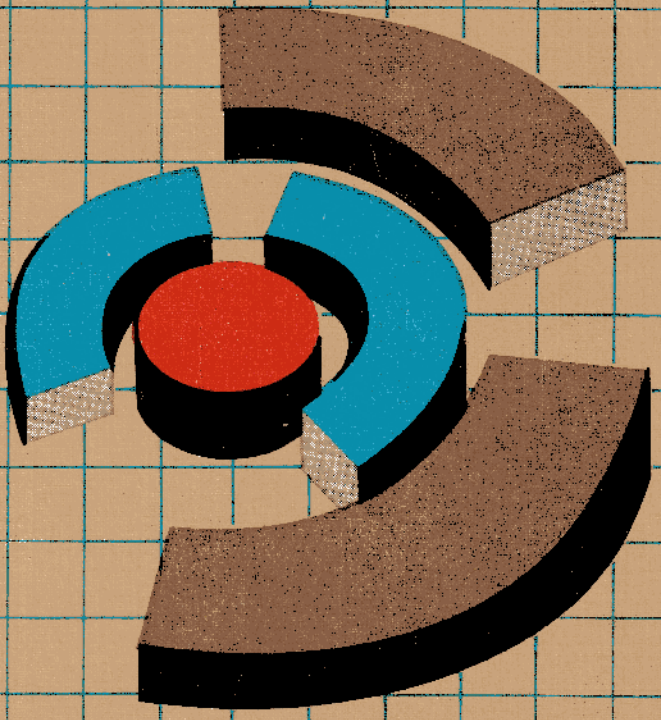
Technische Informatik

Claßen · Oefler

UNIX und C

Ein Anwenderhandbuch



VEB Verlag Technik Berlin

Claßen · Oefler
UNIX und C

Technische Informatik

Herausgegeben von

Dr.-Ing. Ludwig Claßen

Dr.-Ing. Dieter Nedo

Dr. rer. nat. Gerhard Paulin

UNIX und C

Ein Anwenderhandbuch

Dr.-Ing. Ludwig Claßen
Dipl.-Math. Ulrich Oefler

3., durchgesehene Auflage

VEB VERLAG TECHNIK BERLIN

Claßen, Ludwig:
UNIX und C : e. Anwenderhandbuch/Ludwig Claßen ;
Ulrich Oefler. - 3., durchges. Aufl. - Berlin :
Verl. Technik, 1990. - 234 S. : 17 Bilder, 5 Taf.
- (Technische Informatik)
NE: Oefler, Ulrich:

ISBN 3-341-00863-2
ISSN 0863-0860

3., durchgesehene Auflage

VT 201 · 3/5905-3 (6)
Printed in the German Democratic Republic
Druck und buchbinderische Verarbeitung: (140) Druckerei Neues
Deutschland, Berlin
Lektor: Jürgen Reichenbach
Einbandgestaltung: Gabriele Schwesinger
LSV 3054
Bestellnummer: 554 275 1
02400

Vorwort

Ein wesentliches Instrument zur Beschleunigung des wissenschaftlich-technischen Fortschritts ist der breite Einsatz der modernen Computertechnik.
Die Entwicklung der Computertechnik ist in den letzten Jahren durch eine enorme Leistungssteigerung auf dem Hardwaresektor gekennzeichnet. Das gilt nicht nur für Großrechner oder Super-EDV-Anlagen, sondern besonders auch für dezentral eingesetzte Arbeitsplatzcomputersysteme. Durch die Einführung hochintegrierter Mikroprozessor- und Speicherchips (16-/32-Bit-MP; 64-/256-KBit-DRAM) sowie durch den Einsatz moderner Floppy-Disk-/Hard-Disk-Peripherietechnik mit großer Speicherkapazität ermöglichen Arbeitsplatzcomputer heute Problemlösungen, die früher nur Großrechnern vorbehalten waren: die Ausführung komplizierter wissenschaftlicher Berechnungen, die Lösung von Problemen, die bei der Leitung und Planung von ökonomischen Prozessen anfallen, die rechnergestützte Forschung, Entwicklung und Fertigung (CAD/CAM) und vieles andere mehr.

Einen entscheidenden Einfluß auf die effektive Nutzung der modernen Computersysteme hat das Betriebssystem und die zur Problemlösung eingesetzte Programmiersprache.
Auf diesem Gebiet hat in den letzten Jahren wohl kaum eine Entwicklung so eine Bedeutung erlangt, wie die außerordentlich breite Nutzung des Betriebssystemkonzeptes UNIX und der problemorientierten Sprache C.
Arbeiten zur Implementation von UNIX-kompatiblen Betriebssystemen auf in der DDR verfügbaren Großrechnern (ESER-Reihe), Kleinrechnern (SKR-Reihe) und Mikrorechnern (U8000, K1810WM86) wurden richtungweisend ab 1982 im Zentralinstitut für Kybernetik und Informationsprozesse der AdW/Berlin (MUTOS-U), im VEB Leitzentrum für Anwendungsforschung/Berlin (PSU), im VEB Kombinat Robotron/Dresden (MUTOS-1600; MUTOS-7100) und im Zentrum für Forschung und Technologie des Kombinates EAW Berlin-Treptow "Friedrich Ebert" (WEGA/P8000) durchgeführt.
Im Ergebnis dieser Arbeiten steht dem Anwender in der DDR heute eine abgestufte Palette von UNIX- und C-kompatiblen Programmierumgebungen auf Rechnersystemen aller Leistungsklassen zur Verfügung.

Der vorliegende Band der Fachbuchreihe TECHNISCHE INFORMATIK soll in übersichtlicher Form über den Aufbau des Betriebssystems UNIX, über die UNIX-Kommandosprache Shell und über die Programmiersprache C informieren. Die Darlegungen wurden unter dem Aspekt erarbeitet, sowohl die Einarbeitung in UNIX und C als auch die ständige Nutzung zu erleichtern.

Nach der Einleitung, die die Historie und den Standort von UNIX in der großen Familie der Betriebssysteme beschreibt, behandelt der zweite Abschnitt die UNIX-Basiskonzepte (Datei-, Prozeß- und Ein-/Ausgabeverwaltungssystem).
Der dritte Abschnitt vermittelt, ausgehend von den Hardwarekomponenten eines typischen UNIX-Rechners auf Mikroprozessorbasis, Kenntnisse über die Systeminitialisierung. Dabei werden die sogenannten 'Standalone'-Programme und der routinemäßige Start eines UNIX-Systems beschrieben.
Der vierte Abschnitt enthält eine Auswahl von Standardoperatorprozeduren. Er soll vor allem die schnelle Einarbeitung an einem vom Leser nutzbaren UNIX-Terminalarbeitsplatz unterstützen.
Der fünfte Abschnitt stellt eine Zusammenstellung der wichtigsten UNIX-Systemkommandos dar. Die für den Anwender wichtigen Grundkommandos werden umfassend mit allen Optionen dargestellt. Weniger wichtige Kommandos werden überblicksmäßig behandelt.
Im sechsten Abschnitt werden Kenntnisse über die Möglichkeiten der Kommandosprache Shell vermittelt. Sie stellt die Standardschnittstelle des UNIX-Betriebssystems zum Benutzer dar.
Der siebente Abschnitt ist eine Beschreibung der Programmiersprache C. Dabei werden in kompakter Form die Sprachelemente von C und eine Auswahl wichtiger Standardbibliotheksfunktionen beschrieben. Den Abschluß bildet die Darlegung der Handhabung des C-Compilers unter dem Betriebssystem UNIX.

Der Band ist sowohl zur Einarbeitung geeignet als auch für die Aus- und Weiterbildung an den Universitäten, Hoch- und Fachschulen verwendbar. Dabei werden Grundlagenkenntnisse der Programmierung und des Aufbaus von Betriebssystemen vorausgesetzt. Für den erfahrenen UNIX-Anwender soll er den Charakter eines Nachschlagewerkes mit hoher Informationsdichte haben.

Besonderer Dank gilt dem Herausgeber, Herrn Dr. Paulin, für viele wertvolle Hinweise, dem verantwortlichen Lektor des VEB Verlag Technik, Herrn J. Reichenbach, für die außerordentlich kooperative Zusammenarbeit sowie all den Fachkollegen des ZFT-KEAW und des ZKI/AdW, die durch vielfältige Anregungen, durch Diskussionen und durch gemeinsame Arbeit am Zustandekommen dieses Bandes ihren Anteil haben.

Ludwig Claßen Ulrich Oefler

Inhaltsverzeichnis

1. Einleitung

Die Einleitung vermittelt einige Fakten über die Entstehungsgeschichte des heute weltweit verbreiteten Standardbetriebssystems UNIX, es wird sein Standort in der großen Familie der Computerbetriebssysteme bestimmt, es werden die Aufgaben eines Betriebsystems erläutert und es werden die Hauptmerkmale benannt, die das UNIX-System besonders auszeichnen.

1.1. Entwicklung des UNIX-Betriebssystemkonzeptes

Die Entwicklung des Betriebssystemkonzeptes UNIX nahm im Jahre 1969/70 ihren Anfang. Ausgangspunkt war die Unzufriedenheit mit der vom Rechnerhersteller angebotenen Betriebssystemsoftware für einen Minicomputer. Zielstellung des sich schnell entfaltenden Projektes war die Schaffung eines universellen Betriebssystems für Kleinrechner mit standardisierter Benutzerschnittstelle. Außerdem sollte die Zugänglichkeit des Rechners für den Programmierer durch interaktiven Mehrbenutzerbetrieb (Mensch-Maschine-Dialog) erhöht und das koordinierte Zusammenwirken eines ganzen Teams von Softwarebearbeitern während einer Produktentwicklung unterstützt werden.
Die hauptsächlich von Thompsen und Ritchie [1] [6] vorangetriebenen Arbeiten wurden zunächst vorwiegend auf der Basis von Minicomputern durchgeführt. Erreicht wurde die Zielstellung des Projektes durch die Einarbeitung einer ganzen Reihe von richtungweisenden neuen Basiskonzepten (hierarchische Dateiverwaltungsstruktur; Ein-/Ausgaberedirektion u.a.m.), durch eine hohe Modularisierung des gesamten Systems und durch die konsequente Anwendung der parallel entwickelten maschinenunabhängigen höheren Programmiersprache 'C'. Etwa 95% der Systemsoftware von UNIX wurden in C geschrieben und nur 5% des auf einen konkreten Rechner Bezug nehmenden Teils von UNIX bedient sich der jeweiligen maschinenabhängigen Assemblersprache.
Das entstandene Betriebssystem, auf den Namen 'UNIX' getauft, setzte sich sehr schnell bei vielen Minicomputeranwendern durch, vor allem in Lehre und Forschung. Bis 1978 wurde auf mehr als 600 Computeranlagen das Betriebssystem UNIX installiert. Das führte in der Folge zu einer Kettenreaktion, bezogen auf die Verbreitung dieses Softwaresystems.
Es fand und findet seine Anwendung natürlich in ingenieurtechnischen und naturwissenschaftlichen Bereichen, stark zunehmend aber auch bei der Rationalisierung von Verwaltungsprozessen im Büro, bei der Textverarbeitung, bei CAD/CAM-Aufgabenstellungen und bei vielen anderen Anwendungen, die durch den Computereinsatz effektiv unterstützt werden können (Tafel 1.1).
Für die schnelle Verbreitung des Systems sorgten nicht nur seine objektiv vorhandenen Vorzüge, sondern auch ein geschicktes UNIX-Lizenzvergabemarketing durch die Urheber (UNIX ist ein eingetragenes Warenzeichen und der originale UNIX Quell- und Maschinenkode ist durch das Copyright-Urheberrecht vor unlizensierter Weitergabe geschützt). Universitäten und Hochschulen wurden umsonst mit UNIX bedacht. Von kommerziellen Anwendern verlangte man dagegen eine Lizenzgebühr für seine Nutzung.
Neue starke Impulse für die Applikation von UNIX und C gingen ab 1981 von der breiten Einführung sehr leistungsfähiger 16-Bit-Mikroprozessoren, hochintegrierter 64-KBit-Speicherchips sowie 5 1/4-Zoll- und 8-Zoll-Winchesterplattenlaufwerken aus.

Jahr	UNIX-Version	Rechnertyp	Bemerkungen
1969	UNIX Version 1	PDP 7	Beginn der Arbeiten am UNIX Projekt mit ersten Assemblerprogrammen.
1970	UNIX Version 2	PDP 11/20	Übergang zur Sprache 'B'. Der Name 'UNIX' entsteht.
1973	UNIX Version 5		Übergang zur Sprache 'C'.
1975	UNIX Version 6		Der Vertrieb von UNIX beginnt.
1978	UNIX Version 7	PDP 11/70	Version 7 wird die erste UNIX-Standardversion.
1980	UNIX Version 7	Z8000 M68000	UNIX-Implementation auf 16-Bit-Mikroprozessoren.
1982	UNIX System III		Verfügbarkeit einer neuen Standardversion.
1984	UNIX System V/2		Schaffung einer einheitlichen UNIX-Version für unterschiedliche Mikroprozessortypen.

Tafel 1.1. Entwicklung des Betriebssystems UNIX

Heute kann festgestellt werden, daß das Time-Sharing-Betriebssystem UNIX und seine maschinenunabhängige Implementierungsprogrammiersprache C zu einem Standard bei Klein- und Mikrorechnern geworden ist, der von allen führenden Bauelemente- und Systemherstellern unterstützt wird.
Im Ergebnis dieser Entwicklung entstand und entsteht eine Fülle von in C für das Betriebssystem UNIX geschriebener Software, die für ganz unterschiedliche Aufgabenklassen Programme bereitstellt. Durch ihre Anpassung an eine einheitliche Systemschnittstelle (UNIX-Systemcalls) und durch ein einheitliches Sprachkonzept (HLL-Sprache 'C' / HLL high level language) ist diese Software auf ganz unterschiedlichen Rechnersystemen vom Mikroprozessor (M68000, Z8000, iAPX286) über Kleinrechner (PDP11, VAX11) bis zum Großrechner (IBM, AMDAHL) lauffähig.
Das Anwenderspektrum reicht heute von Prozeßsteuerungen durch UNIX-Mikrorechner über UNIX-gesteuerte Nachrichtennetze, bis zum 'automatischen Büro' und CAD/CAM-Entwurfssystemen. Für Mikrorechnerentwicklungssysteme hat sich UNIX zum internationalen Standard schlechthin entwickelt.
Für die umfangreiche Verbreitung (Tafel 1.2), die das Betriebssystem UNIX gefunden hat, sind schwerpunktmäßig wohl vor allem drei Gründe zu nennen:

- UNIX ist ein sehr leistungsfähiges, umfangreich erprobtes Multi-User-/Time-Sharing-Betriebssystem, dessen Systemphilosophie modernsten Auffassungen und Anforderungen entspricht.

- UNIX bietet durch konsequent durchgängige Anwendung der maschinenunabhängigen Programmiersprache C und durch volle Transparenz der verbleibenden hardwarebezogenen Assemblerprogrammteile (Ein-/Ausgabe, Memory Management ...) günstigste

Voraussetzungen zur Implementation auf unterschiedlichsten Hardwarekonfigurationen vom Mikrorechner bis zum Großrechner.

- UNIX stellt das erste Software-Betriebssystem in der Computergeschichte dar, das alle Rechnerklassen (Mikrocomputer Minicomputer; Großrechner) überstreicht.

- UNIX wird nicht von einem einzelnen Hersteller beherrscht und ist nicht an einen einzelnen Rechnertyp gebunden.

An dieser Stelle noch einige Bemerkungen zu der etwas verwirrenden Namensgebung von UNIX-Betriebssystemprodukten.
Es ist grundsätzlich zu beachten, daß der Name UNIX als Bezeichnung für ein Computerbetriebssystem nur für originale Produkte des die Urheberrechte besitzenden Rechtsträgers (Bell Systems Software /1/,/2/,/3/) zulässig ist.
Wurde von diesem Rechtsträger eine UNIX-Weitergabelizenz an eine andere Firma oder Institution vergeben, darf der ursprüngliche Name nicht mehr benutzt werden. Das übernommene, notwendigenfalls etwas modifizierte und auf andere CPU-Systeme (CPU central processing unit) zugeschnittene UNIX muß einen neuen Namen bekommen (XENIX von Microsoft für I8086, OS/1 von Onyx für Z8000, MUNIX von PCS für M68000 ...).

	1981	1982	1983	1984	1985	1986
UNIX auf Großrechnern	9	20	115	248	325	370
UNIX auf Minirechnern	1310	2200	3375	5050	6710	7690
UNIX auf Mikrorechnern	1320	23710	110600	269800	414800	495500
Summe	2639	25930	114090	275098	421835	503560

Tafel 1.2. Verbreitung des UNIX-Betriebssystems nach /8/ (In der Tafel wurde nur der Vertrieb von UNIX-Originalversionen berücksichtigt.)

