFINITION

```
void encrypt (block, edflag)
char block[64];
int edflag;
```

BESCHREIBUNG

Die Funktion *encrypt()* bietet (einen ziemlich einfachen) Zugriff auf einen Verschlüsselungsalgorithmus. Der druch *setkey()* erzeugte Schlüssel wird dazu verwendet, die Zeichenkette *block* mit der Funktion *encrypt()* zu verschlüsseln.

Das Argument *block* der Funktion *encrypt()* ist ein Vektor der Länge 64 Bytes, der nur Elemente mit den numerischen Werten 0 und 1 enthält. Dieser Vektor wird direkt in einen ähnlichen Vektor geändert, wobei der von *setkey()* gesetzte Schlüssel verwendet wird. Wenn *edflag* gleich 0 ist, dann wird das Argument ver-, ansonsten wird es entschlüsselt.

ERGEBNIS

Die Funktion *encrypt()* liefert kein Ergebnis.

FEHLER

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

[ENOSYS] Diese Funktionalität wird unter der aktuellen Implementierung nicht unterstützt. Dieser Fehler kann nur dann auftreten, wenn das Programm auf ein anderes X/Open-kompatibles System portiert wird.

HINWEIS

Da die Funktion *encrypt()* kein Ergebnis liefert, sollten Anwendungen, die auf Fehler prüfen wollen, die Variable *errno* gleich 0 setzen, danach *encrypt()* aufrufen und dann *errno* prüfen. Wenn *errno* dann ungleich 0 ist, dann muß ein Fehler angenommen werden.

PORTABILITÄT

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

SIEHE AUCH

crypt(), *setkey()*.

NAME

end - data location in programm
Erste Adresse hinter dem nicht initialisierten Datenbereich

DEFINITION

```
extern char *end;
```

BESCHREIBUNG

Die Adresse *end* ist der erste Speicherplatz hinter dem nicht initialisierten Datenbereich.

Weitere Adressen sind:

etext	erste Adresse hinter dem Programmcode (s. *etext*)
edata	erste Adresse hinter dem initialisierten Datenbereich (s. *edata*)

Zu Programmbeginn fällt der "break" (erster Speicherplatz hinter dem Datenbereich) mit *end* zusammen. Er kann jedoch durch die Funktionen *brk()*, *malloc()*, Standard Ein-/Ausgabe usw. neu gesetzt werden. Daher sollte der aktuelle Wert für den "break" mit *sbrk(0)* (s. *brk()*) bestimmt werden.

PORTABILITÄT

Die Variable *end* ist im X/Open-Standard nicht mehr enthalten.

SIEHE AUCH

brk(), *edata()*, *etext()*.

NAME

endcfent - Speicher freigeben

DEFINITION

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

void endcfent()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen zur Druckerbeschreibung.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getcfadent()-getcftynam()*.

PORTABILITÄT

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

NAME

endpdent - Speicher freigeben

DEFINITION

```
#include <lpr.h>

void endpdent()
```

DEFINITION

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie Auftragsdateien für Drucker durchsuchen können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getpdjbent()-getpdprnam()*.

PORTABILITÄT

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

NAME

endpwent - Schließen der Kennwortdatei

DEFINITION

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

void endpwent ()
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie die Kennwortdatei */etc/passwd* bearbeiten können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getpwent()*.

PORTABILITÄT

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

NAME

endutent - close utmp file
Bearbeitung der utmp-Datei beenden

DEFINITION

```
nclude <utmp h>

void endutent()
```

BESCHREIBUNG

Die Funktion *endutent()* gehört zu einer Gruppe von Funktionen, die Einträge in der Datei */etc/utmp* bearbeiten. Sie beendet die Bearbeitung der *utmp*-Datei. Eine ausführliche Beschreibung finden Sie unter *getutent()*.

PORTABILITÄT

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

NAME

environ - array of character pointers to the environment strings
Zeiger-Vektor auf die Umgebungsvariablen

DEFINITION

```
extern char **environ;
```

BESCHREIBUNG

Siehe unter *exec*.

PORTABILITÄT

Der Vektor *environ* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

erand48 - generate uniformly distributed pseudo-random numbers
Gleichmäßig verteilte Pseudo-Zufallszahlen erzeugen

DEFINITION

```
double erand48 (xsubi)
unsigned short xsubi[3];
```

BESCHREIBUNG

Siehe unter *drand48()*.

PORTABILITÄT

Die Funktion *erand48()* ist im X/Open-Standard (Ausgabe 3) definiert.

NAME

erf, erfc - error und complementary error functions
Fehler- und komplementäre Fehlerfunktion

DEFINITION

```
#include <math.h>

double erf (x)
double x;

double erfc (x)
double x;
```

BESCHREIBUNG

Die Funktion *erf()* berechnet die Fehlerfunktion zu x, die wie folgt definiert ist:

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Die Funktion *erfc(x)* berechnet 1.0 - *erf(x)*.

ERGEBNIS

Bei erfolgreicher Beendigung liefern die Funktionen *erf()* und *erfc()* den Wert der Fehlerfunktion bzw. der komplementären Fehlerfunktion zu x.

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

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

FEHLER

Die Funktionen *erf()* und *erfc()* können unter anderen X/Open-kompatiblen Implementierungen fehlschlagen, wenn gilt:

[EDOM] Der Wert von x ist NaN.

HINWEIS

Die Funktion *erfc()* wird angeboten, da bei einem Aufruf von *erf(x)* für ein großes x und einer anschließenden Subtraktion des Ergebnisses von 1.0 ein hoher Verlust an relativer Genauigkeit eintritt.

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

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

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

cc programm.c -lm

PORTABILITÄT

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

SIEHE AUCH

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

NAME

errno - XSI error return value
Globale Fehlernummer

DEFINITION

```
#include <errno.h>

extern int errno;
```

BESCHREIBUNG

Die externe Variable *errno* wird von den Funktionen in diesem Handbuch verwendet, um Fehlernummern zurückzugeben. Die für *errno* jeweils möglichen symbolischen Werte werden bei jeder Funktion im Abschnitt FEHLER dokumentiert.

PORTABILITÄT

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

SIEHE AUCH

<errno.h>, Abschnitt 2.2, Fehlernummern.

NAME

etext - Erste Speicheradresse hinter dem Programmcode

DEFINITION

```
extern char *etext;
```

BESCHREIBUNG

Die Adresse *etext* ist der erste Speicherplatz hinter dem Programmcode.

Weitere Adressen sind:

end	erste Adresse hinter dem nicht initialisierten Datenbereich (s. *end*)
edata	erste Adresse hinter dem initialisierten Datenbereich (s. *edata*)

PORTABILITÄT

Die Variable *etext* ist im X/Open-Standard nicht enthalten.

SIEHE AUCH

brk(), *edata*, *end*.

NAME

environ, execl, execv, execle,
execve, execlp, execvp - execute a file
Datei ausführen

DEFINITION

```
extern char **environ;

int execl (path, arg0, arg1, ..., argn, (char *)0)
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv[ ];

int execle (path, arg0, arg1, ..., argn, (char *)0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

int execve (path, argv, envp)
char *path, *argv[ ], *envp[ ];

int execlp (file, arg0, arg1, ..., argn, (char *)0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];
```

BESCHREIBUNG

Die Funktionen der *exec*-Familie ersetzen das aktuelle Prozeßabbild durch ein neues. Das neue Prozeßabbild wird aus einer normalen, ausführbaren Datei erzeugt, die *neuen Abbilddatei des Prozesses* genannt wird. Ein Aufruf von *exec* kehrt niemals zurück, da das aufrufende Prozeßabbild durch das neue Prozeßabbild überlagert wird.

Wenn als Ergebnis dieses Aufrufs ein C-Programm ausgeführt wird, dann wird dieses wie folgt als ein C-Funktionsaufruf angesprungen:

int main (*argc, argv*)
int *argc*; char ***argv*;

wobei *argc* der Argumenten-Zähler und *argv* ein Vektor von Zeigern auf *char* ist, die auf die Argumente selbst zeigen.

Zusätzlich wird die Variable

extern char **environ;

als die Adresse eines Vektors von Zeigern auf *char* initialisiert, die auf die Umgebungs-Zeichenketten zeigen. Die Vektoren *argv* und *environ* werden beide durch den Nullzeiger abgeschlossen. Der Nullzeiger, der den Vektor *argv* abschließt, wird in *argc* nicht mitgezählt.

Die von einem Programm bei einer der *exec*-Funktionen angegebenen Argumente werden an das neue Prozeßabbild in den entsprechenden Argumenten von *main()* übergeben.

Das Argument *path* zeigt auf einen Pfadnamen, der die neue Abbilddatei des Prozesses angibt.

Das Argument *file* wird verwendet, um einen Pfadnamen aufzubauen, der die neue Abbilddatei des Prozesses identifiziert. Wenn das Argument *file* nicht mit einem Schrägstrich beginnt, dann wird der Pfadnamenanfang durch eine Durchsuchung derjenigen Dateiverzeichnisse ermittelt, die in der Umgebungsvariablen *PATH* übergeben werden. Wenn es diese Umgebungsvariable nicht gibt, dann durchsuchen diese Funktionen unter SINIX die Dateiverzeichnisse */bin* und */usr/bin*. Andere X/Open-kompatible Systeme können für diesen Fall andere Vorgaben definieren.

Wenn die Abbilddatei des Prozesses kein gültiges ausführbares Objekt ist, dann verwenden die Funktionen *execlp()* und *execvp()* den Inhalt dieser Datei als die Standardeingabe eines Kommando-Interpreters, analog zu *system()*. In diesem Fall wird der Kommando-Interpreter das neue Prozeßabbild.

Die Argumente *arg0, arg1, ..., argn* sind Zeiger auf Zeichenketten, die mit dem Nullbyte abgeschlossen sind. Diese Zeichenketten bilden die Argumentliste, die dem neuen Prozeßabbild zur Verfügung steht. Die Liste wird durch einen Nullzeiger abgeschlossen. Das Argument *arg0* sollte normalerweise auf einen Dateinamen zeigen, der dem Prozeß zugeordent ist.

Das Argument *envp* ist ein Vektor von Zeigern auf Zeichenketten, die mit dem Nullbyte abgeschlossen sind. Diese Zeichenketten bilden die Umgebung für das neuen Prozeßabbild. Der Vektor *envp* wird durch einen Nullzeiger abgeschlossen.

Bei den Formen, die keinen Zeiger *envp* enthalten (*execl()*, *execv()*, *execlp()* und *execvp()*), wird die Umgebung für das neue Prozeßabbild aus der externen Variablen *environ* des aufrufenden Prozesses gewonnen.

Die Anzahl von Bytes, die für die die Argument- und Umgebungsliste des Prozesses zur Verfügung steht, ist {ARG_MAX}. Unter SINIX schließt die Konstante {ARG_MAX} den Platz für abschließende Nullbytes, Zeiger und/oder Füllbytes mit ein. Andere X/Open-kompatible Systeme können hier andere Vereinbarungen treffen.

Offene Dateikennzahlen des aufrufenden Prozeßabbilds bleiben auch im neuen Prozeßabbild offen, außer denen, für die das sbe-Bit FD_CLOEXEC gesetzt ist (siehe auch *fcntl()*). Für die Dateien, die offen bleiben, bleiben auch alle Attribute der offenen Dateibeschreibung bestehen, einschließlich von Sperren der Datei.

Der Zustand von Dateiverzeichnis-Strömen und Meldungskatalog-Deskriptoren im neuen Prozeßabbild ist undefiniert.

Signale, die im aufrufenden Prozeßabbild auf die Standard-Aktion SIG_DFL gesetzt sind, werden im neuen Prozeßabbild auf die Standard-Aktion gesetzt. Signale, die im aufrufenden Prozeßabbild ignoriert werden (SIG_IGN) werden auch im neuen Prozeßabbild ignoriert. Signale, die im aufrufenden Prozeßabbild abgefangen werden, werden im neuen Prozeßabbild auf die Standard-Aktion gesetzt (siehe auch *<signal.h>*).

Wenn das s-Bit für den Eigentümer bei der neuen Abbilddatei des Prozesses gesetzt ist (siehe auch *chmod()*), dann wird die effektive Benutzernummer des neuen Prozeßabbilds auf die Benutzernummer des Eigentümers der neuen Abbilddatei des Prozesses gesetzt. Analog dazu wird, wenn das s-Bit für die Gruppe der neuen Abbilddatei des Prozesses gesetzt ist, die effektive Gruppennummer des neuen Prozeßabbilds auf die Gruppennummer der neuen Abbilddatei des Prozesses gesetzt. Die reale Benutzer- und Gruppennummer, sowie die weiteren Gruppennummern des neuen Prozeßabbilds bleiben dieselben wie die des aufrufenden Prozeßabbilds. Die effektive Benutzer- und die effektive Gruppennummer des neuen Prozeßabbilds werden für eine Verwendung durch die Funktion *setuid()* als die *gesicherte Benutzer-* und die *gesicherte Gruppennummer* gespeichert.

Gemeinsam genutzte Speicherbereiche, die an das aufrufende Prozeßabbild angehängt sind, werden nicht an das neue Prozeßabbild angehängt (siehe auch *shmat()*). Gemeinsam genutzter Hauptspeicher ist ein *optionaler* Dienst.

Der neue Prozeß erhält auch die folgenden Attribute aus dem aufrufenden Prozeßabbild.

- Priorität (siehe auch *nice()*)
- *semadj*-Werte (siehe auch *semop()*)
- Prozeßnummer
- Vaterprozßnummer
- Prozeßgruppennummer
- Mitgliedschaft in einer Sitzung
- reale Benutzernummer
- reale Gruppennummer
- weitere Gruppennummern
- Restzeit bis zu einem Alarmuhr-Signal (siehe auch *alarm()*)
- aktuelles Dateiverzeichnis
- Root-Dateiverzeichnis
- Schutzbitmaske (siehe auch *umask()*)
- maximale Dateigröße (siehe auch *ulimit()*)
- Signalmaske (siehe auch *sigprocmask()*)
- anstehende Signale (siehe auch *sigpending()*)
- *tms_utime* , *tms_stime* , *tms_cutime* , und *tms_cstime* (siehe auch *times()*)

Alle diese Prozeßattribute sind im neuen und im alten Prozeßabbild identisch.

Bei erfolgreicher Beendigung markieren die *exec*-Funktionen das Feld *st_atime* der Datei zum Ändern. Wenn eine *exec*-Funktion fehlschlägt, aber die Abbilddatei des Prozeses gefunden hatte, dann ist nicht festgelegt, ob das Feld *st_atime* zum Ändern markiert ist. Sollte die *exec*-Funktion erfolgreich sein, dann nimmt man an, daß die Abbilddatei des Prozesses geöffnet wurde. Das korrespondierende Schließen wird für einen Zeitpunkt nach dem Öffnen angesetzt, aber vor der Beendigung des Prozeses oder der erfolgreichen Beendigung eines nachfogenden Aufrufs einer der *exec*-Funktionen.

ERGEBNIS

Wenn eine der *exec*-Funktionen zum aufrufenden Prozeßabbild zurückkehrt, so ist ein Fehler aufgetreten, der Rückgabewert gleich -1 und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

Die *exec*-Funktionen schlagen fehl, wenn gilt:

[E2BIG] Die Anzahl der Bytes, die von der neuen Argument- und Umgebungsliste des Prozeßabbilds verwendet werden, ist größer als die vom System festgelegte Grenze von {ARG_MAX} Bytes.

[EACCES]
Die Durchsucherlaubnis für ein Dateiverzeichnis im Pfadnamen-Anfang der neuen Abbilddatei des Prozesses ist nicht gegeben oder die neue Abbilddatei des Prozesses verweigert die Ausführungserlaubnis oder ist keine normale Datei und die Implementierung unterstützt die Ausführung von Dateien dieses Typs nicht.

[ENAMETOOLONG]
Die Länge der Argumente *path* oder *file* oder ein Element der Umgebungsvariablen *PATH*, das einer Datei voangestellt wird, überschreitet {PATH_MAX}, oder eine Pfadnamen-Komponente ist länger als {NAME_MAX} und {_POSIX_NO_TRUNC} ist für diese Datei aktiv.

[ENOENT]
Eine oder mehrere Komponeten des Pfadnamens der neuen Abbilddatei des Prozesses existiert nicht oder das Argument *path* oder *file* zeigt auf eine leere Zeichenkette.

[ENOEXEC]
Die neue Abbilddatei des Prozesses besitzt die benötigten Zugriffrechte, hat jedoch nicht das richtige Format.

[ENOMEM]
Unter SINIX können diese Funktionen auch dann fehlschlagen, wenn nicht genügend Speicher für die Ausführung zur Verfügung steht. Dieser Fehler ist im XPG3 [6] nicht definiert.

[ENOTDIR]
Eine Komponente des Pfadnamen-Anfangs der neuen Abbilddatei des Prozesses ist kein Dateiverzeichnis.

Die *exec*-Funktionen können in anderen X/Open-kompatiblen Implementierungen zusätzlich fehlschlagen, wenn gilt:

[ETXTBSY]
Die neue Abbilddatei des Prozesses ist eine reine Prozedurdatei (gemeinsam genutzer Text), die gerade von einem anderen Prozeß beschrieben wird.

HINWEIS

Da der Zustand von Dateiverzeichnis-Strömen und Meldungskatalog-Deskriptoren im neuen Prozeßabbild undefiniert ist, sollten sich portable Anwendungen nicht auf deren Verwendung abstützen und diese vor der Verwendung einer der *exec*-Funktionen schließen.

Aus Gründen der Rückwärtskompatibilität mit Ausgabe 2 unterstützen X/Open-kompatible Implementierungen noch immer den folgenden Aufruf von *main()*:

int main (*argc, argv, envp*)
int *argc*; char *******argv*, *******envp*;

Die Verwendung von *envp* ist veraltet und Anwendungen sollten die Funktionen *getenv()* und *putenv()* verwenden, um ihre Umgebung zu manipulieren.

Wenn für eines der Argumente eine ungültige Adresse angegeben wird, dann wird für den aufrufenden Prozeß das Signal SIGSEGV generiert. Früher wurde in diesem Fall die Variable *errno* mit dem Wert [EFAULT] besetzt.

PORTABILITÄT

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

BEISPIEL

Erstellen Sie folgendes Programm:

```
#include <stdio.h>
extern int errno;
extern char *sys_errlist[];
extern int sys_nerr;

char *umgbszg[]={0};

main()
{
   static char *zeiger[]={ "/usr1/felix/ces.prog/ich",
                           "ich", NULL};
   int a;
   if ((a=execve(zeiger[0], zeiger+1, umgbszg))==-1)
      perror("Fehler");
   printf("Ende");     /* Bei Erfolg wird ENDE nie ausgegeben */
}
```

Erstellen Sie die Prozedur "ich" mit dem angegebenen Inhalt und ändern Sie die Zugriffsrechte (chmod u+x ich).

Prozedur *ich*:

```
#!/bin/sh
echo Das war's!
```

Compilieren Sie nun das Programm und rufen Sie anschließend a.out auf: "Das war's!"

2. Beispiel

Das Beispiel zeigt, wie die Funktion *execle* arbeitet: Wenn Sie den Teil des Programmes, der von den Kommentarzeichen eingeschlossen ist, auch übersetzen lassen, wird die Umgebungsvariable PATH neu definiert und die Shell findet *ps* nicht mehr.

```
#include <stdio.h>

extern char **environ;

main()
{
   int i;

   for (i = 0; environ[i] != (char*)NULL; i++)
   {
   /*   if (strncmp (environ[i], "PATH=", 5) == 0)  */
   /*      environ[i] = "PATH=:/usr/bin";           */
        printf("environ[%d]: %s\n", i, environ[i]);
   }
   if (execle ("/bin/sh", "sh", "-c", "ps", NULL, environ) == -1)
   {
      perror("execle");
      exit(13);
   }
}
```

SIEHE AUCH

alarm(), *exit()*, *fcntl()*, *fork()*, *getenv()*, *nice()*, *putenv()*, *semop()*, *shmat()*, *sigaction()*, *system()*, *times()*, *ulimit()*, *umask()*.

NAME

exit, _exit - terminate process
Prozeß beenden

DEFINITION

```
#include <stdlib.h>

void exit (status)
int status;

void _exit (status)
int status;
```

BESCHREIBUNG

Die Funktion *exit()* leert alle Augabepuffer und schließt alle offenen Ströme. Die Funktionen *_exit()* und *exit()* beenden den aufrufenden Prozeß mit den folgenden Konsequenzen:

- Alle Dateikennzahlen, Dateiverzeichnis-Ströme und Meldungskatalog-Deskriptoren, die für den aufrufenden Prozeß offen sind, werden geschlossen.
- Wenn der Vaterprozeß des aufrufenden Prozesses *wait()* oder *waitpid()* ausführt, dann wird dieser von der Beendigung des aufrufenden Prozesses benachrichtigt und die niederwertigen acht Bit von *status*, d.h die Bits 0377, werden ihm verfügbar gemacht. Siehe auch *wait()* und *waitpid()*.
- Wenn der Vaterprozeß nicht wartet, dann wird der Status des Sohnprozesses diesem dann verfügbar gemacht, wenn der Vaterprozeß anschließend *wait()* oder *waitpid()* ausführt.
- Wenn der Vaterprozeß des aufrufenden Prozesses kein *wait()* oder *waitpid()* ausführt, dann wird der aufrufende Prozeß in einen sogenannten Zombieprozeß umgewandelt. Ein Zombieprozeß ist ein inaktiver Prozeß; er wird zu einem späteren Zeitpunkt gelöscht, nämlich wenn sein Vaterprozeß *wait()* oder *waitpid()* ausführt.
- Die Beendigung eines Prozesses beendet nicht unmittelbar dessen Söhne. Das Sendes des Signals SIGHUP beendet, wie unten beschrieben, Sohnprozesse indirekt unter bestimmten Umständen.
- Unter SINIX wird zusätzlich das Signal SIGCHLD an den Vaterprozeß gesendet.

- Die Vaterprozeßnummer aller existierenden Sohn- oder Zombieprozesse des aufrufenden Prozesses wird gleich der Prozeßnummer eines speziellen Systemprozeses gesetzt. Das heißt, diese Prozesse werden von einem speziellen Systemprozeß geerbt, unter SINIX ist dies der *init*-Prozeß.
- Jedes angehängte Segment gemeinsam genutzten Speichers wird abgehängt und der Wert von *shm_nattch* (siehe auch *shmget()*) in der seiner Nummer für gemeinsam genutzten Speicher zugeordneten Datenstruktur wird um 1 dekrementiert.
- Für jedes Semaphor, für das der aufrufende Prozeß einen *semadj*-Wert gesetzt hat (siehe auch *semop()*), wird dieser Wert zum *semval* des angegebenen Semaphors addiert.
- Wenn der Prozeß ein kontrollierender Prozeß ist, dann wird das Signal SIGHUP an jeden Prozeß in der Vordergrundprozeßgruppe des kontrollierenden Terminals gesendet, das zu dem aufrufenden Prozeß gehört.
- Wenn der Prozeß ein kontrollierender Prozeß ist, dann wird das kontrollierende Terminal, das dieser Sitzung zugeordnet ist, wieder freigegeben, wodurch es von einem neuen kontrollierenden Prozeß belegt werden kann.

ERGEBNIS

Diese Funktionen kehren nicht zurück.

HINWEIS

Normalerweise sollten Anwendungen *exit()* anstelle von *_exit()* verwenden.

PORTABILITÄT

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

SIEHE AUCH

fclose(), *semop()*, *shmget()*, *sigaction()*, *wait()*, *<stdlib.h>*.

NAME

exp - exponential function
Exponentialfunktion

DEFINITION

```
#include <math.h>

double exp (x)
double x;
```

BESCHREIBUNG

Die Funktion *exp()* berechnet die Exponentialfunktion zu x, die als e^x definiert ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *exp()* die Exponentialfunktion zu x.

Wenn der korrekte Wert einen Überlauf verursachen würde, dann liefert *exp()* den Wert HUGE_VAL und setzt *errno* gleich [ERANGE].

Wenn der korrekte Wert einen Unterlauf verursachen würde, dann liefert *exp()* den Wert 0 und setzt *errno* gleich [ERANGE].

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

FEHLER

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

[ERANGE] Das Ergebnis würde einen Über- oder Unterlauf verursachen.

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

[EDOM] Der Wert von x ist ein NaN.

HINWEIS

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

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

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Die Funktion berechnet e^x für einen eingelesenen Wert *x*:

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

main()
{
   double x,y;
   if ( scanf("%lf",&x) == 1)
   {
      y =exp(x);
      perror("exp");
      printf("exp(%g) = %g\n",x,y);
   }
}
```

SIEHE AUCH

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

NAME

fabs - absolute value function
Absolutbetrag einer Zahl bestimmen

DEFINITION

```
#include <math.h>

double fabs (x)
double x;
```

BESCHREIBUNG

Die Funktion *fabs()* berechnet den Absolutbetrag von x, nämlich $|x|$.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fabs()* den Absolutbetrag von x.

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

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

FEHLER

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

[EDOM] Der Wert von x ist ein NaN.

HINWEIS

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

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

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Absolutbetrag einer eingelesenen Gleitkommazahl berechnen:

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

main()
{
   double x,y;
   if (scanf("%lf",&x) == 1)
   {
      y = fabs(x);
      perror("fabs");
      printf("|%g| = %g\n", x,y);
   }
}
```

SIEHE AUCH

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

NAME

fclose - close a stream
Datenstrom schließen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fclose()* sorgt dafür, daß der Puffer des Datenstroms, auf den *stream* zeigt, geleert und die zugehörige Datei geschlossen wird. Alle noch nicht geschriebenen, gepufferten Daten für diesen Datenstrom werden der Rechnerumgebung zugestellt um in die Datei geschrieben zu werden; alle noch nicht gelesenen, gepufferten Daten werden aufgegeben. Die Zuordnung des Datenstroms zur Datei wird aufgehoben. Sofern der zugehörige Puffer automatisch reserviert wurde, wird er wieder freigegeben. Die Funktion markiert die Felder *st_ctime* und *st_mtime* der zugrundeliegenden Datei zum Ändern, wenn der Datenstrom beschreibbar war und wenn die gepufferten Daten noch nicht in die Datei geschrieben worden sind. Die Funktion *fclose()* führt ein *close()* für die Dateikennzahl aus, die dem Datenstrom *stream* zugeordnet ist.

Wenn die Datei noch nicht EOF erreicht hat und wenn in der Datei positioniert werden kann, dann wird die Position in der Datei bei der zugrundeliegenden offenen Dateibeschreibung so angepaßt, daß die nächste Operation auf dieser offenen Dateibeschreibung mit dem Byte nach dem letzten von einer Schreib- oder Lesoperation des geschlossenen Datenstroms betroffenen Byte arbeitet.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fclose()* den Wert 0. Andernfalls liefert sie EOF und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[EAGAIN]
Das Kennzeichen O_NONBLOCK ist für die *stream* zugrundeliegende Dateikennzahl gesetzt und der Prozeß würde durch die Schreiboperation zum warten gezwungen.

[EBADF]
Die dem Datenstrom zugrundeliegende Dateikennzahl ist ungültig.

[EFBIG] Es wurde ein Versuch unternommen, eine Datei zu beschreiben, deren Größe die Grenze des Prozesses für die Dateigröße oder die maximale Dateigröße überschreitet. Siehe auch *ulimit()*.

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

[ENOSPC]
Auf dem Gerät, auf dem sich die Datei befindet, war kein Platz mehr frei.

[EPIPE] Es wurde der Versuch unternommen, auf eine Pipe oder FIFO zu schreiben, die durch keinen Prozeß zum Lesen geöffnet war. An den Prozeß wird auch das Signal SIGPIPE gesendet.