Eine dritte Gruppe von UNIX-Anwendern arbeitet lizenzfrei mit eigenen Softwareprodukten, die mehr oder weniger 100%ig die originalen UNIX-Basiskonzepte in selbst geschriebenen Programmen übernehmen (unlicensed reverse-engineered). Diese Betriebssysteme, meist als 'UNIX kompatibel' oder 'UNIX like' bezeichnet, werden, bezogen auf das Vorbild, weder von Warenzeichen- noch von Lizenzproblemen tangiert (COHERENT von Mark Williams für PDP-11, MUTOS von Robotron für K1600, IDRIS von Whitesmiths für M68000, WEGA von ZFT/KEAW für P8000 ...).

1.2. Standort von UNIX als Computerbetriebssystem

Ein Betriebssystem ist das Programm eines digitalen Rechnersystems, das, zusammen mit den Hardwareeigenschaften, die Basis seiner möglichen Betriebsarten bildet und die Abwicklung aller Dienst- und Anwenderprogramme steuert und überwacht [7]. Es ist die Schnittstelle des Rechnernutzers zum Computer und verwaltet die Hardware- und Softwareressourcen des Systems. Außerdem stellt es für Dienst- und Anwenderprogramme ein Softwareinterface zu den Peripheriegeräten und Systemprogrammen zur Verfügung. Die Eigenschaften eines Betriebssystems sind in ihren grundsätzlichen Funktionen für alle Rechnersysteme gleich, Sie unterscheiden sich ausschließlich in bestimmten Teilkomponenten, mit denen sie an spezielle Aufgabenkomplexe angepaßt werden.
Betriebssysteme lassen sich nach verschiedenen Gesichtspunkten systematisieren, sowohl in bezug auf die Anwendungsgebiete als auch in bezug auf ihre Arbeitsweise. Im Zusammenhang mit UNIX sind Echtzeitbetriebssysteme, Single-User-Betriebssysteme und Multi-User-Betriebssysteme bedeutsam.

1.2.1. Echtzeitbetriebssysteme (realtime operating systems)

Echtzeitbetriebssysteme sind definitionsgemäß dadurch gekennzeichnet, daß der Rechner, auf dem sie abgearbeitet werden, auf echte, real existierende zeitkritische Ereignisabläufe in seiner Umwelt reagieren muß. Als ein entscheidendes Maß für die Leistungsfähigkeit eines Computers mit einem Echtzeitbetriebssystem ist deshalb die Reaktionszeit anzusehen (im Mikrosekundenbereich), die beim Umschalten (task switch) von einem Prozeß (Programm) mit niedrigerer Priorität auf einen Prozeß (Programm) mit höherer Priorität (z.B. Uhrimpuls, Bedienung einer Ein-/Ausgabeaktivität, Alarmmeldung ...) vergeht.
Unter Prozeß soll hier und im folgenden eine Softwareumgebung verstanden werden, die für die Abarbeitung eines bestimmten Programmes kennzeichnend ist (Programmkode, Registerinhalte, Stack- und Datenspeicherbereiche, CPU-Status usw.).
Das Betriebssystem UNIX ist standardmäßig nicht auf Echtzeitanforderungen zugeschnitten. Seine Prozeßwechselalgorithmen und Reaktionszeiten sind an die Reaktionszeiten des Menschen an einer Bedienkonsole, an intelligente Steuerungen für Festplattenlaufwerke, an Drucker u.s.w. angepaßt. Der Mensch kann nötigenfalls auch einmal einige Sekunden warten, bis auf seine Aktion eine Reaktion vom Computersystem folgt.
Trotzdem existieren spezielle UNIX-Adaptionen, die nach Aussagen ihrer Urheber echtzeitfähig sind (z.B.: UNOS Charles River Data Systems, MASSCOMP Littleton, VENIX VenturCom [9]).

1.2.2. Einplatzbetriebssysteme (single user operating systems)

Die Single-User-Betriebssysteme stellen die hauptsächliche Softwarebasis für die heute weltweit massenhaft verbreiteten Arbeitsplatzcomputer, aber auch für viele Mikrorechnerentwicklungssysteme, für Kassenterminals und anderes mehr dar. Sie beruhen in den meisten Fällen als Kernstück auf einem 8- oder 16-Bit-Standardmikroprozessorsystem, mit 64 bis 256 KByte RAM-Speicher (RAM random access memory / Schreib-/Lesespeicher) und Floppy-Disk als externem Massendatenspeicher. Durch ein solches Betriebs-

system wird jeweils nur ein Anwender bei der sequentiellen Abarbeitung seiner Aufgaben unterstützt. Eine quasisimultane Mehrfachverarbeitung von Programmen (multi task) findet, wenn angewendet, nur für nebengeordnete Aufgaben (Drucken, serielle Datenübertragung ...) statt. Die Leistungsfähigkeit eines Single-User-Betriebssystems wird gemessen an der Leistungsfähigkeit seines Dateiverwaltungssytems, an der Nutzerfreundlichkeit seiner Kommandosprache und an der Flexibilität, mit der man es an veränderte Umgebungsbedingungen anpassen kann.
Das bekannteste Single-User-Betriebssystem für Mikrorechner mit der größten internationalen Verbreitung ist ohne Frage CP/M (Digital Research).
Obwohl UNIX nicht primär für den Single-User-Einsatzfall konzipiert wurde, existieren abgerüstete Versionen, die, speicherplatzoptimiert mit Festplattenlaufwerken geringer Kapazität (5 1/4-Zoll-Winchester-Hard-Disk / manchmal auch nur mit Floppy-Disk-Laufwerken), die Abarbeitung von Programmen in einer UNIX-Softwareumgebung ermöglichen (z.B.: OS-1 Software Labs für Z80, MARC Vortex Technology für Z80, LSX Lycklama für LSI-11, MUTOS-U ZKI/IfR/Robotron für BC5120.16 mit U8000 [47,107]).

1.2.3. Mehrplatzbetriebssysteme (multi user operating systems)

Multi-User-Betriebssysteme unterstützen parallel mehrere Anwender, die über unterschiedliche Bedienerterminals an ein zentrales Computersystem angeschlossen sind. Durch die Anwendung des Time-Sharing-Verfahrens, das jedem Anwender meist rein zyklisch, manchmal allerdings auch zyklisch prioritätsgesteuert, eine feste oder auch variable Zeitscheibe der CPU zuteilt, bekommt jeder Nutzer einen Teil der Rechenzeit des Systems zugewiesen. Auf Grund der hohen Rechengeschwindigkeit ist für den einzelnen Nutzer die Zeitteilung nicht oder kaum spürbar. Die übrigen Systemressourcen, wie Zugriff zum externen Massendatenspeicher, Druckerausgaben, Verfügbarkeit von bestimmten Ein-/Ausgabekanälen usw., werden prioritätsgerecht unter den Nutzern aufgeteilt.
Das Time-Sharing-Verfahren bildet die Grundlage für jedes Programm-Multitasking, bei dem mehrere Programme quasisimultan von einer CPU abgearbeitet werden. Es ist die Voraussetzung für ein Multi-User-Betriebssystem, wo mehrere Anwender quasisimultan, bei gemeinsamer Nutzung der Systemressourcen, an einer Rechnerzentraleinheit arbeiten können.
Wichtig für die Anwendung von Multi-User-/Time-Sharing-Betriebssystemen sind hohe Rechenleistung der CPU, ein hinreichend großer Hauptspeicher (mindestens 256 KByte ... bis zu 4 MByte) und große und schnelle externe Direktzugriffsmassenspeicher (Hard-Disk), um die Programmabarbeitung und die Prozeßumschaltungen zeitlich zu optimieren.
UNIX wurde primär als Multi-User-/Time-Sharing-Betriebssystem konzipiert und entwickelt. Es besitzt, wie noch zu zeigen ist, eine Fülle von Merkmalen und Eigenschaften, die die quasisimultane Arbeit mehrerer Nutzer an einem Rechner effektiv unterstützen.

1.3. Aufgaben eines Computerbetriebssystems

Die grundsätzlichen Aufgaben eines Computerbetriebssystems sind:

- Verwaltung von Prozessen
- Verwaltung des internen Hauptspeichers
  Verwaltung von Datenbeständen (Dateien) auf externen Speichermedien (insbesondere Direktzugriffsdatenbestände)
- Verwaltung der Ein-/Ausgabekanäle

Ein Computerbetriebssystem verwaltet alle wesentlichen Systemressourcen eines Rechners und teilt sie, den Forderungen und Möglichkeiten entsprechend, unterschiedlichen Prozessen zu. Da jede der vorhandenen Ressourcen immer nur jeweils einem Prozeß zugeordnet werden kann, müssen sie in ihrer Verfügbarkeit steuerbar sein. Bei Überschreitung der Möglichkeiten des Systems (mehr Anforderungen als momentan verfügbare Möglichkeiten) muß nach einer im Betriebssystem festgelegten Strategie eine sequentielle Bedienung erfolgen.

1.3.1. Prozeßverwaltung

Die Prozeßverwaltung bestimmt in entscheidendem Maße die Leistungsfähigkeit eines Betriebssystems. Sie muß den Ablauf der Bearbeitung unterschiedlicher Prozesse so steuern, daß Zeiten, in denen die Computerzentraleinheit nutzlos auf das Eintreten eines Ereignisses wartet, minimiert werden, solange noch Nutzeranforderungen vorliegen.
Die Prozeßverwaltung eines Time-Sharing-Betriebssystems muß außerdem eine Zeitscheibenzuteilung (time sharing) für die verschiedenen Anwenderprozesse übernehmen, so daß eine nach außen quasisimultane Arbeitsweise des Rechners entsteht.
Der Rechner hat in der Regel mehr als einen internen Prozeß gleichzeitig zur Abarbeitung zur Verfügung. Wird die Abarbeitung eines Prozesses unterbrochen, weil auf ein externes Ereignis gewartet werden muß oder weil seine Zeitscheibe abgelaufen ist, wird an einem anderen Prozeß weitergearbeitet. Externe Ereignisse sind zum Beispiel die Beendigung von Ein-/Ausgabeaktivitäten, das Erreichen bestimmter Zustände anderer Prozesse und vieles andere mehr.
Basis der Prozeßverwaltung sind Funktionen zur Sicherung der Prozeßstatusinformationen bei der Unterbrechung und Wiederfortsetzung eines Prozesses sowie die Unterbrechung und Neuzuweisung von Prozessen (task switch) nach festgelegten Kriterien (sowohl zu im Prozeß festgelegten Zeitpunkten als auch vom Betriebssystem initiiert).
Ferner erfordert die quasisimultane Abarbeitung von mehreren Prozessen ihre sorgfältige Steuerung und Überwachung zur Vermeidung von Prozeßverklemmungen und Zugriffskonflikten mit Hilfe der Prozeßkoordination und der Prozeßsynchronisation.
Die Prozeßkoordination realisiert die sequentielle Abarbeitung von zueinander kritischen Aktionen, die nicht parallel ablaufen dürfen (z.B.: gleichzeitiges Lesen und Schreiben ein und desselben Datensatzes auf einen externen Massendatenspeicher).
Die Prozeßsynchronisation realisiert erforderlichenfalls die gesteuerte sequentielle Abarbeitung von mehreren Prozessen (z.B.: erst muß ein Datensatz vom Floppy-Disk gelesen - danach kann er erst gedruckt werden).

1.3.2. Hauptspeicherverwaltung

Eine der wichtigsten Ressourcen jedes Computers ist sein Hauptspeicher. In ihm erfolgt die Abspeicherung der Befehle eines Programmes (Programmkode) vor der Abarbeitung durch die Computerzentraleinheit und die Speicherung von zu jedem Prozeß gehörenden programm- und prozeßspezifischen Daten (z.B. Operativdaten). Zur Sicherung der effektiven Nutzung des Hauptspeichers realisiert die Speicherverwaltung Funktionen zur temporären und - bezogen auf die Bereichsgröße - gegebenenfalls auch dynamischen Zuweisung von Arbeitsbereichen für einen oder für mehrere Prozesse.
Gebräuchliche Verfahren zur effektiven Nutzung und Verwaltung des verfügbaren Hauptspeichers sind das Overlay- und das Swappingverfahren.
Das Overlayverfahren geht von der Aufteilung des Hauptspeicherbereiches für den Programmkode in einen Programmkodewurzelbereich, einen Programmkodeoverlaybereich sowie einen gemeinsamen Bereich (Commonbereich) mit programm- und prozeßspezifischen Daten aus. Der Inhalt des Programmkodewurzelbereiches befindet sich, solange ein Prozeß bearbeitet wird, fest und unveränderlich im Hauptspeicher; unterschiedliche Programmkodeoverlayteile werden nacheinander auf ein und demselben Overlayspeicherbereich eingelesen und abgearbeitet. In einem allen Programmkodeteilen gemeinsamen Datenbereich werden alle notwendigen programm- und prozeßspezifischen Daten gespeichert und übergeben. Die Overlaybereichsaufteilung wird vom Programmierer vorgenommen. Das Betriebssystem übernimmt insbesondere das aufeinanderfolgende Einlesen von jeweils neuen Programmkodesegmenten in den Programmkodeoverlaybereich bei gleichzeitiger Sicherung des aktuellen Prozeßstatus.
Mit Swapping wird ein Verfahren bezeichnet, bei dem nacheinander unterschiedlichen Prozessen feste oder auch variable Bereiche des Hauptspeichers für die Abspeicherung von allen zu einem Prozeß gehörenden Informationen (Programmkode, Programmdaten, Prozeßdaten ...) zugewiesen werden. Bei einem Prozeßwechsel erfolgt - wenn notwendig - eine komplette Auslagerung dieser Informationen auf einen externen Massendatenspeicher.
Stellt sich bei der Abarbeitung eines Prozesses heraus, daß der vom Betriebssystem bereitgestellte Hauptspeicherbereich zu klein ist, erfolgt ebenfalls eine Auslagerung und das anschließende Wiedereinlesen, diesmal auf einen größeren Bereich.
Voraussetzung für die Anwendung des Swappingverfahrens ist Massendatenspeicher mit kurzen Zugriffszeiten (Hard-Disk).
Neben der Hauptspeicherverwaltung durch das Betriebssystem existiert in modernen Rechnersystemen eine MMU-Hardware-Speicherverwaltungseinheit (MMU memory management unit) zur Umsetzung programmierlogischer Adressen in reale physische Speicheradressen, zur Zuweisung unterschiedlicher Programm- und Datenstrukturen auf unterschiedliche Speicherbereiche (z.B.: Programmkode, Operativdaten, Stackdaten), zum Schutz vor fehlerhaften Speicherbereichsüberschreitungen durch die Betriebs- und Anwendersoftware und zur Realisierung virtueller Adressierungsmechanismen. Solche Funktionen können von einem Betriebssystem nur unterstützt werden - aus Zugriffszeitgründen ist ihre alleinige Realisierung durch Softwaremaßnahmen nicht denkbar.

1.3.3. Dateiverwaltung

Die Dateiverwaltung hat die Aufgabe, logisch zusammengehörige Datenbestände zu speichern, zu verwalten und bei Bedarf wieder zur Verfügung zu stellen. Sie realisiert die Verbindung zwischen Betriebssystem und dem eine Schlüsselrolle zukommenden externen Direktzugriffsmassendatenspeicher (Hard-Disk; Floppy-Disk). Sie übernimmt die Realisierung von Zugriffsoperationen (Eröffnen, Schließen, Verändern von Datenbeständen) und die Gewährleistung von Dateischutzmechanismen. Die Dateiverwaltung standardisiert den Dateizugriff und befreit den Nutzer von physischen Dateiprogrammen.
Durch die Einführung der Hard- und Floppy-Disk-Technik - insbesondere in moderne Mikrocomputersysteme - ergibt sich ein qualitativer Sprung vor allem auf dem Softwaresektor, dessen Bedeutung kaum überschätzt werden kann. Der Anwender hat zur effektiven Nutzung dieser Speichermedien allerdings einen etwas höheren Softwareaufwand im Betriebssystem zu treiben als bei den vordem üblichen externen Speichermedien (Lochband, Magnetband). Dieser höhere Aufwand findet vor allem in dem Dateiverwaltungssystem des Betriebssystems seinen Niederschlag Physisch sind diese Speichermedien dadurch gekennzeichnet, daß die in Spuren (tracks) auf konzentrischen Kreisen angeordneten Informationen in eine bestimmte Anzahl von Sektoren (Kreissegmente) gleicher Größe aufgeteilt werden. Die in einem Sektor einer Spur aufgezeichneten Daten werden als Record oder Datensatz bezeichnet. Ein Record ist die kleinste physisch schreib- oder lesbare Informationsmenge. Er besteht je nach Datenträger und Aufzeichnungsverfahren aus 128, 256 oder 512 Byte. Die Anzahl der Spuren (z.B. 8" Standard-Floppy: 77 Spuren) ist abhängig vom Laufwerkstyp und von der Anschlußsteuerung. Das gleiche gilt für die Anzahl der Sektoren pro Spur. Sind die Anzahl der Laufwerke (drives), die Anzahl der Spuren (tracks) pro Laufwerk, die Anzahl der Sektoren pro Spur und die Recordlänge bekannt, kann die Speicherkapazität leicht ermittelt werden. Drivenummer, Tracknummer und Sektornummer lokalisieren physisch jeden Datensatz.
Neben dieser hardwaretechnischen Betrachtungsweise des Speichermediums Festplatte oder Diskette hat sich eine softwaretechnisch bedingte Betrachtungsweise als außerordentlich zweckmäßig herausgestellt, die darauf basiert, daß die kleinste vom Betriebssystem und den Anwenderprogrammen zu behandelnde Einheit eine aus Datensätzen bestehende Datei (file) darstellt. Eine Datei ist definiert als die logisch zusamengehörige geordnete Menge von einzelnen Records (z.B.: ein BASIC-Interpreter, ein Applikationsprogramm oder eine Meßwertreihe). Die genaue Anzahl der Dateien, ihre physische Lokalisierung in Tracks und Sektoren, die zugeordneten Namen zur Kennung und andere Spezifikationen werden in dem Inhaltsverzeichnis (directory) zusammengefaßt. Der Anwender kann über den Dateinamen und die zugehörigen Verweise im Directory jede Datei lesen oder umgekehrt schreiben. Directory und die auf dem Datenträger vorhandenen Dateien stellen die Basis des Dateiverwaltungssystems dar. Der Anwender braucht sich um die Verwaltung der Records nicht zu kümmern. Diese Arbeit, ebenso wie Zugriffsoptimierungen, eine ganze Reihe von Hilfsfunktionen und vieles andere mehr, übernimmt automatisch das Betriebssystem.

1.3.4. Ein-/Ausgabekanalverwaltung

Die meist langsamen (bezogen auf die Rechengeschwindigkeit der CPU) Ein-/Ausgabekanäle sind der Teil des Rechners, über den die Kommunikation mit der Umwelt abläuft. Die Ein-/Ausgabekanalverwaltung des Betriebssystem übernimmt die Verwaltung des Status der Ein-/Ausgabekanäle (besetzt, frei usw.), die Bereitstellung der Gerätekanalprogramme und die Abwicklung der Ein-/Ausgabeaktivitäten nach dem Polling- oder Interruptverfahren.
Das einfachste Verfahren zur Steuerung der peripheren Ein-/Ausgabekanäle ist das Pollingverfahren, bei dem zyklisch der Status des Ein-/Ausgabekanals abgefragt wird und je nach Ergebnis die gewünschte Ein-/Ausgabe erfolgt oder nicht. Dieses Verfahren ist letztlich allerdings uneffektiv, da es die zur Verfügung stehende CPU- und Ein-/Ausgabegerätezeiten schlecht ausnutzt.
Eine wesentlich effektivere Methode zur Ausnutzung der CPU-Rechenzeit ist das Interruptverfahren. Hier wird vom peripheren Gerät an die Computerzentraleinheit ein spezielles Programmunterbrechungssignal (Interruptmeldung) gesendet, das die kurzzeitige Aussetzung des momentan laufenden Prozesses veranlaßt und die Ausführung einer dem Interrupt entsprechenden Maßnahme in einer Interruptserviceroutine einleitet. Damit wird die CPU von zeitraubenden Abfragezyklen entlastet. Die Interruptbearbeitung kann bei Ein-/Ausgabegeräten zur Realisierung von Prioritäten auch verschachtelt ablaufen.

1.4. Hauptmerkmale des Betriebssystems UNIX

Wird die Entwicklung auf dem Computersektor (Hardware und Software) einer Analyse unterzogen, so ist vor allem anderen die Tendenz zur Standardisierung kennzeichnend. Bei den Hardwaresystemen stehen dabei die Prozessortypen, die Bussysteme, parallele und serielle Interfaces, Leiterkartenformate usw. im Mittelpunkt. Bei der Software ist die 'de facto'-Standardisierung von Betriebssystemen, von Programmiersprachen, von Programminterfaces usw. unverkennbar.
Ursache dieser Entwicklung sind die Vorteile, die sich aus einer maximalen Austauschbarkeit und Koppelbarkeit von Hardware und Softwarekomponenten ergeben, sowohl innerhalb eines Rechners als auch zwischen Rechnern der gleichen oder einer anderen Klasse (Mikrorechner untereinander, aber auch Mikrorechner mit Minicomputern oder Großrechnern). Die bei der Software dadurch realisierbare Austauschbarkeit auf allen Ebenen,

- auf der **Programmebene** (Austausch von Anwenderprogrammen),
- auf der **Dateiebene** (Austausch oder gemeinsame Nutzung Datenbeständen) und
- auf der **Nachrichtenebene** (Austausch von Nachrichten /Messages/ zwischen Anwenderprogrammen in einem Rechner bzw. in Rechnernetzen),

spielt bei der überwiegenden Anzahl der Anwender heute außerordentlich wichtige Rolle.
Der Erfolg des Betriebssystems UNIX ist letztlich auch als ein Ergebnis dieser Entwicklung zu betrachten, stellt es doch ein umfangreiches Instrumentarium von universellen Unterstützungswerkzeugen für die Realisierung dieser Forderungen bereit.

Als allgemein hervorstechende Merkmale von UNIX können daneben insbesondere angesehen werden:

Hierarchische Dateiverwaltung - Das hierarchische, baumstrukturierte Dateiverwaltungssystem von UNIX (tree structured hierarchical file system) stellt das modernste existierende Dateiverwaltungskonzept dar. Es geht davon aus, daß jede Datei (file) Mitglied einer durch ein Dateiverzeichnis (directory) zusammengefaßten Gruppe von Dateien ist, die ihrerseits wieder selbst Teil einer übergeordneten Gruppe von Dateien sein kann. Die dadurch entstehende Baumstruktur von Dateien und Dateiverzeichnissen (beides wird vom UNIX-Dateiverwaltungssystem gleich behandelt) beginnt mit einem Wurzelverzeichnis (root directory) und ist nach oben und in der Breite nur durch die Hardwaresystemgrenzen beschränkt (Bild 2.1). Durch den vom Anwender in Ausdehnung und Struktur dynamisch definierbaren Dateibaum mit Directories und Subdirectories ist es möglich, beliebige Dateigruppen nach Zugriffsrechten, nach Projektzugehörigkeit, nach Anwendergruppen usw. aufzubauen und vom Betriebssystem zu verwalten. Hinzu kommt, daß, neben der Gleichbehandlung von Dateien und Dateiverzeichnissen, eine Gleichbehandlung von Dateien und Ein-/Ausgabegeräten (bzw. von den Ein-/Ausgabegerätekanalprogrammen / drivern) im UNIX-Dateiverwaltungssystem implementiert wurde.

Ein-/Ausgaberedirektion - Die Ein-/Ausgaberedirektion von Dateien, Geräten und Programmdaten basiert auf einheitlichen, untereinander kompatiblen Datenströmen. Sie ermöglicht, durch einfachste Maßnahmen, die Umleitung einer für ein bestimmtes Ein-/Ausgabegerät vorgesehene Programmausgabe auf ein anderes Ausgabegerät oder auf eine neu anzulegende bzw. schon vorhandene Datei.

Pipes/Filter - Das Pipekonzept (pipe Kanal) ermöglicht dem UNIX-Anwender, die Ausgabedaten eines Prozesses dynamisch als Eingabedaten für einen nächsten Prozeß zuzuweisen.
Programme, die, auf diesem Konzept aufbauend, einen vorgegebenen Eingabedatenstrom in einen neu erzeugten Ausgabedatenstrom umsetzen, werden unter UNIX als Filter bezeichnet.

Shell - Das Betriebssystem UNIX beinhaltet eine außerordentlich leistungsfähige Benutzerschnittstelle, den sog. Shell-Kommandointerpreter (shell Schale). Dieses, wie eine Schale um den UNIX-Systemkern gelegte, Softwareinterface ermöglicht den Aufruf von Programmen - sowohl der Aufruf einzelner Programme als auch die parallele Bearbeitung mehrerer Programme und deren rekursiver Aufruf werden unterstützt -, und es stellt dem UNIX-Nutzer einen Satz von Variablen und Kontrollstrukturen, ähnlich einer höheren Programmiersprache, zur Verfügung. Dadurch können zum Beispiel zu sogenannten Shellskripten zusammengestellte wiederkehrende Aufgaben automatisiert werden.
Das Shell-Benutzerinterface ist sowohl ein Kommandointerpreter als auch eine Programmiersprache in enger Verbindung zum Betriebssystem.

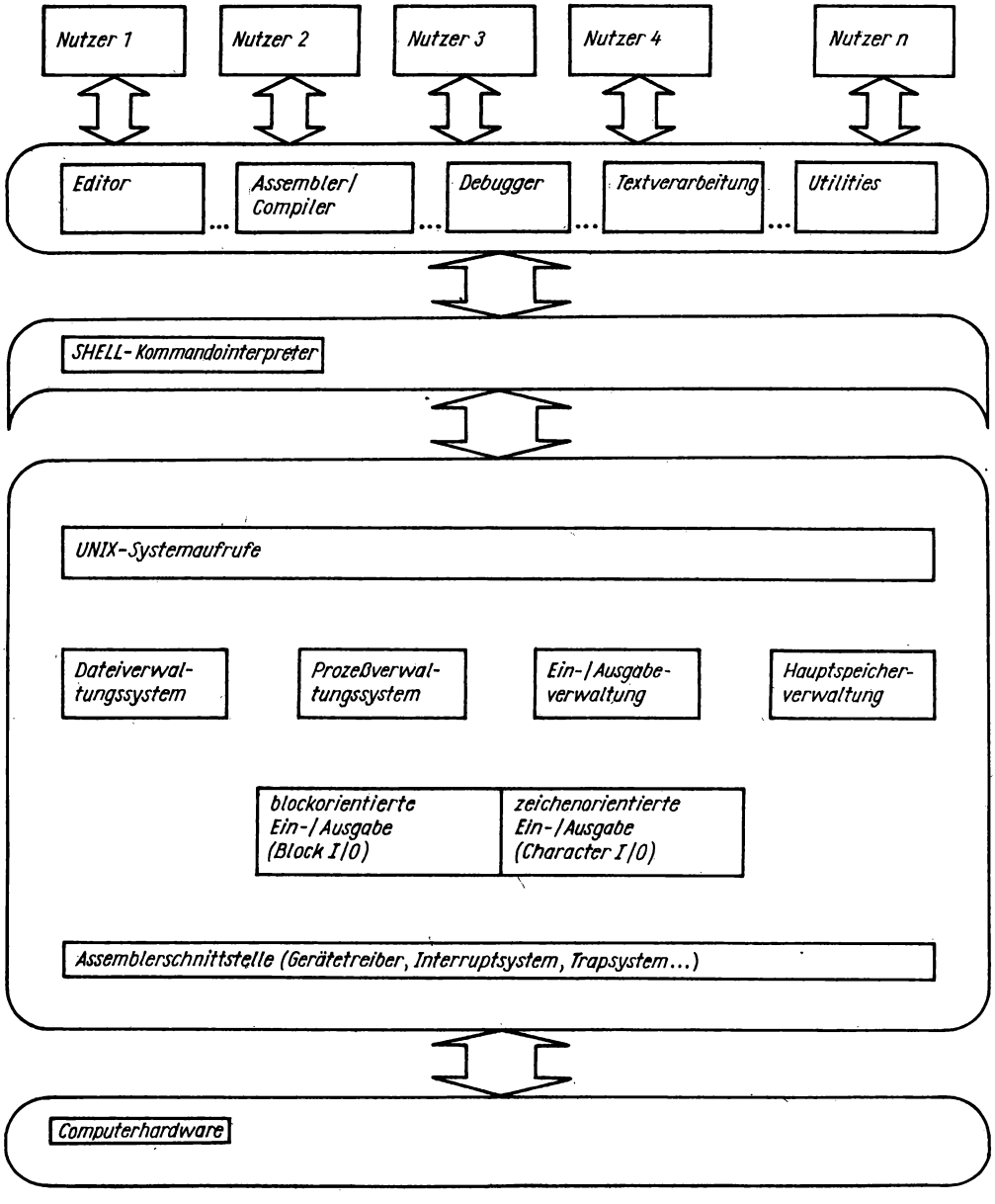


Bild 1.1. Struktur des Betriebssystems UNIX

Vom Betriebssystem UNIX wird eine Shell-Standardvariante (Bourne-shell) bereitgestellt, die, wenn sinnvoll und notwendig, jederzeit vom Anwender durch eine eigene Benutzerschnittstelle ersetzt werden kann (Bild 1.1).

C-Systemprogrammiersprache - Die Systemprogrammiersprache von UNIX ist die im Zusammenhang mit diesem Betriebssystem entstandene höhere Programmiersprache 'C'. Sie kombiniert die Effizienz einer Assemblersprache mit den Steuerstrukturen moderner HLL-Sprachkonzepte. Die Programmerarbeitung unter UNIX ist allerdings wegen der Verfügbarkeit einer ganzen Reihe von Sprachcompilern nicht an die Sprache C gebunden. Trotzdem empfiehlt sich ihre Anwendung wegen des dann leichten Transportes der erarbeiteten Software von einem UNIX-System auf ein anderes (Softwareportabilität auf Quellprogrammebene). Für Systemprogramme sollte sie zwingend vorgeschrieben werden.

Neben diesen Hauptmerkmalen des Betriebssystems UNIX existiert eine Reihe von weiteren Besonderheiten, wie zum Beispiel die mögliche Aufteilung des Anwenderbereiches in leicht identifizierbare Gruppen oder die automatische Projektfortschreibung (SCCS source code control system), die die besondere Unterstützung des Anwenders bei der Programmerarbeitung für ein großes Softwareprojekt absichern. UNIX bietet außerdem eine ganze Kollektion modular aufgebauter, spezialisierter und effizienter Softwarewerkzeuge (tools) für die Programmentwicklung, die durch die Verfügbarkeit mehrerer Editoren, einer ganzen Reihe von Sprachcompilern (C, FORTRAN, PASCAL, COBOL, BASIC u.a.), von speziellen Programmtestunterstützungsmitteln (debuggern), von Textverarbeitungssystemen und Datenbanksystemen u.a.m. ergänzt werden.
Die hervorragenden Multi-User-Eigenschaften (simultane Arbeit mehrer Anwender an einem Computer bei gemeinsamer Nutzung der Systemressourcen) und die implementierten effizienten Multi-Task-Algorithmen (quasisimultane Abarbeitung mehrerer Prozesse) sichern dem Betriebssystem UNIX zusätzlich eine große Anwendungsbreite.
Es soll an dieser Stelle nicht verschwiegen werden, daß natürlich auch kritische Bemerkungen zum Betriebssystem UNIX existieren. Als Hauptpunkt ist dabei wohl die minimalistische Philosophie bei der Ein-/Ausgabekommunikation zu nennen. Mit dem Ziel, bei der Eingabe von Kommandos durch den Anwender mit möglichst wenigen Tastenanschlägen auszukommen und bei der Ausgabe von Meldungen durch das Betriebssystem auf einer Bildschirmseite ein Maximum an Informationen unterzubringen, wurden bei der Gestaltung des Ein-/Ausgabedialogs geradezu spartanische Befehls- und Kommandozeichenfolgen festgelegt. Das mag besonders den Anfänger etwas irritieren und akademisch erscheinen, ist aber für den fortgeschrittenen Nutzer bald zur Gewohnheit geworden. Voraussetzung für die effektive Arbeit mit dem Betriebssystem UNIX ist deshalb auch und besonders die exakte Beherrschung der Kommandosyntax und die sorgfältige Arbeit an dem Bedienterminal.

Ein Vergleich unterschiedlicher Betriebssysteme wird immer stark vom Standort des Betrachters und vom Ziel des Vergleiches abhängen. Es soll deshalb an dieser Stelle kein Vergleich zwischen UNIX und Betriebssystemen von Großrechnern oder Minicomputern folgen. Die hier seit längerer Zeit mit einer universellen Verbreitung vorhandenen Betriebssysteme werden durch UNIX keine existentielle Konkurrenz bekommen können, sondern nur an spezifischen Applikationsstellen eine Ergänzung erfahren.