Auf anderen X/Open-kompatiblen Systemen, die Auftragskontrolle unterstützen, kann zusätzlich noch der folgende Fehler auftreten:

[EIO]
Der Prozeß ist Mitglied einer Hintergrundprozeßgruppe die auf ihr kontrollierendes Terminal schreiben will, TOSTOP ist gesetzt, Das Signal SIGTTOU wird vom Prozeß weder ignoriert noch blockiert und die Prozeßgruppe des Prozesses ist verwaist. Dieser Fehler kann dann auch unter weiteren implementierungsabhängigen Bedingungen auftreten.

PORTABILITÄT

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

BEISPIEL

Legen Sie eine Datei *datei1* an. Wenn Sie dann das übersetzte Programm aufrufen, wird diese Datei vom Programm geöffnet und geschlossen.

```
#include <stdio.h>

main()
{  FILE *dp_1, *dp_s;
   int rc;

   if ((dp_1 =fopen("datei1","r")) == NULL )
   {  perror("fopen");
      exit(1);
   }

   if ((rc = fclose(dp_1)) == -1)
   {  perror("fclose");
      exit(2);
   }
   printf("rc = %d\n",rc);
}
```

SIEHE AUCH

close(), *fopen()*, *<stdio.h>*.

NAME

fcntl - file control
Datei kontrollieren

DEFINITION

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

int fcntl (fildes, cmd, ...)
int fildes, cmd;
```

BESCHREIBUNG

Die Funktion *fcntl()* ermöglicht die Kontrolle offener Dateien. Das Argument *fildes* ist eine Dateikennzahl.

Die möglichen Werte für *cmd* sind in der Include-Datei <*fcntl.h*> definiert. Dazu gehören unter anderem:

F_DUPFD liefert eine neue Dateikennzahl, welche die niedrigste, verfügbare Dateikennzahl größer oder gleich dem dritten Argument *arg*, dargestellt als ein Argument vom Typ *int*, ist. Die neue Dateikennzahl verweist auf dieselbe offene Dateibeschreibung wie die ursprüngliche Dateikennzahl und teilt mit dieser alle Sperren. Das der neuen Dateikennzahl zugeordnete sbe-Bit FD_CLOEXEC wird gelöscht, um die Datei auch bei einem Aufruf einer der *exec*-Funktionen offen zu halten.

F_GETFD Liefert das in der Datei <*fcntl.h*> definierte Prozeß-Dateistatusbyte, das der Dateikennzahl *fildes* zugeordnet ist. Das Dateistatusbyte ist einer einzelnen Dateikennzahl zugeordnet und beeinflußt keine anderen, auf dieselbe Datei verweisenden Dateikennzahlen.

F_SETFD Setzt das in <*fcntl.h*> definierte und zu *fildes* gehörende Prozeß-Dateistatusbyte gleich dem dritten Argument *arg*, das als ein *int* interpretiert wird. Wenn das sbe-Bit FD_CLOEXEC im dritten Argument gleich 0 ist, dann bleibt die Datei auch beim Aufruf einer der *exec*-Funktionen offen; andernfalls wird die Datei bei erfolgreicher Ausführung der *exec*-Funktionen geschlossen.

F_GETFL Liefert das System-Dateistatusbyte und die Zugriffsarten, definiert in *<fcntl.h>*, für die *fildes* zugeordnete Dateibeschreibung. Die Zugriffsarten der Datei können aus dem Ergebnis extrahiert werden, indem die Maske O_ACCMODE verwendet wird, die in *<fcntl.h>* definiert ist. System-Dateistatusbyte und Zugriffsrechte sind der Dateibeschreibung zugeordnet und beeinflussen keine anderen Dateikennzahlen, die derselben Datei, jedoch mit einer anderen offenen Dateibeschreibung, zugeordnet sind.

F_SETFL Setzt das in *<fcntl.h>* definierte System-Dateistatusbyte für die der Dateikennzahl *fildes* zugeordnete Dateibeschreibung, die auf die entsprechenden Bits des dritten Arguments *arg*, das als ein *int* betrachtet wird. Die Zugriffsarten werden nicht geändert. Wenn in *arg* andere Bits gesetzt sind, dann ist das Ergebnis implementierungs-abhängig.

Die folgenden Kommandos stehen für das Sperren von Dateibereichen zur Verfügung. Das Sperren von Dateibereichen wird für normale Dateien unterstützt und kann für andere Dateien unterstützt werden.

F_GETLK Liefert die erste Sperre, die die Sperrenbeschreibung blockiert, auf die das dritte Argument *arg* zeigt. Dieses Argument wird als ein Zeiger auf eine Struktur vom Typ *struct flock* interpretiert (siehe auch *<fcntl.h>*). Die ermittelten Informationen überschreiben die Informationen, die an *fcntl()* in der Struktur *flock* übergeben werden. Wird keine Sperre gefunden, die das Setzen dieser Sperre verhindern könnte, so bleibt die Struktur unververändert, außer daß der Sperrtyp auf F_UNLCK gesetzt wird.

F_SETLK Setzt oder hebt eine Sperre für einen Dateibereich auf, entsprechend der Sperrenbeschreibung, auf die das dritte Argument *arg* zeigt, das als Zeiger vom Typ *(struct flock*)* interpretiert wird (siehe auch *<fcntl.h>*). F_SETLK wird verwendet, um gemeinsame bzw. Lesesperren einzurichten (F_RDLCK), um exklusive bzw. Schreibsperren einzurichten (F_WRLCK), oder um einen beliebigen Sperrtyp aufzuheben (F_UNLCK). F_RDLCK, F_WRLCK und F_UNLCK sind in der Datei *<fcntl.h>* definiert. Wenn eine gemeinsame oder exklusive Sperre nicht eingerichtet werden kann, dann kehrt *fcntl()* sofort mit dem Ergebnis -1 zurück.

F_SETLKW Dieses Kommando leistet dasselbe wie F_SETLK, außer daß der Prozeß solange wartet, bis die Sperre eingerichtet werden kann, wenn eine gemeinsame oder exklusive Sperre wegen anderer Sperren nicht eingerichtet werden kann. Wird während dieses Wartens von *fcntl()* ein Signal empfangen, das abgefangen werden soll, dann wird die Funktion *fcntl()* unterbrochen. Nach der Rückkehr aus der Signalbehandlungsfunktion des Prozesses liefert *fcntl()* den Wert -1, setzt *errno* gleich [EINTR] und führt die Sperroperation nicht aus.

In *<fcntl.h>* können noch weitere, implementierungs-abhängige Kommandos definiert sein. Deren Namen beginnen immer mit F_. Unter SINIX sind dies *F_GETOWN* und *F_SETOWN*.

Wenn eine Lesesperre für einen Dateibereich eingerichtet worden ist, dann können auch andere Prozesse Lesesperren für diesen Bereich oder Teile davon einrichten. Eine Lesesperre verhindert, daß ein anderer Prozeß eine Schreibsperre für irgeneinen beliebigen Teil des geschützten Bereichs einrichtet. Der Versuch, eine Lesesperre einzurichten schlägt fehl, wenn die Dateikennzahl vorher nicht zum Lesen geöffnet wurde.

Eine Schreibsperre verhindert, daß ein anderer Prozeß Schreib- oder Lesesperren für einen beliebigen Teil des geschützen Bereichs einrichtet. Der Versuch, eine Schreibsperre einzurichten schlägt fehl, wenn die Dateikennzahl vorher nicht zum Schreiben geöffnet wurde.

Die Struktur *flock* beschreibt den Typ *(l_type)*, den Ausgangspunkt *(l_whence)*, die relative Anfangsposition *(l_start)*, die Größe *(l_len)* und die Prozeßnummer (*l_pid*) für den zu sperrenden Dateibereich.

Der Wert von *l_whence* ist SEEK_SET, SEEK_CUR oder SEEK_END, je nachdem, ob die relative Anfangsposition *l_start* vom Anfang, von der aktuellen Position oder vom Ende in der Datei an gerechnet werden soll. Der Wert von *l_len* gibt die Anzahl der zu sperrenden, aueinanderfolgenden Bytes an. *l_len* kann negativ sein (wenn die Definition von *off_t* negative Werte für *l_len* erlaubt, wie unter SINIX). Die Strukturkomponetente *l_pid* wird nur für das Kommando F_GETLK verwendet, um die Prozeßnummer des Prozesses zurückzuliefern, der die blockierende Sperre besitzt. Nach einer erfolgreichen Ausführung des Kommandos F_GETLK ist der Wert von *l_whence* gleich SEEK_SET.

Wenn *l_len* positiv ist, dann beginnt der gesperrte Bereich bei *l_start* und endet bei *l_start + l_len-1*. Wenn *l_len* negativ ist, dann beginnt der gesperrte Bereich bei *l_start + l_len* (bzw. bei *l_start - |l_len|*) und endet bei *l_start-1*. Sperren können hinter dem aktuellen Dateiende beginnen oder über dieses hinausgehen, sie dürfen jedoch nicht vor dem Dateianfang beginnen. Eine Sperre kann so gesetzt sein, daß sie immer bis zum größtmöglichen Wert für ihre Ausdehnung reicht, indem *l_len* gleich 0 gesetzt wird. Wenn für solch eine Sperre *l_start* gleich 0 und *l_whence* gleich SEEK_SET gesetzt ist, dann wird die gesamte Datei gesperrt. Das Ändern oder Entsperren eines Teils aus der Mitte eines größeren gesperrten Bereichs hinterläßt jeweils einen kleineren Teil an jedem Ende. Das Sperren eines Bereichs, der bereits vom aufrufenden Prozeß gesperrt wurde, sorgt dafür, daß der alte Sperrtyp durch den neuen ersetzt wird. Alle Sperren, die von einem Prozeß einer Datei zugeordnet sind, werden aufgehoben, wenn die Dateikennzahl für diese Datei geschlossen wird oder sich der betreffende Prozeß beendet. Sperren werden mit der Funktion *fork()* nicht an Sohnprozsse vererbt.

Die Gefahr einer Verklemmung (deadlock) besteht dann, wenn ein Prozeß, der einen gesperrten Bereich kontrolliert, auf die Möglichkeit wartet, einen Bereich zu sperren, der von einem anderen Prozeß gesperrt ist. Wenn das System in solch einem Fall entdeckt, daß das Warten auf die Aufhebung einer eine Verklemmung verursachen würde, dann schlägt die Funktion *fcntl()* fehl, wobei der Fehler [EDEADLK] angezeigt wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion einen Wert, der wie folgt von *cmd* abhängt:

cmd	Ergebnis
F_DUPFD	eine neue Dateikennzahl
F_GETFD	der Wert des Prozeß-Statusbytes, der in <*fcntl.h*> definiert ist. Das Ergebnis ist nicht negtiv.
F_SETFD	ein Wert ungleich -1
F_GETFL	der Wert des System-Statusbytes und der Zugriffsarten. Das Ergebnis ist nicht negativ.
F_SETFL	ein Wert ungleich -1
F_GETLK	ein Wert ungleich -1
F_SETLK	ein Wert ungleich -1
F_SETLKW	ein Wert ungleich -1

Andernfalls wird der Wert -1 zurückgegeben und *errno* ist gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] oder [EAGAIN]
: Das Argument *cmd* ist gleich F_SETLK; der Sperrtyp ist eine Lese- oder Schreibsperre und der zu sperrende Bereich der Datei ist bereits von einem anderen Prozeß mit einer Schreibsperre versehen worden. Oder der Sperrtyp ist eine Schreibsperre und ein Teil des zu sperrenden Bereichs der Datei ist bereits von einem anderen Prozeß mit einer Schreib- oder Lesesperre versehen wurden.

[EBADF]
: Das Argument *fildes* ist keine gültige, offene Dateikennzahl. Oder das Argument *cmd* ist F_SETLK oder F_SETLKW, der Sperrtyp *l_type* ist eine Lesesperre (F_RDLCK) und *fildes* ist keine gültige Dateikennzahl, die zum lesen geöffnet ist. Oder der Sperrtyp *l_type* ist eine Schreibsperre (F_WRLCK) und *fildes* ist keine gültige Dateikennzahl, die zum Schreiben geöffnet ist. Oder es wurde versucht, eine Sperre für eine Datei zu setzen, ohne daß die notwendigen Zugriffsberechtigungen vorhanden sind.

[EINTR] Das Argument *cmd* ist gleich F_SETLKW und die Funktion wurde durch ein Signal unterbrochen.

[EINVAL]
Das Argument *cmd* ist ungültig. Oder das Argument *cmd* ist gleich F_DUPFD und *arg* ist negativ bzw. größer oder gleich {OPEN_MAX}. Oder das Argument *cmd* ist gleich F_GETLK, F_SETLK oder F_SETLKW und die Daten, auf die *arg* zeigt sind ungültig. Oder *fildes* bezieht sich auf eine Datei, für die ein Sperren nicht unterstützt wird.

[EMFILE]
Das Argument *cmd* ist gleich F_DUPFD und {OPEN_MAX} Dateikennzahlen sind bereits für den aufrufenden Prozeß geöffnet, oder es sind keine Dateikennzahlen größer oder gleich *arg* verfügbar.

[ENOLCK]
Das Argument *cmd* ist gleich F_SETLK oder F_SETLKW und durch das Einrichten oder Aufheben dieser Sperre würde die Zahl der gesperrten Bereiche die im System festgelegte Grenze überschreiten.

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

[EDEADLK]
Das Argument *cmd* ist gleich F_SETLKW, das Sperren ist aufgrund einer Sperre eines anderen Prozesses nicht möglich und ein Warten des aufrufenden Prozesses auf die Aufhebung der Sperre würde eine Verklemmung verursachen.

PORTABILITÄT

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

SIEHE AUCH

close(), *exec* , *open()*, *sigaction()*, *<fcntl.h>* , *<sys/types.h>*, *<unistd.h>*.

NAME

fcvt - Gleitkommazahl in ASCII-Zeichenkette (Fortran F Format) umwandeln

DEFINITION

```
char *fcvt (wert, anz, dezpkt, vz)
double wert;
int anz, *dezpkt, *vz;
```

BESCHREIBUNG

Wie *ecvt()* konvertiert *fcvt()* einen Gleitkommawert *wert* zu einer mit einem Nullbyte abgeschlossenen ASCII-Zeichenkette und gibt einen Zeiger darauf zurück. Dabei sorgt *fcvt()* zusätzlich dafür, daß das Ausgabeformat dem Fortran F-Format entspricht. Mit *anz* geben Sie die Anzahl der Ziffern rechts vom Komma in der Ergebniszeichenkette an. Die höchstwertige Ziffer ist ungleich Null, es sei denn, der Wert von *wert* ist Null (0). Die niedrigstwertige Ziffer ist gerundet.

Die Position des Dezimalpunkts relativ zum Anfang der Zeichenkette wird indirekt über *dezpkt* zurückgegeben (ein negativer Wert bedeutet links vom ersten Zeichen der zurückgegebenen Zeichenkette). Der Dezimalpunkt selbst ist nicht in der zurückgegebenen Zeichenkette enthalten.

Das Zeichen für den Dezimalpunkt wird aus der internationalen Umgebung des Programms ermittelt (Kategorie LC_NUMERIC), wenn zuvor ein erfolgreicher Aufruf von *setlocale()* erfolgt ist.

Bei negativem Vorzeichen ist der durch *vz* refenzierte Wert ungleich Null, andernfalls ist er Null.

HINWEIS

Die Rückgabewerte von *ecvt()* und *fcvt()* verweisen auf denselben statischen Datenbereich, dessen Inhalt bei jedem Aufruf überschrieben wird.

BEISPIEL

Siehe das Beispiel bei *ecvt()*

PORTABILITÄT

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

SIEHE AUCH

ecvt(), *gcvt()*, *printf()*.

NAME

fdopen - associate a stream with a file descriptor
Datenstrom zu gegebener Dateikennzahl zuordnen

DEFINITION

```
#include <stdio.h>

FILE *fdopen (fildes, mode)
int fildes;
char *mode;
```

BESCHREIBUNG

Die Funktion *fdopen()* ordnet einer gegebenen Dateikennzahl einen Datenstrom zu.

Das Argument *mode* ist eine Zeichenkette, die eine der folgenden Erscheinungsformen annehmen kann:

"r"	zum Lesen öffnen
"w"	zum Schreiben öffnen
"a"	zum Anfügen öffnen
"r+"	zum Ändern öffnen (lesen und schreiben)
"w+"	zum Ändern öffnen (lesen und schreiben)
"a+"	zum Ändern am Dateiende öffnen (lesen und schreiben)

Die Bedeutung dieser Zeichenketten ist dieselbe, die vom C-Standard für *fopen()* definiert ist, außer daß "w" und "w+" die Datei nicht löschen.

Das Argument *mode* für den Datenstrom darf nur diejenigen Zugriffsarten enthalten, die durch die Zugriffsarten der offenen Datei festgelegt sind. Der Zeiger für die Position des neuen Datenstroms wird auf die Dateiposition gesetzt, die zu der offenen Dateikennzahl gehört.

Die Anzeiger für Fehler und Dateiende des Datenstroms werden gelöscht.

Die Funktion *fdopen()* kann veranlassen, daß das Feld *st_atime* der zugrundeliegenden Datei zum Ändern markiert wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fdopen()* einen Dateizeiger. Andernfalls wird der Nullzeiger zurückgegeben und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

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

[EINVAL]
Das Argument *mode* ist ungültig.

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

[ENOMEM]
Es ist nicht genügend Platz vorhanden, um einen Puffer zu reservieren.

PORTABILITÄT

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

BEISPIEL

Das Beispiel zeigt, wie mit *fdopen()* zu einer Dateikennzahl ein Dateizeiger angelegt werden kann:

```
#include <stdio.h>

main()
{
   FILE *fp, *fdopen();
   int fd;
   char buf[10];
   int c, i, zaehler;

   /* zuerst mit Dateikennzahl arbeiten: */
   if((fd = open("dat",2)) < 0);
   {
      perror("open");
      exit(1);
   }

   if((zaehler = read(fd,buf,10)) > 0)
      write(1,buf,zaehler);

   /* Dateizeiger mit Dateikennzahl verbinden: */
   fp = fdopen(fd,"w");
   if (fp==NULL)
   {  perror("fopen");
      exit(2);
   }

   while((c = getchar()) != EOF)
      putc(c,fp);
   fclose(fp);
}
```

SIEHE AUCH

fclose(), *fopen()*, *open()*, *<stdio.h>*.

NAME

feof - test end-of-file indicator on a stream
Datenstrom auf Dateiende prüfen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *feof()* prüft das Dateiende-Kennzeichen für den Datenstrom, auf den *stream* zeigt.

ERGEBNIS

Die Funktion *feof()* liefert einen Wert ungleich 0 genau dann, wenn das Dateiende-Kennzeichen für *stream* gesetzt ist.

FEHLER

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

[EBADF] Die *stream* zugrundeliegende Dateikennzahl ist ungültig.

PORTABILITÄT

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

SIEHE AUCH

clearerr(), *ferror()*, *<stdio.h>* .

NAME

ferror - test error indicator on a stream
Fehlerkennzeichen eines Datenstroms prüfen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *ferror()* prüft das Fehlerkennzeichen für den Datenstrom, auf den *stream* zeigt.

ERGEBNIS

Die Funktion *ferror()* liefert einen Wert ungleich 0 genau dann, wenn das Fehlerkennzeichen für *stream* gesetzt ist.

FEHLER

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

[EBADF] Die *stream* zugrundeliegende Dateikennzahl ist ungültig

PORTABILITÄT

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

SIEHE AUCH

clearerr(), *feof()*, *fopen()*, *<stdio.h>*.

NAME

fflush - flush a stream
Datenstrom-Puffer bereinigen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Wenn *stream* auf einen Ausgabestrom oder einen Ein- und Ausgabestrom zeigt, dessen letzte Operation nicht das Lesen war, dann sorgt die Funktion *fflush()* dafür, daß alle noch nicht geschriebenen Daten für diesen Datenstrom an die Rechnerumgebung überstellt werden. Dadurch werden sie auf die Datei geschrieben und die Felder *st_ctime* und *st_mtime* der zugrundeliegenden Datei werden zum Ändern markiert.

Wenn der Datenstrom zum Lesen geöffnet ist, dann werden alle noch nicht gelesenen Daten verworfen.

Für einen Datenstom, der zum Lesen geöffnet ist, wird die Dateiposition so angepaßt, daß die nächste Operation auf der offenen Dateibeschreibung genau das Byte betrifft, das auf das letzte gelesene oder geschriebene Byte dieses Datenstroms folgt, sofern noch nicht das Dateiende erreicht ist und in der Datei positioniert werden kann.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fflush()* den Wert 0. Andernfalls liefert sie den Wert EOF und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[EAGAIN]
: Das Kennzeichen O_NONBLOCK ist für die *stream* zugrundeliegende Dateikennzahl gesetzt und der Prozeß würde durch die Schreiboperation angehalten werden.

[EBADF]
: Die *stream* zugrundeliegende Dateikennzahl ist nicht gültig.

[EFBIG] Es wurde versucht, auf eine Datei zu schreiben, deren Größe die Grenze des Prozesses für die Dateigröße oder die maximale Dateigröße überschreitet. Siehe auch *ulimit()*.

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

[ENOSPC] Das Gerät, auf dem sich die Datei befindet verfügt nicht über genügend freien Platz.

[EPIPE] Es wurde der Versuch unternommen, auf eine Pipe oder FIFO zu schreiben, die von keinem Prozeß zum Lesen geöffnet war. Außerdem wird das Signal SIGPIPE an den Prozeß gesendet.

Auf anderen X/Open-kompatiblen Systemen, die Auftragskontrolle unterstützen, kann zusätzlich noch der folgende Fehler auftreten:

[EIO] Der Prozeß ist Mitglied einer Hintergrundprozeßgruppe und will auf deren kontrollierendes Terminal schreiben, TOSTOP ist gesetzt, das Signal SIGTTOU wird vom Prozeß weder ignoriert noch blockiert und die Prozeßgruppe des Prozesses ist verwaist. Dieser Fehler kann dann auch unter weiteren implementierungsabhängigen Bedingungen auftreten.

PORTABILITÄT

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

SIEHE AUCH

<stdio.h>.

NAME

fgetc - get a character from a stream
Zeichen aus Datenstrom lesen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fgetc()* erhält das nächste Zeichen, sofern vorhanden, als ein Zeichen vom Typ *unsigned char*, umgewandelt in den Typ *int* , aus dem Eingabestrom, auf den *stream* zeigt. Sie schaltet den zugehörigen Zeiger auf die Dateiposition des Datenstroms weiter (sofern definiert).

Die Funktion *fgetc()* kann das Feld *st_atime* der *stream* zugeordneten Datei zum Ändern markieren. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung von *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* zum Ändern markiert, die für *stream* verwendet wird und Daten zurückliefert, die nicht durch einen vorangegangen Aufruf von *ungetc()* bereitgestellt wurden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fgetc()* das nächste Zeichen aus dem Eingabestrom, auf den *stream* zeigt. Wenn der Datenstrom das Dateiende erreicht hat, dann wird das Dateiende-Kennzeichen dieses Datenstroms gesetzt und *fgetc()* liefert den Wert EOF. Wenn ein Lesefehler auftritt, dann wird das Fehler-Kennzeichen des Datenstroms gesetzt, *fgetc()* liefert den Wert EOF und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[EAGAIN] Für die *stream* zugrundeliegende Dateikennzahl ist das Kennzeichen O_NONBLOCK gesetzt und der Prozeß würde durch die Operation *fgetc()* angehalten werden.

[EBADF] Die *stream* zugrundeliegende Dateikennzahl ist keine gültige, zum Lesen geöffnete Dateikennzahl.

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

[ENOMEM]
Es ist nicht genügend Speicherplatz vefügbar.

[ENXIO] Die Anforderung wurde für ein nicht existierendes Gerät vorgenommen oder lag außerhalb der Möglichkeiten des Geräts.

In anderen X/Open-kompatiblen Implementierungen können zusätzlich die folgenden Fehler auftreten:

[EINTR] Die Leseoperation wurde durch den Empfang eines Signals beendet und es wurden entweder keine Daten übertragen oder die Implementierung meldet keine teilweise Übertragung für diese Datei.

[EIO] Die Implementierung unterstützt Auftragskontrolle, der Prozeß ist Mitglied einer Hintergrund-Prozeßgruppe und versucht von seinem kontrollierenden Terminal zu lesen, das Signal SIGTTIN wird vom Prozeß entweder blockiert oder ignoriert oder die Prozeßgruppe ist verwaist. Dieser Fehler kann auch aus weiteren, von der Implementierung festgelegten Gründen auftreten.

HINWEIS

Wenn das ganzzahlige Ergebnis von *fgetc()* in einer Variablen vom Typ *char* abgelegt und dann gegen die ganzzahlige Konstante EOF verglichen wird, dann kann es sein, daß dieser Vergleich niemals erfolgreich ist, da die Vorzeichen-Erweiterung eines Zeichens bei der Umwandlung in eine Ganzzahl rechnerabhängig ist. Daher sollte eine portable Anwendung immer eine *int*-Variable für das Ergebnis von *fgetc()* verwenden.

PORTABILITÄT

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

BEISPIEL

Folgendes Programm liest aus einer Reihe (maximal 10) beim Aufruf übergebener Dateien nacheinander jeweils ein Zeichen und gibt es auf die Standardausgabe aus:

```
#include <stdio.h>

int main(argc, argv)
int argc;
char *argv[ ];
{
   FILE *dz[10], **app;
   int c, i ;
   for (i = 1; i < argc; i++)
       if ((dz[i-1]=fopen(argv[i], "r")) == NULL)
       {  perror(argv[i])
          exit(1);
       }
   app = dz;
   while (*app != NULL)
   {   c = fgetc(*app++);
       putchar(c);
   }
   putchar("\n");
   exit(0);
}
```

SIEHE AUCH

getc(), *<stdio.h>*.

NAME

fgetgrent - get group entry
Zeiger auf die nächste Gruppenstruktur

DEFINITION

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

struct group *fgetgrent (f)
FILE *f;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie die Gruppendatei */etc/group* bearbeiten können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getgrent()*.

PORTABILITÄT

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

NAME

fgetpwent - get password entry
Zeiger auf nächste Kennwort-Struktur

DEFINITION

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

struct passwd *fgetpwent (f)
FILE *f;
```

BESCHREIBUNG

Diese Funktion gehört zu einer Gruppe von Funktionen, mit denen Sie die Kennwortdatei */etc/passwd* bearbeiten können.

Die Beschreibung dieser und der anderen Funktionen finden Sie zusammengefaßt unter *getpwent()*.

PORTABILITÄT

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

NAME

fgets - get a string from a stream
Zeichenkette von Datenstrom lesen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fgets()* liest Zeichen von Datenstrom *stream* in den Vektor, auf den *s* zeigt, bis *n*-1 Zeichen gelesen wurden oder bis das Zeichen '\n' gelesen und in *s* eingetragen wurde oder bis das Dateiende erkannt wurde.

Die Funktion *fgets()* kann das Feld *st_atime* der *stream* zugeordneten Datei zum Ändern markieren. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung von *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* zum Ändern markiert, die für *stream* verwendet wird und Daten zurückliefert, die nicht durch einen vorangegangen Aufruf von *ungetc()* bereitgestellt wurden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fgets()* die Adresse *s*. Wenn der Datenstrom das Dateiende erreicht hat, dann wird das Dateiende-Kennzeichen dieses Datenstroms gesetzt und *fgets()* liefert den Nullzeiger. Wenn ein Lesefehler auftritt, dann wird das Fehler-Kennzeichen des Datenstroms gesetzt, *fgets()* liefert den Nullzeiger und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fgetc()*.

PORTABILITÄT

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

SIEHE AUCH

fread(), *gets()*, *<stdio.h>*.