Anders ist die Situation bei Mikrorechnern. Hier findet seit Jahren eine hauptsächlich von großen internationalen Rechnerherstellern geführte Auseinandersetzung um das dominierende Betriebssystem statt. Auf den in ihrer Leistungsfähigkeit durch die gegebenen Hardwaremöglichkeiten stark begrenzten 8-Bit-Mikrorechnern ist diese Auseinandersetzung klar zugunsten des Betriebssystems CP/M-80 ausgegangen.
Auf dem Gebiet der leistungsstarken 16-Bit-Mikrorechner dagegen existieren neben UNIX zwei weitere Betriebssysteme mit hohem Verbreitungsgrad, die in Vergleichen immer wieder genannt werden - das weiterentwickelte CP/M und MS-DOS.
Das Betriebssystem CP/M war das erste signifikante Mikrocomputerbetriebssystem. Primär für den Einsatz mit Floppy-Disk-Laufwerken vorgesehen, existiert eine geradezu unendliche Fülle von Anwendungsprogrammpaketen für die 8-Bit-Version. Die Schwächen von CP/M resultieren letztlich aus den außerordentlich begrenzten Möglichkeiten der Mikrocomputertechnik zur Zeit der Entstehung (1973/74) von CP/M. Multi-User- und Multi-Task-Möglichkeiten ebenso wie Portabilität auf verschiedenen Mikrorechnern wurden im Laufe der Zeit schrittweise nachgebessert, ohne allerdings am Grundkonzept etwas zu verändern. Mit dem Aufkommen der ersten 16-Bit-Mikrorechner wurde versucht, den Erfolg von CP/M-80 auf diese Rechnerklasse zu übertragen. CP/M-86 ist eine direkte Umsetzung der Version 2.2 von CP/M-80 auf den 16-Bit-Mikroprozessor I8086. Der große Vorteil - unter CP/M-80 geschriebene Disketten können sofort ohne jegliche Konvertierung gelesen werden. Die wechselseitige Nutzung von Datenbeständen ist also uneingeschränkt möglich. Programme sind, wegen unterschiedlicher Befehlssätze der verwendeten Mikroprozessoren, nicht direkt austauschbar.
Das Betriebssystem MS-DOS wurde für einen der ersten 16-Bit-Mikrorechner auf Basis des Schaltkreises I8086 neu entwickelt. In vielem lehnte man sich bei der Konzeption von MS-DOS an das erfolgreiche CP/M an. Das Besondere an MS-DOS ist die Tatsache, daß ein großer internationaler Rechnerhersteller (IBM) dieses Betriebssystem für seinen in sehr großen Stückzahlen hergestellten 16-Bit-Personalcomputer als Softwarebasis ausgewählt hat. Die Unterstützung und Verbreitung, die MS-DOS von dieser Seite erfährt, ist deshalb außerordentlich groß. Die anfänglichen Probleme von MS-DOS - nur für I8086, nicht Multi-User usw. - wurden nach und nach beseitigt, und mit Beginn des Siegeszuges von UNIX wurden fast alle im vorhergehenden Abschnitt aufgeführten Hauptmerkmale von UNIX in fortgeschrittenen MS-DOS-Versionen übernommen. Das gilt für das hierarchische Dateiverwaltungssystem ebenso wie für die Ein-/Ausgaberedirektion, Pipestrukturen und das C-Sprachkonzept. Bei dieser Entwicklung ist jedoch zu beachten, daß MS-DOS nach wie vor nicht UNIX-kompatibel ist. Für MS-DOS geschriebene Programme werden, auch wenn sie in der Programmiersprache C geschrieben wurden, nicht durch eine Neu-Compilation unter UNIX sofort lauffähig sein (z.B. unterschiedliche Systemaufrufe und verschiedene Ein-/Ausgabebibliotheken).
Die Qual der Wahl zwischen MS-DOS und UNIX bleibt bei dieser Entwicklung letztlich wohl keinem Anwender erspart. Als wesentliches Kriterium für den Einsatz von UNIX bleibt die Tatsache, daß MS-DOS auch in seiner allerneusten Weiterentwicklungsvariante (OS/2) immer ein Betriebssystem für ein Ein-Platz-Mikrorechnersystem bleibt und keine gleichberechtigten Multi-User-Fähigkeiten wie UNIX besitzt. Außerdem wird wohl keiner auf die Idee kommen, MS-DOS sowohl auf Mikrorechnern als auch auf Minicomputern und Großrechnern implementieren zu wollen.

2. Basiskonzepte des Betriebssystems UNIX

Der zweite Abschnitt gibt einen Überblick über den Aufbau und die Arbeitsweise des UNIX-Betriebssystemkerns. Das für den Anwender zum Verständnis der Basiskonzepte von UNIX notwendige Wissen über das Dateiverwaltungssystem, über die Prozeßverwaltung und über das Ein-/Ausgabesystem wird vermittelt. Die überblicksmäßige Darstellung der Systemaufrufe (system calls) des UNIX-Kerns schließt dieses Kapitel ab.

2.1. Aufbau und Arbeitsweise des UNIX-Betriebssystemkerns

Die Gründe für die Verbreitung, die das UNIX-Betriebssystemkonzept gefunden hat, sind sicher vielfältiger Natur. Fest steht, daß eine ganze Reihe der in UNIX realisierten Ideen bis heute in wenigen anderen Betriebssystemen zu finden sind. Das gilt, wie bereits angedeutet, für das hierarchische, baumstrukturierte Dateisystem, für die einheitliche Behandlung von Dateien und Ein-/Ausgabegeräten, besonders aber auch für die Verfügbarkeit eines umfangreichen Satzes modularer, aufeinander abgestimmter Softwarewerkzeuge um einen von allem unnötigen Ballast befreiten Systemkern.
Der UNIX-Betriebssystemkern (UNIX kernel) läßt sich im wesentlichen in die drei Basiskomponenten Dateiverwaltung, Prozeßverwaltung und Ein-/Ausgabeverwaltung aufgliedern. Ferner existieren Module zur Absicherung der Interprozeßkommunikation, für die nutzerbezogene Rechenzeitabrechnung u.a.m.
Das Dateiverwaltungssystem übernimmt die Verwaltung aller auf externen Speichermedien, wie Magnetplatte, Floppy-Disk oder Magnetband, gespeicherten Datenbestände.
In der Prozeßverwaltung sind alle die Komponenten zusammengefaßt, die zur Ablaufsteuerung von System- und Anwenderprozessen benötigt werden.
Das Ein-/Ausgabesystem ist für die Kommunikation mit der Umwelt verantwortlich. Es unterteilt sich in eine strukturierte (block i/o-system) und in eine unstrukturierte (character i/o-system) Ein-/Ausgabe. Beide Ein-/Ausgabesysteme sind zur Erhöhung der Zugriffsgeschwindigkeit mit einem speziellen Cachepuffer ausgestattet, der die physischen Ein-/Ausgabetransfers durch ein vorgreifendes Lesen (read ahead) und ein nachziehendes Schreiben (write after) reduziert.

2.1.1. Dateiverwaltungssystem

Das Dateiverwaltungssystem ist ein Basiselement jedes Betriebssystems. Unter einer Datei (file) soll - wie im ersten Abschnitt schon erläutert ein Satz von Informationen verstanden werden, auf den über einen zugehörigen Dateinamen zugegriffen werden kann. Dateien werden auf externen Speichermedien angelegt. Das Dateiverwaltungssystem (file system) des Betriebssystems sichert den Zugriff und die Verwaltung dieser externen Datenbestände.

Auf der physischen Ebene kennt UNIX bei der Speicherung von Datenbeständen auf externen Speichermedien grundsätzlich nur eine Ein-/Ausgabe, die einheitlich mit Datensätzen einer Länge von 512 Byte arbeitet. Eine UNIX-Datei besteht dann aus 0, 1, 2, 3 Datenblöcken (0, 512, 1024, 1536 Byte). Die maximale Größe einer von UNIX verwalteten Datei liegt bei über zwei Millionen Datenblöcken.
Auf der für den UNIX-Anwender maßgeblichen programmierlogischen Ebene wird zwischen drei Dateitypen unterschieden

- reguläre Dateien (ordinary files),
- Dateiinhaltsverzeichnisse (directories) und
- spezielle Dateien (special files).

Neben einigen in der Folge zu vermittelnden Kenntnissen über die unterschiedlichen Dateitypen sind für den Anwender Dateinamenskonventionen, Mehrfachverweismöglichkeiten bei Dateien mit gleichem Inhalt und vom Betriebssystem unterstützte Basisschutzmechanismen im UNIX-Dateisystem zu beachten.

Reguläre Dateien (ordinary files)

Die 'regulären Dateien' enthalten alle im UNIX-Dateisystem gespeicherten Datenbestände, wie Anwenderprogramme, Anwenderdaten, Dienstprogramme und Systemprogramme. Vom Betriebssystem wird eine solche Datei als endliche Folge von Zeichen ohne irgendeine interne Struktur behandelt. Quellprogramm- und Textdateien bestehen zum Beispiel aus einer Kette von ASCII-Zeichen, in der das jeweilige Ende einer Zeile durch ein Wagenrücklaufzeichen (CR=0x0D 0x=hexadezimal) gekennzeichnet ist. Dateien mit ausführbaren Programmen (binary) bestehen aus einer Folge von Worten (mit einer Headerinformation), die den Speicherabzug eines Programmes darstellen usw.
Alle System-, Dienst- und Anwenderprogramme müssen, wenn erforderlich, den Aufbau und die Verwaltung der internen Struktur einer Datei selbst übernehmen.
Es existieren also prinzipiell nur Dateien, die eine lineare Folge von Bytes darstellen und auf die in einer beliebigen Reihenfolge zugegriffen werden kann. Das hat zur Folge, daß der Benutzer 'regulärer Dateien', wenn irgendeine Struktur in einer Datei verwaltet werden muß, sich die dazu notwendigen Funktionen selbst programmieren muß. Da das Betriebssystem keine bestimmte Struktur voraussetzt, hat der Benutzer zum einen den Vorteil, daß beliebige Strukturen realisierbar sind und zum anderen ergibt sich aus dieser Tatsache, daß alle Geräte, die eine zeichenweise Ein-/Ausgabe besitzen, vom Anwender wie Dateien behandelt werden können.

Dateiinhaltsverzeichnisse (directories)

Ein Dateiinhaltsverzeichnis (directory) ist ein Datensatz, der das Inhaltsverzeichnis einer Gruppe von Dateien darstellt. Es enthält alle Dateinamen einer Gruppe in Verbindung mit einem Pointer, der einen Zusammenhang zwischen den Dateinamen und der Datei selbst herstellt.
Der interne Aufbau einer Dateiinhaltsverzeichnisdatei und ihre Behandlung durch das Betriebssystem UNIX stimmt mit dem internen Aufbau und der

Behandlung von 'regulären Dateien' bis auf eine Ausnahme überein - Directories können nicht direkt durch unpriviligierte Anwenderprogramme, sondern nur vom Betriebssystem selbst erstellt und verändert (geschrieben) werden.
Im Gegensatz zu anderen Betriebssystemen kann ein Dateiinhaltsverzeichnis unter UNIX einen oder mehrere Verweise auf Subdirectories enthalten, die wiederum das Dateiinhaltsverzeichnis von zugehörigen Subdateigruppen darstellen. Dieser Vorgang läßt sich rekursiv beliebig verschachtelt wiederholen, so daß insgesamt ein hierarchisch, umgekehrt baumstrukturiertes UNIX-Dateisystem entsteht (Bild 2.1). Die Summe der Informationen aus allen in einem konkreten System zu einem bestimmten Zeitpunkt existierenden Dateiinhaltsverzeichnissen bildet die momentane interne Struktur des Dateisystems ab. Diese Struktur ist dynamisch veränderlich, da jeder Nutzer mit Hilfe des Betriebssystems nicht nur 'reguläre Dateien', sondern auch Directories neu erstellen oder löschen kann. Der Nutzer kann so, neben den Dateiinhaltsverzeichnissen, die vom UNIX-System selbst benötigt werden, seinen speziellen Anwendungserfordernissen entsprechende Dateigruppen aufbauen.
Der Ausgangspunkt eines UNIX-Dateibaums ist immer die Root-Directory (root Wurzel). In ihr befinden sich neben dem Betriebssystemkern die Systemsubdirectories der ersten Stufe:

bin	UNIX-Standardsystemkommandoprogramme
dev	peripheriegerätebezogene 'special files'
usr	anwenderbezogene Systemkommandoprogramme und Dateien
etc	Programme zur Systeminitialisierung, sowie zur Sicherung, zum Test und zur Wiederherstellung des UNIX-Dateisystems (Arbeitsmittel des UNIX-Superusers / UNIX-Systemprogrammierer)
lib	Bibliotheksprogramme
tmp	temporäre Dateien
lost+found	Sicherungsdirectory beim Dateisystemtest ('fsck')

Die spezielle Dateiinhaltsverzeichnisstruktur jedes UNIX-Systems wird im einzelnen wesentlich durch die spezielle UNIX-Implementierung des Rechnerherstellers, durch den jeweiligen Systemprogrammierer (UNIX-Superuser) und die Anwender des Computersystems bestimmt.

Spezielle Dateien (special files)

Die 'speziellen Dateien' stellen einen Bezug zwischen den an einem UNIX-Rechner angeschlossenen peripheren Ein-/Ausgabegeräten und dem Betriebssystem her. Sie können wie die 'regulären Dateien' gelesen und geschrieben werden, mit dem Unterschied, daß sich der Lese- oder Schreibaufruf als Ein- oder Ausgabeanforderung an das der 'speziellen Datei' zugeordnete Peripheriegerät (device) widerspiegelt. Dadurch entsteht eine nahezu identische Behandlung von Dateien und Peripheriegeräten mit gleichen Dateinamenskonventionen und mit gleichem Datenstromaufbau, die letztlich für eine ganze Reihe der besonderen Merkmale des Betriebssystems UNIX (Ein-/Ausgaberedirektion, Pipekonzept) eine der wichtigsten Voraussetzungen darstellt.

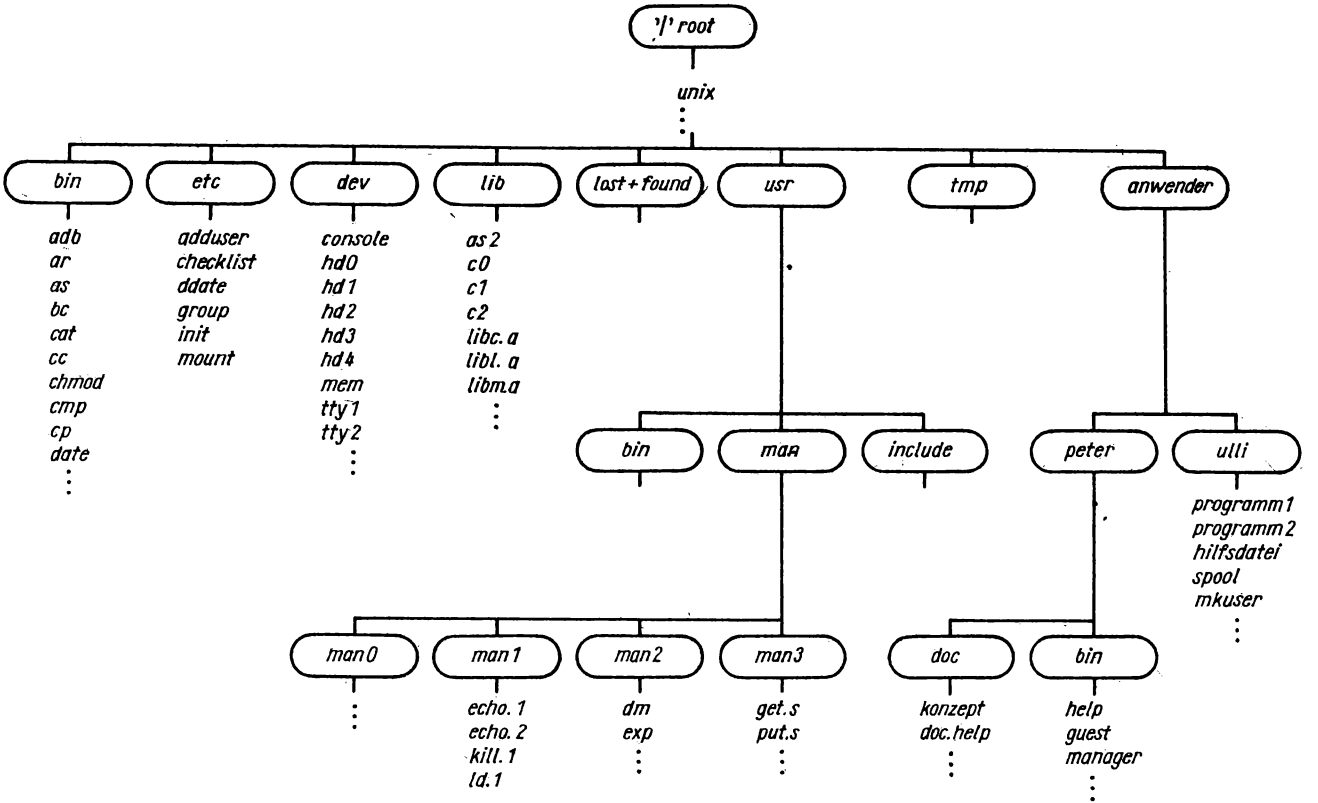


Bild 2.1. Struktur des UNIX-Dateisystems

Alle von einer speziellen UNIX-Generierung unterstützten Ein-/Ausgabegeräte, wie Platteneinheiten, Floppy-Disks, Drucker, Terminals usw., müssen mit mindestens einer 'speziellen Datei' korrespondieren, die einheitlich in dem Systemdirectory 'dev' (devices) zusammengefaßt werden.
Die 'speziellen Dateien' belegen keinen Platz auf dem externen Massenspeicher, auf dem die 'regulären Dateien' und die Directories abgelegt sind, sondern werden in der Systemdirectory 'dev' nur durch einen Namen und durch zwei Zahlen - anstelle der i-node - repräsentiert ('major-number' Gerätetreiber / 'minor-number' Gerätenummer in einer Klasse von Geräten mit gleichem Gerätetreiber), die die jeweiligen Eintrittspunkte der Ein-/Ausgabegerätetreiber darstellen.

Dateinamenskonventionen

Das im Bild 2.1 wiedergegebene Beispiel eines umgekehrt baumstrukturierten UNIX-Dateisystems geht von der Root-Directory an der Spitze des Dateibaums aus. An diese durch einen Schrägstrich ('/') gekennzeichnete Dateisystemwurzel schließen sich die Systemsubdirectories ('bin' 'etc', 'dev', 'lib', 'lost+found', 'usr', 'tmp' und 'anwender') der ersten Stufe an. Danach kommen u.a. die ersten Anwenderdateiverzeichnisse in unserem Beispiel 'ulli' und 'peter' (Bild 2.1). Diese auch als 'login directories' bezeichneten Anwenderdateiverzeichnisse werden beim Zuschalten (login) des jeweiligen Anwenders in ein laufendes Multi-User-UNIX sofort aktiv, d.h., der Anwender befindet sich in 'seinem' - dem dann momentan aktiven - Arbeitsdateiverzeichnis ('working directory' oder 'current directory'). Durch Editieren, Assemblieren, Linken oder andere Operationen kann er in diesem Directory neue Dateien anlegen, durch Ausführen spezieller UNIX-Systemkommandos kann er neue Subdirectories kreieren (Kommando: mkdir / make directory), sich im Dateibaum seiner Zugriffserlaubnis entsprechend hin und her bewegen (Kommando: cd / change directory) oder sich das momentan aktive Dateiinhaltsverzeichnis ausgeben lassen, in dem er sich befindet (Kommando: pwd / print working directory).
Alle Dateien im UNIX-Dateibaum werden durch einen absoluten und einen relativen Dateipfadnamen ('pathname') identifiziert. Der absolute oder volle Pfadname beginnt mit dem Schrägstrich der Root-Directory, gefolgt von den jeweils durch einen Schrägstrich getrennten Subdirectories. Am Ende dieser Folge befindet sich dann der Name der Datei:

```
/anwender/peter/doc/konzept
```

Der relative Dateipfadname geht nicht von der Dateibaumwurzel, sondern von dem momentan aktiven Arbeitsdirectory des jeweiligen Nutzers aus. Ist also 'peter' das momentane 'working directory', dann ergibt sich der relative Pfadname der Datei 'konzept' zu:

```
doc/konzept
```

(Ohne beginnenden Schrägstrich!)

Ist die bezeichnete Datei in dem momentanen Arbeitsdirectory direkt enthalten, stimmt der eigentliche Dateiname mit dem relativen Pfadnamen überein.
Die Länge eines Dateinamens kann unter UNIX maximal 14 Zeichen betragen Buchstaben, Ziffern und Punktsonderzeichen ('.'). Andere Sonderzeichen, wie Doppelpunkt (':'), Semikolon (';'), Fragezeichen (' ') usw. sind zwar generell zulässig, können aber an verschiedenen Stellen bei der Arbeit mit UNIX zu Verwechslungen führen, so daß auf ihre Verwendung, wenn möglich, ganz verzichtet werden sollte.

Mehrfachverweismöglichkeiten

Der UNIX-Betriebssystemkern stellt einen Satz von Elementaroperationen, bezogen auf die Arbeit mit Dateien, zur Verfügung. Dazu gehören Funktionen zum Eröffnen, zum Löschen, zum Schreiben, zum Lesen und zum Positionieren. Eine ganz spezielle Eigenschaft des UNIX-Dateiverwaltungssystems ist die Möglichkeit, eine Datei, die sich physisch nur ein einziges Mal auf dem Dateisystemdatenträger befindet, programmierlogisch in mehrere Directories, wenn notwendig, mit unterschiedlichen Dateinamen aufzunehmen. Diese Mehrfachverweismöglichkeit wird als 'file link' bezeichnet.

Dateischutzmechanismen

Zu jeder Datei unter UNIX gehört ein Satz von zehn Schutzattributen (protection bits). Diese Schutzattribute spezifizieren die Zugriffsrechte der einzelnen, vom System durch eine Benutzeridentifikation (UID user identification) und eine Gruppenzugehörigkeit (GID group identification) eindeutig voneinander unterscheidbaren Benutzer. Dabei kennt das Betriebssystem UNIX drei Arten von Anwendern - die einfachen Nutzer ('user' oder abgekürzt 'u'), die Nutzergruppen ('group' oder abgekürzt 'g') und alle sonstigen anonym am System arbeitenden Anwender ('other' oder abgekürzt 'o').
Wird vom Systemprogrammierer ein neuer zeitweiliger oder ständiger Nutzer dem Betriebssystem bekanntgemacht, muß er einer schon bestehenden oder neu zu installierenden Nutzergruppe zugeordnet werden. Durch Setzen des 'group protection mode' kann dann jedes Mitglied einer Nutzergruppe den Zugriff auf jede einzelne seiner Dateien erlauben oder verwehren.
Zuzüglich zu diesem einfachen Schutzmechanismus für den Dateizugriff existieren drei Kennzeichnungen für jede Datei - 'read', 'write' und 'execute' die die Zugriffseigenschaften durch andere Programme spezifizieren.
Werden diese dreimal drei Schutzebenen zusammengerechnet (user, group, other und read, write, execute) ergeben sich neun der zehn möglichen Schutzbits. Das zehnte Schutzbit bewirkt eine temporäre Veränderung der Benutzeridentifikation, die es ermöglicht, bestimmte Programme oder Datenbestände zeitweise zu privilegieren.
Die Dateischutzmechanismen sind beim Zugriff des UNIX-Superusers nicht wirksam.

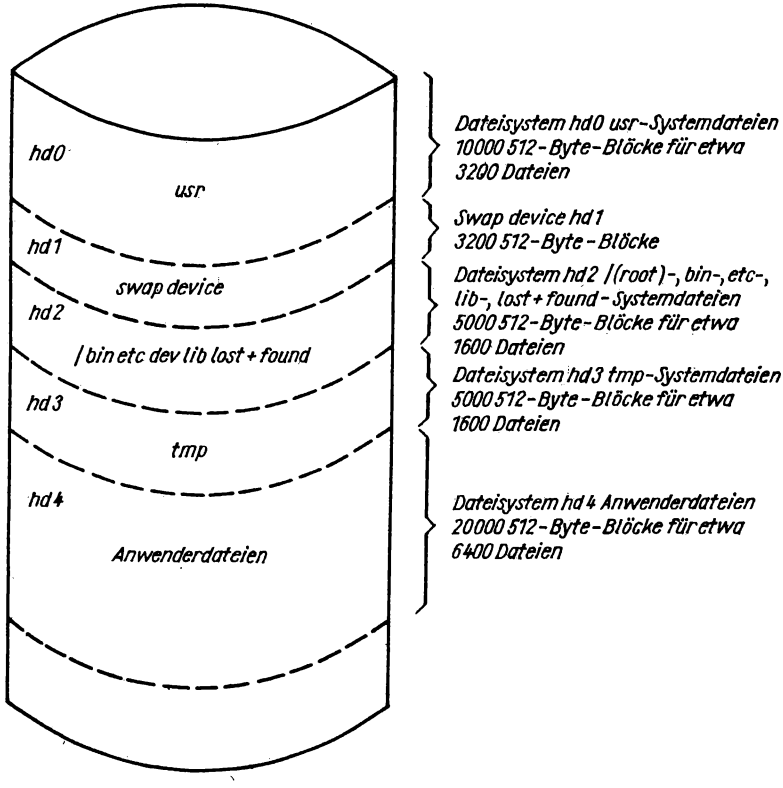


Bild 2.2. Beispiel für den Aufbau eines UNIX-Dateisystems

Interne Struktur des Dateiverwaltungssystems

Neben dem für jeden Anwender des Betriebssystems UNIX wichtigen Wissen über die allgemeine Struktur des Dateiverwaltungssystems folgen an dieser Stelle einige Erläuterungen zum UNIX-Dateisystem, deren Kenntnis nicht Voraussetzung für die Anwendung dieses Betriebssystems ist.
Die physischen Datenträger für das Dateiverwaltungssystem (z.B. Hard-Disk-Laufwerke) werden bei vielen UNIX-Implementationen programmierlogisch in mehrere Teile aufgespalten. Dadurch ist es möglich, auf einem Plattenlaufwerk mehrere voneinander unabhängige und gegeneinander abgesicherte Dateisysteme anzulegen. Zusätzlich muß ein Swappingbereich für die automatische Ein-/Auslagerung von Programmen und Daten durch das UNIX-Betriebssystem existieren, der nicht in das Dateiverwaltungssystem einbezogen wird (Bild 2.2).

Jedes einzelne der programmierlogischen Dateisysteme von UNIX hat dann den für alle gleichen internen Aufbau:

logischer Block 0	512 Byte für einen Boot-Anfangslader
- logischer Block 1	512 Byte für den Superblock
- logischer Block 2 bis n	512 x (n-1) Byte für die 'i-list'
logischer Block n+1 bis m	512 x (m-n) Byte für die eigentliche Speicherung von Dateien und für die vom Betriebsystem verwaltete (dem Anwender nicht zugängliche) Liste der freien 512-Byte-Blöcke eines Dateisystems (Freispeicherliste)

Der logische Block 0 für den ersten 512 Byte langen UNIX-Anfangslader (primary bootstrapper) wird in der Regel nur im physisch ersten Dateisystem echt genutzt, in allen weiteren ist dieser Block meist leer.
Der Superblock - ebenfalls 512 Byte lang - enthält die allen Dateien eines Dateisystems gemeinsamen Basisinformationen, wie die genaue Größe des Dateisystems, die maximal mögliche Anzahl von Dateien in diesem Dateisystem, einen Verweis auf die ersten und (über einen Pointer) auf die folgenden freien Blöcke zur Aufnahme der eigentlichen Dateien (zusammen die sogenannte Freispeicherliste) und anderes mehr.
Die 'i-list' ist eine zusammenfassende Liste aller Beschreibungsblöcke - der sogenannten 'i-node' - der einzelnen Dateien in einem logischen Dateisystem. Jeder dieser 'i-node'-Beschreibungsblöcke stellt eine 64-Byte-Struktur dar, die die Charakteristiken einer Datei genau spezifiziert:

- Typ der Datei ('reguläre Datei' / 'Directory' / 'spezielle Datei'),
- Zugriffserlaubnis (protection bits) z.B.: 'read', 'write' und 'exec',
- Anzahl der Verweise (links) auf diese Datei in verschiedenen Directories,
- Eigentümer und Gruppenzugehörigkeit der Datei,
- Größe der Datei,
- Lokalisierung der Datei im Dateisystem (Adresse der Datei auf der Hard-Disk, Floppy-Disk oder auf dem Magnetband) und erstes Erstellungs-, letztes Verwendungs- und letztes Modifikationsdatum (die UNIX-Zeit wird intern grundsätzlich in Sekunden ab 1970 gerechnet - daraus können Zeit und Datum ermittelt werden).

Der Versatz - bezogen auf den Anfang der Liste - 'i-node' einer 'i-list' wird als 'i-number' bezeichnet.
Ist bekannt, auf welchem physischen und logischen Gerät sich eine Datei befindet, d.h., sind die 'major-number' und die 'minor-number' bekannt und ist außerdem die 'i-number' dieser Datei gegeben, ist unter UNIX eindeutig definiert, um welche Datei es sich handelt.

Der in der 'i-node' enthaltene Adreßverweis zur Lokalisierung der 512-Byte-Blöcke einer Datei besteht normalerweise aus 13 Blockadressen. Die ersten 10 Blockadressen zeigen direkt auf die ersten zehn Dateiblöcke insgesamt einen Bereich von 5 KByte umfassend. Falls die Datei größer als 5 KByte ist (und nur in diesem Fall), zeigt die 11. Blockadresse in der 'i-node' auf einen 512-Byte-Block mit den nächsten 128 Adressen von 512-Byte-Blöcken der Datei. Reicht auch dieser Bereich (5 KByte plus 64 KByte) wegen der Größe der Datei nicht aus, erfolgt in der 12. Blockadresse in der 'i-node' ein Verweis auf einen Block mit 128 Blockadressen, die ihrerseits wiederum (zweimal indirekt) auf 128 Blöcke mit Dateiblockadressen verweisen. Ist dies immer noch nicht genug, besteht über die 13. 'i-node'-Blockadresse (dreimal indirekt) die Möglichkeit, einen noch größeren Bereich zu adressieren. Insgesamt ergibt sich so die maximal mögliche Größe einer Datei zu

(10 + 128 + 128 x 128 + 128 x 128 x 128) x 512 Bytes.

Handelt es sich bei einer Datei um eine 'special file', sind in der 'i-node' keine Adreßverweise auf die Lokalisierung der 512-Byte-Blöcke zur Speicherung des Dateiinhaltes notwendig. An ihrer Stelle wird der physische Gerätetyp und spezielle Gerätename für Geräte gleichen Typs, die schon erwähnte 'major-number' und 'minor-number', eingetragen.
Die meisten der in einer 'i-node' enthaltenen Informationen einer Datei werden beim Öffnen (open) dem Betriebssystem durch Kopieren der 'i-node' in die 'activ i-node table' bekanntgemacht - beim Schließen der Datei (close) werden die dann aktualisierten Angaben auf den Datenträger zurückgeschrieben.
Ein ähnliches Verfahren gilt für den Superblock beim Eröffnen und Schließen eines ganzen Dateisystems.
Im Bild 2.3 sind die Datenstrukturelemente des UNIX-Dateiverwaltungssystems und ihre Beziehungen untereinander dargestellt.
Jeweils für einen Anwenderprozeß (siehe auch Abschnitt 2.1.2.) existiert eine Tabelle der von diesem Prozeß eröffneten Dateien ('per user open file table'). Außerdem existiert eine systemglobale Tabelle ('open file table'), die die notwendigen Informationen für den parallelen Zugriff mehrerer UNIX-Prozesse auf eine gemeinsame Datei sichert. Daneben wird zentral vom UNIX-Systemkern die schon erwähnte Tabelle von dateispezifischen Steuerinformationen ('active i-node table') verwaltet, die Auskunft über die momentan aktiven Dateien gibt.

2.1.2. Prozeßverwaltung

Anwenderprogramme werden in einer Softwareumgebung abgearbeitet, die, als 'user process' bezeichnet, alle Komponenten einschließt, die bei der Abarbeitung eines Programmes signifikant sind. Das sind Speicher- und Registerinhalte, der CPU-Status, ein bestimmter Zustand von Dateien (open, close) und anderes mehr.
Benötigt ein in Abarbeitung befindlicher Anwenderprozeß Dienste des UNIX-Betriebssystems, wird über einen Systemaufruf ('system call') eine entsprechende Subroutine aktiviert. Der Aufruf und die Abarbeitung einer Systemroutine ist mit einem Wechsel der Softwareumgebung der Programme

verbunden - von einem 'user process' zu einem 'system process'. Nach dem Wechsel von einem Anwenderprogramm zu einem zum UNIX-Kern gehörenden Systemprogramm arbeitet der laufende Prozeß im sogenannten Systemmode.