NAME

fileno - map stream pointer to file descriptor
Dateikennzahl abfragen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fileno()* liefert die ganzzahlige Dateikennzahl, die zu dem Datenstrom gehört, auf den *stream* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fileno()* den ganzzahligen Wert der Dateikennzahl, die zu *stream* gehört. Anderfalls wird der Wert -1 geliefert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EBADF] Das Argument *stream* ist kein gültiger Datenstrom.

PORTABILITÄT

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

SIEHE AUCH

fdopen(), *fopen()*, *<stdio.h>*.

NAME

floor - floor function
Abrundungsfunktion

DEFINITION

```
#include <math.h>

double floor (x)
double x;
```

BESCHREIBUNG

Die Funktion *floor()* berechnet den größten ganzzahligen Wert, der nicht größer als *x* ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *floor()* den größten ganzzahligen Wert, der nicht größer als *x* ist, dargestellt als ein Wert vom Typ *double*.

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

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

FEHLER

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

[ERANGE] Das Ergebnis würde einen Überlauf verursachen.

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

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

HINWEIS

Der ganzzahlige, als *double* dargestellte Wert, den *floor()* liefert, kann möglicherweise nicht als *int* oder *long* dargestellt werden. Das Ergebnis sollte vor einer Zuweisung an einen ganzzahligen Typ geprüft werden, um undefinierte Resultate eines ganzzahligen Überlaufs zu vermeiden.

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

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

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

cc programm.c -lm

PORTABILITÄT

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

SIEHE AUCH

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

NAME

fmod - remainder value function
Restfunktion für Gleitpunktzahlen

DEFINITION

```
#include <math.h>

double fmod (x, y)
double x, y;
```

BESCHREIBUNG

Die Funktion *fmod()* liefert den Gleitpunktrest der Division *x/y*.

ERGEBNIS

Die Funktion *fmod()* liefert den Wert *x - i * y*, für eine ganze Zahl *i*, die so aussieht, daß das Vorzeichen des Ergebnisses gleich dem von *x* ist und dessen Betrag kleiner ist als *y*, sofern *y* ungleich 0 ist (Modulo-Operation für Gleitpunktzahlen).

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

Wenn *y* gleich 0 ist, dann wird der Wert 0 zurückgegeben. Unter anderen X/Open-kompatiblen Implementierungen können zusätzlich oder statt dessen eine oder alle der folgenden beiden Aktionen ausgeführt werden:
— *errno* wird gleich [EDOM] gesetzt;
— oder ein NaN wird zurückgegeben.

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

FEHLER

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

[EDOM] Das Argument *y* ist gleich 0 oder eines der Argumente ist ein NaN.

HINWEIS

Eine Anwendung sollte sicherstellen, daß *y* ungleich 0 ist, bevor sie *fmod()* aufruft.

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

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

Außerdem gilt, daß eine portable Anwendung *fmod()* nicht mit dem Argument *y* gleich 0 aufrufen sollte, da das Ergebnis in diesem Fall implementierungs-abhängig ist.

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

cc programm.c -lm

PORTABILITÄT

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

SIEHE AUCH

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

NAME

fopen - open a stream
Datenstrom öffnen

DEFINITION

```
#include <stdio.h>

FILE *fopen (filename, mode)
char *filename, *mode;
```

BESCHREIBUNG

Die Funktion *fopen()* öffnet die Datei, auf deren Namen *filename* zeigt, und ordnet ihr einen Datenstrom zu.

Das Argument *mode* zeigt auf eine Zeichenkette, die mit einer der folgenden Sequenzen beginnt:

"r"	Datei zum Lesen öffnen
"w"	Datei auf die Länge 0 kürzen oder Datei zum Schreiben erzeugen
"a"	anfügen; Datei zum Schreiben am Dateiende öffnen oder erzeugen
"r+"	Datei zum Ändern öffnen (Lesen und Schreiben)
"w+"	Datei auf die Länge 0 kürzen oder Datei zum Schreiben erzeugen
"a+"	anfügen; Datei zum Ändern öffnen oder erzeugen, Schreiben nur am Dateiende

Das Öffnen einer Datei zum Lesen, d.h. mit dem 'r' als ersten Zeichen im Argument *mode*, schlägt fehl, wenn die Datei nicht existiert oder nicht gelesen werden kann.

Das Öffnen einer Datei zum Anfügen, d.h. mit dem 'a' als ersten Zeichen im Argument *mode*, sorgt dafür, daß alle nachfolgenden Schreiboperationen auf die Datei am aktuellen Dateiende erfolgen, ohne daß dabei eventuell zwischendurch erfolgte Aufrufe von Funktion *fseek()* eine Rolle spielen.

Wenn eine Datei zum Ändern geöffnet wurde, d.h. mit dem '+' als zweiten Zeichen im Argument *mode*, dann können auf dem zugeordneten Datenstrom sowohl Ein- als auch Ausgabeoperationen durchgeführt werden. Dennoch darf auf eine Ausgabeoperation nicht unmittelbar eine Eingabeoperation erfolgen, ohne daß inzwischen ein Aufruf der Funktion *fflush()* oder einer der Funktionen zur Positionierung, *fseek()* oder *rewind()*, erfolgt ist.

Auf eine Eingabeoperation darf nicht unmittelbar eine Ausgabeoperation erfolgen, ohne daß zwischendurch eine der Funktionen zur Positionierung augerufen wurde, außer die Eingabeoperation stieß auf das Dateiende.

Wenn dann Datenstrom geöffnet wird, dann wird dieser genau dann voll gepuffert, wenn entschieden werden kann, daß dieser Datenstrom kein interaktives Gerät anspricht. Die Kennzeichen für Dateiende und Fehler dieses Datenstroms werden gelöscht.

Wenn das Argument *mode* gleich "w", "a", "w+" oder "a+" ist, und wenn die Datei vor diesem Aufruf nicht existierte, dann markiert die Funktion *fopen()* bei erfolgreicher Beendigung die Felder *st_atime*, *st_ctime* und *st_mtime* der Datei und die Felder *st_ctime* und *st_mtime* des übergeordneten Dateiverzeichnisses zum Ändern.

Wenn *mode* gleich "w" oder "w+" ist, und wenn die Datei vor dem Aufruf bereits existierte, dann markiert die Funktion *fopen()* bei erfolgreicher Beendigung die Felder *st_ctime* und *st_mtime* der Datei zum Ändern. Die Funktion *fopen()* reserviert eine Dateikennzahl, wie dies die Funktion *open()* macht.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fopen()* einen Zeiger auf das Objekt, das den Datenstrom kontrolliert. Andernfalls wird der Nullzeiger zurückgeliefert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] Die Durchsucherlaubnis für eine Komponete des Pfadnamen-Anfangs wird verweigert. Oder die Datei existiert und für *mode* benötigten Zugriffrechte werden verweigert. Oder die Datei existiert nicht und die Schreiberlaubnis wird für das übergeordnete Dateiverzeichnis der zu erzeugenden Datei verweigert.

[EINTR] Währen der Funktion *fopen()* wurde ein Signal abgefangen.

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

[EISDIR] Die benannte Datei ist ein Dateiverzeichnis und *mode* fordert den Schreibzugriff.

[EMFILE] Für den aufrufenden Prozeß sind bereits {FOPEN_MAX} Dateikennzahlen, Dateiverzeichnisse und Meldungskataloge geöffnet.

[ENAMETOOLONG]
Die Länge von *filename* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX} während {_POSIX_NO_TRUNC} aktiv ist.

[ENFILE] Die Dateitabelle des Systems ist voll.

[ENOENT] Die benannte Datei existiert nicht oder das Argument *filename* zeigt auf die leere Zeichekette.

[ENOSPC] Das Dateiverzeichnis oder das Dateisystem, das die neue Datei aufnehmen würde kann nicht erweitert werden, die Datei existiert nicht und muß erzeugt werden.

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

[ENXIO] Die benannte Datei ist eine Gerätedate für ein zeichen- oder blockorientiertes Gerät und das dieser Datei zugeordnete Gerät existiert nicht.

[EROFS] Die benannte Datei befindet sich in einem, nur zum Lesen eingehängten Dateisystem und *mode* fordert Schreibzugriff.

[ETXTBSY]
Die Datei ist eine reine Prozedurdatei (gemeinsam genutzter Text), die gerade von einem anderen Prozeß ausgeführt wird, und *mode* fordert den Schreibzugriff.

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

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

PORTABILITÄT

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

BEISPIEL

Programm zum Kopieren von *datei1* und *datei2* auf *datei3*:

```
#include <stdio.h>

main()
{  FILE *dp_1, *dp_s;
   void copy();

   if ((dp_1 = fopen("datei1","r"))==NULL
      || (dp_s = fopen("datei3","w")) == NULL)
   {  /* Programmabbruch bei Fehler mit Rueckgabewert 1       */
      perror("fopen");
      exit(1);
   }
   copy(dp_1, dp_s);

   /* Umhaengen des Dateizeigers von datei1 auf datei2 */
   if ((freopen("datei2","r",dp_1)) == NULL)
      /* Programmabbruch bei Fehler */
      /* mit Rueckgabewert 2        */
      exit(2);
   copy(dp_1, dp_s);
   fclose(dp_1); fclose(dp_s);
   /* Korrektes Programmende: */
   exit(0);
}

void copy(dp_1, dp_s)
FILE *dp_1, *dp_s;
{  int c;
   while((c=getc(dp_1)) != EOF)
      putc(c,dp_s);
}
```

SIEHE AUCH

fclose(), *fdopen()*, *freopen()*, *<stdio.h>*.

NAME

fork - create a new process
Neuen Prozeß erzeugen

DEFINITION

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

pid_t fork()
```

BESCHREIBUNG

Die Funktion *fork()* erzeugt einen neuen Prozeß. Der neuen Prozeß, der Sohnprozeß, ist eine exakte Kopie des aufrufenden Prozesses (Vaterprozeß), mit den unten dargestellten Ausnahmen.

Jede der Dateikennzahlen des Sohnprozesses teilt eine gemeinsame, offene Dateibeschreibung mit der entsprechenden Dateikennzahl des Vaterprozesses.

Der Sohnprozeß hat seine eigene Kopie der offenen Dateiverzeichnisströme des Vaterprozesses. Jeder offene Dateiverzeichnisstrom des Sohnprozesses kann die Positionierung im Dateiverzeichnisstrom mit dem entprechenden offenen Dateiverzeichnisstrom des Vaterprozesses teilen.

Der Sohnprozeß besitzt unter SINIX seine eigene Kopie der Meldungskatalog-Deskriptoren des Vaterprozesses.

Der Sohnprozeß unterscheidet sich vom Vaterprozeß in den folgenden Punkten:

- Der Sohnprozeß besitzt eine eindeutige Prozeßnummer.
- Die Prozessnummer des Sohnprozesses entspricht nicht der aktiven Prozeßgruppennummer.
- Der Sohnprözeß besitzt eine anderen Vaterprozeßnummer (d.h. die Prozeßnummer des Vaterprozesses).
- Die Werte des Sohnprozesses für *tms_utime*, *tms_stime*, *tms_cutime* und *tms_cstime* sind gleich 0 gesetzt.
- Die Restzeit bis zu einem Alarmuhr-Signal ist auf 0 zurückgesetzt.

- Alle *semadj*-Werte sind gelöscht.
- Dateisperren des Vaterprozesses werden vom Sohnprozeß nicht geerbt. Siehe auch *fcntl()*.
- Die Menge der für den Sohnprozeß anstehenden Signale wird mit der leeren Menge initialisiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fork()* den Wert 0 an den Sohnprozeß und die Prozeßnummer des Sohnprozesses an den Vaterprozeß. Andernfalls wird der Wert -1 an den Vaterprozeß zurückgeliefert, kein Sohnprozeß erzeugt und *errno* gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[EAGAIN]
: Die vom System festgelgte Grenze für die Anzahl der systemweit oder für eine einzelne Benutzernummer gleichzeitig ablaufenden Prozesse {CHILD_MAX} würde überschritten werden.

Unter anderen X/Open-kompatiblen Implementierungen kann die Funktion *fork()* zusätzlich auch fehlschlagen, wenn gilt:

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

PORTABILITÄT

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

BEISPIEL

Das folgende Beispiel zeigt eine typische Aufruffolge von *fork* und *exec*:
Zuerst wird *fork* aufgerufen, um einen neuen Prozeß zu erzeugen, dann ruft der neue Sohnprozeß mit *exec* ein Programm auf (*sh -c "echo Sohn"*), das den aufrufenden Sohnprozeß überlagert:

```
#include <stdio.h>

main()
{
   switch (fork())
   {
     case 0:                                  /* Sohn   */
        execl("/bin/sh","sh","-c","echo Sohn",NULL);
        break;
     case -1:                                 /* Fehler */
        perror("fork");
        exit(1);
     default:                                 /* Vater  */
        printf("Vaterprozess\n");
        break;
   }
}
```

SIEHE AUCH

alarm(), *exec*, *fcntl()*, *semop()*, *signal()*, *times()*, *<sys/types.h>*.

NAME

fpathconf - get configurable pathname variables
Konfigurierbare Pfadnamen-Variablen ermitteln

DEFINITION

```
#include <unistd.h>

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

BESCHREIBUNG

Siehe unter *pathconf()*.

PORTABILITÄT

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

NAME

fprintf - print formatted output
Formatierte Ausgabe

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Siehe unter *printf()*.

PORTABILITÄT

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

BEISPIEL

Folgendes Programmschema gibt in jedem Schleifendurchlauf die Werte der Variablen *var1*, *var2* und *var3* in eine Datei *protokoll* aus:

```
#include <stdio.h>

main()
{
   int i, var1;
   char *var2;
   double var3;
   FILE *dz;

   dz = fopen("protokoll","w");
      /* Datei protokoll zum Schreiben oeffnen */

   if (dz==NULL)
   {
      perror("fopen");
      exit(1)
   }

   var1 = 123;
   var2 = "var2 ist diese Zeichenkette";
   var3 = 50e20;

   for( i = 1; i < 11; i++ )
   {
      fprintf(dz,"\nWerte beim %d-ten Lauf: \n\n", i);
      fprintf(dz,"1.Variable : %d ", (var1 += i));
      fprintf(dz,"2.Variable : %s ", var2);
      fprintf(dz,"3.Variable : %f \n", var3);
   }
}
```

NAME

fputc - put character on a stream
Zeichen auf Datenstrom ausgeben

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fputc()* schreibt das durch *c* angegeben Zeichen, umgewandelt in *unsigned char*, auf den Ausgabestrom, auf den *stream* zeigt. Die Ausgabe erfolgt an der Position, die durch den zugehörigen Zeiger auf die Dateiposition, sofern definiert, angegeben wird und erhöht diese Zeiger entsprechend. Wenn die Datei Positionierungs-Anforderungen nicht unterstützen kann, oder der Datenstrom zum Anfügen geöffnet wurde, dann wird das Zeichen an den Ausgabestrom angefügt.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *fputc()* und dem nächsten erfolgreichen Aufruf von *fflush()* oder *fclose()* für denselben Datenstrom oder einem Aufruf von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fputc()* den geschriebenen Wert. Andernfalls liefert sie EOF und setzt das Fehlerkennzeichen für den Datenstrom und *errno*, um den Fehler anzuzeigen.

FEHLER

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

[EAGAIN]
: Das Kennzeichen O_NONBLOCK ist für die *stream* zugrundeliegende Dateikennzahl gesetzt und der Prozeß würde durch die Schreiboperation angehalten werden.

[EBADF]
: Die *stream* zugrundeliegende Dateikennzahl ist keine gültige, zum Schreiben geöffnete Dateikennzahl.

[EFBIG] Es wurde versucht, auf eine Datei zu schreiben, deren Größe die Grenze des Prozesses für die Dateigröße oder die maximale Dateigröße überschreitet. Siehe auch *ulimit()*.

[EINTR] Die Schreiboperation wurde durch den Empfang eines Signals beendet und es wurden entweder keine Daten übertragen oder es wird keine teilweise Übertragung für diese Datei gemeldet.

[ENOSPC] Das Gerät, auf dem sich die Datei befindet verfügt nicht über genügend freien Platz.

[ENXIO] Die Anforderung fand für ein nicht vorhandenes Gerät statt oder lag außerhalb der Fähigkeiten des Geräts.

[EPIPE] Es wurde der Versuch unternommen, auf eine Pipe oder FIFO zu schreiben, die von keinem Prozeß zum Lesen geöffnet war. Außerdem wird das Signal SIGPIPE an den Prozeß gesendet.

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

[ENOMEM] Es ist nicht genügend Speicherplatz vorhanden.

Auf anderen X/Open-kompatiblen Systemen, die Auftragskontrolle unterstützen, kann zusätzlich noch der folgende Fehler auftreten:

[EIO] Der Prozeß ist ein Mitglied einer Hintergrund-Prozeßgruppe und versucht, auf sein kontrollierendes Terminal zu schreiben TOSTOP ist gesetzt, Das Signal SIGTTOU wird vom Prozeß weder blockiert noch ignoriert und die Prozeßgruppe des Prozesses ist verwaist. Dieser Fehler kann auch dort unter anderen, von der Implementierung definierten Bedingungen auftreten.

PORTABILITÄT

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

SIEHE AUCH

ferror(), *fopen()*, *putc()*, *puts()*, *setbuf()*, *<stdio.h>*.

NAME

fputs - put a string on a stream
Zeichenkette auf Datenstrom ausgeben

DEFINITION

```
#include <stdio.h>

int fputs (s, stream)
char *s;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *fputs()* schreibt die, mit dem Nullbyte abgeschlossene Zeichenkette, auf die *s* zeigt, auf den Datenstrom, auf den *stream* zeigt. Das abschließende Nullbyte wird nicht geschrieben.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *fputs()* und dem nächsten Ende eines Aufrufs von *fflush()* oder *fclose()* für den selben Datenstrom oder einem Aufruf von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fputs()* eine nichtnegative ganze Zahl. Andernfalls liefert sie EOF und setzt das Fehlerkennzeichen für den Datenstrom und *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fputc()*.

HINWEIS

Die Funktion *puts()* fügt ein Neue-Zeile-Zeichen an, die Funktion *fputs()* nicht.

PORTABILITÄT

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

BEISPIEL

Zeichenketten von Standardeingabe einlesen und auf *datei* ausgeben:

```
#include <stdio.h>

main()
{
   FILE *fp;
   char s[BUFSIZ];

   if ((fp = fopen("datei","w")) == NULL)
   {
      perror("fopen");
      exit(10);
   }

   while (gets(s) != NULL)
   {  fputs(s,fp);        /* Zeichenkette ausgeben */
      fputc('\n', fp);    /* Zeilenvorschub ausgeben */
   }
}
```

SIEHE AUCH

fopen(), *putc()*, *puts()*, *<stdio.h>*.

NAME

fread - binary input
Binäre Eingabe

DEFINITION

```
#include <stdio.h>

size_t fread (ptr, size, nitems, stream)
void *ptr;
size_t size, nitems;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *fread()* liest *nitems* Elemente der Größe *size* aus dem Datenstrom, auf den *stream* zeigt, in den Vektor, auf den *ptr* zeigt. Der Zeiger auf die Position des Datenstroms in der Datei wird, wenn er definiert ist, um die Anzahl von Bytes weitergeschaltet, die erfolgreich gelesen wurden. Wenn ein Fehler auftritt, dann ist der sich daraus ergebende Wert des Zeigers auf die Dateiposition unbestimmt. Wenn ein Element teilweise gelesen wird, dann ist sein Wert unbestimmt.

Die Funktion *fread()* kann das Feld *st_atime* der *stream* zugeordneten Datei zum Ändern markieren. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung von *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* zum Ändern markiert, die für *stream* verwendet wird und Daten zurückliefert, die nicht durch einen vorangegangen Aufruf von *ungetc()* bereitgestellt wurden.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fread()* die Anzahl der erfolgreich gelesenen Elemente. Diese ist nur dann kleiner als *nitems*, wenn das Dateiende erreicht wird, oder wenn ein Lesefehler auftritt. Andernfalls, wenn ein Lesefehler auftritt, wird das Fehlerkennzeichen für den Datenstrom gesetzt und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fgetc()*.

HINWEIS

Die Funktionen *ferror()* oder *feof()* müssen verwendet werden, um zwischen dem Auftreten des Dateiendes und eines Lesefehlers zu unterscheiden.

PORTABILITÄT

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

BEISPIEL

Folgendes Programmstück liest sieben Personeneinträge aus der Datei mit Dateizeiger *p_liste*:

```
 /* Eintraege im aktuellen Dateiverzeichnis interaktiv löschen */

#include <stdio.h>

struct person
{
   short    pnummer;
   char     abt[10]
   char     name[14];
   unsigned tel;
} pe[7];

FILE *p_liste;

main()
{   int i;

    if ((p_liste = fopen("pers.dat","r")) == NULL)
       /* Datei zum Lesen oeffnen */
    {  perror("pers.dat");
       exit(1);
    }

    if ((i=fread((void *)pe,sizeof(struct person),7,p_liste)) != 7)
       fprintf(stderr, "Es wurden nur %d Einträge gelesen\n", i);
}
```

SIEHE AUCH

feof(), *ferror()*, *fopen()*, *getc()*, *gets()*, *scanf()*, *<stdio.h>*.

NAME

free - free allocated memory
Reservierten Speicher freigeben

DEFINITION

```
#include <stdlib.h>

void free (ptr)
void *ptr;
```

BESCHREIBUNG

Die Funktion *free()* sorgt dafür, daß der Speicherplatz, auf den *ptr* zeigt, freigegeben wird. Das heißt, er wird für zukünfige Reservierung wieder verfügbar gemacht. Wenn *ptr* gleich dem Nullzeiger ist, dann wird keine Aktion ausgeführt. Andernfalls, wenn das Argument keine Adresse ist, die vorher von einer der Funktionen *calloc()*, *malloc()* oder *realloc()* zurückgeliefert wurde, oder wenn der Speicherplatz bereits vorher durch einen Aufruf von *free()* oder *realloc()* freigegeben wurde, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *free()* liefert kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

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

PORTABILITÄT

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

SIEHE AUCH

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

NAME

freopen - open a stream
Datenstrom öffnen

DEFINITION

```
#include <stdio.h>

FILE *freopen (filename, mode, stream)
char *filename, *mode;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *freopen()* versucht zuerst, für den *stream* zugeordneten Datenstrom, den Puffer zu leeren und die zugehörige Dateikennzahl zu schließen. Ein Fehler beim Leerens des Puffers oder beim Schließen der Datei wird ignoriert. Die Kennzeichen für Fehler oder Dateiende des Datenstroms werden gelöscht.

Die Funktion *freopen()* öffnet dann die Datei, deren Name durch die Zeichenkette gegeben ist, auf die *filename* zeigt, und ordnet dieser den durch *stream* angegebenen Datenstrom zu. Das Argument *mode* wird genauso verwendet, wie bei der Funktion *fopen()*.

Der ursprüngliche Datenstrom wird geschlossen, ohne Rücksicht darauf, ob das nachfolgende Öffnen erfolgreich ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *freopen()* den Wert von *stream*. Andernfalls wird der Nullzeiger zurückgegeben und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] Die Durchsucherlaubnis für eine Komponete des Pfadnamen-Anfangs wird verweigert. Oder die Datei existiert und die für *mode* benötigten Zugriffrechte werden verweigert. Oder die Datei existiert nicht und die Schreiberlaubnis wird für das übergeordnete Dateiverzeichnis der zu erzeugenden Datei verweigert.

[EINTR] Während der Funktion *freopen()* wurde ein Signal abgefangen.

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

[EISDIR] Die angegebene Datei ist ein Dateiverzeichnis und *mode* fordert den Schreibzugriff.

[EMFILE] Für den aufrufenden Prozeß sind bereits {FOPEN_MAX} Dateikennzahlen, Dateiverzeichnisse und Meldungskataloge geöffnet.

[ENAMETOOLONG]
Die Länge von *filename* überschreitet {PATH_MAX} oder eine Komponente des Pfadnamens ist länger als {NAME_MAX} während {_POSIX_NO_TRUNC} aktiv ist.

[ENFILE] Die Dateitabelle des Systems ist voll.

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

[ENOSPC] Das Dateiverzeichnis oder das Dateisystem, das die neue Datei aufnehmen würde kann nicht erweitert werden, die Datei existiert nicht und muß erzeugt werden.

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

[ENXIO] Die benannte Datei ist eine Gerätedate für ein zeichen- oder blockorientiertes Gerät und das dieser Datei zugeordnete Gerät existiert nicht.

[EROFS] Die benannte Datei befindet sich in einem, nur zum Lesen eingehängten Dateisystem und *mode* fordert Schreibzugriff.

[ETXTBSY]
Die Datei ist eine reine Prozedurdatei (gemeinsam genutzter Text), die gerade von einem anderen Prozeß ausgeführt wird, und *mode* fordert den Schreibzugriff.

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

[ENOMEM] Es ist nicht genügend Speicherplatz verfügbar.

HINWEIS

Die Funktion *freopen()* wird typischerweise dazu verwendet, die standardmäßig geöffneten Datenströme, die *stdin*, *stdout* und *stderr* zugeordnet sind, anderen Dateien zuzuordnen.

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *fopen()*

SIEHE AUCH

fclose(), *fdopen()*, *fopen()*, *<stdio.h>*.

NAME

frexp - extract mantissa und exponent from double precision number
Mantisse und Exponent einer doppelt genauen Gleitpunktzahl ermitteln

DEFINITION

```
#include <math.h>

double frexp (value, exp)
double value;
int *exp;
```

BESCHREIBUNG

Die Funktion *frexp()* zerlegt eine Gleitpunktzahl in eine normalisierte Mantisse und einen ganzzahligen Exponenten von 2. Der ganzzahlige Exponent wird in der *int*-Variablen abgelegt, auf die das Argument *exp* zeigt.

ERGEBNIS

Die Funktion *frexp()* liefert den Wert x, so daß x vom Typ *double* aus dem Intervall [0.5 , 1) oder gleich 0 ist und *value* entspricht $x * 2^{*exp}$.

Wenn *value* gleich 0 ist, dann sind beide Teile des Ergebnisses gleich 0.

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

Wenn *value* glaich HUGE_VAL ist, dann wird der Wert HUGE_VAL zurückgeliefert.

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

FEHLER

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

[EDOM] Der Wert von *value* ist ein NaN oder eine unendliche Größe.

HINWEIS

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

Unter Fehlerbedingung wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* geschrieben.

Hier kann die Fehlerbehandlung durch die Bereitstellung einer *matherr()*-Funktion geändert werden (siehe auch *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

Normierte Darstellung der Zahl 5 zur Basis 2:

```
#include <stdio.h>

double frexp();
int exp;

main()
{
   double z;
   z = frexp((double)5,&exp);
   printf("5 = %g *2 ** %d\n",z,exp);
}
```

SIEHE AUCH

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

NAME

fscanf - convert formatted input
Formatierte Eingaben umwandeln

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Siehe unter *scanf()*.

PORTABILITÄT

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

NAME

fseek - reposition a file-position indicator in a stream
Positionszeiger eines Datenstroms neu positionieren

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *fseek()* setzt den Zeiger für die Dateiposition des durch *stream* angegebenen Datenstroms.

Die neue Position, gemessen in Bytes vom Anfang der Datei, wird durch die Addition von *offset* zu der durch *whence* angegebenen Position ermittelt. Der angegebene Punkt ist der Dateianfang bei SEEK_SET, der aktuelle Wert des Positionszeigers bei SEEK_CUR oder das Dateiende bei SEEK_END.

Ein erfolgreicher Aufruf der Funktion *fseek()* löscht das Kennzeichen für Dateiende und macht alle Wirkungen der Funktion *ungetc()* für denselben Datenstrom rückgängig. Nach einem Aufruf von *fseek()* kann die nächste Operation auf einem zum Ändern geöffneten Datenstrom sowohl eine Schreib- als auch eine Leseoperation sein.

Wenn die letzte Operation ungleich *ftell()* auf einem gegebenen Datenstrom die Operation *fflush()* ist, dann wird die Dateiposition der zugrundeliegenden Dateibeschreibung angepaßt, um den durch *fseek()* vorgegebenen Ort wiederzuspiegeln.

Die Funktion *fseek()* erlaubt, daß der Zeiger für die Dateiposition hinter das Ende der existierenden Daten in der Datei gesetzt wird. Wenn später Daten an diese Stelle geschrieben werden, dann liefern anschließende Leseoperationen für Daten in dieser Lücke Nullbytes, bis wirklich Daten in diese Lücke geschrieben wurden.

Die Funktion *fseek()* erweitert nicht selbst die Größe einer Datei. Das Verhalten von *fseek()* auf Geräten, die nicht zur Positionierung fähig sind, wird von der jeweiligen Implementierung und dem jeweiligen Gerät festgelegt. Der Wert der Dateiposition einer solch einem Gerät zugeordneten Datei ist undefiniert.

Wenn der Datenstrom zum Schreiben geöffnet ist und gepufferte Daten noch nicht auf die zugrundeliegende Datei geschrieben wurden, dann sorgt *fseek()* dafür, daß die noch nicht geschriebenen Daten auf die Datei geschrieben werden und markiert die Felder *st_ctime* und *st_mtime* der Datei zum Ändern.

ERGEBNIS

Die Funktion *fseek()* liefert den Wert 0 bei Erfolg; andernfalls liefert sie den Wert -1 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

[EBADF]
: Die *stream* zugrundeliegende Dateikennzahl ist nicht zum Schreiben geöffnet und es muß der Puffer geleert werden.

[EFBIG]
: Es wurde versucht, auf eine Datei zu schreiben, deren Größe die Grenze des Prozesses für die Dateigröße oder die maximale Dateigröße überschreitet. Siehe auch *ulimit()*.

[EINVAL]
: Das Argument *whence* ist ungültig.

[ENOSPC]
: Das Gerät, auf dem sich die Datei befindet verfügt nicht über genügend freien Platz.

[ESPIPE]
: Die *stream* zugrundeliegende Dateikennzahl ist einer Pipe oder FIFO zugeordnet.

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

[EAGAIN]
: Für die *stream* zugrundeliegende Dateikennzahl ist das Kennzeichen O_NONBLOCK gesetzt und der Prozeß würde durch die Schreiboperation angehalten werden.

[EINTR] Die Schreiboperation wurde durch den Empfang eines Signals beendet und es wurden entweder keine Daten übertragen oder die Implementierung meldet keine teilweise Übertragung für diese Datei.

[EINVAL] Der berechnete Zeiger für die Dateiposition würde einen negativen Wert annehmen.

[EIO] Ein Ein- oder Ausgabefehler ist aufgetreten.

[EPIPE] Es wurde der Versuch unternommen, auf eine Pipe oder FIFO zu schreiben, die von keinem Prozeß zum Lesen geöffnet war. Außerdem wird das Signal SIGPIPE an den Prozeß gesendet.

HINWEIS

Das Argument *offset* muß unbedingt vom Typ *long* sein. Ein *fseek()* in einer Anwendung, die auf Systeme portiert wird, bei denen *long* und *int* verschiedene Größen haben, könnte sonst undefinierte Ergebnisse zur Folge haben.

PORTABILITÄT

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

BEISPIEL

datei ab dem elften Zeichen bis zum Dateiende lesen:

```
#include <stdio.h>

main()
{
   FILE *fp;
   int c;

   if ((fp = fopen("datei","r")) != NULL)
   {
      /* die ersten 10 Zeichen ueberspringen */
      fseek(fp,10L,SEEK_SET);
      while ((c = getc(fp)) != EOF)
          putc(c,stdout);
      fclose(fp);
   }
}
```

SIEHE AUCH

fopen(), *ftell()*, *rewind()*, *ungetc()*, *<stdio.h>*.

NAME

fstat - get file status
Dateiinformation ermitteln

DEFINITION

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

int fstat (fildes, buf)
int fildes;
struct stat *buf;
```

BESCHREIBUNG

die Funktion *fstat()* ermittelt Informationen über die offene Datei, der die Dateikennzahl *fildes* zugeordnet ist.

Das Argument *buf* wird als Zeiger auf eine Struktur vom Typ *struct stat* interpretiert, so wie in der Include-Datei *<sys/stat.h>* definiert, in welche die Dateiinformationen abgelegt werden.

Andere X/Open-kompatible Implementierungen, die erweiterte Sicherheitskontrollen anbieten, können die Funktion *fstat()* unter implementierungsabhängigen Bedingungen zum Scheitern bringen.

Die Funktion *fstat()* ändert gegebenenfalls alle zeitbezogenen Felder, wie im Glossar unter **Dateizeiten-Änderung** beschrieben, bevor sie die Struktur *stat* füllt.

ERGEBNIS

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

FEHLER

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

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

PORTABILITÄT

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

SIEHE AUCH

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

NAME

fstatfs - get file system information
Informationen zum Dateisystem ermitteln

DEFINITION

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

int fstatfs(fildes, buf, len, fstyp)
int fildes;
struct statfs *buf;
int len, fstyp;
```

BESCHREIBUNG

Die Funktion *fstatfs()* liefert einen sogenannten *generischen Superblock*, der ein Dateisystem beschreibt. Eine ausführliche Beschreibung finden Sie unter *statfs()*.

PORTABILITÄT

Diese Funktion ist im X/Open-Standard (Ausgabe 3) nicht enthalten.

NAME

fsync - synchronise a file's state
Dateizustand synchronisieren

DEFINITION

```
int fsync(fildes)
int fildes;
```

BESCHREIBUNG

Die Funktion *fsync()* bewirkt, daß alle geänderten Daten und Attribute der durch *fildes* angegebenen Datei an die zugrundeliegende Hardware übergeben wird.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *fsync()* den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

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

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

[EINTR] Die Funktion *fsync()* wurde von einem Signal unterbrochen.

[EINVAL] Das Argument *fildes* bezieht sich nicht auf eine Datei, für die diese Operation möglich ist.

[EIO] Ein Ein- oder Ausgabefehler trat beim Lesen bzw. Schreiben der Datei auf.

HINWEIS

Die Funktion *fsync()* sollte von Programmen verwendet werden, die einen eindeutigen Zustand einer Datei benötigen. So kann zum Beispiel ein Programm, das einfache Transaktions-Fähigkeiten besitzt, diese Funktion verwenden, um sicherzustellen, daß alle Änderungen auf einer Datei oder auf Dateien, die von einer Transaktion verursacht wurden, auch wirklich vorgenommen wurden.

PORTABILITÄT

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

NAME

ftell - return a file offset in a stream
Position im Datenstrom liefern

DEFINITION

```
#include <stdio.h>

long ftell (stream)
FILE *stream;
```

BESCHREIBUNG

Die Funktion *ftell()* ermittelt den derzeitigen Wert des Zeigers auf die Dateiposition für den Datenstrom, auf den *stream* zeigt.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *ftell()* den aktuellen Wert des Zeigers auf die Position der Datei für den Datenstrom, angegeben in Bytes vom Anfang der Datei.

Andernfalls liefert die Funktion *ftell()* den Wert -1L und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fseek()*, mit Ausnahme des Fehlers EINVAL, der nicht auftritt.

PORTABILITÄT

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

BEISPIEL

Ab dem elften Zeichen wird jedes weitere von *datei* mit der Position des Lese/Schreibzeigers ausgegeben:

```
#include <stdio.h>

main()
{
   FILE *fp;
   int c;
   if ((fp = fopen("datei","r")) != NULL)
   {
      /* die ersten 10 Zeichen werden uebersprungen */
      fseek(fp,10L,0);
      while((c=getc(fp)) != EOF)
          printf("%Position: %ld, Zeichen : %c\n",ftell(fp),c);
      fclose(fp);
   }
}
```

SIEHE AUCH

fopen(), *fseek()*, *lseek()*, *<stdio.h>*.

NAME

ftok - Standard Interprozess-Kommunikationspaket

DEFINITION

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

key_t fotk (path, id)
char * path;
char id;
```

BESCHREIBUNG

Alle Interprozess-Kommunikationsanwendungen erwarten, daß der Anwender einen Schlüssel unterstützt, der von den Systemaufrufen *msgget()*, *semget()*, *shmget()* benutzt wird, um einen Bezeichner für die Interprozess-Kommunikation zu erhalten. Eine vorgeschlagene Methode einen Schlüssel zu bilden ist, die Funktion *ftok()* zu benutzen, die anschließend beschrieben wird. Eine andere Möglichkeit einen Schlüssel zu bilden ist, das höchstwertige Byte mit einem Projektbezeichner zu belegen und den restlichen Teil als eine Sequenznummer herzunehmen. Es gibt viele andere Möglichkeiten, einen Schlüssel zu bilden, aber es ist dann für jeden Rechner nötig nach einem Standard vorzugehen. Wenn nicht nach einem Standard vorgegangen wird, können fremde Prozesse unabsichtlich mit Operationen eines anderen Prozesses kollidieren. Deshalb wird eindringlich vorgeschlagen, daß sich das höchstwertige Byte eines Schlüssels in irgendeiner Weise auf ein Projekt bezieht, so daß es zu keinen Schlüsselkollisionen auf einem Rechner kommt.

ftok() liefert als Rückgabewert einen Schlüssel, der auf den Argumenten *path* und *id* basiert und der in nachfolgenden Aufrufen von den Systemaufrufen *msgget()*, *semget()* und *shmget()* verwendet werden kann. *path* muß der Pfadname einer existierenden Datei sein, für die der aufrufende Prozeß Zugriffsrechte hat. *id* ist ein Zeichen, das ein Projekt eindeutig kennzeichnet. Achtung: *fotk()* liefert den gleichen Schlüssel für Dateien, für die ein Querverweis besteht, wenn für *id* das gleiche Zeichen verwendet wird. Allerdings wird für das gleiche Argument *path* zusammen mit einem unterschiedlichem Zeichen für das Argument *id* ein unterschiedlicher Schlüssel erzeugt.

ERGEBNIS

ftok() gibt den Wert (key_t) -1 zurück falls *path* nicht existiert oder für den Prozess keine Zugriffsrechte bestehen.

HINWEIS

Achtung

Falls die Datei, deren Pfadname an *ftok* übergeben wurde, gelöscht wird solange noch Schlüssel-Referenzen auf die Datei bestehen, ergeben nachfolgende Aufrufe an *ftok()* mit denselben Argumenten *path* und *id* einen Fehlerwert zurück. Falls die Datei mit dem gleichen Namen nochmals erzeugt wird, gibt *ftok()* normalerweise einen anderen Schlüssel zurück, als bei dem früheren Aufruf mit denselben Argumenten *path* und *id*.

PORTABILITÄT

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

SIEHE AUCH

msgget(), *semget()*, *shmget()*, *<sys/ipc.h>*, *<sys/types.h>*.

NAME

ftw - traverse (walk) a file tree
Dateibaum durchlaufen

DEFINITION

```
#include <ftw.h>

int ftw (path, fn, ndirs)
char *path;
int (*fn)();
int ndirs;
```

BESCHREIBUNG

Die Funktion *ftw()* wandert rekursiv durch die Dateiverzeichnishierarchie hinab, die bei *path* beginnt. Für jedes Objekt der Hierarchie ruft *ftw()* die Funktion auf, auf die *fn* zeigt, und übergibt ihr einen Zeiger auf eine, mit dem Nullbyte abgeschlossene Zeichenkette, die den Namen des Objekts enthält, einen Zeiger auf eine Struktur vom Typ *struct stat* (siehe auch *<sys/stat.h>*), die Informationen über das Objekt enthält, und eine ganze Zahl. Mögliche Werte für die ganze Zahl, sind die in der Include-Datei *<ftw.h>* definierten Werte:

FTW_F	für eine Datei,
FTW_D	für ein Dateiverzeichnis
FTW_DNR	Für ein Dateiverzeichnis, das nicht gelesen werden kann
FTW_NS	für ein Objekt, für das *stat()* nicht erfolgreich ausgeführt werden konnte.

Wenn die ganze Zahl gleich FTW_DNR ist, dann werden Unterbäume dieses Dateiverzeichnisses nicht bearbeitet. Wenn die ganze Zahl gleich FTW_NS ist, dann enthält die Struktur *stat* undefinierte Werte. Ein Beispiel für ein Objekt, für das FTW_NS an die Funktion, auf die *fn* zeigt, übergeben werden würde, ist eine Datei in einem Dateiverzeichnis mit Lese- aber ohne Sucherlaubnis.

Die Funktion *ftw()* durchläuft zuerst ein Dateiverzeichnis, bevor einer seiner Nachfolger bearbeitet wird.

Die Baumdurchquerung hält solange an, bis der Baum vollständig durchsucht ist, ein Aufruf von *fn* einen Wert ungleich 0 zurückgibt oder in *ftw* ein Fehler entdeckt wird.

Das Argument *ndirs* gibt die maximale Anzahl von Dateiverzeichnisströmen und/oder Dateikennzahlen an, die für die Verwendung durch *ftw()* bei der Bearbeitung des Dateibaums zur Verfügung stehen. Wenm *ftw()* zurückkehrt, dann schließt sie alle Dateiverzeichnisströme und Dateikennzahlen, die sie verwendet hat, ohne dabei diejenigen zu berücksichtigen, die durch die Funktion *fn()* des Benutzers geöffnet wurden.

ERGEBNIS

Wenn der Dateibaum abgearbeitet ist, dann liefert *ftw()* den Wert 0. Wenn die Funktion, auf die *fn* zeigt, einen Wert ungleich 0 zurückliefert, dann beendet *ftw()* die Abarbeitung des Dateibaums und liefert des Ergebnis der Funktion zurück, auf die *fn* zeigt. Entdeckt *ftw()* einen Fehler, so wird -1 zurückgegeben und *errno* besetzt, um den Fehler anzuzeigen.

FEHLER

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

[EACCES] Die Durchsucherlaubnis für eine Komponente von *path* oder die Leseerlaubnis für *path* wird verweigert.

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

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

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

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

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

Zusätzlich kann, wenn die Funktion, auf die *fn* zeigt, auf Systemfehler trifft, *errno* entsprechend besetzt sein.

HINWEIS

Da *ftw()* rekursiv ist, kann diese Funktion mit einem Speicherfehler abbrechen, wenn sie auf sehr tiefe Dateibäume angewendet wird.

Die Funktion *ftw()* verwendet *malloc()*, um während ihres Ablaufs dynamisch Speicherplatz zu reservieren. Wenn *ftw()* zum Abbruch gezwungen wird, wie zum Beispiel durch *longjmp()* oder *siglongjmp()*, ausgeführt von der Funktion, auf die *fn* zeigt, oder aus einer Signalbehandlungsroutine heraus, dann hat *ftw()* keine Möglichkeit, diesen Speicher freizugeben, so daß dieser ständig reserviert bleibt. Ein sicherer Weg, Unterbrechungen zu behandeln ist der, sich das Auftreten der Unterbrechung zu merken und die Funktion, auf die *fn* zeigt, zu veranlassen, bei ihrem nächsten Aufruf einen Wert ungleich 0 zurückzuliefern.

PORTABILITÄT

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

BEISPIEL

Das Programm gibt alle im Baum vorhandenen Dateien und Dateiverzeichnisse aus:

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

main (argc, argv)
int argc;
char **argv;
{  int nach(), t = OPEN_MAX;

   ftw(argv[1], nach, t);
}

nach(s, sptr, i)
char *s;
struct stat *sptr;
int i;
{
   printf ("i ist %d\n", i);
   switch(i)
   {
       case FTW_D   : printf("%s ist ein DVZ\n", s);
                      break;
       case FTW_F   : printf("%s ist eine Datei\n", s);
                      break;
       case FTW_DNR : printf("%s ist ein nicht lesbares DVZ\n", s);
                      break;
       case FTW_NS  : printf("%s: Datei nicht existent oder stat nicht ausfuehrbar\n", s
                      break;
       default      : printf("%s ist unbekannt\n", s);
   }
   return (0);
}
```

SIEHE AUCH

longjmp(), *malloc()*, *siglongjmp()*, *stat()*, *<ftw.h>*, *<sys/stat.h>*.

NAME

fwrite - binary output
Binäre Ausgabe

DEFINITION

```
#include <stdio.h>

size_t fwrite (ptr, size, nitems, stream)
void *ptr;
size_t size, nitems;
FILE *stream;
```

BESCHREIBUNG

Die Funktion *fwrite()* schreibt *nitems* Elemente der Größe *size* aus dem Vektor, auf den *ptr* zeigt, auf den Datenstrom, auf den *stream* zeigt. Der Zeiger auf die Position des Datenstroms in der Datei wird, wenn er definiert ist, um die Anzahl von Bytes weitergeschaltet, die erfolgreich geschrieben wurden. Wenn ein Fehler auftritt, dann ist der sich daraus ergebende Wert des Zeigers auf die Dateiposition unbestimmt.

Die Felder *st_ctime* und *st_mtime* der Datei werden zwischen der erfolgreichen Ausführung von *fwrite()* und der nächsten erfolgreichen Beendigung eines Aufrufs von *fflush()* oder *fclose()* für denselben Datenstrom oder einem Aufruf von *exit()* oder *abort()* zum Ändern markiert.

ERGEBNIS

Die Funktion *fwrite()* liefert die Anzahl der erfolgreich geschriebenen Elemente. Diese kann dann kleiner als *nitems* sein wenn ein Schreibfehler auftritt. Wenn *size* oder *nitems* gleich 0 ist, dann liefert *fwrite()* den Wert 0 und der Inhalt des Vektors wie auch der Zustand des Datenstroms bleiben unverändert. Andernfalls, wenn ein Schreibfehler auftritt, wird das Fehlerkennzeichen für den Datenstrom gesetzt und *errno* wird besetzt, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fputc()* .

PORTABILITÄT

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

BEISPIEL

Folgendes Programmstück überträgt zwei Personeneinträge auf die Datei mit Dateizeiger *p_liste*:

```
#include <stdio.h>

main ()
{
   int gesch;
   FILE *p_liste,*fopen();
   struct eintrag
   {
     int Klasse;
     int alter;
   } person[2];
   person[0].Klasse = 2;
   person[0].alter = 11;
   person[1].Klasse = 55;
   person[1].alter = 35;

   if ((p_liste = fopen("pdat","w")) == NULL)
   {
      perror("fopen");
      exit(1);
   }
   printf ("Da bin ich:\n");
   gesch = fwrite ((void *) person,sizeof(struct eintrag),(size_t) 2,p_liste);
   printf ("%d Elemente geschrieben.\n",gesch);
}
```

SIEHE AUCH

ferror(), *fopen()*, *printf()*, *putc()*, *puts()*, *write()*, *<stdio.h>*.

NAME

gamma, signgam - log gamma function
logarithmische Gamma-Funktion

DEFINITION

```
#include <math.h>

double gamma (x)
double x;

extern int signgam;
```

BESCHREIBUNG

Die Funktion *gamma()* berechnet ln (|Γ(x)|), wobei Γ(*x*) als

$$\int_0^\infty e^{-t}\, t^{x-1}\, dt.$$

definiert ist. Das Vorzeichen von Γ(x) wird in der externen Variablen *signgam* zurückgegeben. Das Argument *x* darf keine negative ganze Zahl sein (Γ(x) ist für alle reele Zahlen definiert, außer für die negativen ganzen Zahlen).

BESCHREIBUNG

Bei erfolgreicher Beendigung liefert *gamma()* das logarithmische Gamma von *x* als Ergebnis zurück.

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

Falls x eine negative ganze Zahl ist, wird HUGE_VAL zurückgegeben und *errno* wird gleich [EDOM] gesetzt. Andere X/Open-kompatble Implementierungen können hier auch ein NaN zurück und *errno* gleich [EDOM] setzen.

Falls der korrekte Wert einen Überlauf erzeugen würde, liefert *gamma()* den Wert HUGE_VAL und setzt *errno* auf [ERANGE].

Andernfalls wird entweder 0 zurückgegeben und *errrno* gesetzt, um den Fehler anzuzeigen oder es wird ein NaN zurückgegeben und *errno* kann gesetzt sein, um den Fehler anzuzeigen.

FEHLER

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

[EDOM] Der Wert von *x* ist eine negative ganze Zahl.

[ERANGE] Das Ergebnis würde einen Überlauf verursachen.

HINWEIS

Eine Anwendung, die Fehlersituationen portabel überprüfen will, sollte *errno* gleich 0 setzen, bevor sie *gamma()* aufruft. Falls *errno* nach Aufruf der Funktion gesetzt oder das Ergebnis ein NaN ist, ist ein Fehler aufgetreten.

Im Fehlerfall wird bei SINIX und einigen anderen Implementierungen eine Fehlermeldung auf *stderr* ausgegeben. Hier kann die Fehlerbehandlung geändert werden, indem eine *matherr()*-Funktion bereitgestellt wird, (siehe *matherr()*).

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

cc programm.c -lm

PORTABILITÄT

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

SIEHE AUCH

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

NAME

gcvt - Gleitkommazahl in Zeichenkette umwandeln

DEFINITION

```
char *gcvt (wert, anz, puf)
double wert;
int anz;
char *puf;
```

BESCHREIBUNG

gcvt() konvertiert *wert* zu einer mit dem Nullzeichen abgeschlossenen ASCII-Zeichenkette und legt diese in dem durch *puf* referenzierten Zeichenvektor ab; der Zeiger auf *puf* wird zurückgegeben.

Wenn möglich, werden *anz* signifikante Stellen im FORTRAN F-Format erzeugt, andernfalls wird Fortran E-Format für Druckausgabe erzeugt. Ein eventuell vorhandenes Minuszeichen wie auch ein Dezimalpunkt sind in der zurückgegebenen Zeichenkette enthalten. Abschließende Nullen werden unterdrückt.

Das Zeichen für den Dezimalpunkt wird aus der internationalen Umgebung des Programms ermittelt (Kategoric LC_NUMERIC), wenn zuvor ein erfolgreicher Aufruf von *setlocale()* erfolgt ist.

BEISPIEL

Siehe Beispiel bei *ecvt()*.

PORTABILITÄT

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

SIEHE AUCH

ecvt(), *fcvt()*, *printf()*.

NAME

getc - get character from a stream
Zeichen von Datenstrom lesen

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *getc()* ist äquivalent zu *fgetc()*, außer daß sie, wenn sie als Makro definiert ist, das Argument *stream* öfter als einmal auswertet. Daher sollte dieses Argument niemals ein Ausdruck mit Seiteneffekten sein.

ERGEBNIS

Siehe unter *fgetc()*.

FEHLER

Siehe unter *fgetc()*.

HINWEIS

Wenn das ganzzahlige Ergebnis von *getc()* in einer Variablen vom Typ *char* abgelegt und dann gegen die ganzzahlige Konstante EOF verglichen wird, dann kann es unter verschiedenen anderen Systemen sein, daß dieser Vergleich niemals erfolgreich ist, da die Vorzeichen-Erweiterung eines Zeichens bei der Umwandlung in eine Ganzzahl rechnerabhängig ist. Portable Anwendungen sollten daher darauf achten, daß das Ergebnis von *getc()* stets in einer Variablen des Typs *int* abgelegt wird.

Da *getc()* als Makro implementiert sein kann, kann ein Argument *stream* mit Seiteneffekten inkorrekt behandelt werden. So wird zum Beispiel der Ausdruck **f++* im Aufruf *getc(*f++)* mehrfach ausgewertet! Statt dessen sollte die Funktion *fgetc()* benutzt werden.

BEISPIEL

Programmstück für zeichenweises Einlesen aus der Datei mit Dateizeiger *eingabe*, bis Datei- oder Pufferende erreicht ist:

```
#include <stdio.h>

.
.
int c, i=0;
char buf[4096];
FILE *eingabe;
.
.
if ((eingabe=fopen("eingabe","r")) != NULL)
   while ((c=getc(eingabe)) != EOF  &&  i < 4096)
      buf[i++] = c;
```

PORTABILITÄT

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

SIEHE AUCH

fgetc(), *<stdio.h>*.

NAME

endcfent, getcfadent, getcfadnam, getcfgrent, getcfgrnam, getcfprent, getcfprnam, getcftyent, getcftynam, setcfadent, setcfgrent, setcfprent, setcftyent - Funktionen zur C-Schnittstelle des Druckspools, Druckerbeschreibung

DEFINITION

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

struct dlist *getcfprnam(name)
char *name
        /* Druckername, entspricht ddnam in struct dlist */

struct glist *getcfgrnam(name)
char *name
        /* Name der Druckergruppe, entspricht gnam in struct glist */

struct alist *getcfadnam(name)
char *name
        /* Name des Drucker-Verwalters, entspricht anam */
        /*  in struct alist */

struct tlist *getcftynam(name)
char *name
        /* Name des Bildschirms bzw. Benutzers, entspricht tnam */
        /*  in struct tlist */

struct dlist *getcfprent()
struct glist *getcfgrent()
struct alist *getcfadent()
struct tlist *getcftyent()

void setcfprent()
void setcfgrent()
void setcfadent()
void setcftyent()

void endcfent()
```

BESCHREIBUNG

getcfprnam(), *getcfgrnam()*, *getcfadnam()* und *getcftynam()* liefern je einen Zeiger auf eine der Strukturen *alist*, *glist*, *dlist* und *tlist*, die die Angaben je einer Zeile aus der Datei CONFIG.bin enthalten:

```
struct dlist                /* Struktur eines konfigurierten Druckers */
       char *ddnam;           /* Druckername */
       char *dexec;           /* Name des Backend-Aufrufes */
       char *dfile;           /* Name der Drucker-Geraetedatei */
       long dmask;            /* interne Druckernummer */
       char *dnull;           /* Nullzeiger */
       struct dlist  *dnext;  /* Zeiger auf naechstes Listen-Element */

struct glist                /* Struktur einer konfigurierten Druckergruppe */
       char *gnam;            /* Name der Druckergruppe */
       char *gdnam;           /* Namen der in dieser Gruppe konfigurierten Drucker */
       char *gcomm;           /* Kommentar, der sich auf diese Gruppe bezieht */
       long gdmask;           /* Maske fuer in dieser Gruppe enthaltene Drucker */
       struct flag  *gflags;  /* Zeiger auf Tabelle mit Backend-Schaltern */
       char *gnull;           /* Nullzeiger */
       struct glist  *gnext;  /* Zeiger auf naechstes Listen-Element */

struct alist                /* Struktur eines konfigurierten Drucker-Verwalters */
       char *anam;            /* Name des Drucker-Verwalters */
       char *adnam;           /* Namen der verwalteten Drucker */
       long *admask;          /* Maske fuer hier enthaltene Drucker */
       char *anull;           /* Nullzeiger */
       struct alist  *anext;  /* Zeiger auf naechstes Listen-Element */

struct tlist                /* Struktur einer Bildschirm- oder Benutzerzuordnung */
       char *tnam;            /* Bildschirm- oder Benutzername */
       char *tgnam;           /* Name der zugehoerigen Druckergruppe */
       char *tnull;           /* Nullzeiger */
       struct tlist  *tnext;  /* Zeiger auf naechstes Listen-Element */

struct flag                 /* Struktur eines Backend-Schalters */
       char *flagstr;         /* Name des Backend-Schalters */
       char flagtype;         /* Typ des Schalters */
       char para;             /* Parameterflag, Schalter mit oder ohne */
                              /*  Parameter erwartet */
       char fevent;           /* Ereignis, das aus dem Aufruf des */
                              /*  Schalters zu folgen hat */
```

CONFIG.bin ist eine statische Datei. Sowohl *getcfprnam()*, *getcfgrnam()*, *getcfadnam()* als auch *getcftynam()* suchen ab Anfang der jeweils zutreffenden Struktur in CONFIG.bin, bis ein passender Name gefunden wird oder EOF erreicht ist.

getcfprent(), *getcfgrent()*, *getcfadent()* und *getcftyent()* lesen die nächste Zeile in der jeweils zutreffenden Struktur, so daß aufeinanderfolgende Aufrufe dazu benutzt werden können, um die gesamte Datei zu durchsuchen, bis ein passender Eintrag gefunden wird oder EOF erreicht ist.