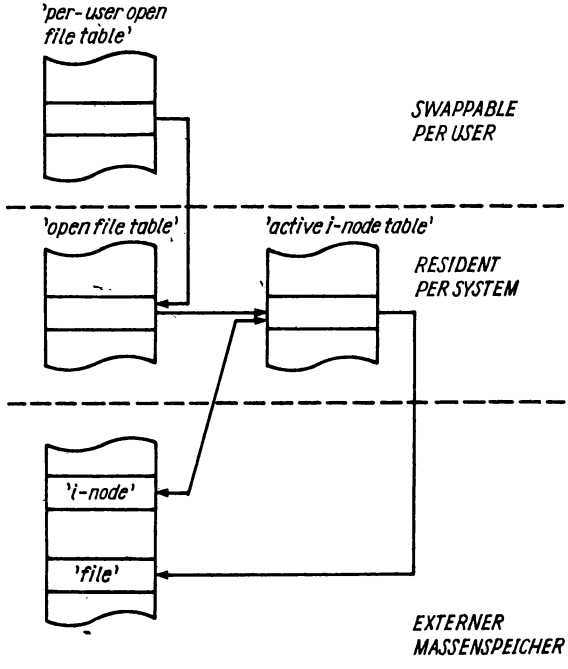


Bild 2.3. Steuerdatenstrukturen des UNIX-Dateiverwaltungssystems (nach Thompson [4])

Wesentliches Element der UNIX-Prozeßverwaltung sind die verschiedenen Programmdatenstrukturen von Anwender- und Systemprozeß. Dabei ist zu beachten, daß der segmentierte Adreßraum eines unter Steuerung des Betriebssystems UNIX laufenden Programmes aus einem Programmkodesegment ('code segment' oder auch 'text segment'), aus einem Datensegment ('data segment') und aus einem Stackdatensegment ('stack segment') besteht. Werden beide Prozesse - der Anwenderprozeß und der Systemprozeß - als zwei unterschiedliche Teile ein und desselben übergeordneten Prozesses aufgefaßt, kann zwischen den folgenden sieben (jeweils eine programmierlogische

Einheit darstellenden) Prozeßdatenstrukturelementen unterschieden werden:

Textsegment Anwendermode (Befehlskode des Anwenderprogrammes)
- Datensegment Anwendermode
- Stacksegment Anwendermode

Textsegment Systemmode (Befehlskode des Betriebssystems)
- globales Datensegment Systemmode (globale Daten des Betriebssystems für alle Prozesse)
- lokales Datensegment Systemmode (lokale Daten des Operationssystems für einen Prozeß - oft auch als sog. 'user structur' bezeichnet)
- lokales Stacksegment Systemmode (lokale Stackdaten des Betriebssystems für einen Prozeß)

Das Bild 2.4 gibt einen Überblick über die Prozeßkontrollstrukturen des Betriebssystems. Insbesondere wird deutlich, daß mit dem Daten- und Stacksegment des gesteuerten Programmes (user) ein prozeßspezifisches Systemdatensegment (system) abgelegt wird, auf das ausschließlich der UNIX-Systemkern Zugriffsrechte besitzt. Es enthält alle die prozeßspezifischen Steuerinformationen, die nur dann benötigt werden, wenn der entsprechende Prozeß in dem Arbeitsspeicher geladen ist. Die für die Prozeßverwaltung zu jeder Zeit wichtigen Informationen werden resident in einer Prozeßkontrolltabelle ('process table') und einer Programmkodesegmenttabelle ('text table') zusammengefaßt.
Besonders wichtiges Element der UNIX-Prozeßverwaltung ist die prioritätsgesteuerte CPU-Vergabe. Dabei ist grundsätzlich zwischen höher priorisierten Systemprozessen und niedriger priorisierten Anwenderprozessen zu unterscheiden.
Tritt eine Unterbrechung der Abarbeitung eines Systemprogrammes ein, bekommt es in Abhängigkeit von dem Ereignis, welches das Programm nach der Unterbrechung wieder in den ablauffähigen Zustand versetzen wird, eine bestimmte Priorität zugeordnet.
Anwenderprogramme haben eine Priorität, die von dem Verhältnis der Prozessorbenutzungszeit zur realen Gesamtprozeßzeit abhängig ist, so daß Prozesse hoher Priorität nach intensiver Prozessornutzung auf eine niedrigere Priorität herabgesetzt werden und Prozesse, die mit niedriger Priorität längere Zeit den Prozessor nicht zugeteilt bekommen haben, in ihrer Priorität steigen. Prozesse gleicher Priorität werden zyklisch zeitgesteuert ('round robin' Sekundenrhythmus) abgearbeitet.
Durch diesen Prioritätsvergabealgorithmus werden insbesondere interaktive Prozesse bevorzugt. Die Neuberechnung der Priorität findet bei den meisten UNIX-Implementationen im Sekundenabstand statt.
Neben Fragen der Prozeßkontrollstrukturen und Fragen der Prioritätsvergabe sind vor allem Fragen der Prozeßerzeugung, der Prozeßabwicklung und der Prozeßkoordinierung für die UNIX-Prozeßverwaltung maßgeblich.
Die Erzeugung eines Prozesses wird mit dem Systemaufruf 'fork' abgewickelt. Der durch diesen Systemdienst neu angelegte Prozeß (Tochterprozeß) ist zunächst in allem eine Kopie des erzeugenden Prozesses (des sog. Elternprozesses / parent process), insbesondere gilt das für die Prozeßverwaltungsstrukturen. Beim Laden und Ausführen eines Programmes (Systemaufruf 'exec') wird der mit 'fork' angelegte Prozeß (bezogen auf den Programmkode-, Daten- und Stackadreßraum) entsprechend dem aktuellen Programm konkretisiert.

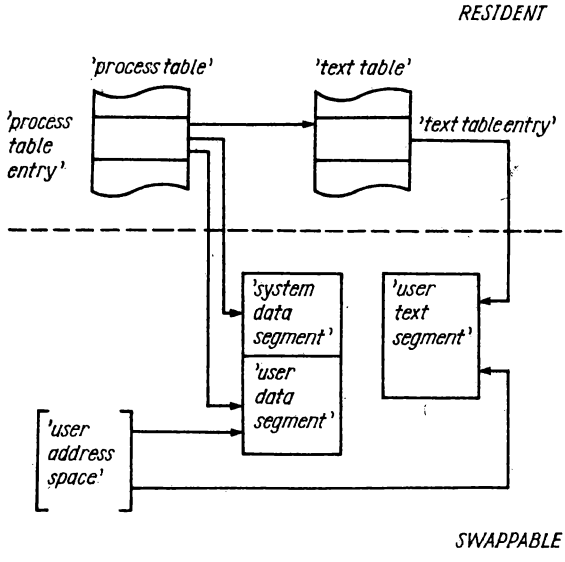


Bild 2.4. Steuerdatenstrukturen der UNIX-Prozeßverwaltung (nach Thompson [4])

Anmerkung:
Das Anwendertextsegment und das Anwenderdatensegment können in Abhängigkeit von einer Option (-i) beim Übersetzen und Linken des Programmes entweder in zwei getrennte Segmente oder auch in ein gemeinsames Segment abgelegt werden. Wird der Programmkode in einem seperaten Segment abgelegt, bekommt dieses Segment die 'read only'-Eigenschaft. Wird ein solcher Programmkode von mehereren Prozessen gleichzeitig abgearbeitet, braucht er nur ein einziges Mal im Hauptspeicher abgelegt zu werden.

Für die Prozeßkoordinierung stehen die Systemrufe 'wait' (warten auf ein Ereigniskode) und 'signal' (Versetzen eines Programmes in den lauffähigen Zustand) zur Verfügung.
Weitere existentielle Systemdienste der Prozeßverwaltung sind die Speicherverwaltung (software memory management) und das Ein-/Auslagern von Prozessen (swapping).
Verantwortlich für die Ein-/Auslagerung von Prozessen von (zum) Systemspeicher zum (vom) externen Massendatenspeicher ist der 'process scheduler', ein separater Teil der UNIX-Prozeßverwaltung. Er überprüft die Prozeßtabelle auf laufbereite Prozesse, die im Moment nicht abgearbeitet werden können, da sie sich nicht im Hauptspeicher befinden. Existiert solch ein Prozeß, wird im Hauptspeicher ein entsprechender Speicherbereich als belegt gemeldet und der Prozeß mit allen seinen Teilsegmenten eingelesen und ausgeführt. Ist kein freier Hauptspeicherbereich verfügbar, überprüft der 'process scheduler' so lange die im Hauptspeicher befindlichen Prozesse, bis ein Prozeß gefunden wird, der ausgelagert werden kann. Darauf folgt dann das als Swapping bezeichnete Auslagern des im Moment nicht bearbeiteten Prozesses und das Einlesen des neuen auf seine Abarbeitung bereits wartenden Prozesses.
Beim Ein-/Auslagern von Prozessen werden zwei Kriterien vom 'process scheduler' bei der Prozeßauswahl angewendet. Das eine bestimmt den auf dem externen Speichermedium befindlichen abarbeitungsbereiten Prozeß als nächsten in den Hauptspeicher einzulesenden Prozeß, der bereits am längsten auf dem externen Speicher auf die Abarbeitung wartet. Das zweite Kriterium bestimmt die Prozesse aus dem Hauptspeicher zur Auslagerung, die entweder momentan nicht abarbeitungsbereit sind oder die auf Fertigmeldungen von langsamen Ein-/Ausgabegeräten (z.B. Terminals) warten.
Das Prozeßswapping beeinflußt ausschließlich die Abarbeitungszeit eines Programmes. Es wird um so weniger aktiv, je größer der Hauptspeicherbereich eines UNIX-Computers ist. Bei einem Rechner mit 256 KByte und 16 gleichzeitig an verschiedenen Terminals arbeitenden Nutzern wird sehr viel Zeit für die Ein-/Auslagerung von Prozessen benötigt - stehen dagegen 1 MByte für vier Nutzer zur Verfügung, wird die für das Swapping benötigte Zeit sehr gering sein. Ein Dimensionierungsrichtwert für angemessene Swappingaktivitäten eines UNIX-Rechners sollte sein, daß für jeweils vier gleichzeitig an verschiedenen Terminals arbeitende Nutzer mindestens 512 KByte Hauptspeicher bereitstehen.

2.1.3. Ein-/Ausgabesystem

Wie schon erwähnt, kennt das UNIX-Ein-/Ausgabesystem zwei unterschiedliche E/A-Typen, die strukturierte Ein-/Ausgabe (block i/o-system) und die unstrukturierte Ein-/Ausgabe (character i/o-system).
Die strukturierte Ein-/Ausgabe arbeitet mit systeminternen 512-Byte-Puffern, auf die bei Ausgabe- bzw. Eingabeanforderungen von Programmen zugegriffen wird. Der eigentliche Ein-/Ausgabeverkehr erfolgt, wenn der Inhalt dieser Puffer nicht der gewünschten Information entspricht (Eingabe) bzw. wenn die Puffer voll sind, so daß kein Pufferschreibzugriff mehr möglich ist (Ausgabe). Wenn dieser Fall nicht eintritt, wird der Inhalt der Puffer zyklisch in bestimmten Abständen (meist alle 30 Sekunden) durch Ausführung des 'sync'-Systemaufrufes den realen Ein-/Ausgabewünschen angepaßt. Vorteilhaft ist ohne Frage die durch diesen Cachepufferspeicher erhöhte Abarbeitungsgeschwindigkeit von Programmen. Zu

beachten ist allerdings, daß durch einen nicht beabsichtigten Systemzusammenbruch (Netzausfall) eine auf dem internen Puffer befindliche Information verlorengehen kann. Letzeres kann unter anderem zu Inkonsistenzproblemen von Dateisystemen führen.
Die unstrukturierte Ein-/Ausgabe entspricht der klassischen Einzelzeichenein-/ausgabe an und von Terminals, Druckern, seriellen Datenkanälen usw.
Für jedes einzelne Ein-/Ausgabegerät (gekennzeichnet durch seine Zugehörigkeit zu einem oder auch zu beiden Ein-/Ausgabesystemen sowie durch die 'major' device number' und die 'minor device number') existiert als eindeutige Schnittstelle zwischen Kanalprogramm (Treiber) und Betriebssystem eine Konfigurationstabelle ('configuration table'). In dieser Tabelle befinden sich fest definierte Eintrittspunkte für standardisierte Treiberfunktionen.
Im engen Zusammenhang mit dem Aufbau und der Struktur des UNIX-Ein-/Ausgabesystems steht die Benutzeroberfläche dieses Betriebssystems, der sogenannte Shellkommandointerpreter. Da der UNIX-Systemkern nur Elementaroperationen für die Dateiverwaltung, die Prozeßverwaltung und die eben beschriebenen beiden Ein-/Ausgabetypen enthält, kann die Benutzerschnittstelle beliebig austauschbar gestaltet werden. Der Shellkommandointerpreter legt sich wie eine Schale um den kompletten Systemkern und stellt ein beliebiges Anwenderprogramm wie jedes andere dar. Zwei Standard-Shell-Versionen von UNIX sind die sog. 'Bourne-Shell' (Abschnitt 6.) und 'C-Shell". Der aktuell definierte Shellkommandointerpreter wird nach dem Start von UNIX automatisch aufgerufen (der Shellaufruf ist letztes Element in den anwenderbezogenen Zeilen der Initialisierungsdatei 'etc/passwd'). Er eröffnet bei seinem Aufruf drei Dateien, die alle das Terminal des Benutzers darstellen. Es sind die Dateien

stdin (standard input), die Datei der Tastatur des Terminals
stdout (standard output), die Datei des Bildschirmterminals, verwendet für alle Ausgaben, außer den Fehlermeldungen des Systems
stderr (standard error output), die für die Fehlermeldungen des Systems verwendete Ausgabedatei

Über die angegebenen Dateien läuft die gesamte Bedienerkommunikation des UNIX-Betriebssystems.

2.2. Systemaufrufe des UNIX-Betriebssystems

In diesem Abschnitt soll eine kurze, nach dem UNIX-Systemaufrufsprungverteiler geordnete Übersicht über die Systemaufrufe (system calls) des UNIX-Betriebssystemkerns gegeben werden.
Die Systemaufrufe stellen die unterste Stufe des Zusammenwirkens zwischen Betriebssystem und allen unter Steuerung von UNIX laufenden Programmen dar. Sie betreffen das Ein-/Ausgabesystem, das Dateiverwaltungssystem, die Prozeßsteuerung und anderes mehr. Mit ihrer Hilfe erfolgt die Initialisierung neuer Prozesse, das Eröffnen einer Datei, das Lesen bzw. das Schreiben einer Datei, die Abfrage der UNIX-Systemuhr, der Stop eines laufenden Programmes oder der Wechsel der Zugriffsberechtigung.
Die in den vorhergehenden Abschnitten behandelten UNIX-Basiskonzepte spiegeln sich in den Systemaufrufen in vielfältiger Weise wider.

Die Nutzung der UNIX-Systemfunktionen durch den Anwender erfolgt grundsätzlich über Systemaufrufe. Auf Assemblersprachniveau stellt der Systemaufrufname eine Einsprungstelle in der internen UNIX-Tabelle 'sysent' dar. Die Übergabe der Argumente erfolgt in Registern oder im Stack.
Bei der Programmierung in höheren Programmiersprachen erfolgt die Auflösung der Systemdienste über Assemblerelementaroperationen in den UNIX-Standardbibliotheken, so daß der Anwender des Betriebssystems UNIX, auch wenn von ihm neue System-, Dienst- oder Anwenderprogramme selbst entwickelt werden, mit den UNIX-Systemaufrufen nicht in Berührung kommt, solange er sich Programmiersprachen wie C, PASCAL, FORTRAN77 usw. und der für diese Sprachen vorhandenen Standardbibliotheken bedient.
Eine detaillierte Beschreibung der UNIX-Systemaufrufe ist in [17] zu finden.

Tafel 2.1. Übersicht über die UNIX-Systemaufrufe

`exit(status).`	BEENDEN EINES PROZESSES Schließt alle Dateien eines Prozesses und informiert den Elternprozess, falls dieser auf die Beendigung des Prozesses wartet.
`pid = fork()`	ERZEUGEN EINES PROZESSES Es wird ein neuer Prozeß erzeugt, dessen Prozeßabbild eine direkte Kopie des Abbildes des aufrufenden Prozesses (Elternprozeß) ist. Es wird die Prozeßidentifikationsnummer (pid) des neuen Prozesses zurückgeben.
`n = read(fd,buffer,nbytes)`	LESEN AUS EINER DATEI Es werden 'nbytes' Bytes aus der Datei 'fd' in den angegebenen Puffer 'buffer' eingelesen. Die Anzahl der übertragenen Bytes wird zurückgeben (Null bei EOF).
`n = write(fd,buffer,nbytes)`	SCHREIBEN IN EINE DATEI Es werden 'nbytes' Bytes ab der Adresse 'buffer' in die Datei 'fd' geschrieben. Die Anzahl der geschriebenen Bytes wird zurückgegeben.
`fd open(name,mode)`	ERÖFFNEN EINER DATEI Unter dem angegebenen Namen und Zugriffserlaubnis wird eine Datei zum Lesen oder Schreiben (oder zu beidem gleichzeitig) eröffnet. Der zurückgegebene Wert 'fd' (Dateideskriptor) muß bei weiteren Dateizugriffen verwendet werden (-1, wenn nicht eröffnet werden kann).
`n = close(fd)`	SCHLIESSEN EINER DATEI Schließt die zum angegebenen Dateideskriptor 'fd' gehörende Datei. Wenn die Datei geschlossen werden kann, wird Null zurückgegeben.
`pid = wait(status)`	WARTEN AUF PROZESSENDE Wartet auf ein Signal oder darauf, daß einer der erzeugten Tochterprozesse beendet wird (wurde). Es wird der Prozeßidentifikator 'pid' des beendeten Prozesses zurückgegeben.

fd creat(name,mode)	ERZEUGEN EINER DATEI Unter dem angegebenen Namen und Zugriffserlaubnis wird eine neue Datei erzeugt oder der Inhalt einer bereits existierenden Datei gelöscht. Die Datei wird danach zum Schreiben eröffnet. Der zurückgegebene Wert 'fd' (Dateideskriptor) muß bei weiteren Dateizugriffen verwendet werden (-1, wenn nicht eröffnet werden kann).
n link(name1,name2)	VERBINDUNG ZU EINER DATEI Es wird eine Verbindung zu einer existierenden Datei (name1) über den zweiten Namen (name2) innerhalb eines Dateisystems hergestellt (gleiche Datei - verschiedene Pfadnamen).
n unlink(name)	STREICHEN DIRECTORYEINTRAG Es wird der zum angegebenen Dateinamen gehörende Directoryeintrag gestrichen. Wenn dies die letzte Verbindung zu einer Datei war, so wird die Datei gelöscht.
exec(name,argv)	AUSFÜHREN EINES PROZESSES Veranlaßt den Start des in der angegebenen Datei 'name' abgelegten Programmes (Prozesses) mit den spezifiziertenden Argumenten 'argv'. Bei erfolgreichem exec-Aufruf geht der aufrufende Prozeß verloren.
n = chdir(dirname)	WECHSEL DER ARBEITSDIRECTORY Veranlaßt den Wechsel zum angegebenen (Pfadname) Directory. Dieses Directory ist danach das aktuelle Arbeitsdirectory.
nn = time(tloc)	ERMITTELN DER SYSTEMZEIT Übergibt als Rückgabewert die aktuelle Systemzeit (in Sekunden nach 0h:1.1.1970). Falls 'tloc' ungleich Null ist, wird der Zeitwert zusätzlich auf den durch 'tloc' adressierten Speicherplatz abgelegt.
n = mknod(name,mode,addr) (*)	ERZEUGEN EINER i-node Unter dem angegebenen Namen und Zugriffserlaubnis wird eine i-node und der zugehörige Directory-eintrag erzeugt. Wird hauptsächlich zur Erzeugung von Gerätedateien ('special files') benutzt.
n = chmod(name,mode)	ÄNDERN DATEIMODE Für die angegebene Datei wird die Zugriffserlaubnis auf den Wert von 'mode' gesetzt.

n chown(name,owner,group) (*)	ÄNDERN DATEIEIGNER Für die angegebene Datei werden der Eigner und die zugehörige Gruppe verändert.
n = brk(addr)	ÄNDERN SPEICHERZUWEISUNG Setzt den im UNIX-Betriebssystemkern gespeicherten Wert für die niedrigste vom Programm nicht benutzte Adresse ('break'-Adresse genannt) auf den Wert von 'addr'. Ein Zugriff auf eine Adresse größer 'addr' veranlaßt eine Speicherschutzverletzung.
n = stat(name,buf)	ERMITTELN DATEISTATUS Für die angegebene Datei ('name') wird der Dateistatus in den durch 'buf' adressierten Pufferspeicher kopiert.
nn = lseek(fd,offset,mode)	POSITIONIEREN DATEIZEIGER Für die durch 'fd' gekennzeichnete eröffnete Datei wird der Dateizeiger verändert. (mode=0: auf offset, mode=1: aktueller Wert+offset, mode=2: Dateigröße+offset)
pid = getpid()	ERMITTELN PROZESSIDENTIFIKATION Es wird die Prozessidentifikation ('pid') des augenblicklichen Prozesses übergeben.
n = mount(special,name,rwflg)	ANSCHLIESSEN EINES DATEISYSTEMS Es wird ein durch 'special' gekennzeichnetes (d.h. auf einem bestimmten Gerät befindliches) Dateisystem angeschlossen. Der angegebene Dateiname verweist auf das Wurzeldirectory des angeschlossenen Dateisystems.
n umount(special)	ABKOPPELN EINES DATEISYSTEMS Ein vorher über das angegebene Gerät ('special') angeschlossenes Dateisystem wird vom System abgekoppelt.
n = setuid(uid) (*)	SETZEN NUTZERIDENTIFIKATION Die Nutzeridentifikation ('uid') des augenblicklichen Prozesses wird auf den angegebenen Wert gesetzt.
uid = getuid()	ERMITTELN NUTZERIDENTIFIKATION Es wird die Nutzeridentifikation des augenblicklichen Prozesses übergeben.
n = stime(tp) (*)	SETZEN SYSTEMZEIT Die Systemzeit wird auf den durch 'tp' adressierten Wert (in Sekunden nach 0h:1.1.1970) gesetzt.

n	ptrace(req,pid,addr,data)	PROZESSTRACE Erlaubt dem Elternprozeß die Steuerung des Ablaufs eines Tochterprozesses. Wenn der Tochterprozeß sich im Stopzustand befindet, kann ggf. über 'ptrace' sein Speicherabbild verändert und die Fortsetzung veranlaßt werden.
time	alarm(seconds)	AUSLÖSEN EINES SIGNALS Löst ein vorher durch einen Signalaufruf ('signal') spezifiziertes Signal nach Ablauf der angegebenen Zeit in Sekunden (maximal: 65535) aus.
n	fstat(fd, buf)	ERMITTELN DATEISTATUS Analog stat-Systemaufruf für eine eröffnete Datei (durch 'fd' gekennzeichnet).
pause()		STOP BIS SIGNAL Programmstop bis zum Auftreten eines Signals, Weiterlauf dann entsprechend Signalaufruf.
n	utime(file,timep)	SETZEN DATEIZEITEN Die Zugriffs- und Modifikationszeit der angegebenen Datei wird durch zwei durch 'timep' adressierte Zeiten ersetzt.
n	stty(fd,argp)	SETZEN TERMINALSTATUS An das angegebene Terminal (Gerätedatei 'fd') werden die durch 'argp' adressierten Werte ausgegeben.
n	gtty(fd,argp)	ERMITTELN TERMINALSTATUS Für das angegebene Terminal (Gerätedatei 'fd') werden die aktuellen Statusinformationen ermittelt und auf den durch 'argp' adressierten Platz abgelegt.
n	access(name,mode)	ÜBERPRÜFEN DATEIZUGRIFFSERLAUBNIS Es wird die durch 'mode' spezifizierte Zugriffserlaubnis für die angegebene Datei ('name') überprüft.
n	nice(incr)	SETZEN DER PROZESSPRIORITÄT Die aktuelle Prozeßwechselpriorität des augenblicklich in Bearbeitung befindlichen Prozesses wird zu dem angegebenen Wert 'incr' addiert (seine Priorität verringert sich) bzw. subtrahiert (seine Priorität erhöht sich - letzteres nur Superuser).

`n = ftime(tp)`	ERMITTELN DER SYSTEMZEIT Übergibt die Systemzeit in Form der Struktur 'timeb' (einschließlich Zeitzone) an den durch 'tp' adressierten Speicherlatz.
`sync()`	AKTUALISIEREN PLATTENINHALT Veranlaßt das Schreiben von Speicherblöcken aus dem Cachepufferspeicher auf den Plattenspeicher.
`n = kill(pid,sig)`	AUSSENDEN EINES ABBRUCHSIGNALS Es wird an den angegebenen Prozeß ('pid') das durch 'sig' spezifizierte Abbruchsignal ausgesendet.
`fd = dup(fd)` `n = dup2(fd1,fd2)`	DUPLIZIEREN DATEIDESKRIPTOR Übergibt in der ersten Form einen weiteren Dateideskriptor für die durch 'fd' gekennzeichnete Datei. In der zweiten Form wird veranlaßt, daß beide angegebenen Dateideskriptoren auf die gleiche Datei verweisen.
`fd = pipe()`	ERZEUGEN PIPEKANAL Es werden zwei Dateideskriptoren (für 'read' und 'write') zur Arbeit mit einem Pipekanal zurückgegeben. Über den Pipekanal können dann zwei Prozesse direkt Daten austauschen.
`n = times(buffer)`	ERMITTELN PROZESSZEITEN Es werden Zeitinformationen für den aktuellen Prozeß, einschließlich seiner beendeten Tochterprozesse, in einer durch 'buffer' adressierten Struktur übertragen.
`n = profil(buff,` `bufsiz,offset,scale)`	AUSFÜHRUNG LAUFZEITPROFIL Nach diesem Systemaufruf wird alle 1/60 Sekunden der Befehlszählerwert ermittelt, eine Transformation durchgeführt und die entsprechende Zelle im angegebenen Puffer inkrementiert. Dadurch entsteht das Laufzeitprofil eines Programmes.
`n = setgid(gid)` (*)	SETZEN GRUPPENIDENTIFIKATION Die Gruppenidentifikation des aktuellen Prozesses wird auf den angegebenen Wert ('gid') gesetzt.
`gid = getpid()`	ERMITTELN GRUPPENIDENTIFIKATION Es wird die Gruppenidentifikation des augenblicklichen Prozesses übergeben.

n signal(sig,func)	SETZEN SIGNALBEHANDLUNG Es wird spezifiziert, welche Aktion ('func') bei Eintreten eines bestimmten Signals ('sig') ausgeführt werden soll.
n acct(file)	EIN/AUS ABRECHNUNG FÜR EINEN PROZESS Bei Ausgabe eines Dateinamens als Argument wird für jeden beendeten Prozeß ein Abrechnungsdatenzatz an die angegebene Datei angehängt.
lock(flag) (*)	EIN/AUS PROZESSCHUTZ Ist das Argument ungleich Null, wird der Prozeß nicht mehr ausgelagert (swapped). Ausnahme: Ein Datenbereich muß vergrößert werden. Wenn flag=0, wird dieser Schutz ausgeschaltet.
n ioctl(fd,request,argp)	AUSFÜHREN GERÄTESTEUERFUNKTIONEN Für das angegebene Gerät (Gerätedatei 'fd') wird die spezifizierte Funktion ('request') mit den entsprechenden Argumenten ('argp') ausgeführt.
exece(name,argv,envp)	AUSFÜHREN EINES PROZESSES Die Wirkung des exece-Systemaufrufs entspricht der des exec-Systemaufrufs, zusätzlich zeigt 'envp' auf ein Feld von Variablendefinitionen. Diese Werte sind für das gestartete Programm (Prozeß) verfügbar.
n = umask(complmode)	SETZEN DATEIERZEUGUNGSMODE Es wird die Standardzugriffserlaubnis, mit der alle Dateien implizit erzeugt werden, gesetzt.
n = chroot(dirname) (*)	SETZEN WURZELDIRECTORY Es wird das Wurzeldirectory auf die angegebene Datei ('dirname') gesetzt.

Anmerkung: Die Systemaufrufe geben i. allg. einen Integerwert zurück. Der Rückgabewert kann je nach Systemaufruf eine spezielle Bedeutung haben. Ein aufgetretener Fehler wird durch die Rückgabe einer '-1' gekennzeichnet.
Die mit (*) gekennzeichneten Systemaufrufe sind nur vom Systemprogrammierer (Superuser) ausführbar.

3. Initialisierung eines UNIX-Systems

Ausgangspunkt des dritten Abschnitts ist der Überblick über die Hardwarekomponenten eines typischen UNIX-Computers. Eine kurze Beschreibung des meist vorhandenen gerätebezogenen EPROM-Monitors leitet zu den für das Einrichten eines UNIX-Filesystems - der sog. Systeminitialisierung - verantwortlichen Standalone-Programmen ('format' / 'mkboot' / 'mkfs' / 'restor') über. Die Behandlung der UNIX-Anfangsladeprozedur (initial boot) vermittelt dem Leser Kenntnisse über den routinemäßigen Start eines UNIX-Rechners.

3.1. Struktur eines typischen UNIX-Computers

Ausgangspunkt der Behandlung des Betriebssystems UNIX, seiner Standardkommandos, der Utilities und einiger Dienstprogramme soll das im Bild 3.1 dargestellte Computersystem sein.
Eine leistungsfähige 16-Bit-Zentraleinheit (CPU) übernimmt den Befehlsaufruf und die Befehlsabarbeitung. Wesentliche Teile jedes Computers, wie Arithmetik-/Logikeinheit, Interruptsystem, Registersatz, Statusflags, Befehlszähler und vieles andere mehr, sind hier konzentriert.
Der Systemspeicher unterteilt sich in zwei Hauptkomponenten, den nach dem UNIX-Systemstart abschaltbaren - für den Hardwareeigentest, für einige maschinennahe Softwarebasisfunktionen und den Anfangsladeprozeß vorgesehenen - EPROM-Monitorspeicher (EPROM electrically programmable read only memory / elektrisch programmierbarer Programmfestwertspeicher) und den eigentlichen zwischen 256 KByte und 8 MByte großen RAM-Hauptspeicher (RAM random access memory / Lese-/Schreibspeicher) des Computersystems.
Besonders wichtiges Element jedes UNIX-Rechners ist die zwischen Zentraleinheit und Hauptspeicher angeordnete MMU-Speicherverwaltungseinheit (MMU memory management unit). Sie übernimmt die Übersetzung der von der CPU ausgegebenen programmierlogischen Speicheradressen in reale physische Speicheradressen sowie die Gewährleistung einer Reihe von Speicherschutzfunktionen (protection violation). Die Speicherverwaltungseinheit realisiert insbesondere die jedem unter Steuerung des Betriebssystems UNIX laufenden Prozeß zugrunde gelegte Speicheraufteilung mit Programmkode ab Adresse 0 aufsteigend, Stackspeicherbereich ab Adresse 65536 (64K) fallend sowie einem sich im gleichen Adessenbereich wie der Stack ab Aresse 0 befindlichen Operativdatenspeicherbereich (Bild 3.2).
Ohne MMU-Speicherverwaltungseinheit wird an dieser Stelle der verfügbare Hauptspeicher schlecht genutzt. Jeder unter Steuerung des Betriebssystems laufende Prozeß belegt komplette 64 KByte (oder mehr), wie klein oder groß er auch sein mag. In einem 256-KByte-System bleibt dann neben dem UNIX-Systemkern gerade noch Platz für zwei quasisimultan laufende Prozesse.