Ein Aufruf von *setcfprent()*, *setcfgrent()*, *setcfadent()* oder *setcftyent()* hat den Effekt, daß der Zeiger wieder auf den Anfang des Strukturfeldes positioniert und damit wiederholtes Suchen ermöglicht wird.

endcfent() kann aufgerufen werden, sobald die Bearbeitung abgeschlossen ist, um den durch CONFIG.bin belegten Speicher wieder freizugeben.

ERGEBNIS

Im Fehlerfall oder bei EOF wird ein Nullzeiger zurückgeliefert.

PORTABILITÄT

Alle aufgeführten Funktionen sind im X/Open-Standard nicht enthalten.

SIEHE AUCH

getpdjbent()-getpdprnam(), *<sys/lpr.h>*.

NAME

getchar - get character from *stdin* stream
Zeichen von Standardeingabe lesen

DEFINITION

```
#include <stdio.h>

int getchar()
```

BESCHREIBUNG

Der Funktionsaufruf *getchar()* ist äquivalent zu *getc(stdin)*.

ERGEBNIS

Siehe unter *fgetc()*.

FEHLER

Siehe unter *fgetc()*.

HINWEIS

Wenn das ganzzahlige Ergebnis von *getchar()* in einer Variablen vom Typ *char* abgelegt und dann gegen die ganzzahlige Konstante EOF verglichen wird, dann kann es sein, daß dieser Vergleich niemals erfolgreich ist, da die Vorzeichen-Erweiterung eines Zeichens bei der Umwandlung in eine Ganzzahl rechnerabhängig ist.

PORTABILITÄT

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

SIEHE AUCH

getc(), *<stdio.h>*.

NAME

getcwd - get pathname of current working directory
aktuelles Dateiverzeichnis ermitteln

DEFINITION

```
char *getcwd (buf, size)
char *buf;
int size;
```

BESCHREIBUNG

Die Funktion *getcwd()* legt den absoluten Pfadnamen des aktuellen Dateiverzeichnisses in dem Vektor ab, auf den *buf* zeigt, und liefert *buf* zurück. Das Argument *size* sollte die Größe des Vektors, auf den das Argument *buf* zeigt, in Bytes sein.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *getcwd()* einen Zeiger auf eine Zeichenkette, die den absuluten Pfadnamen des aktuellen Dateiverzeichnisses enthält. Andernfalls liefert *getcwd()* den Nullzeiger und besetzt *errno*, wenn *size* nicht groß genug ist oder wenn ein interner Fehler auftritt. Der Inhalt des Vektors, auf den *buf* zeigt, ist dann undefiniert.

FEHLER

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

[EACCES] Die Lese- oder Durchsucherlaubnis für eine Komponente des Pfadnamens wurde verweigert.

[EINVAL] Das Argument *size* ist gleich 0 oder negativ.

[ERANGE]
Das Argument *size* ist größer als 0, aber kleiner als die Länge des Pfadnamens + 1.

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

[ENOMEM]
Es steht nicht genügend Speicherplatz zur Verfügung.

PORTABILITÄT

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

NAME

getegid - get effective group ID
Effektive Gruppennummer ermitteln

DEFINITION

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

gid_t getegid()
```

BESCHREIBUNG

Die Funktion *getegid()* liefert die effektive Gruppennummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *getegid()* ist immer erfolgreich und kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

getgid(), *setgid()*, *<sys/types.h>*.

NAME

getenv - get value of environment variable
Umgebungsvariable ermitteln

DEFINITION

```
#include <stdlib.h>

char *getenv (name)
char *name;
```

BESCHREIBUNG

Die Funktion *getenv()* sucht in der Liste der Umgebungsvariablen nach einer Zeichenkette der Form *"name=wert"*, und liefert einen Zeiger auf die Zeichenkette, die *wert* für den angegebenen Namen enthält. Andernfalls wird der Nullzeiger zurückgeliefert.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *getenv()* einen Zeiger auf ein Zeichenkette, die den *wert* für das angegeben Argument *name* enthält oder den Nullzeiger, wenn *name* nicht gefunden werden kann. Bei Mißerfolg wird der Nullzeiger zurückgeliefert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

PORTABILITÄTSHINWEIS

Unter anderen X/Open-kompatiblen Implementierungen kann das Ergebnis von *getenv()* auf statische Daten zeigen und daher bei jedem Aufruf überschrieben werden.

BEISPIEL

Das folgende Programm liefert Ihnen den Wert der als Argument angegebenen Umgebungsvariablen. Wenn Sie es z.B. mit *a.out USER* aufrufen, dann liefert es Ihnen den aktuellen Benutzernamen:

```
#include <stdio.h>

main (argc, argv)
int argc;
char **argv;
{
   if (argc < 2)
      exit(3);
   if ((getenv(argv[1])) != NULL)
      printf ("%s = %s\n", argv[1], getenv(argv[1]));
   else
      printf ("%s : nicht vorhanden\n", argv[1]);
}
```

SIEHE AUCH

exec, putenv(), <stdlib.h>. Abschnitt 2., Umgebungsvariablen.

NAME

geteuid - get effective user ID
Effektive Benutzernummer ermitteln

DEFINITION

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

uid_t geteuid()
```

BESCHREIBUNG

Die Funktion *geteuid()* liefert die effektive Benutzernummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *geteuid()* ist immer erfolgreich und es ist kein Ergebnis für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *getuid()*.

SIEHE AUCH

getuid(), *setuid()*, *<sys/types.h>*.

NAME

getgid - get real group ID
Reale Gruppennummer ermitteln

DEFINITION

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

gid_t getgid()
```

BESCHREIBUNG

Die Funktion *getgid()* liefert die reale Gruppennummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktin *getgid()* ist immer erfolgreich und es ist kein Ergebnis für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *getuid()*.

SIEHE AUCH

getuid(), *setgid()*, *<sys/types.h>*.

NAME

endgrent, fgetgrent, getgrent, getgrgid, getgrnam, setgrent - Eintrag aus Gruppendatei abfragen

DEFINITION

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

struct group *getgrent ()

void setgrent ()

void endgrent ()

struct group *fgetgrent (f)
FILE *f;
```

BESCHREIBUNG

getgrent(), gibt Zeiger auf *struct group* zurück; diese Struktur enthält Einzelelemente einer Zeile der Gruppendatei.

```
struct group {
    char    *gr_name;
    char    *gr_passwd;
    int     gr_gid;
    char    **gr_mem;
    };
```

Die Komponenten dieser Struktur sind:

gr_name	Gruppenkennung
gr_passwd	Verschlüsseltes Kennwort der Gruppe
gr_gid	Gruppennummer
gr_mem	Mit dem Nullzeiger abgeschlossener Vektor von Zeigern auf die einzelnen Benutzer in dieser Gruppe

getgrent() liest die nächste Zeile in der Datei, so daß aufeinanderfolgende Aufrufe zum Durchsuchen der gesamten Datei benutzt werden können.

Ein Aufruf von *setgrent()* bewirkt das Rücksetzen der Gruppendatei auf Dateianfang, um so wiederholte Suche zu ermöglichen. Der Aufruf von *endgrent()* kann zum Schließen der Gruppendatei verwendet werden, wenn die Dateibearbeitung abgeschlossen ist.

fgetgrent() gibt einen Zeiger auf die nächste *group* Struktur in der Datei *f* zurück; diese Datei muß das Format von */etc/group* haben.

ERGEBNIS

getgrent() gibt einen Zeiger auf eine Struktur vom Typ *struct group* zurück.

Bei Dateiende oder einem auftretenden Fehler wird von dieser Funktion der NULL-Zeiger zurückgegeben.

Die Funktionen *setgrent()* und *endgrent()* haben kein Ergebnis.

HINWEIS

Die gesamte Information wird in einem statischen Bereich abgelegt, so daß sie gegebenenfalls kopiert werden muß.

Es gibt keine befriedigende Möglichkeit, in */etc/group* ein Kennwort einzutragen. Die Verwendung von Kennwörtern für Gruppen wird nicht empfohlen, da sie naturgemäß nur schwachen Schutz bieten. Wahrscheinlich wird es sie zukünftig nicht mehr geben.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard nicht mehr enthalten.

BEISPIEL

Folgendes Beispiel gibt die Einträge von */etc/group* mit entsprechenden Kommentaren aus:

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

struct group *getgrent();
struct group *k;

main ()
{
   int i;
   while ((k = getgrent()) != NULL)
   {
       printf ("Name: %s\n", k->gr_name);
       printf ("Passwort: %s\n", k->gr_passwd);
       printf ("Gruppennummer: %d\n", k->gr_gid);
       i=0;
       while (k->gr_mem[i] != NULL)
           printf ("Mitglied: %s\n", k->gr_mem[i++]);
   }
   endgrent ();
}
```

SIEHE AUCH

getgrgit(), *getlogin()*, *getgrnam()*, *getpwent()*, *<grp.h>*.

NAME

getgrgid - get group database entry for particular group ID
Eintrag für Gruppennummer aus Gruppendatei ermitteln

DEFINITION

```
#include <grp.h>

struct group * getgrgid(gid)
gid_t gid;
```

BESCHREIBUNG

Die Funktion *getgrgid()* sucht in der Gruppendatei nach einem Eintrag, dessen Komponente *gr_gid* zu *gid* paßt.

ERGEBNIS

Die Funktion *getgrgid()* liefert einen Zeiger auf eine *struct group*, so wie in *<grp.h>* definiert, mit einem passenden Eintrag, sofern einer gefunden wird. Sie liefert den Nullzeiger im Fehlerfall, oder wenn der gesuchte Eintrag nicht gefunden wird.

FEHLER

Die Funktion *getgrgid()* kann in folgenden und weiteren, konfigurierungsabhängigen Fällen fehlschlagen, wenn gilt:

[EIO] Ein Ein- oder Ausgabefehler ist aufgetreten.

[EINTR] Während des Ablaufs der Funktion *getgrgid()* wurde ein Signal abgefangen.

[EMFILE] Für den aktuellen Prozeß sind derzeit zu viele Dateikennzahlen offen.

[ENFILE] Die Dateitabelle des Systems ist voll.

HINWEIS

Das Ergebnis kann auf einen statischen Bereich zeigen, der von jedem nachfolgenden Aufruf von *getgrgid()* oder *getgrnam()* überschrieben wird.

PORTABILITÄT

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

SIEHE AUCH

getgrent(), *getgrnam()*, *<grp.h>*.

NAME

getgrnam - search group database for particular name
Eintrag für Gruppenname aus Gruppendatei ermitteln

DEFINITION

```
#include <grp.h>

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

BESCHREIBUNG

Die Funktion *getgrnam()* sucht in der Gruppendatei nach einem Eintrag, dessen Komponente *gr_name* zu *name* paßt.

ERGEBNIS

Die Funktion *getgrnam()* liefert einen Zeiger auf eine *struct group*, so wie in *<grp.h>* definiert, mit einem passenden Eintrag, sofern einer gefunden wird. Sie liefert den Nullzeiger im Fehlerfall, oder wenn der gesuchte Eintrag nicht gefunden wird.

FEHLER

Die Funktion *getgrnam()* kann in folgenden und in weiteren, konfigurierungsabhängigen Fällen fehlschlagen, wenn gilt:

[EIO] Ein Ein- oder Ausgabefehler ist aufgetreten.

[EINTR] Während des Ablaufs der Funktion *getgrnam()* wurde ein Signal abgefangen.

[EMFILE] Für den aktuellen Prozeß sind derzeit zu viele Dateikennzahlen offen.

[ENFILE] Die Dateitabelle des Systems ist voll.

HINWEIS

Das Ergebnis kann auf einen statischen Bereich zeigen, der von jedem nachfolgenden Aufruf von *getgrgid()* oder *getgrnam()* überschrieben wird.

PORTABILITÄT

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

SIEHE AUCH

getgrgid(), *getgrent()*, *<grp.h>*.

NAME

getgroups - get supplementary group IDs
weitere Gruppennummern lesen

DEFINITION

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

int getgroups (gidsetsize, grouplist)
int gidsetsize;
gid_t grouplist[];
```

BESCHREIBUNG

Die Funktion *getgroups()* füllt den Vektor *grouplist* mit den aktuell definierten weiteren Gruppennummern des aufrufenden Prozesses.

Das Argument *gidsetsize* gibt die Anzahl der Elemente im Vektor *grouplist* an. Die aktuelle Anzahl von weiteren Gruppennummern wird als Ergebnis zurückgegeben. Die Werte von Vektorelementen mit Indizes größer oder gleich dem Funktionsergebnis sind undefiniert.

Falls *gidsetsize* gleich 0 ist, liefert die Funktion *getgroups()* die Anzahl der weiteren Gruppennummern, die dem aufrufenden Prozeß zugeordnet sind, ohne den durch *grouplist* gegebenen Vektor zu verändern.

Es ist nicht festgelegt, ob die effektive Gruppennummer des aufrufenden Prozesses in die zurückgegebene Liste weiterer Gruppennummern aufgenommen wird oder nicht.

ERGEBNIS

Bei erfolgreicher Beendigung wird die Anzahl der weiteren Gruppennummern als Ergebnis zurückgeliefert. Dieser Wert ist gleich 0, wenn {NGROUPS_MAX} gleich 0 ist. Ein Ergebnis von -1 zeigt einen Fehler an und *errno* ist dann gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[EINVAL] Das Argument *gidsetsize* ist ungleich 0 und kleiner als die Anzahl der weiteren Gruppennummern.

PORTABILITÄT

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

SIEHE AUCH

getegid(), *setgid()*, *<sys/types.h>*.

NAME

gethz - get frequency
Taktfrequenz ermitteln

DEFINITION

```
int gethz()
```

BESCHREIBUNG

Die Funktion *gethz()* ermittelt die Taktfrequenz des Rechners in CLK_TCK Einheiten je Sekunde. Dazu durchsucht die Funktion in ihrer derzeitigen Implementierung die Umgebung des aufrufenden Prozesses nach einer Zeichenkette der Form HZ=*wert* und liefert den Wert von *wert*.

ERGEBNIS

Bei Erfolg liefert die Funktion *gethz()* die Taktfrequenz des Rechners in CLK_TCK Einheiten je Sekunde. Existiert keine Umgebungsvariable mit dem Namen HZ oder kann *wert* nicht als ganze Zahl interpretiert werden, dann liefert die Funktion *gethz()* den Wert 0.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

<environ.h>.

NAME

getlogin - get login name
Benutzernamen ermitteln

DEFINITION

```
char *getlogin()
```

BESCHREIBUNG

Die Funktion *getlogin()* liefert einen Zeiger auf den Benutzernamen, der durch das Anmelden dem kontrollierenden Terminal des aktuellen Prozesses zugeordnet wurde. Wenn *getlogin()* einen Zeiger ungleich dem Nullzeiger zurückliefert, dann zeigt dieser Zeiger auf den Namen, unter dem sich der Benutzer angemeldet hat, selbst wenn es mehrere Benutzernamen mit der selben Benutzernummer gibt.

Wenn *getlogin()* aus einem Prozeß heraus aufgerufen wird, dessen Benutzername nicht herausgefunden werden kann, dann liefert sie den Nullzeiger.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *getlogin()* einen Zeiger auf den Benutzernamen. Andernfalls wird der Nullzeiger zurückgeliefert und *errno* wird gesetzt, um den Fehler anzuzeigen.

FEHLER

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

[EMFILE] Es werden von diesem Prozeß zu viele Dateikennzahlen verwendet.

[ENFILE] Die Dateitabelle des Systems ist voll.

[ENXIO] Der aufrufende Prozeß besitzt kein kontrollierendes Terminal.

HINWEIS

Das Ergebnis zeigt normalerweise auf statische Daten, deren Inhalt von jedem Aufruf überschrieben werden. Eine portable Anwendung sollte daher den Benutzernamen umspeichern, wenn dieser über einen weiteren Aufruf der Funktion hinaus benötigt wird.

Drei, zum aktuellen Prozeß gehörende Namen können bestimmt werden: *cuserid()* liefert den Namen, der der effektiven Benutzernummer des Prozesses zugeordnet ist; *getlogin()* liefert den Namen, der den aktuellen Anmeldungs-Aktivitäten zugeordnet ist und *getpwuid* (*getuid()*) liefert den Namen, der zur realen Benutzernummer des Prozesses gehört.

PORTABILITÄT

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

SIEHE AUCH

cuserid(), *getpwnam()*, *getpwuid()*, *getuid()*.

NAME

getopt, optarg, opterr, optind - get option character from argument vector
Optionen im Argumentvektor ermitteln

DEFINITION

```
#include <stdio.h>

int getopt (argc, argv, optstring)
int argc;
char **argv, *optstring;

extern char *optarg;
extern int optind, opterr;
```

BESCHREIBUNG

Die Funktion *getopt()* ist ein Parser für die Aufrufzeile. Sie liefert das nächste Optionszeichen in *argv*, das einem Optionszeichen in *optstring* entspricht. Obwohl die Funktion *getopt()* es erlaubt, beliebige Zeichen für Optionen zu verwenden, sollten doch nur Zeichen aus ISO 8859-1:1987 verwendet werden, um ein Höchstmaß an Portabilität zu erzielen.

Das Argument *argv* zeigt auf einen Vektor von *argc+1* Elementen, der *argc* Zeiger auf Zeichenketten, gefolgt vom Nullzeiger enthält.

Das Argument *optstring* zeigt auf eine Zeichenkette von Optionszeichen; wenn ein Optionszeichen von einem Doppelpunkt gefolgt wird, dann wird für diese Option ein Argument erwartet, das von der Option durch ein Leerzeichen getrennt sein kann oder nicht. Die externe Variable *optarg* wird gesetzt, um nach der Rückkehr von *getopt()* auf das Optionsargument zu zeigen.

Die Funktion *getopt()* trägt in *optind* den Index in *argv* des nächsten zu verarbeitenden Arguments ein. Das System initialisiert die externe Variable *optind* vor dem ersten Aufruf von *getopt()* mit dem Wert 1.

Sobald alle Optionen verarbeitet wurden (d.h. bis zum ersten Argument, das keine Option ist), liefert *getopt()* den Wert EOF. Die spezielle Option -- kann dazu verwendet werden, das Ende der Optionen zu kennzeichnen; EOF wird zurückgegeben und -- wird ausgelassen.

Die Funktion *getopt()* verarbeitet Ziffern gemäß der internationalen Umgebung des Programms (Kategorie LC_CTYPE), wenn zuvor ein erfolgreicher Aufruf der Funktion *setlocale()* erfolgt ist. Ansonsten richtet sich die Zeichenklassifizierung nach der Umgebungsvariablen LANG. Ist diese nicht definiert, dann finden die Regeln des US-ASCII-Zeichensatzes Anwendung.

ERGEBNIS

Die Funktion *getopt()* gibt eine Fehlermeldung auf *stderr* aus und liefert das Fragezeichen '?', wenn sie auf ein Optionszeichen trifft, das nicht in *optstring* enthalten ist. Diese Fehlermeldung kann durch das Setzen von *opterr* auf den Wert 0 ausgeschaltet werden. Andernfalls liefert sie das gefundene Optionszeichen. Die Sprache der Fehlermeldung ist von der Umgebungsvariablen LANG abhängig. Dies funktioniert nur dann korrekt, wenn vorher die Funktion *setlocale()* aufgerufen wurde.

FEHLER

Es sind keine Fehler definiert.

BEISPIEL

Das folgende Programmstück zeigt, wie die Argumente für ein Kommando bearbeitet werden können, das zwei exklusive Optionen *-a* und *-b* kennt, und die Optionen *-f* und *-o*, die beide Argumente benötigen:

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

main (argc, argv)
int argc;
char **argv;
{
    int c;
    int aflg, bflg, errflg;
    char *ifile = NULL;
    char *ofile = NULL;
    extern char *optarg;
    extern int optind;
    .
    .
    .
```

```
    aflg = bflg = errflg = 0;
    while ((c = getopt(argc, argv, "abf:o:")) != EOF)
            switch (c) {
            case 'a':
                    if (bflg)
                            errflg++;
                    else
                            aflg++;
                    break;
            case 'b':
                    if (aflg)
                            errflg++;
                    else
                            bflg++;
                    break;
            case 'f':
                    ifile = optarg;
                    break;
            case 'o':
                    ofile = optarg;
                    break;
            case '?':
                    errflg++;
                    break;
            }
    if (errflg) {
            fprintf(stderr, "usage: . . . ");
            exit(2);
    }
    for ( ; optind < argc; optind++) {
            if (access(argv[optind], R_OK)) {
    .
    .
    .
}
```

HINWEIS

Die Zeichen '-' und '?' sollten nicht für Optionen verwendet werden, da sie in dieser Funktion eine spezielle Bedeutung haben.

PORTABILITÄT

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

SIEHE AUCH

<stdio.h>.

NAME

getpass - read a string of characters without echo
Zeichenkette ohne Echo lesen

DEFINITION

```
char *getpass (prompt)
char *prompt;
```

BESCHREIBUNG

Die Funktion *getpass()* öffnet die Datei */dev/tty* (d.h. die aktuelle Datensichtstation), schreibt die mit dem Nullbyte abgeschlossene Zeichenkette *prompt* auf dieses Gerät, schaltet das lokale Echo ab, liest eine Zeichenkette bis zum nächsten Neue-Zeile-Zeichen oder bis EOF ein, stellt den Zustand der Datensichtstation wieder her und schließt dann die Datei */dev/tty* wieder.

Die Funktion *getpass()* markiert die Felder *st_atime* und *st_mtime* der Datei */dev/tty* zum Ändern.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *getpass()* einen Zeiger auf eine mit dem Nullbyte abgeschlossene Zeichenkette von höchstens {PASS_MAX} von der Datensichtstation eingelesenen Zeichen. Wenn ein Fehler auftritt, so wird der Zustand der Datensichtstation wiederhergestellt und der Nullzeiger zurückgeliefert.

FEHLER

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

[EINTR] Die Funktion *getpass()* wurde von einem Signal unterbrochen.

[ENXIO] Der Prozeß besitzt kein kontrollierendes Terminal.

HINWEIS

Das Ergebnis zeigt auf statische Daten, deren Inhalt bei jedem Aufruf überschrieben wird.

PORTABILITÄT

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

BEISPIEL

Einlesen eines "Kennworts" mit *getpass()*:

```
main()
{
   char *getpass();
   char *z;

   z = getpass("Geheimcode 007:");
   printf("%s\n",z);
}
```

SIEHE AUCH

<limits.h>.

NAME

endpdent, getpdjbent, getpdjbnam, getpdjbnum, getpdprent, getpdprnam, getpooldat, setpdjbent, setpdprent - functions for spool-management
Funktionen zur C-Schnittstelle des Druckspools, Durchsuchen der Auftragsdateien

DEFINITION

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

getpooldat()

struct prstat *getpdprent()

struct prstat *getpdprnam(name)
char *name
     /* Druckername, entspricht st_dnam in struct prstat */

struct jobpar *getpdjbent()

struct jobpar *getpdjbnum(jobnr)
long jobnr
     /* Auftrags-(ID-)Nummer, entspricht jb_num in struct */
     /*  jobpar bzw. st_num in struct prstat */

struct jobpar *getpdjbnam(name)
char *name
     /* Auftragsname, entspricht jb_anam in struct jobpar */

void setpdprent()

void setpdjbent()

void endpdent()
```

BESCHREIBUNG

POOLDAT ist eine Datei, deren Inhalt sich bei laufendem Spoolbetrieb ständig ändert. *getpooldat()* liest die gesamte Datei POOLDAT in einen intern bereitgestellten Speicher. *getpooldat()* wird benutzt, um einen momentanen, widerspruchsfreien Überblick über das gesamte Spool-System zu erhalten, z.B. um die Auftrags-Warteschlange oder die Druckerzustände aufzulisten.

getpdprent() und *getpdprnam()* liefern Zeiger auf die Struktur *prstat*. Diese Struktur enthält die einzelnen Felder aus der Datei POOLDAT, mit Informationen über die Druckerzustände:

```
struct prstat {
   char  st_dnam[DNAML+1];  /* Druckername */
   char  st_cstate;         /* aktueller Zustand des Druckers */
   char  st_admreq;         /* anliegende Administrations-Auftraege
                               (du,dd,dk,tst,dg,...) */
   char  st_evcod;          /* aktuell eingetretenes Ereignis
                               im Spoolbetrieb */
   int   st_cmdnr;          /* Nummer des laufenden Kommandos */
   long  st_num;            /* Nummer des laufenden Auftrags */
   long  st_pages;          /* Zaehler fuer bereits gedruckte Seiten */
   long  st_seek;           /* Zeiger auf das Ende der letzten vollstaendig
                               ausgedruckten Seite in der Datei */
   int   st_ncout;          /* Anzahl der bereits gedruckten Kopien */
   long  st_wecker;         /* Zeit des naechsten Alarms zum
                               Druckerpolling */
   int   st_excode;         /* exit-Wert des Backend */
   int   st_pip[2];         /* Kennzahlen der Nachrichtenwarteschlangen zur
                               Kommunikation zwischen daemon und lpr */

   int   st_bpid;           /* Prozessnummer des Backends */
   int   st_lprpid;         /* Prozessnummer des lpr, fuer Rueckmeldungen
                               beim Probedruckbetrieb */
   long  st_dmask;          /* Maske der Druckernummer */
   short st_aform;          /* aktuelle Formular-Einstellung des Druckers */
   char  st_ofence;         /* Prioritaets-Grenzwert für die Druckausgabe */
   char  st_flags;          /* Druck-Flag (DR_LKMOD, ...) */
   char  st_msg[MESSL+1];   /* aktuelle Meldungen des Backend */
   char  st_ldcnt;          /* Lade-Zaehler des Backends */
   char  *st_null;          /* Nullzeiger */
   };
```

getpdprent(), getpdprnam(), getpdjbent(), getpdjbnum und *getpdjbnam()* liefern Zeiger auf die Struktur *jobpar*, die alle Informationen über einen Druckauftrag enthält, der in die Warteschlange eingereiht ist.

```
struct jobpar {
       char  jb_anam[ANAML+1];  /* Auftragsname */
       char  jb_dgru[DGRUL+1];  /* Name der ausgewaehlten Druckergruppe */
       char  jb_path[PATHL+1];  /* absoluter Pfadname der Druckdatei */
       char  jb_to[TNAML+1];    /* Empfaenger des Auftrags, falls gesetzt */
       char  jb_unam[UNAML+1];  /* Auftraggeber */
       char  jb_bflg[BFLGL+1];  /* Feld fuer Backend-Flags */
       char  jb_lang[LANGL+1];  /* Sprache des Auftraggebers */
       char  jb_dnam[DNAML+1];  /* Name des zur Zeit der Auftrags-
                                   Ausfuehrung zugewiesenen Druckers */
       long  jb_time;           /* Auftragszeit */
       long  jb_num;            /* Auftragsnummer */
       long  jb_gmask;          /* Maske fuer benutzbare Drucker
                                   gemaess jb_dgru[] */
       short jb_prio;           /* Auftragsprioritaet */
       short jb_form;           /* Nummer des fuer den Auftrag
                                   gewuenschten Formulars */
       int   jb_flags;          /* Auftrags-Flags DP-COP, DP-WRK, ... */
       int   jb_nc;             /* Anzahl der Kopien */
       int   jb_ncout;          /* aktuelle Anzahl der gedruckten Kopien */
       int   jb_uid;            /* Benutzernummer */
       int   jb_gid;            /* Gruppennummmer */
       long  jb_pages;          /* Anzahl der bereits gedruckten Seiten */
       long  jb_seek;           /* Zeiger auf das Ende der zuletzt
                                   gedruckten Seite */
       long  jb_size;           /* Groesse der Druckdatei */
       char  *jb_null;          /* Nullzeiger */
       };
```

getpdprent(), *getpdprnam()*, *getpdjbent()*, *getpdjbnum* und *getpdjbnam()* belegen Speicherplatz und lesen POOLDAT, falls dies nicht schon geschehen ist (Aufruf: *getpooldat()*). Um neue Änderungen von POOLDAT zu erhalten, muß *getpooldat()* nochmals aufgerufen werden.

getpdjbnum sucht ab Beginn der Auftrags-Warteschlange, bis eine passende Auftragsnummer sowie eine passende Auftragsdatei gefunden wird oder EOF erreicht ist. *getpdjbnam()* sucht nach einem passenden Auftragsnamen. Da *getpdjbnam()* sehr zeitaufwendig ist, sollte nach Möglichkeit besser *getpdjbnum* benutzt werden. *getpdprnam()* sucht ab Anfang der Struktur *prstat*, bis ein passender Druckername gefunden wird oder EOF erreicht ist.

getpdjbent() und *getpdprent()* lesen die nächste Zeile in der jeweiligen Struktur, so daß aufeinanderfolgende Aufrufe benutzt werden können, um alle Einträge aufzulisten oder um die gesamte Datei zu durchsuchen, bis ein passender Eintrag gefunden wird bzw. EOF erreicht ist.

Der Aufruf von *setpdjbent()* oder *setpdprent()* erlaubt ein erneutes Durchsuchen des Strukturfeldes. Hier ist zu beachten, daß der aktuelle Stand jedoch nur über *getpooldat()* abgerufen werden kann (s.o.).

endpdent() kann aufgerufen werden, um den durch POOLDAT belegten Speicher wieder freizugeben, sobald die Bearbeitung abgeschlossen ist.

ERGEBNIS

Im Fehlerfall oder bei EOF wird ein Nullzeiger zurückgeliefert. *getpooldat()* liefert einen Wert kleiner 0 zurück, wenn POOLDAT nicht gelesen werden kann.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard nicht enthalten.

SIEHE AUCH

getcfadent()-getcftynam(), *<sys/lpr.h>*.

NAME

getpgrp - get process group ID
Prozeßgruppennummer ermitteln

DEFINITION

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

pid_t getpgrp()
```

BESCHREIBUNG

Die Funktion *getpgrp()* liefert die Prozeßgruppennummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *getpgrp()* ist immer erfolgreich und kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *getpid()* und *setpgrp()*.

SIEHE AUCH

exec, *fork()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, *<sys/types.h>*.

NAME

getpid - get process ID
Prozeßnummer ermitteln

DEFINITION

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

pid_t getpid()
```

BESCHREIBUNG

Die Funktion *getpid()* liefert die Prozeßnummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *getpid()* ist immer erfolgreich und kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

exec, *fork()*, *getpgrp()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, *<sys/types.h>*.

NAME

getppid - get parent process ID
Vaterprozeßnummer ermitteln

DEFINITION

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

pid_t getppid()
```

BESCHREIBUNG

Die Funktion *getppid()* liefert die Vaterprozeßnummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *getppid()* ist immer erfolgreich und kein Ergebnis ist für die Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

exec, *fork()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*, *<sys/types.h>*.

NAME

getpw - Benutzernummer aus Kennwortdatei abfragen

DEFINITION

```
getpw (uid, puf)
int uid;
char *puf;
```

BESCHREIBUNG

getpw() sucht in der Kennwortdatei nach der Benutzernummer *uid* und trägt die zugehörige Zeile in *puf* ein. Falls *uid* nicht gefunden werden kann, wird ein Wert ungleich 0 zurückgegeben. Die eingetragene Zeile ist mit dem Nullbyte abgeschlossen.

ERGEBNIS

Bei Fehler ein Wert ungleich 0.

PORTABILITÄT

Die Funktion *getpw()* ist im X/Open-Standard nicht mehr enthalten. Sie wird lediglich aus Gründen der Kompatibilität zu früheren Systemen zur Verfügung gestellt und sollte nicht verwendet werden; stattdessen sollten die Funktionen *getpwnam()* und *getpwuid()* verwendet werden.

BEISPIEL

Das Programm liefert zu einer beim Aufruf übergebenen Benutzernummer den Eintrag in */etc/passwd*, falls er existiert:

```
#include <stdio.h>

int main(argc,argv)
int argc;
char ** argv;
{
   char buf[BUFSIZ];

   if ((getpw(atoi(argv[1]),buf)) == 0)
   {  puts(buf);
      return 0;
   }
   else
      return 1;
}
```

SIEHE AUCH

getpwent(), *getpwnam()*, *getpwuid()*.

NAME

endpwent, fgetpwent, getpwent, setpwent - Eintrag aus Kennwortdatei abfragen

DEFINITION

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

struct passwd *getpwent ()

void setpwent ()

void endpwent ()

struct passwd *fgetpwent (f)
FILE *f;
```

BESCHREIBUNG

getpwent() und *fgetpwent()* geben beide einen Zeiger auf eine Struktur des Typs *struct passwd* zurück, die in den folgenden Komponenten die einzelnen Elemente einer Zeile aus der Kennwortdatei enthält:

```
struct passwd {
    char *pw_name;      /* Login-Name */
    char *pw_passwd;    /* verschluesseltes Kennwort
                           (evtl. leer) */
    int pw_uid;         /* Benutzernummer */
    int pw_gid;         /* Gruppennummer (evtl. leer) */
    char *pw_age;       /* nicht versorgt */
    char *pw_comment;   /* nicht versorgt */
    char *pw_gecos;     /* Kommentar und
                           Login-Universum */
    char *pw_dir;       /* Login-Dateiverzeichnis */
    char *pw_shell;     /* Startprogramm, Standard: /bin/sh */
};

struct comment
{
    char *c_dept;
    char *c_name;
    char *c_acct;
    char *c_bin;
};
```

Der Eintrag *pw_comment* wird nicht benutzt, die Bedeutung der anderen ist in *<pwd.h>* beschrieben.

getpwent() liest die nächste Zeile in der Datei, so daß aufeinanderfolgende Aufrufe dazu benutzt werden können, die gesamte Datei zu durchsuchen (s. Beispiel).

Ein Aufruf von *setpwent()* bewirkt das Rücksetzen des Lesezeigers der Kennwortdatei auf Dateianfang, um so eine wiederholte Suche zu ermöglichen. Der Aufruf von *endpwent()* sollte zum Schließen der Kennwortdatei verwendet werden, wenn die Dateibearbeitung abgeschlossen ist.

Die Prozedur *fgetpwent()* gibt einen Zeiger auf die nächste *passwd*-Struktur in der Datei *f* zurück; diese Datei muß das Format von */etc/passwd* haben.

ERGEBNIS

getpwent() und *fgetpwent()* geben alle einen Zeiger auf eine Struktur des Typs *struct passwd* zurück.

Bei einem Fehler oder EOF wird von diesen Funktionen der Nullzeiger zurückgegeben.

HINWEIS

Die gesamte Information wird in einem statischen Bereich abgelegt, so daß sie bei Sicherungsbedarf kopiert werden muß.

Von dem Gebrauch von *pw_gecos* und *struct comment* wird abgeraten.

PORTABILITÄT

Diese Funktionen sind im X/Open-Standard nicht mehr enthalten. Anwendungen sollten nur noch die Funktionen *getpwnam()* und *getpwuid()* verwenden.

BEISPIEL

Das Programm druckt die Einträge der "Kennwortdatei" mit entsprechenden Kommentaren:

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

struct passwd *tompass;

main()
{
   int i;
   while ((tompass = getpwent()) != NULL)
   {
      printf("::::::::::::::::::::::::::::::::::: %d\n");
      printf("Name: %s\n", tompass->pw_name);
      printf("Passwd: %s\n", tompass->pw_passwd);
      printf("UID: %d\n", tompass->pw_uid);
      printf("GID: %d\n", tompass->pw_gid);
      printf("Age: %s\n", tompass->pw_age);
      printf("Comment: %s\n", tompass->pw_comment);
      printf("Gecos: %s\n", tompass->pw_gecos);
      printf("Dir: %s\n", tompass->pw_dir);
      printf("Shell: %s\n", tompass->pw_shell);
   }
   exit(0);
}
```

SIEHE AUCH

getgrent(), *getlogin()*, *getpwnam()*, *getpwuid()*, *<pwd.h>*.

NAME

getpwnam - search user database for particular name
Eintrag für Namen in der Benutzerdatei ermitteln

DEFINITION

```
#include <pwd.h>

struct passwd *getpwnam (name)
char *name;
```

BESCHREIBUNG

Die Funktion *getpwnam()* sucht in der Benutzerdatei nach einem Eintrag, dessen Komponente *pw_name* zu *name* paßt.

ERGEBNIS

Die Funktion *getpwnam()* liefert einen Zeiger auf eine *struct passwd*, die in *<pwd.h>* definiert ist, mit einem passenden Eintrag, sofern einer gefunden wird. Sie liefert den Nullzeiger bei Fehler, oder wenn der geforderte Eintrag nicht gefunden wird. Das Ergebnis zeigt auf einen statischen Bereich, der durch einen anschließenden Aufruf von *cuserid*, *getpwuid()* oder *getpwnam()* überschrieben wird.

FEHLER

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

[EIO] Ein Ein- oder Ausgabefehler ist aufgetreten.

[EINTR] Während der Funktion wurde ein Signal abgefangen.

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

[ENFILE] Die Dateitabelle des Systems ist voll.

HINWEIS

Die Implementierung der Funktion *cuserid()* verwendet die Funktion *getpwnam()*: daher kann wird das Ergebnis eines Aufrufs einer der Funktionen durch den Aufruf der anderen überschrieben.

PORTABILITÄT

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

SIEHE AUCH

getpwuid(), *<pwd.h>*.

NAME

getpwuid - search user database for particular user ID
Eintrag für Benutzernummer in Benutzerdatei ermitteln

DEFINITION

```
#include <pwd.h>

struct passwd *getpwuid (uid)
uid_t uid;
```

Die Funktion *getpwuid()* sucht in der Benutzerdatei nach einem Eintrag, dessen Komponente *pw_uid* zu *uid* paßt.

ERGEBNIS

Die Funktion *getpwuid()* liefert einen Zeiger auf eine *struct passwd*, die in *<pwd.h>* definiert ist, mit einem passenden Eintrag, sofern einer gefunden wird. Sie liefert den Nullzeiger bei Fehler, oder wenn der geforderte Eintrag nicht gefunden wird.

FEHLER

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

[EIO] Ein Ein- oder Ausgabefehler ist aufgetreten.

[EINTR] Während der Funktion wurde ein Signal abgefangen.

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

[ENFILE] Die Dateitabelle des Systems ist voll.

HINWEIS

Drei, zum aktuellen Prozeß gehörende Namen können bestimmt werden: *cuserid()* liefert den Namen, der der effektiven Benutzernummer des Prozesses zugeordnet ist; *getlogin()* liefert den Namen, der den aktuellen Anmeldungs-Aktivitäten zugeordnet ist und *getpwuid* (*getuid()*) liefert den Namen, der zur realen Benutzernummer des Prozesses gehört.

Das Ergebnis zeigt auf einen statischen Bereich, der durch einen anschließenden Aufruf von *cuserid()*, *getpwnam()* oder *getpwuid()* überschrieben wird.

PORTABILITÄT

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

SIEHE AUCH

getpwnam(), *<pwd.h>*.

NAME

gets - get a string from *stdin* stream
Zeichenkette von stdin lesen

DEFINITION

```
#include <stdio.h>

char *gets (s)
char *s;
```

BESCHREIBUNG

Die Funktion *gets()* liest Zeichen vom Datenstrom *stdin* in den Vektor, auf den *s* zeigt, bis ein Neue-Zeile-Zeichen gefunden oder das Dateiende erreicht wird. Ein Neue-Zeile-Zeichen wird verworfen und unmittelbar hinter das letzte, in den Vektor gelesene Zeichen, wird ein Nullbyte geschrieben.

Die Funktion *gets()* kann das Feld *st_atime* der *stream* zugeordneten Datei zum Ändern markieren. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung von *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* auf dem Datenstrom *stream* zum Ändern markiert, die keine Daten zurückliefert, die vorher durch *ungetc()* bereitgestellt worden sind.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *gets()* den Wert von *s*. Wenn der Datenstrom das Dateiende erreicht hat, dann wird das Kennzeichen dieses Datenstroms für das Dateiende gesetzt und *gets()* liefert den Nullzeiger. Wenn ein Lesefehler auftritt, dann wird das Fehlerkennzeichen des Datenstroms gesetzt, *gets()* liefert den Nullzeiger und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fgetc()*.

HINWEIS

Das Lesen einer Zeile, die die Länge des Vektors *s* überschreitet, liefert undefinierte Ergebnisse Die Verwendung von *fgets()* wird empfohlen.

PORTABILITÄT

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

SIEHE AUCH

feof(), *ferror()*, *fgets()*, *<stdio.h>*.

NAME

getuid - get real user ID
Reale Benutzernummer ermitteln

DEFINITION

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

uid_t getuid()
```

BESCHREIBUNG

Die Funktion *getuid()* liefert die reale Benutzernummer des aufrufenden Prozesses.

ERGEBNIS

Die Funktion *getuid()* ist immer erfolgreich und es ist kein Ergebnis zur Anzeige eines Fehlers reserviert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

BEISPIEL

Schreiben und übersetzen Sie dieses Programm unter Ihrer Kennung (nicht root) und setzen Sie für das ausführbare Programm *a.out* mit dem *chmod(1)* Kommando das s-Bit für den Eigentümer (z. B. chmod 4777 a.out). Rufen Sie dann das lauffähige Programm *a.out* unter einer anderen Kennung (z.B. *root*) mit anderer Gruppennummer auf.

```
#include <stdio.h>

main()
{
   unsigned uid, euid, gid, egid;
   uid = getuid();
   euid = geteuid();
   gid = getgid();
   egid = getegid();

   printf("reale Benutzernummer = %d\n", uid);
   printf("effektive Benutzernummer = %d\n", euid);
   printf("reale Gruppennummer = %d\n", gid);
   printf("effektive Gruppennummer = %d\n", egid);
}
```

SIEHE AUCH

geteuid(), *getgid()*, *setuid()*, *<sys/types.h>*.

NAME

getutent - access utmp file entry
Einträge aus utmp-Datei bearbeiten

DEFINITION

```
nclude <utmp.h>

struct utmp *getutent()

struct utmp *getutline(line)
struct utmp *line;

void setutent()

void endutent()

void utmpname(file)
char *file;
```

BESCHREIBUNG

Diese Funktionen erlauben es dem Benutzer, die Datei */etc/utmp* zu bearbeiten, d.h. Einträge aus dieser Datei zu lesen. Die Datei */etc/utmp/* enthält für alle derzeit angemeldeten Benutzer einen Eintrag mit den Daten des Login-Prozesses.

Die Funktionen *getutent()* und *getutline()* liefern beide einen Zeiger auf eine Struktur des Typs *struct utmp*, die in der Datei *<utmp.h>* wie folgt definiert ist:

```
struct utmp {
  char  ut_user[8];     /* Benutzerkennung                      */
  char  ut_id[4];       /* Nummer aus der /etc/inittab          */
  char  ut_line[12];    /* Datensichtstationsname               */
  short ut_pid;         /* Prozessnummer                        */
  short ut_type;        /* Typ des Eintrags                     */
  struct exit_status {
     short e_termination;  /* Grund fuer Prozessende           */
     short e_exit;         /* Prozess-Endestatus               */
  } ut_exit;            /* Endestatus eines Prozesses, der als
                           DEAD_PROCESS markiert ist            */
  time_t ut_time;       /* Zeit des Eintrags                    */
};
```

Die Funktion *getutent()* liest den nächsten Eintrag aus einer Datei, die denselben Aufbau wie */etc/utmp* haben muß. Wenn diese Datei noch nicht geöffnet ist, dann öffnet die Funktion *getutent()* sie. Wenn die Funktion das Dateiende erreicht hat, dann schlägt sie fehl und liefert den Nullzeiger zurück.

Die Funktion *getutline()* durchsucht die *utmp*-Datei von der aktuellen Position aus solange in Richtung Dateiende, bis sie entweder das Dateiende erreicht, oder einen Eintrag findet, dessen Komponente *ut_line* zu der Komponente *line->ut_line* paßt, die durch das Argument *line* angegeben wird. Findet die Funktion *getutline()* einen passenden Eintrag, so liefert sie einen Zeiger auf eine Struktur des Typs *struct utmp*, in welcher der gefundene Eintrag abgelegt ist. Findet sie das Dateiende, so schlägt sie fehl und liefert den Nullzeiger.

Die Funktion *setutent()* positioniert die *utmp*-Datei wieder an den Dateianfang, so daß die Datei auch mehrfach durchsucht werden kann. Die Funktion *endutent()* beendet die Bearbeitung der *utmp*-Datei und schließt diese.

Mit der Funktion *utmpname()* kann eine Anwendung jede beliebige Datei als zu bearbeitende *utmp*-Datei bestimmen. Voraussetzung ist, daß die über das Argument *file* angegebene Datei dasselbe Format wie die Datei */etc/utmp* besitzt. Dies trifft normalerweise nur für die Datei */etc/wtmp* zu. Die Funktion *utmpname()* schließt eine eventuell offene *utmp*-Datei und speichert den Namen der angegebenen Datei ab. Erst ein nachfolgender Aufruf von *getutent()* öffnet dann die neue *utmp*-Datei.

ERGEBNIS

Bei erfolgreicher Beendigung liefern die Funktionen *getutent()* und *getutline()* einen Zeiger auf eine *utmp*-Struktur, die den nächsten, bzw. den nächsten passenden Eintrag enthält. Andernfalls liefern diese Funktionen den Nullzeiger.

Die Funktionen *setutent()*, *endutent()* und *utmpname()* liefern kein Ergebnis.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Die Ergebnisse der Funktionen *getutent()* und *getutline()* zeigen auf einen statischen Speicherbereich, der bei jedem Aufruf überschrieben wird. Dieser statische Bereich wird auch von jedem Aufruf der Funktion *getutline()* zuerst auf einen passenden Eintrag hin untersucht. Sollen also mit der Funktion *getutline()* mehrere Einträge gesucht werden, die alle dieselbe Komponente *ut_line* besitzen, so muß die Komponente *ut_line* des statischen Bereichs vor jedem weiteren Aufruf gelöscht werden, indem an die Adresse, auf die *ut_line* zeigt das Nullbyte '\0' geschrieben wird.

PORTABILITÄT

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

SIEHE AUCH

ttyslot().

NAME

getutline - access utmp file entry
Nächsten utmp-Eintrag lesen

DEFINITION

```
nclude <utmp.h>

struct utmp *getutline(line)
struct utmp *line;
```

BESCHREIBUNG

Die Funktion *getutline()* gehört zu einer Gruppe von Funktionen, die Einträge in der Datei */etc/utmp* bearbeiten. Sie liest einen Eintrag aus der *utmp*-Datei. Eine ausführliche Beschreibung finden Sie unter *getutent()*.

PORTABILITÄT

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

NAME

getw - get a word from a stream

DEFINITION

```
#include <stdio.h>

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

BESCHREIBUNG

Die Funktion *getw()* liest das nächste Maschinenwort aus dem Datenstrom *stream*. Die Größe eines Maschinenworts ist die Größe eines *int*; sie kann von Rechner zu Rechner unterschiedlich sein. Die Funktion *getw()* nimmt für die Datei keine besondere Ausrichtung an.

Die Funktion *getw()* kann das Feld *st_atime* der *stream* zugeordneten Datei zum Ändern markieren. Das Feld *st_atime* wird von der ersten erfolgreichen Ausführung von *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()* oder *fscanf()* auf dem Datenstrom *stream* zum Ändern markiert, die Daten zurückliefert, die nicht vorher durch *ungetc()* bereitgestellt worden sind.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *getw()* das nächste Wort aus dem Eingabestrom, auf den *stream* zeigt. Wenn der Datenstrom das Dateiende erreicht hat, dann wird das Kennzeichen dieses Datenstroms für das Dateiende gesetzt und *getw()* liefert EOF. Wenn ein Lesefehler auftritt, dann wird das Fehlerkennzeichen des Datenstroms gesetzt, *getw()* liefert EOF und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

Siehe unter *fgetc()*.

HINWEIS

Wegen der möglichen Unterschiede in der Wortlänge und Ausrichtung sind Dateien, die mit *putw()* geschrieben wurden, rechnerabhängig; sie können unter Umständen auf einem anderen Rechner nicht korrekt mit *getw()* gelesen werden.

Weil die Darstellung von EOF eine gültige ganze Zahl sein kann, sollten Anwendungen, die auf Fehler überprüfen wollen, *ferror()* und *feof()* benutzen.

PORTABILITÄT

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

SIEHE AUCH

ferror(), *getc()*, *putw()*, *<stdio.h>*.

NAME

gmtime - convert time value to broken down UTC time
Zeit (UTC) in Struktur aufschlüsseln

DEFINITION

```
#include <time.h>

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

BESCHREIBUNG

Die Funktion *gmtime()* wandelt die Zeit in Sekunden seit dem Epochenwert, auf die *timer* zeigt, um in eine in der Struktur *tm* aufgeschlüsselte Zeit um, ausgedrückt als Coordinated Universal Time (UTC).

ERGEBNIS

Die Funktion *gmtime()* liefert einen Zeiger auf eine Struktur des Typs *struct tm*, in der die Zeit aufgeschlüsselt ist.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

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

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *localtime()*.

SIEHE AUCH

asctime(), *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *time()*, *utime()*, *<time.h>*.

NAME

gsignal - Software-Signal auslösen

DEFINITION

```
#include <signal.h>

int gsignal (sig)
int sig;
```

BESCHREIBUNG

Mit *ssignal()* und *gsignal()* werden dem Benutzer softwaremäßige Funktionen analog zu *signal()* und *kill()* angeboten.

Die Beschreibung von *gsignal()* ist mit der von *ssignal()* unter *ssignal()* zusammengefaßt.

PORTABILITÄT

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

NAME

hcreate - create hash search tables
Hash-Tabelle erzeugen

DEFINITION

```
#include <search.h>

int hcreate (nel)
unsigned nel;
```

BESCHREIBUNG

Siehe unter *hsearch()*.

PORTABILITÄT

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

NAME

hdestroy - destroy hash search tables
Hashtabellen zerstören

DEFINITION

```
#include <search.h>

void hdestroy()
```

BESCHREIBUNG

Siehe unter *hsearch()*.

PORTABILITÄT

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

NAME

hsearch, hcreate, hdestroy - manage hash search tables
Hash-Tabellen verwalten

DEFINITION

```
#include <search.h>

ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy()
```

BESCHREIBUNG

Die Funktion *hsearch()* durchsucht eine Hash-Tabelle. Sie liefert einen Zeiger in eine Hash-Tabelle, der die Position angibt, an der ein Eintrag gefunden werden kann. Das Artgument *item* ist eine Struktur vom Typ *ENTRY* (definiert in der Include-Datei *<search.h>*), die zwei Zeiger enthält: *item.key* zeigt auf den Vergleichsschlüssel und *item.data* zeigt auf beliebige weitere Daten, die zu diesem Schlüssel gehören. (Zeiger auf andere Datentypen als *void* sollten in *(void *)* umgewandelt werden.) Das Argument *action* ist ein Element eines Aufzählungstyps *ACTION*, das die Aktion angibt, die dann ausgeführt werden soll, wenn der Eintrag nicht in der Tabelle gefunden wird. ENTER gibt an, daß der Eintrag an einer passenden Stelle in die Tabelle eingefügt werden soll. FIND gibt an, daß kein Eintrag vorgenommen werden soll. Eine erfolglose Suche wird durch die Rückgabe des Nullzeigers angezeigt.

Die Funktion *hcreate()* reserviert genügend Speicherplatz für die Tabelle, und muß vor der Verwendung von *hsearch()* aufgerufen werden. Das Argument *nel* ist ein Schätzwert für die maximale Anzahl von Einträgen, die in die Tabelle aufgenommen werden sollen. Diese Anzahl kann durch den Algorithmus nach oben angepaßt werden, um bestimmte, mathematisch günstige Umstände zu erzielen.

Die Funktion *hdestroy()* gibt die Hashtabelle frei und kann von einem weiteren Aufruf von *hcreate()* gefolgt werden. Nach einem Aufruf von *hdestroy()* darf auf die Daten nicht länger zugegriffen werden.

ERGEBNIS

Die Funktion *hsearch()* liefert den Nullzeiger, wenn die Aktion entweder gleich FIND ist und der Eintrag nicht gefunden werden kann, oder wenn die Aktion gleich ENTRY und die Tabelle voll ist.

Die Funktion *hcreate()* liefert den Wert 0, wenn sie nicht genügend Speicherplatz für die Tabelle reservieren kann.

BEISPIEL

Das folgende Beispiel liest Zeichenketten und jeweils zwei Nummern ein und speichert diese in der Hashtabelle, wobei doppelte Einträge verworfen werden. Danach liest es Zeichenketten ein, findet den passenden Eintrag in der Hashtabelle und gibt diesen aus.

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

struct info {              /* Diese Informationen stehen     */
                           /* zusaetzlich zum Suchbegriff in */
   int age, room;          /* der Tabelle                    */
};

#define NUM_EMPL    5000  /* Anzahl von Elementen           */

main( )
{
   char string_space[NUM_EMPL*20];   /* Platz fuer Zeichenketten */
   struct info info_space[NUM_EMPL]; /* Platz fuer Informationen */
   char *str_ptr = string_space;     /* Naechster freier Platz
                                      * in string_space */
   struct info *info_ptr = info_space;  /* Naechster freier Platz
                                         * in info_space */
   ENTRY item;
   ENTRY *found_item;
   ENTRY *hsearch( );
   char name_to_find[30];  /* Zu suchender Name */

   int i = 0;

   /* Tabelle erzeugen; keine Fehlerprüfung */
   (void) hcreate(NUM_EMPL);
   while (scanf("%s%d%d", str_ptr, &info_ptr->age,
          &info_ptr->room) != EOF && i++ < \s-1NUM_EMPL\s+1) {

          /* Struktur fuellen und eintragen */
          item.key = str_ptr;
          item.data = (void *)info_ptr;
          str_ptr += strlen(str_ptr) + 1;
          info_ptr++;

          /*  Eintrag in die Tabelle einfuegen */
          (void) hsearch(item, ENTER);
   }
```

```
    /* auf Tabelle zugreifen */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != \s-1NULL\s+1) {

            /* wenn Eintrag in Tabelle ist */
            (void)printf("gefunden %s, age = %d, room = %d\n",
                    found_item->key,
                    ((struct info *)found_item->data)->age,
                    ((struct info *)found_item->data)->room);
  } else {
            (void)printf("kein Eintrag %s\n", name_to_find);
  }
}
```

HINWEIS

Die Funktionen *hsearch()* und *hcreate()* können die Funktion *malloc()* verwenden, um Platz zu reservieren.

PORTABILITÄT

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

SIEHE AUCH

bsearch(), *lsearch()*, *malloc()*, *tsearch()*, *<search.h>*.

NAME

hypot - Euclidean distance function
Euklidische Distanzfunktion

DEFINITION

```
#include <math.h>

double hypot (x, y)
double x, y;
```

BESCHREIBUNG

Die Funktion *hypot()* berechnet die Hypothenuse eines rechtwinkligen Dreiecks, d.h. $\sqrt{(x*x + y*y)}$.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *hypot()* die Länge der Hypothenuse des rechtwinkligen Dreiecks mit Seiten der Länge x und y.

Wenn das Ergebnis einen Überlauf erzeugen würde, dann wird HUGE_VAL zurückgeliefert und *errno* ist gesetzt, um den Fehler anzuzeigen.

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

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

FEHLER

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

[ERANGE]
Das Ergebnis würde einen Überlauf verursachen.

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

[EDOM] Der Wert von x oder y ist ein NaN.

HINWEIS

Die Funktion *hypot()* trifft Vorkehrungen gegen einen Überlauf während der einzelnen Schritte der Berechnung. Wenn das berechnete Ergebnis dennoch über den Wertebereich von *double* hinausgeht, dann liefert *hypot()* den Wert HUGE_VAL.

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

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

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

cc programm.c -lm

PORTABILITÄT

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

BEISPIEL

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

main()
{
   double x,y;
   double alpha,r,pi;

   printf("Koordinaten x,y eingeben:");
   scanf("%lf %lf",&x,&y);

   pi = 2*asin(1.0);

   if (x>0.0)
      alpha = atan(y/x);
   else if (x <0.0)
           if (y >-0.0)
              alpha = atan(y/x) + pi;
           else alpha = atan(y/x) -pi;
   else if (y > 0)
           alpha = pi/2.0;
        else if (y < 0)
                alpha = -pi/2.0;
             else
             {
                printf("Winkel nicht definiert");
                exit(1);
             }

   r = hypot(x,y);
   alpha = alpha *(189.0/pi);

   printf("Die Polarkoordinaten lauten: \n");
   printf("Abstand vom Nullpunkt : %g\n",r);
   printf("Winkel zur x-Achse:");
   printf("%g Grad\n",((y<0.0)? alpha + 360: alpha));
}
```

SIEHE AUCH

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

NAME

ioctl - Gerätesteuerung

DEFINITION

```
int ioctl (datkz, aktion, arg)
int datkz, aktion;
struct termio *arg;
```

BESCHREIBUNG

Die Funktion *ioctl()* bietet eine Vielzahl von Funktionen zur Steuerung von - üblicherweise zeichenorientierten - Geräten über die dazugehörigen Gerätedateien.

Mit dem Argument *datkz* geben Sie die Dateikennzahl einer geöffneten Datei an, für die eine Steuerung erfolgen soll.

Mit dem Argument *aktion* bezeichnen Sie die auszuführende Funktion in Abhängigkeit vom zu steuernden Gerät. *arg*-Wert und -typ sind geräte- und funktionsspezifisch.

Leitungsprozedur

Das Feld *c_lflag* in der Struktur *termio* wird von der Leitungsprozedur verwendet, um die Datensichtstationsfunktionen zu steuern. Die grundlegende Leitungsprozedur sieht folgende Funktionen vor:

ISIG	Signalaktivierung
ICANON	Zeilenorientierte (Standard) Eingabe (Behandlung von ERASE- und KILL-Zeichen)
XCASE	Standardgemäße Darstellung von Groß-/Kleinbuchstaben
ECHO	Echo-Funktion aktivieren
ECHOE	ERASE-Zeichen als BS-SP-BS ausgeben ("Echo")
ECHOK	NL-Zeichen nach KILL-Zeichen ausgeben ("Echo")
ECHONL	NL-Zeichen ausgeben ("Echo")
NOFLSH	Leeren der Ein- und Ausgabewarteschlangen nach INTERRUPT oder QUIT von Tastatur desaktivieren

Bei gesetztem ISIG wird bei jedem eingegebenen Zeichen geprüft, ob es sich um eins der Steuerzeichen INTR, SWTCH und QUIT handelt. Ist dies der Fall, so wird die dazugehörige Funktion ausgeführt. Ist ISIG nicht gesetzt, so wird diese Prüfung nicht durchgeführt. D.h., diese Sonderfunktionen für die Eingabe können nur bei gesetztem ISIG durchgeführt werden. Sie können aber auch einzeln ausgeschaltet werden, indem man ihnen einen unwahrscheinlichen oder unmöglichen Wert als Steuerzeichen zuordnet (z.B. 0377).

Bei gesetztem ICANON wird die zeilenorientierte (standardmäßige) Eingabebehandlung aktiviert. Das heißt, die Funktionen zur Behandlung von ERASE- und KILL-Zeichen werden aktiviert. Die Eingabezeichen werden zu Zeilen aufgesammelt. Das Ende einer Zeile wird mit NL, EOF oder EOL angegeben. Ist ICANON nicht gesetzt, werden Lesecaufträge direkt aus der Eingabewarteschlange bedient, und zwar erst dann, wenn mindestens MIN Zeichen empfangen wurden oder der Zeitüberwachungswert TIME überschritten ist (s.u. ''Wirkung von MIN/TIME''). Dies macht es möglich, daß einerseits schnell aufeinanderfolgende Eingabeschübe auf effiziente Weise gelesen und andererseits weiterhin einzelne Zeichen eingegeben werden können. Die Werte für MIN und TIME werden unter dem EOF- bzw. dem EOL-Zeichen abgelegt. Die Angabe des TIME-Wertes erfolgt in Zehntelsekunden.

Sind XCASE und ICANON gesetzt, so werden Großbuchstaben bei der Eingabe akzeptiert, sofern ihnen ein \ Zeichen vorangestellt ist; bei der Ausgabe wird ihnen ein \ Zeichen vorangestellt. In diesem Betriebsmodus werden folgende Escape-Sequenzen bei der Ausgabe erzeugt und bei der Eingabe akzeptiert:

Bedeutung	Codierung
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

So werden z.B. für A die Zeichen \a eingegeben, für \n die Zeichen \\n und für \N die Zeichen \\\n.

Bei gesetztem ECHO werden eingegebene Zeichen so, wie sie empfangen wurden, wieder auf den Bildschirm ausgegeben.

Bei gesetztem ICANON sind folgende "Echo"-Funktionen möglich: Sind ECHO und ECHOE gesetzt, so wird das ERASE-Zeichen als ASCII BS SP BS zurückgeliefert, wodurch das letzte Zeichen auf dem Bildschirm gelöscht wird. Ist ECHOE gesetzt und ECHO nicht, so wird das ERASE-Zeichen als ASCII SP BS zurückgeliefert. Bei gesetztem ECHOK wird nach dem KILL-Zeichen ein NL-Zeichen auf den Bildschirm ausgegeben und damit angezeigt, daß die Zeile gelöscht wird.

Hinweis
Ein einem ERASE- oder KILL-Zeichen vorausgehendes ESCAPE-Zeichen hebt die dem Zeichen zugeordnete Steuerfunktion auf.

Bei gesetztem ECHONL wird ein NL-Zeichen auch dann auf den Bildschirm ausgegeben, wenn ECHO nicht gesetzt ist. Dies ist bei Datensichtstationen im lokalen Echo-Modus (sog. Halbduplexbetrieb) nützlich. Ein EOF-Zeichen wird nur auf den Bildschirm ausgegeben, wenn es entwertet ist. Da das EOT-Zeichen ('Ende der Übertragung') standardmäßig als EOF-Zeichen verwendet wird, kann man auf diese Weise eine Verbindungsauflösung durch Datensichtstationen, die sich bei Empfang von EOT abmelden, verhindern.

Bei gesetztem NOFLSH findet die normalerweise nach Empfang von QUIT, INTR und SWTCH durchgeführte Löschung der Ein- und Ausgabewarteschlangen nicht statt.

Der Anfangswert für die Leitungsprozedur ist, kein Bit gesetzt.

Die primären *ioctl()*-Systemaufrufe haben folgendes Format:

```
ioctl (datkz, funktion, arg)
int datkz, funktion;
struct termio *arg;
```

funktion kann dabei folgende Werte annehmen:

TCGETA
Die der Datensichtstation zugeordneten Parameter abfragen und in der *termio*-Struktur, auf die *arg* verweist, ablegen.

TCSETA
Die der Datensichtstation zugeordneten Parameter entsprechend der Struktur, auf die *arg* verweist, setzen. Diese Änderung hat sofortige Wirkung.

TCSETAW
Mit dem Setzen der Parameter bis zur Ausgabebeendigung warten. Diese Funktion sollte verwendet werden, wenn sich die Änderung der Parameter auf die Ausgabe auswirkt.

TCSETAF
Auf Ausgabebeendigung warten, dann Eingabewarteschlange löschen und die Parameter neu setzen.

Weitere *ioctl()*-Aufrufe haben folgende Form:

```
ioctl (datkz, funktion, arg)
int arg;
```

In dieser Form kann *funktion* folgende Werte annehmen:

TCSBRK
Auf Ausgabebeendigung warten, dann - sofern *arg* = 0 - "break" absetzen (Nullbits, 0,25 Sekunden).

TCXONC
START/STOP-Steuerung. Bei *arg* = 0: Ausgabe anhalten; bei *arg* = 1: angehaltene Ausgabe fortsetzen.

TCFLSH
Bei *arg* = 0: Eingabewarteschlange löschen;
bei *arg* = 1: Ausgabewarteschlange löschen;
bei *arg* = 2: Ein- und Ausgabewarteschlange löschen.

ERGEBNIS

Im Fehlerfall wird –1 zurückgeliefert und der Fehler in *errno* angezeigt.

FEHLER

ioctl kann nicht ausgeführt werden, wenn einer oder mehrere der folgenden Fehler auftreten:

[EBADF] *datkz* ist keine gültige Dateikennzahl.

[EINTR] Während der *ioctl*-Ausführung wurde ein Signal abgefangen.

[EINVAL] *aktion* oder *arg* ist ungültig.

[ENOTTY] Das der Datei mit Dateikennzahl *datkz* entsprechende Gerät akzeptiert keine Steuerfunktionen.

HINWEIS

Da *ioctl()* äußerst implementierungsabhängig ist, ist eine genauere Beschreibung im Rahmen dieses Manuals nicht sinnvoll. Wenn Sie den Zugang zu einem Gerät über *ioctl()* wünschen, müssen Sie zusätzlich Spezialliteratur hinzuziehen, in der die an Ihrem System angeschlossenen Geräte und die darauf abgestimmten *ioctl()*-Funktionen beschrieben sind. Für weitere Informationen zum Thema Gerätesteuerung verwenden Sie bitte das Handbuch SINIX-Systemverwaltung [4] und vor allem das Handbuch SINIX-Schnittstellen [2].

PORTABILITÄT

Die Funktion *ioctl()* ist im X/Open-Standard ab Ausgabe 3 nicht mehr enthalten. Statt dessen sollten die X/Open-kompatiblen Funktionen *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcgetattr()* und *tcsetattr()* verwendet werden.

BEISPIEL

Die hier gegebene Funktion yes_no_prompt promptet yes/no-Angaben vom Terminal (y=yes, n=no), wobei alle anderen Eingaben außer y/Y/n/N (auch z.B. DEL-Taste) abgewiesen werden. Durch Wegnahme des ECHO-, des ICANON- und des ISIG-Bits geschieht folgendes:

- die Eingabe des Benutzers erscheint nicht am Bildschirm (in der do-Schleife werden nur die Zeichen y/Y/n/N davon ausgenommen)
- das c_cc[VEOL] erhält Bedeutung und Wert des c_cc[VTIME], und das c_cc[VEOF] erhält Bedeutung und Wert des c_cc[VMIN]; [VTIME] und [VMIN] sollten daher explizit gesetzt werden, wobei MIN > 0 und TIME = 0 für die Mindestanzahl der einzugebenden Zeichen ohne Zeitüberwachung für deren Übertragung steht (s. *<termios.h>*).
- das c_cc[VINTR] und das c_cc[VQUIT] erzeugen nicht das Signal 2 bzw. 3.

```
#include <stdio.h>
#include <sys/termio.h>

#define YES_GIVEN(c) ((c) == 'y' ## (c) == 'Y')
#define NO_GIVEN(c)  ((c) == 'n' ## (c) == 'N')

#define BELL '\007'    /* '\a' fuer neuere Systeme */

typedef enum { NO, YES } Answer;
#define DEFAULT YES

main()
{
    extern Answer yes_no_prompt();

    Answer standard = DEFAULT;

    switch ((int) yes_no_prompt("Bitte eingeben: ", standard))
        {
        case YES: printf("Sie haben Y oder y eingegeben. Danke.\n");
                  break;
        case  NO: printf("Sie haben N oder n eingegeben. Danke.\n");
                  break;
        default : printf("Irgendetwas ging schief!");
        }
}
```

```
Answer yes_no_prompt(out_string, default_value)
char *out_string;
Answer default_value;
{
   struct termio tp1;
   struct termio tp2;
   Answer value;
   int c;

   if (ioctl(0, TCGETA, &tp1) != 0)      /* Liegt stdin auf Terminal? */
       return(default_value);

   fputs(out_string, stdout);            /* z.B. "remove file?" */
   fputs(" (y/n)? ", stdout);
   fflush(stdout);                       /* Sicherstellen, dass Ausgabe erscheint */

   tp2 = tp1;
   /* Wenn ein System es nur erlaubt, strukturierte Daten als Referenz- */
   /*  parameter zu uebergeben, sollte statt tp2 = tp1 folgende Funktion  */
   /*  benutzt werden: */
   /*  memcpy ((char*) &tp2, (char*) &tp1, sizeof(struct termio)); */

   tp2.c_lflag       &= ~(ECHO ≠ ICANON ≠ ISIG);
   tp2.c_cc [VMIN]   = 1;
   tp2.c_cc [VTIME]  = 0;

   ioctl(0, TCSETAW, &tp2);              /* Setzen der neuen tty-Parameter */

   do {
       c = fgetc(stdin);
       if ( YES_GIVEN (c) )              /* y oder Y getippt? */
       {
           value = YES;
           break;
        } else if ( NO_GIVEN (c) )       /* n oder N getippt? */
          {
           value = NO;
           break;
          }
      } while (fputc(BELL, stdout));     /* sonst: unerlaubte Eingabe */

      fputc(c, stdout);
      fputc('\n', stdout);
      fflush(stdout);

      ioctl(0, TCSETAW, &tp1);           /* Ruecksetzen der tty-Parameter */

      return  value;
}
```

SIEHE AUCH

cfgetispeed(), *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcgetattr()*, *tcsetattr()*, *<sys/termio_h>*

NAME

isalnum - test for alphanumeric character
Test auf Buchstabe oder Ziffer

DEFINITION

```
#include <ctype.h>

int isalnum (c)
int c;
```

BESCHREIBUNG

Die Funktion *isalnum()* prüft, ob *c* ein alphanumerisches Zeichen ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*).

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Zeichen größer als oktal 0177 der Wert 0 zurückgeliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isalnum()* liefert genau dann den Wert 0, wenn die Funktionen *isalpha()* und *isdigit()* den Wert 0 liefern würden. Andernfalls liefert sie einen Wert ungleich 0.

Wenn das Argument der Funktion *isalnum()* nicht im Definitionsbereich liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalpha(), *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *<stdio.h>*, *Abschnitt.2.5*.

NAME

isalpha - test for alphabetic character
Test auf Buchstabe

DEFINITION

```
#include <ctype.h>

int isalpha (c)
int c;
```

BESCHREIBUNG

Die Funktion *isalpha()* prüft, ob *c* ein Buchstabe ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*).

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Zeichen größer als oktal 0177 der Wert 0 zurückgeliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isalpha()* liefert einen Wert ungleich 0, wenn *c* ein Buchstabe ist, sonst liefert sie den Wert 0. Wenn das Argument der Funktion *isalpha()* nicht im Definitionsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *<stdio.h>*, *Abschnitt 2.5*.

NAME

isascii - test for 7-bit US-ASCII character
Test auf 7-Bit US-ASCII-Zeichen

DEFINITION

```
#include <ctype.h>

int isascii (c)
int c;
```

BESCHREIBUNG

Die Funktion *isascii()* prüft, ob *c* einen 7-Bit US-ASCII Zeichencode besitzt.

Die Funktion *isascii()* ist für alle ganzzahligen Werte definiert.

ERGEBNIS

Die Funktion *isascii()* liefert einen Wert ungleich 0, wenn *c* einen ASCII-Zeichencode zwischen 0 und 0177 einschließlich besitzt. Andernfalls liefert sie den Wert 0.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

<ctype.h>.

NAME

isatty - test for a terminal device
Test auf Datensichtstation

DEFINITION

```
int isatty (fildes)
int fildes;
```

BESCHREIBUNG

Die Funktion *isatty()* prüft, ob die offene Dateikennzahl *fildes* einer Datensichtstation zugeordnet ist.

ERGEBNIS

Bei erfolgreicher Beendigung liefert die Funktion *isatty()* den Wert 1, wenn *fildes* einer Datensichtstation zugeordnet ist. Andernfalls liefert sie 0 und besetzt *errno*, um den Fehler anzuzeigen.

FEHLER

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

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

[ENOTTY]
: Das Argument *fildes* ist nicht einer Datensichtstation zugeordnet.

HINWEIS

Die Funktion *isatty()* zeigt nicht unbedingt an, ob ein Mensch hinter den Interaktionen über *fildes* steht. Es ist durchaus möglich, daß andere Geräte als Datensichtstationen mit der Kommunikationsleitung verbunden sind (z.B. Drucker oder Magnetband, die über serielle Leitungen mit dem Rechner gekoppelt sind).

Wenn bei einer Datensichtstation die Leitungsprozedur (line discipline) nicht auf sys5 eingestellt ist, dann liefert *isatty()* immer den Wert 0 als Ergebnis (siehe auch *<sys/termio.h>*).

PORTABILITÄT

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

BEISPIEL

Siehe das Beispiel unter *ttyname()*

NAME

iscntrl - test for control characters
Test auf Kontrollzeichen

DEFINITION

```
#include <ctype.h>

int iscntrl (c)
int c;
```

BESCHREIBUNG

Die Funktion *iscntrl()* prüft, ob *c* ein Kontrollzeichen ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*).

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert. Die Kontrollzeichen sind die Zeichen mit einem oktalen Wert kleiner 040 und das Zeichen 0177 (DEL).

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *iscntrl()* liefert einen Wert ungleich 0, wenn *c* ein Kontrollzeichen ist, anderrnfalls liefert sie den Wert 0.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

isdigit - test for decimal digit
Test auf Ziffer

DEFINITION

```
#include <ctype.h>

int isdigit (c)
int c;
```

BESCHREIBUNG

Die Funktion *isdigit()* prüft, ob *c* eine Ziffer ist. Die Ziffern sind die Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, und 9, unabhängig vom verwendeten Zeichensatz.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isdigit()* liefert einen Wert ungleich 0, wenn *c* eine dezimale Ziffer ist, andernfalls liefert sie den Wert 0.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *<ctype.h>*.

NAME

isfirst - test for multibyte character
Test auf Multibyte-Zeichen

DEFINITION

```
#include <ctype.h>

int isfirst (c)
int c;
```

BESCHREIBUNG

Die Funktion *isfirst()* prüft, ob *c* das erste Byte eines Zeichens aus mehreren Bytes ist (Multibyte-Zeichen). Solche Zeichen können vor allem in nationalen Zeichensätzen vorkommen. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch *Abschnitt 2.5*.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für alle Zeichen der Wert 0 zurückgeliefert wird.

ERGEBNIS

Die Funktion *isfirst()* liefert einen Wert ungleich 0, wenn *c* das erste Byte eines Multibyte-Zeichens ist; sonst liefert sie den Wert 0. Wenn das Argument der Funktion *isfirst()* nicht im Definitionsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *<stdio.h>*, *Abschnitt 2.5*.

NAME

isgraph - test for visible character
Test auf sichtbares Zeichen

DEFINITION

```
#include <ctype.h>

int isgraph (c)
int c;
```

BESCHREIBUNG

Die Funktion *isgraph()* prüft, ob *c* ein Zeichen mit einer sichtbaren Darstellung ist (d.h. ein druckbares Zeichen ohne das Leerzeichen in ASCII (040)). Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Werte oberhalb von oktal 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isgraph()* liefert einen Wert ungleich 0, wenn *c* ein Zeichen mit einer sichtbaren Darstellung ist. andernfalls liefert sie den Wert 0. Wenn das Argument der Funktion *isgraph()* nicht im Definitionsbereich der Funktion,liegt, so ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt2.5*.

NAME

islower - test for lower-case letter
Test auf Kleinbuchstabe

DEFINITION

```
#include <ctype.h>

int islower (c)
int c;
```

BESCHREIBUNG

Die Funktion *islower()* prüft, ob *c* ein Kleinbuchstabe ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Zeichen oberhalb von 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *islower()* liefert einen Wert ungleich 0, wenn *c* ein Kleinbuchstabe ist, andernfalls liefert sie den Wert 0. Wenn das Argument der Funktion *islower()* nicht im Definitionsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt2.5*.

NAME

isnan - test for NaN
Test auf NaN

DEFINITION

```
#include <math.h>

int isnan (x)
double x;
```

BESCHREIBUNG

Die Funktion *isnan()* überprüft, ob *x* ein NaN ist. Ein NaN (Not a Number) ist jedes beliebige Bitmuster, das kein gültiges Bitmuster einer Gleitpunktzahl ist.

ERGEBNIS

Die Funktion *isnan()* gibt einen Wert ungleich 0 zuück, falls *x* ein NaN ist. Andernfalls wird 0 zurückgegeben.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Unter SINIX ist z.B. jedes Bitmuster ein NaN, bei dem alle Bits des Exponenten gleich 1 gesetzt sind. Auf einigen Systemen liefert die Funktion *isnan()* immer den Wert 0, d.h. dort sind alle Bitmuster für Gleitpunktzahlen gültig.

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

cc programm.c -lm

PORTABILITÄT

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

SIEHE AUCH

<math.h>.

NAME

isprint - test for printing character
Test auf druckbares Zeichen

DEFINITION

```
#include <ctype.h>

int isprint (c)
int c;
```

BESCHREIBUNG

Die Funktion *isprint()* prüft, ob *c* ein druckbares Zeichen ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Werte oberhalb von oktal 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isprint()* liefert einen Wert ungleich 0, wenn *c* ein druckbares Zeichen ist, andernfalls liefert sie den Wert 0. Wenn das Argument der Funktion *isprint()* nicht im Definitionsbereich liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

ispunct - test for punctuation character
Test auf Interpunktionszeichen

DEFINITION

```
#include <ctype.h>

int ispunct (c)
int c;
```

BESCHREIBUNG

Die Funktion *ispunct()*, prüft, ob *c* ein Interpunktionszeichen ist (d.h ein beliebiges druckbares Zeichen außer dem Leerzeichen (040), den Ziffern und den Buchstaben in US-ASCII).

Die Funktion *ispunct()* klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Werte oberhalb von oktal 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *ispunct()* liefert einen Wert ungleich 0, wenn *c* ein Interpunktionszeichen ist, andernfalls liefert sie den Wert 0. Wenn das Argumnt von *ispunct()* nicht im Definitionsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

isspace - test for "white space" character
Test auf Leerzeichen

DEFINITION

```
#include <ctype.h>

int isspace (c)
int c;
```

BESCHREIBUNG

Die Funktion *isspace()* prüft, ob *c* ein Zeichen ist, das die Ausgabe von Leerezeichen in angezeigtem Text verursacht (d.h. Leerezeichen, Tabulator, Wagenrücklauf, Zeilenvorschub, Vertikaltabulator oder Seitenvorschub in US-ASCII).

Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Werte oberhalb von oktal 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isspace()* liefert einen Wert ungleich 0, wenn *c* ein solches Leerzeichen ist, andernfalls liefert sie den Wert 0. Wenn das Argument von *isspace()* außerhalb des Definitonsbereichs der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

isupper - test for upper-case letter
Test auf Großbuchstabe

DEFINITION

```
#include <ctype.h>

int isupper (c)
int c;
```

BESCHREIBUNG

Die Funktion *isupper()* prüft, ob *c* ein Großbuchstabe ist. Sie klassifiziert als Zeichen codierte ganzzahlige Werte entsprechend den Regeln des Zeichensatzes, der durch die Zeichentyp-Information in der internationalen Umgebung des Programms definiert ist. (Kategorie *LC_CTYPE*), siehe auch Abschnitt 2.5.

In der internationalen Umgebung für die Programmiersprache C oder in einer internationalen Umgebung, für die keine Zeichentyp-Information definiert ist, werden Zeichen nach den Regeln des US-ASCII 7-Bit Zeichensatzes klassifiziert, wobei für Werte oberhalb von oktal 0177 der Wert 0 geliefert wird.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isupper()* liefert einen Wert ungleich 0, wenn *c* ein Großbuchstabe ist, andernfalls liefert sie den Wert 0. Wenn das Argument von *isupper()* nicht im Definitonsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*, *setlocale()*, *<ctype.h>*, *Abschnitt 2.5*.

NAME

isxdigit - test for hexadecimal digit
Test auf Hexadezimalziffer

DEFINITION

```
#include <ctype.h>

int isxdigit (c)
int c;
```

BESCHREIBUNG

Die Funktion *isxdigit()* prüft, ob *c* eine hexadezimale Ziffer ist. Die hexadezimalen Ziffern sind die Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E und F, unabhängig vom verwendeten Zeichensatz.

In allen Fällen ist das Argument *c* vom Typ *int*, dessen Wert als *unsigned char* darstellbar, oder gleich dem Wert des Makros EOF sein muß. Wenn das Argument irgendeinen anderen Wert besitzt, dann ist das Verhalten undefiniert.

ERGEBNIS

Die Funktion *isxdigit()* liefert einen Wert ungleich 0, wenn *c* eine hexadezimale Ziffer ist, andernfalls liefert sie den Wert 0. Wenn das Argument von *isxdigit()* nicht im Definitionsbereich der Funktion liegt, dann ist das Ergebnis undefiniert.

FEHLER

Es sind keine Fehler definiert.

HINWEIS

Um die Portabilität von Anwendungen sicherzustellen, insbesondere über verschiedene Landessprachen, sollten für die Klassifikation von Zeichen nur diese und die im Abschnitt SIEHE AUCH aufgeführten Funktionen verwendet werden.

PORTABILITÄT

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

SIEHE AUCH

isalnum(), *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *<ctype.h>*.

A Anhang

dezi-mal	oktal	hexa-dez.		Bedeutung	Control
0	00	00	NUL	Null, keine Operation	@
1	01	01	SOH	Start of Heading Vorspannanfang	A
2	02	02	STX	Start of Text Textanfang	B
3	03	03	ETX	End of Text Textende	C
4	04	04	EOT	End of Transmission Taste END Übertragungsende	D
5	05	05	ENQ	Enquiry Stationsanruf	E
6	06	06	ACK	Acknowledge Bestätigung	F
7	07	07	BEL	Bell Klingel	G
8	10	08	BS	Backspace Korrekturtaste	H
9	11	09	HT	Horizontal Tabulation Tabulatorzeichen	I
10	12	0A	LF	Line Feed Zeilenvorschub, neue Zeile	J
11	13	0B	VT	Vertical Tabulation	K
12	14	0C	FF	Form Feed Formularvorschub	L
13	15	0D	CR	Carriage Return Wagenrücklauf	M
14	16	0E	SO	Shift Out Umschalten Zeichensatz	N
15	17	0F	SI	Shift In Zurückschalten Zeichensatz	O
16	20	10	DLE	Data Link Escape Austritt aus der Datenverbindung	P
17	21	11	DC1	Device Control 1 Gerätesteuerung 1, Ausgabe fortsetzen	Q
18	22	12	DC2	Device Control 2	R
19	23	13	DC3	Device Control 3 Ausgabe anhalten	S
20	24	14	DC4	Device Control 4	T
21	25	15	NAK	Negative Acknowledge Fehlermeldung	U
22	26	16	SYN	Synchronous Idle Synchronisierung	V
23	27	17	ETB	End of Transm. Block Datenblockende	W
24	30	18	CAN	Cancel ungültig, Zeilenlöscher	X
25	31	19	EM	End of Medium Datenträgerende, quit (Signal3)	Y
26	32	1A	SUB	Substitute Character Zeichen ersetzen	Z

dezimal	oktal	hexadez.		Bedeutung		Control
27	33	1B	ESC	Escape	Rücksprung	
28	34	1C	FS	File Separator	Dateitrennung	\
29	35	1D	GS	Group Separator	Gruppentrennung	]
30	36	1E	RS	Record Separator	Satztrennung	^
31	37	1F	US	Unit Separator	Einheitentrennung	_
32	40	20	SP	SPACE	Leerzeichen	ø oder DEL
33	41	21	!			
34	42	22	"			
35	43	23	#	Nummernzeichen		
36	44	24	$	oder nationales Währungssymbol		
37	45	25	%			
38	46	26	&			
39	47	27	'			
40	50	28	(			
41	51	29	)			
42	52	2A	*	(Stern gilt oft als Multiplikationszeichen)		
43	53	2B	+			
44	54	2C	,			
45	55	2D	-			
46	56	2E	.			
47	57	2F	/	(dient als Divisionszeichen)		
48	60	30	0			
49	61	31	1			
50	62	32	2			
51	63	33	3			
52	64	34	4			
53	65	35	5			
54	66	36	6			
55	67	37	7			
56	70	38	8			
57	71	39	9			
58	72	3A	:			
59	73	3B	;			
60	74	3C	<			
61	75	3D	=			
62	76	3E	>			
63	77	3F	?			
64	100	40	@	(kaufmännisches "at" oder §)		
65	101	41	A			
66	102	42	B			
67	103	43	C			
68	104	44	D			
69	105	45	E			
70	106	46	F			
71	107	47	G			
72	110	48	H			
73	111	49	I			
74	112	4A	J			
75	113	4B	K			
76	114	4C	L			
77	115	4D	M			

dezi-mal	oktal	hexa-dez.		Bedeutung	Control
78	116	4E	N		
79	117	4F	O		
80	120	50	P		
81	121	51	Q		
82	122	52	R		
83	123	53	S		
84	124	54	T		
85	125	55	U		
86	126	56	V		
87	127	57	W		
88	130	58	X		
89	131	59	Y		
90	132	5A	Z		
91	133	5B	[	oder Ä	
92	134	5C	\	Gegenschrägstrich oder Ö	
93	135	5D	]	oder Ü	
94	136	5E	^	oder ↑	
95	137	5F	_	Unterstrich oder >	
96	140	60	`		
97	141	61	a		
98	142	62	b		
99	143	63	c		
100	144	64	d		
101	145	65	e		
102	146	66	f		
103	147	67	g		
104	150	68	h		
105	151	69	i		
106	152	6A	j		
107	153	6B	k		
108	154	6C	l		
109	155	6D	m		
110	156	6E	n		
111	157	6F	o		
112	160	70	p		
113	161	71	q		
114	162	72	r		
115	163	73	s		
116	164	74	t		
117	165	75	u		
118	166	76	v		
119	167	77	w		
120	170	78	x		
121	171	79	y		
122	172	7A	z		
123	173	7B	{	oder ä	
124	174	7C	|	oder ö	
125	175	7D	}	oder ü	
126	176	7E	~	oder ß	
127	177	7F	DEL	Delete Löschzeichen, Interrupt (Signal2)	

Fachwörter englisch - deutsch

absolute pathname	absoluter Pfadname
access	Zugriff
access mode	Zugriffsart
access permission	Zugriffsberechtigung
action	Aktion
address	Adresse
address space	Adreßraum
alarm clock	Alarmuhr
alignment bytes	Füllbytes
APPLICATION USAGE	HINWEIS
appropriate privileges	besondere Rechte
archive	Bibliothek
argument	Argument
assignment	Zuweisung
back space key	Korrekturtaste
backgoound process group	Hintergrund-Prozeßgruppe
backslash	Gegenschrägstrich
base adress	Basisadresse
baud rate	Baudrate
binary file	Binärdatei
binary operator	zweistelliger Operator
block special device	blockorientiertes Gerät
block special file	Gerätedatei für b.o. Gerät
break	break-Anweisung
broken-down time	aufgeschlüsselte Zeit
buffer area	Pufferbereich
buffer	Puffer
buffered I/O	gepufferte Ein- und Ausgabe
byte	Byte = 8 Bit
call	aufrufen, Aufruf
canonical mode input processing	Standard-Eingabeverarbeitung
cast	Typumwandlung mit dem Cast-Operator
cast	cast-Operator
catch	abfangen (Signale)
category	Kategorie
change	ändern
character set	Zeichensatz,Zeichenvorrat
character special device	zeichenorientiertes Gerät
character special file	Gerätedatei für z.o. Gerät
character	Zeichen

child process	Sohnprozeß
cleanup	Bereinigung,Aufräumarbeiten
close-on-exec-bit	sbe-Bit (schließe bei exec)
code in use	aktuell gültiger Zeichensatz
code set	Zeichensatz
collating element	Zeicheneinheit
collating sequence	Sortierreihenfolge
collating symbol	Zeicheneinheits-Symbol
command interpreter	Kommandointerpreter
command prompt	Eingabeaufforderung
compile time	Übersetzungzeit
completion	Beendigung
concatenate	verketten
concatenation	Verkettung
console	Konsole
control character	Kontrollzeichen
control mode (Terminal Interface)	Hardware-Eigenschaften
controlling process	kontrollierender Prozeß
controlling terminal	kontrollierendes Terminal
core (dump|file|image)	Speicherabzug
currency symbol	Währungszeichen
current	aktuell
current working directory	aktuelles Verzeichnis
data transmission	Datenübertragung
decimal	Dezimal-, zur Basis 10
default	Standard-
delete	löschen,entfernen
description	Beschreibung
device	Gerät
device driver	Gerätetreiber
device ID	Gerätenummer
diagnostics	Diagnoseteile
digit groupng symbol	Tausendertrennzeichen
digit	Ziffer
directory	Dateiverzeichnis
directory entry	Dateiverzeichniseintrag
directory stream	Dateiverzeichnis-Strom
disconnect the line	die Verbindung abbrechen
disk	Festplatte
double floating point	doppelt genaue Gleitpunktzahl
driver	Treiber
duration	Dauer
effective group ID	effektive Gruppennummer
effective user ID	effektive Benutzernummer

empty directory	leeres Dateiverzeichnis
empty string	leere Zeichenkette
end-of-file	Dateiende
environment	Umgebung
environment variables	Umgebungsvariablen
Epoch	Epochenwert (1.1.1070, 0:00 Uhr)
equivalence class	Äquivalenzklasse
error	Fehler
error condition	Fehlerbedingung
error handling	Fehlerbehandlung
error number	Fehlernummer
escape character	Fluchtsymbol
escape	entwerten
escape sequence	Escape-Sequenz
execute	ausführen
execute permission	Ausführberechtigung
exit status	Endestatus
expression	Ausdruck
extended security controls	erweiterte Sicherheitskontrollen
FIFO	FIFO
FIFO special file	FIFO-Gerätedatei
file	Datei
file access permissions	Zugriffsrechte für Dateien
file description	Datei-Beschreibung
file descriptor	Dateikennzahl
file descriptor flag	Prozeß-Dateistatus-Byte
file flag	System-Dateistatus-Byte
file group class	Benutzerklasse Gruppe
file hierarchy	Dateibaum
file mode	Dateistatus
file name	Dateiname
file offset	Dateiposition
file other class	Benutzerklasse übrige Benutzer
file owner class	Benutzerklasse Eigentümer
file permission bits	Schutzbits der Datei
file pointer	Dateizeiger
file serial number	Dateinummer
FILE structure	FILE-Struktur
file system	Dateisystem
file times update	Änderung der Dateizeiten
flag	Anzeiger,Schalter,Bit
floating point (number)	Gleitkommazahl,-punktzahl
floppy disk	Diskette

foreground process group	Vordergrund-Prozeßgruppe
foreground process group ID	Vordergrund-Prozeßgruppennr.
formatted I/O	formatierte Ein- und Ausgabe
free	freigeben
functionatity	Funktionalität
group	Gruppe
group ID	Gruppennummer
group name	Gruppenname
header (file)	Include-Datei
hexadecimal	hexadezimal
home directory	HOME-Dateiverzeichnis
implementation	Implemenierung, implementierungs-
include	einbinden, -schließen
include-file	Include-Datei
initialize	initialisieren
input	Eingabe
input mode	Eingabeverarbeitung
input stream	Eingabestrom
integer	ganze Zahl, Ganzzahl
inter-process communication	Interprozeßkommunikation
interface	Schnittstelle
interrupt	Unterbrechung, unterbrechen
issue	Ausgabe
job control	Auftragskontrolle
key	Schlüssel,Taste
link	Verweis
link count	Verweiszähler
local mode (Terminal Interface)	lokale Verarbeitung
locale	internationale Umgebung
macro	Makro
mandatory	obligatorisch
match	passen,gefunden
memory	Speicher(platz)
message (NLS)	Meldung
message catalogue	Meldungskatalog
message catalogue descriptor	Meldungskatalog-Deskriptor
mode (of a file)	Dateistatus
mount	einhängen
NaN (Not a Number)	NaN
native language	Landessprache

new process image file	neue Abbilddatei des Prozesses
nice value	Priorität
node	Knoten,Struktur
non-canonical mode input processing	rohe Eingabeverarbeitung
non-spacing character	Leerzeichen ohne Schreibmarkenbewegung
null byte (\0)	Nullbyte
null character	Nullbyte
null pointer	Nullzeiger
null string	leere Zeichenkette
null terminated string	mit Nullbyte abgeschlossene Zeichenkette
octal	oktal
offset	Offset,Abstand
open file	offene Datei
open file description	geöffnete Dateibeschreibung
open file decriptor	offene Dateikennzahl
ordinary file	Textdatei
orphaned	verwaist
output	Ausgabe
output mode	Ausgabeverarbeitung
output stream	Ausgabestrom
overflow	Überlauf
owner	Eigentümer
parent directory	übergeordnetes Dateiverzeichnis
parent process	Vaterprozeß
parent process ID	Prozeßnummer des Vaterprozesses
path	Pfad
path prefix	Pfadnamen-Anfang
pathname	Pfadname
pathname component	Pfadnamen-Komponente
pathname resolution	Pfadnamen-Auflösung
pattern	Muster
pipe	Pipe
pointer	Zeiger
portable	portabel
portable filename character set	Zeichensatz für portable Dateinamen
portable pathname	portabler Pfadname
preprocessor	Präprozessor
privilege	Recht
process	Prozeß

process group	Prozeßgruppe
process group ID	Prozeßgruppennummer
process group leader	Prozeßgruppenchef
process group lifetime	Lebensdauer einer Prozeßgruppe
process ID	Prozeßnummer
process image	Prozeßabbild
process lifetime	Lebensdauer eines Prozesses
process termination	Prozeßende
program message	Meldungstext
protection bit	Schutzbit
radix character	Dezimalpunkt
read	lesen
read-only file system	nur zum Lesen eingehängtes Dateisystem
read permission	Leseerlaubnis
real group ID	reale Gruppennummer
real user ID	reale Benutzernummer
regular file	normale Datei
regular expression	regulärer Ausdruck
relative pathname	relativer Pfadname
released	freigegeben
request	Anfrage
result	Ergebnis
resume	fortsetzen
return	zurückliefern,-kehren,Rückkehr
root-directory	Root-Dateiverzeichnis
run time	Laufzeit
saved set-group-ID	gesicherte Gruppennummer
saved set-user-ID	gesicherte Benutzernummer
schedule	steuern,festlegen
screen	Bildschirm
search	durchsuchen
search permission	Durchsuchberechtigung
security mechanism	Sicherheitsmechanismus
security policy	Sicherheitsverfahren
session	Sitzung
session leader	Sitzungsführer
session lifetime	Lebensdauer einer Sitzung
set-group-ID mode bit	s-Bit für die Gruppe
set-user-ID mode bit	s-Bit für den Eigentümer
shared text	gemeinsamer Text
sign	Vorzeichen
signal	Signal

signal mask	Signalmaske
size	Größe
slash	Schrägstrich
space character	Zwischenraumzeichen
space	Zwischenraum
special character	Sonderzeichen
special file	Gerätedatei
static data area	statischer Datenbereich
stream	Strom
string	Zeichenkette
structure	Struktur
superuser (root)	Superuser (root)
supplementary group ID	weitere Gruppennummer
suspend (a process)	anhalten (einen Prozeß
synopsis	Definition
system	System
system administrator	Systemverwalter
system call	Systemaufruf
system process	Systemprozeß
system time	Systemzeit
tab	Tabulator(zeichen)
temporary	temporär
terminal (device)	Datensichtstation
territory	Gebiet
timezone	Zeitzone
transmission	Übertragung
transmit	übertragen
underflow	Unterlauf
underscore	Unterstrich
user	Benutzer
user ID	Benutzernummer
user time	Benutzerzeit
value	Wert
variable	variabel, Variable
white space character	Zwischenraumzeichen
withdrawn	zurückgezogen
working directory	aktuelles Datciverzeichnis
write permission	Schreiberlaubnis
zombie process	Zombieprozeß

Fachwörter deutsch - englisch

abfangen (Signale)	catch
absoluter Pfadname	absolute pathname
Adresse	address
Adreßraum	address space
ändern	change
Änderung der Dateizeiten	file times update
Äquivalenzklasse	equivalence class
Aktion	action
aktuell	current
aktuell gültiger Zeichensatz	code in use
aktuelles Dateiverzeichnis	working directory
aktuelles Verzeichnis	current working directory
Alarmuhr	alarm clock
Anfrage	request
anhalten (einen Prozeß	suspend (a process)
Anzeiger,Schalter,Bit	flag
Argument	argument
aufgeschlüsselte Zeit	broken-down time
aufrufen, Aufruf	call
Auftragskontrolle	job control
Ausdruck	expression
Ausführberechtigung	execute permission
ausführen	execute
Ausgabe	issue
Ausgabe	output
Ausgabestrom	output stream
Ausgabeverarbeitung	output mode
Basisadresse	base adress
Baudrate	baud rate
Beendigung	completion
Benutzer	user
Benutzerklasse Eigentümer	file owner class
Benutzerklasse Gruppe	file group class
Benutzerklasse übrige Benutzer	file other class
Benutzernummer	user ID
Benutzerzeit	user time
Bereinigung,Aufräumarbeiten	cleanup
Beschreibung	description
besondere Rechte	appropriate privileges
Bibliothek	archive

Bildschirm	screen
Binärdatei	binary file
blockorientiertes Gerät	block special device
break-Anweisung	break
Byte = 8 Bit	byte
cast-Operator	cast
Datei	file
Datei-Beschreibung	file description
Dateibaum	file hierarchy
Dateiende	end-of-file
Dateikennzahl	file descriptor
Dateiname	file name
Dateinummer	file serial number
Dateiposition	file offset
Dateistatus	file mode
Dateistatus	mode (of a file)
Dateisystem	file system
Dateiverzeichnis	directory
Dateiverzeichnis-Strom	directory stream
Dateiverzeichniseintrag	directory entry
Dateizeiger	file pointer
Datensichtstation	terminal (device)
Datenübertragung	data transmission
Dauer	duration
Definition	synopsis
Dezimal-, zur Basis 10	decimal
Dezimalpunkt	radix character
Diagnoseteile	diagnostics
die Verbindung abbrechen	disconnect the line
Diskette	floppy disk
doppelt genaue Gleitpunktzahl	double floating point
Durchsuchberechtigung	search permission
durchsuchen	search
effektive Benutzernummer	effective user ID
effektive Gruppennummer	effective group ID
Eigentümer	owner
einbinden, -schließen	include
Eingabe	input
Eingabeaufforderung	command prompt
Eingabestrom	input stream
Eingabeverarbeitung	input mode
einhängen	mount
Endestatus	exit status

entwerten	escape
Epochenwert (1.1.1070, 0:00 Uhr)	Epoch
Ergebnis	result
erweiterte Sicherheitskontrollen	extended security controls
Escape-Sequenz	escape sequence
Fehler	error
Fehlerbedingung	error condition
Fehlerbehandlung	error handling
Fehlernummer	error number
Festplatte	disk
FIFO	FIFO
FIFO-Gerätedatei	FIFO special file
FILE-Struktur	FILE structure
Fluchtsymbol	escape character
formatierte Ein- und Ausgabe	formatted I/O
fortsetzen	resume
freigeben	free
freigegeben	released
Funktionalität	functionatity
Füllbytes	alignment bytes
ganze Zahl, Ganzzahl	integer
Gebiet	territory
Gegenschrägstrich	backslash
gemeinsamer Text	shared text
gepufferte Ein- und Ausgabe	buffered I/O
Gerät	device
Gerätedatei	special file
Gerätedatei für b.o. Gerät	block special file
Gerätedatei für z.o. Gerät	character special file
Gerätenummer	device ID
Gerätetreiber	device driver
gesicherte Benutzernummer	saved set-user-ID
gesicherte Gruppennummer	saved set-group-ID
geöffnete Dateibeschreibung	open file description
Gleitkommazahl,-punktzahl	floating point (number)
Gruppe	group
Gruppenname	group name
Gruppennummer	group ID
Größe	size
Hardware-Eigenschaften	control mode (Terminal Interface)
hexadezimal	hexadecimal

Hintergrund-Prozeßgruppe	backgoound process group
HINWEIS	APPLICATION USAGE
HOME-Dateiverzeichnis	home directory
Implemenierung, implementierungs-	implementation
Include-Datei	header (file)
Include-Datei	include-file
initialisieren	initialize
internationale Umgebung	locale
Interprozeßkommunikation	inter-process communication
Kategorie	category
Knoten,Struktur	node
Kommandointerpreter	command interpreter
Konsole	console
kontrollierender Prozeß	controlling process
kontrollierendes Terminal	controlling terminal
Kontrollzeichen	control character
Korrekturtaste	back space key
Landessprache	native language
Laufzeit	run time
Lebensdauer einer Prozeßgruppe	process group lifetime
Lebensdauer einer Sitzung	session lifetime
Lebensdauer eines Prozesses	process lifetime
leere Zeichenkette	empty string
leere Zeichenkette	null string
leeres Dateiverzeichnis	empty directory
Leerzeichen ohne Schreibmarkenbewegung	non-spacing character
Leseerlaubnis	read permission
lesen	read
lokale Verarbeitung	local mode (Terminal Interface)
löschen,entfernen	delete
Makro	macro
Meldung	message (NLS)
Meldungskatalog	message catalogue
Meldungskatalog-Deskriptor	message catalogue descriptor
Meldungstext	program message
mit Nullbyte abgeschlossene Zeichenkette	null terminated string
Muster	pattern
NaN	NaN (Not a Number)

neue Abbilddatei des Prozesses	new process image file
normale Datei	regular file
Nullbyte	null byte (\0)
Nullbyte	null character
Nullzeiger	null pointer
nur zum Lesen eingehängtes Dateisystem	read-only file system
obligatorisch	mandatory
offene Datei	open file
offene Dateikennzahl	open file decriptor
Offset,Abstand	offset
oktal	octal
passen,gefunden	match
Pfad	path
Pfadname	pathname
Pfadnamen-Anfang	path prefix
Pfadnamen-Auflösung	pathname resolution
Pfadnamen-Komponente	pathname component
Pipe	pipe
portabel	portable
portabler Pfadname	portable pathname
Priorität	nice value
Prozeß	process
Prozeß-Dateistatus-Byte	file descriptor flag
Prozeßabbild	process image
Prozeßende	process termination
Prozeßgruppe	process group
Prozeßgruppenchef	process group leader
Prozeßgruppennummer	process group ID
Prozeßnummer	process ID
Prozeßnummer des Vaterprozesses	parent process ID
Präprozessor	preprocessor
Puffer	buffer
Pufferbereich	buffer area
reale Benutzernummer	real user ID
reale Gruppennummer	real group ID
Recht	privilege
regulärer Ausdruck	regular expression
relativer Pfadname	relative pathname
rohe Eingabeverarbeitung	non-canonical mode input processing
Root-Dateiverzeichnis	root-directory
s-Bit für den Eigentümer	set-user-ID mode bit
s-Bit für die Gruppe	set-group-ID mode bit

sbe-Bit (schließe bei exec)	close-on-exec-bit
Schlüssel,Taste	key
Schnittstelle	interface
Schreiberlaubnis	write permission
Schrägstrich	slash
Schutzbit	protection bit
Schutzbits der Datei	file permission bits
Sicherheitsmechanismus	security mechanism
Sicherheitsverfahren	security policy
Signal	signal
Signalmaske	signal mask
Sitzung	session
Sitzungsführer	session leader
Sohnprozeß	child process
Sonderzeichen	special character
Sortierreihenfolge	collating sequence
Speicher(platz)	memory
Speicherabzug	core (dump|file|image)
Standard-	default
Standard-Eingabeverarbeitung	canonical mode input processing
statischer Datenbereich	static data area
steuern,festlegen	schedule
Strom	stream
Struktur	structure
Superuser (root)	superuser (root)
System	system
System-Dateistatus-Byte	file flag
Systemaufruf	system call
Systemprozeß	system process
Systemverwalter	system administrator
Systemzeit	system time
Tabulator(zeichen)	tab
Tausendertrennzeichen	digit groupng symbol
temporär	temporary
Textdatei	ordinary file
Treiber	driver
Typumwandlung mit dem Cast-Operator	cast
übergeordnetes Dateiverzeichnis	parent directory
Überlauf	overflow
Übersetzungzeit	compile time
übertragen	transmit
Übertragung	transmission
Umgebung	environment

Umgebungsvariablen	environment variables
Unterbrechung, unterbrechen	interrupt
Unterlauf	underflow
Unterstrich	underscore
variabel, Variable	variable
Vaterprozeß	parent process
verketten	concatenate
Verkettung	concatenation
verwaist	orphaned
Verweis	link
Verweiszähler	link count
Vordergrund-Prozeßgruppe	foreground process group
Vordergrund-Prozeßgruppennr.	foreground process group ID
Vorzeichen	sign
weitere Gruppennummer	supplementary group ID
Wert	value
Währungszeichen	currency symbol
Zeichen	character
Zeicheneinheit	collating element
Zeicheneinheits-Symbol	collating symbol
Zeichenkette	string
zeichenorientiertes Gerät	character special device
Zeichensatz	code set
Zeichensatz für portable Dateinamen	portable filename character set
Zeichensatz,Zeichenvorrat	character set
Zeiger	pointer
Zeitzone	timezone
Ziffer	digit
Zombieprozeß	zombie process
Zugriff	access
Zugriffsart	access mode
Zugriffsberechtigung	access permission
Zugriffsrechte für Dateien	file access permissions
zurückgezogen	withdrawn
zurückliefern,-kehren,Rückkehr	return
Zuweisung	assignment
zweistelliger Operator	binary operator
Zwischenraum	space
Zwischenraumzeichen	space character
Zwischenraumzeichen	white space character

Abkürzungen

Die englischsprachige Bezeichnung ist in Klammern gesetzt.

<datei>	
<sys/datei>	Abkürzende Schreibweise für Include-Dateien, deren vollständiger Pfadname lautet: */usr/include/datei* bzw. */usr/include/sys/datei*
bzw.	beziehungsweise
CPU	Zentraleinheit (central process unit)
d.h.	das heißt
DVZ	Dateiverzeichnis
E/A	Ein-/Ausgabe
egid	effektive Gruppennummer (effective group ID)
EOF	Datei-Ende, in <stdio.h> definiert (End Of File)
euid	effektive Benutzernummer (effective user ID)
FIFO	Abkürzung für first-in-first-out
GID, gid	Gruppennummer, gemeint ist die reale Gruppennummer ((real) group ID)
GMT	Abkürzung für Greenwich Mean Time
ID	Identifikations-Nummer (identification)
I/O	Ein-/Ausgabe (input/output)
i. allg.	im allgemeinen
i.d.R.	in der Regel
MET	Abkürzung für Mean European Time
MEZ	Mitteleuropäische Zeit
o.g.	oben genannt
PID, pid	Prozeßnummer (Process ID)
PPID, ppid	Vaterprozeßnummer (Parent Process ID)
r-Bit	Schutzbit für Leseberechtigung (read)
s.	siehe
s-Bit	Schutzbit, um einem Programmanwender die effektive Benutzernummer des Programmeigentümers während des Programmablaufs zu verleihen (set-user-ID Bit)
sbe-Bit	"schließe-bei-exec-Bit"; früher verwandte Bezeichnung für das close-on-exec-Bit
s.o.	siehe oben
sog.	sogenannte, -r, -s
s.u.	siehe unten
t-Bit	Sticky-Bit
usw.	und so weiter

UID, uid	Benutzernummer, gemeint ist die reale Benutzernummer ((real) user ID)
u.U.	unter Umständen
vgl.	vergleiche
w-Bit	Schutzbit für Schreibberechtigung (write)
wg.	wegen
x-Bit	Schutzbit für Ausführberechtigung (execute)
z.B.	zum Beispiel

Literatur

SINIX-Handbücher

Die mit * gekennzeichneten Titeln sind nicht von der Siemens AG herausgegeben.

[1] Betriebssystem SINIX V5.22
Kommandos

[2] Betriebssystem SINIX V5.22
SINIX-Schnittstellen
Benutzerhandbuch

[3] Betriebssystem SINIX V5.22
Einführung

[4] Betriebssystem SINIX V5.22
Systemverwaltung

[5] Internationalisation in SINIX
Benutzerhandbuch

X/OPEN Guide

*

[6] X/OPEN Portability Guide (December 1988)
Issue 3, Volumes 1 - 7
Eaglewood Cliffs: Prentice Hall 1989

Literatur zu UNIX

* T. Baggenstos, R. Marty u.a.
UNIX als Basis für Softwareentwicklung
Berlin, Heidelberg, New York, Tokyo: Springer 1986

* S. R. Bourne
Das UNIX System
o.O.: Addison-Wesley Verlag (Deutschland) GmbH 1985

* Jürgen Gulbins
UNIX
Berlin: Springer-Verlag , 2. Auflage 1985

* M. J. Rochkind
Advanced UNIX Programming
Englewood Cliffs: Prentice Hall 1985

* R. Thomas, L. R. Rogers, J. L. Yates
Advanced Programmer's Guide To UNIX System V
Berkeley: Osborne McGraw-Hill 1986

* M. J. Bach
The Design Of The UNIX System
Englewood Cliffs: Prentice Hall 1986

Literatur zur Sprache C

* H. Herold, W. Unger
Das C-Buch
München: te-wi Verlag 1986

B. W. Kernighan, D. M. Ritchie
Programmieren in C
Zweite Ausgabe - ANSI C
Coedition Hanser/Prentice Hall 1990
Sonderausgabe SIEMENS AG 1990

C. L. Tondo, S. E. Gimpel
Das C-Lösungsbuch zu Kernighan/Ritchie, Programmieren in C
Zweite Ausgabe - ANSI C
Coedition Hanser/Prentice Hall 1990
Sonderausgabe SIEMENS AG 1990

* T. Plum
Das C-Lernbuch
München, Wien: Hanser, London: Prentice Hall 1985

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis Datentechnik*. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen Datentechnik*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Siemens-Zweigniederlassung; außerhalb der Bundesrepublik Deutschland hilft Ihnen die zuständige Siemens-Vertretung.

Stichwörter

Bestell-Nr. des leeren Ordners U3764-J-Z18-1 (950)

SIEMENS

Betriebssystem SINIX V5.22
CES C-Entwicklungssystem
Teil 1

SIEMENS

SINIX Computer

Daten- und Informationstechnik
Data and Information Systems