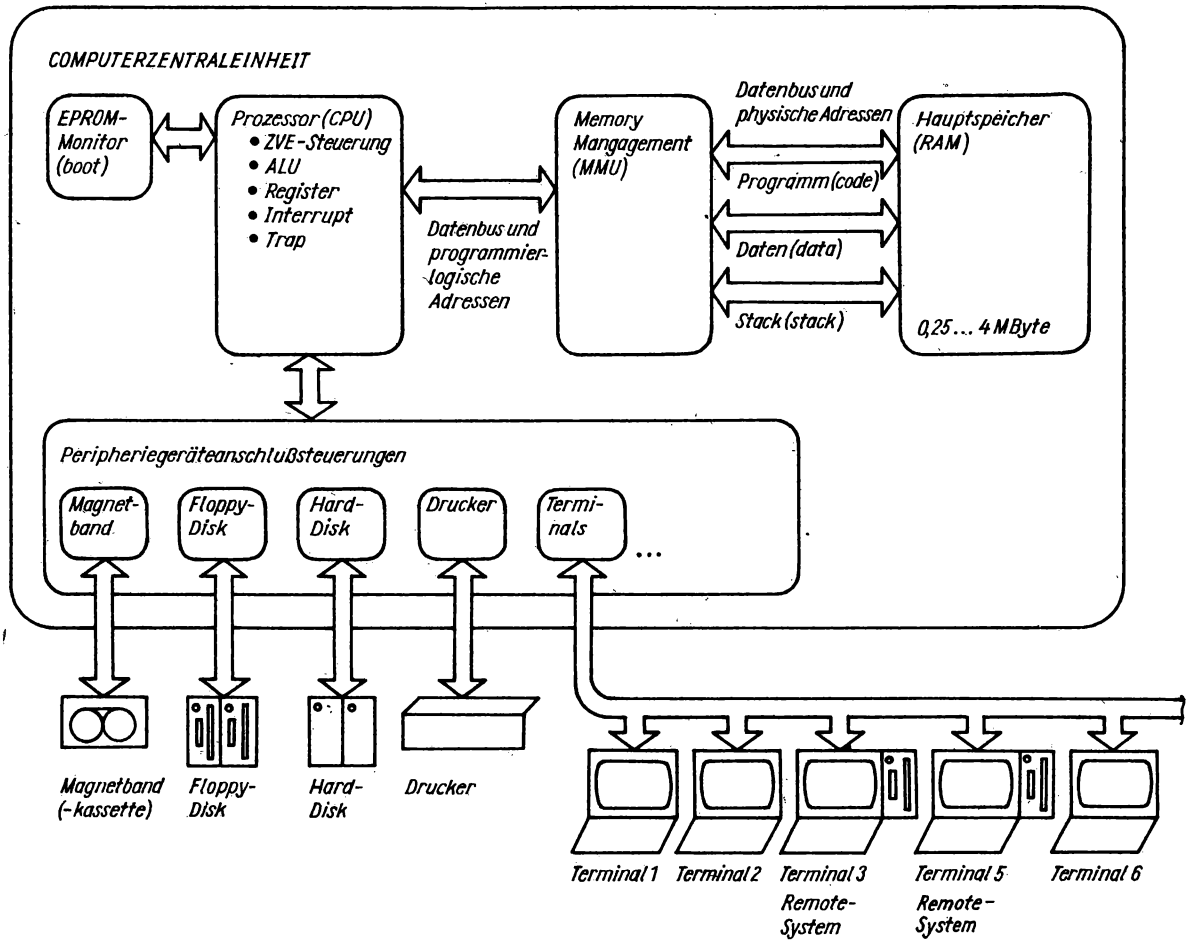


Bild 3.1. Standardkonfiguration eines UNIX-Computers

Von Multi-Task oder gar Multi-User kann nahezu keine Rede mehr sein. Erst eine leistungsfähige Speicherverwaltungseinheit, die den verfügbaren Speicherraum durch Zusammenschieben nicht benötigter Speicherbereiche effektiv ausnutzt und trotzdem die vorgeschriebene programmierlogische Adressenverteilung realisiert, garantiert hier die Ausnutzung der im Betriebssystem UNIX implementierten Leistungscharakteristiken.
Wesentliches viertes Element, neben CPU, MMU und Speicher, sind die an den Rechner anschließbaren peripheren Geräte mit den zugehörigen Anschlußsteuereinheiten, das sind zunächst insbesondere die externen Massendatenspeicher Hard-Disk, Floppy-Disk und Magnetband(kassette).
Die Hard-Disk, das eigentliche Master-Massenspeichermedium mit den hier transient abgelegten UNIX-Betriebssystemteilen, wird in den meisten Fällen ein sogenanntes Winchesterlaufwerk sein, das - besonders wartungsarm und robust - auch in computeruntypischen Umgebungsbedingungen zuverlässig seinen Dienst tut.
Ein Floppy-Disk-Laufwerk dient besonders der einfachen und problemlosen Datenübergabe/-übernahme von Anwenderprogrammen oder Anwenderdatenbeständen. Hier ist zu beachten, daß das Speichermedium Diskette in den letzten Jahren eine sehr große Verbreitung auf kleineren und mittleren Rechnern gefunden hat, dem auch bei der Nutzung von UNIX Rechnung zu tragen ist. Der Anschluß von Floppy-Disk-Laufwerken kann, wie gleich noch zu zeigen sein wird, neben der direkten Ankopplung an die UNIX-Zentraleinheit auch über abgesetzte Remote-Systeme erfolgen.
Der dritte wichtige Massenspeicher ist die Magnetband- oder Magnetbandkassetteneinheit, meist als Streamer-Laufwerk mit Einzelspuraufzeichnung ausgebildet und ohne aufwendige Start-Stop-Blockungsmechanismen. Es dient zur periodischen Back-up-Sicherung aller auf der Hard-Disk gespeicherten Datenbestände sowohl der UNIX-Systemkomponenten als auch der Anwenderprogramme und Anwenderdaten. Der zyklische Systemabzug (sog. Back-up) je nach Größe, Frequentierung und Bedeutung des Systems - täglich, wöchentlich oder gar monatlich ist eine wichtige Maßnahme, um im denkbaren Fall des totalen Systemzusammenbruchs (z.B. nach einem besonders ungünstig kommenden Netzausfall) ohne besondere Datenverlustprobleme die Reinitialisierung des Systems gewährleisten zu können.
Eine weitere, nicht weniger wichtige Gruppe von Ein-/Ausgabegeräten sind die für die Multi-User-Arbeit notwendigen seriellen V24-Terminals. Vier, acht, sechzehn oder mehr serielle Kanäle an einer UNIX-Zentraleinheit gestatten den voneinander unabhängigen Anschluß einer entsprechenden Anzahl von Terminals. Die Terminals, jeweils aus einer Tastatur-/Bildschirmkombination bestehend, stellen die eigentlichen UNIX-Anwenderarbeitsplätze dar. Ein Terminalarbeitsplatz kann zusätzlich durch ein oder zwei Floppy-Disk-Laufwerke ergänzt werden, so daß der Nutzer bei den unterschiedlichsten Aufgabenstellungen durch einen seinem Arbeitsplatz speziell zugeordneten externen Speicher unterstützt wird. Ein solches, dann als Remote-System (remote entfernt) bezeichnetes Terminal (auch autonom arbeitsfähig z.B. unter Steuerung eines Betriebssystems wie CP/M) wird vom zentralen System mit speziellen UNIX-Dateitransferprogrammen ('infile' / 'outfile' / 'back') in seiner Arbeit unterstützt.
Natürlich sind auch weitere dem externen Arbeitsplatz speziell zugeordnete Peripheriegeräte, wie grafische Displays, Plotter, EPROM-Programmierer und vieles andere mehr, an einem solchen Rechner denkbar.

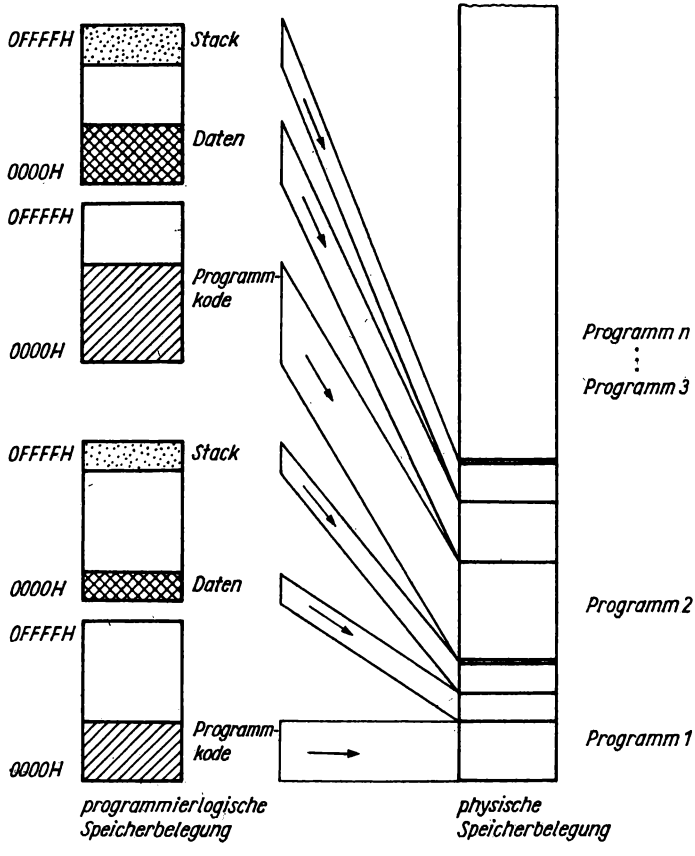


Bild 3.2. Programmierlogische und physische Speicheraufteilung in einem UNIX-Rechner

Das gilt besonders auch für den in unserem Fall zentral an den Computer angeschlossenen Drucker. Handelt sich es um einen schnellen, hochproduktiven und damit sicher auch teuren Drucker, wird er für alle Nutzer als gemeinsames Ausgabegerät genutzt, das an die UNIX-Zentraleinheit direkt angeschlossen ist und mit speziellen Softwaretreiberprogrammen unterstützt wird. Ein solcher Drucker, im sog. Spooler-Betrieb (line printer spooler) arbeitend, führt nacheinander die Druckaufträge der einzelnen Benutzer aus.

Wird der Drucker dezentral einem Terminalarbeitsplatz zugeordnet, ist zwar der unmittelbare sofortige Zugriff in jedem Fall sichergestellt, seine Auslastung jedoch sicher nicht sehr hoch.

3.2. EPROM-Monitor

Die erste nach dem Systemneustart (Systemreset) aktivierte Basissoftwarekomponente eines modernen UNIX-Mikrocomputersystems wird wohl immer ein sog. EPROM-Monitor sein. Dieses als Firmware bezeichnete und in einem nicht veränderbaren Programmfestwertspeicher geladene Programm übernimmt den automatischen hardwarebezogenen Eigentest, es enthält gerätenahe Softwaretestfunktionen, sichert die Möglichkeit der Herstellung eines Speicherabzugs und ermöglicht den manuellen oder automatischen UNIX-Anfangsladestart.
Es ist wichtig, an dieser Stelle darauf hinzuweisen, daß der EPROM-Monitor kein Teil des UNIX-Betriebssystems darstellt. Er wird von jedem Computerhersteller ganz spezifisch, abhängig von den Hardwarebesonderheiten und dem Anwendungsumfeld seines Rechners, entworfen (Anwendung des UNIX-Rechners als Bürocomputer, als Mikrorechnerentwicklungssystem, als Steuerungsrechner usw.). Der Monitor wird nach oder während des Anfangsladeprozesses meist abgeschaltet, so daß der UNIX-Nutzer, von speziellen Anwendungen abgesehen, nur dann mit ihm in Berührung kommt, wenn nach einem Systemzusammenbruch oder aus anderen Gründen eine neue Initialisierung des gesamten Softwaresystems erforderlich ist.

Hardwareeigentest

In einem UNIX-Computer werden meist zwei Testsoftwarepakete existieren der EPROM-residente Hardwareeigentest und ein ganz spezielles, aber auch sehr umfassendes Standalone-Hardwaretestprogramm, welches von einem externen Datenträger (Diskette oder Magnetband) in den Rechner geladen werden kann.
Der EPROM-residente Hardwareeigentest sollte die folgenden Basisfunktionsprüfungen umfassen

- Instruktionstest der CPU
- Test der Speicherverwaltungseinheit (MMU)
- Hauptspeichertest (RAM)
- Test der Ein-/Ausgabeperipherie

Natürlich wird der Hardwareeigentest nicht umfassend die angegebenen Baugruppen auf ihre Funktionstüchtigkeit prüfen können. Dazu ist der im EPROM für den Eigentest zur Verfügung stehende Programmspeicherbereich zu begrenzt. Die umfassende Funktionsprüfung aller Komponenten des UNIX-Rechners (CPU, Hauptspeicher, Memory Management, Hard-Disk, Floppy-Disk, Magnetband usw.) bleibt dem Standalone-Testprogrammpaket vorbehalten. Der ('power up') Eigentest gibt jedoch dem Anwender einen ersten Hinweis, ob der Computer ordnungsgemäß arbeitet. Die Fehlersignalisierung erfolgt über ein Bedienerterminal oder (falls es gar nicht soweit kommt) über eine vom Eigentestprogramm speziell angesteuerte Luminiszenzdiodenkombination auf der Rechnerleiterkarte.

Softwaretestfunktionen

Die hardwarenahen Softwaretestfunktionen im EPROM-Monitor dienen zum Test von Programmkomponenten, deren Test so basisnah erfolgen muß, daß jedes Betriebssystem den Test nur kompliziert, verfälscht oder gar unmöglich macht (z.B: Interruptserviceroutinen, Gerätekanalprogramme usw.).
In einem Standardmonitor sollten die folgenden Testkommandos zur Verfügung stehen:

DISPLAY	Adresse	Lesen und Schreiben Speicherinhalten
REGISTER	Registerbezeichnung	Lesen und Schreiben Registerinhalten
IOPORT	Ein-/Ausgabeportadresse	Lesen und Schreiben Ein-/Ausgabekanälen
BREAK	Adresse	Setzen und Löschen von Softwarehaltepunkten
GO		Sprung zum letzten Befehlszählerinhalt
JUMP	Adresse	Sprung zu der angegebenen Adresse
NEXT	---	Einzelschrittabarbeitung
FILL	Adresse1 Adresse2 Daten	Speicherschreiben mit einer Konstante
MOVE	Adresse1 Adresse2 n	Umladen eines Speicherbereiches
COMPARE	Adresse1 Adresse2 n	Vergleichen von Speicherbereichen
LOAD	Adresse Dateiname	Laden eines Hauptspeicherbereiches mit einer Datei von einem externen Datenträger
SEND	Datei Adresse	Abspeichern eines Hauptspeicherbereiches als Datei auf einem externen Datenträger
TEST		Manuelle Aktivierung des Hardwareeigentestes
BOOT		Manuelle Aktivierung des UNIX-Anfangsladers

Anfangslader

Nach dem Netzeinschalten wird, abhängig von den Hardwarebesonderheiten des Rechners, entweder der sofortige automatische UNIX-Anfangsladeprozeß initialisiert, oder dieser Start muß nach der EPROM-Monitormeldung manuell durch Eingabe eines Boot-Kommandos erfolgen. Wie im zweiten Kapitel bereits angedeutet wurde, beginnt jede UNIX-Initialisierung mit dem Einlesen eines 512 Byte langen Anfangsladers von der physischen Adresse 0 des Master-Massenspeichermediums (Hard-Disk). Unmittelbar nach dem Einlesen dieses ersten Boot-Programmes (primary bootstrapper) geht die Steuerung vom EPROM-Monitor auf den Anfangslader über. Er wird dann alles weitere einleiten, um den UNIX-Systemstart zu vollziehen.

3.3. Standalone-Softwarekomponenten des UNIX-Betriebssystems

Nachdem der Aufbau und die prinzipielle Arbeitsweise des UNIX-Dateiverwaltungssystems im Abschnit 2. behandelt wurde, soll nun der in den ersten Abschitten 3.1. und 3.2. dieses Kapitels vorgestellte Computer zu einem Multi-User-System unter Steuerung des Betriebssystems UNIX ausgebaut werden. Die ersten UNIX-spezifischen Softwarekomponenten, denen ein Anwender bei einer Systeminitialisierung begegnet, sind vier Standalone-Programme:

format	format	Formatierung des Master-Massendatenspeichers Hard-Disk (oder Floppy-Disk) als Systemspeicher.
mkboot	make primary bootstrapper	Schreiben des ersten 512-Byte-Anfangsladers auf den physischen Block 0 des vorher formatierten Master-Massendatenspeichers.
mkfs	make file system	Einrichten eines UNIX-Dateisystems.
restor	restore	Rückspeichern des UNIX-Betriebssystems von einem externen Back-Up-Datenträger (Magnetband, Magnetbandkassette oder Floppy-Disk).

Diese vier Programme existieren jeweils in zwei Versionen: einer Version, die ein ganz normales Superuser-Systemkommando darstellt, das unter Steuerung des Betriebssystems UNIX aufgerufen und abgearbeitet werden kann sowie einer Version, die einen Standalone-Charakter besitzt und demzufolge nicht ein lauffähiges Betriebssystem zur Abarbeitung voraussetzt. Die Standalone-Utilities finden immer dann Anwendung, wenn das UNIX-Betriebssystem noch nicht oder nicht mehr über den normalen Anfangsladevorgang aktiviert werden kann.
Die Standalone-Programme besitzen naturgemäß einen sehr engen Bezug zur gegebenen Computerhardware, so daß hier, obwohl es sich um UNIX-Standardsystemkomponenten handelt, unterschiedliche Programmversionen existieren können.

format

Wurde der Inhalt oder (und) die Struktur des Dateisystems auf dem Master-Massendatenspeicher - durch welches Ereignis auch immer - so zerstört, daß eine Wiederherstellung durch Softwaremaßnahmen (z.B mit dem Standalone-Programm 'fsck') nicht mehr möglich ist oder soll eine fabrikneue Hard-Disk (Floppy-Disk) als Systemspeicher initialisiert werden, muß als erstes die Standalone-Version des Formatierungsprogrammes 'format' angewendet werden, um schrittweise das UNIX-Betriebssystem wieder lauffähig zu machen. Das Formatierungsprogramm übernimmt die im Abschnitt 1.3.3. beschriebene Einteilung der Spur (track) einer Magnetplatte in eine bestimmte Anzahl von Sektoren gleicher Größe (512 Byte), es schreibt eine Reihe von logischen Feldern (ID-Feld, Datenfelder CRC-Feld) und übernimmt verschiedene Basisverifikationsüberprüfungen.
Zu beachten ist in jedem Fall, daß der Formatierungsprozeß alle auf dem magnetischen Datenträger eventuell noch vorhandenen Informationen zerstört und eine Back-Up-Kopie wertvoller Datenbestände zur Wiederherstellung des alten Dateninhaltes unbedingt notwendig ist.

mkboot

Das Standalone-Programm 'mkboot' übernimmt das Schreiben eines fest definierten 512-Byte-Anfangsladers auf den physischen Block 0 des vorher formatierten Master-Massendatenspeichers Hard-Disk (oder Floppy-Disk). Dieser 'primary bootstrapper' übernimmt beim manuellen oder automatischen Systemstart das Einlesen des eigentlichen UNIX-Systemladers ('secondary bootstrapper').

mkfs

Die Standalone-Version des Programmes 'mkfs' baut ein leeres UNIX-Dateisystem auf einem vorher formatierten externen Master-Massenspeicher Hard-Disk (oder Floppy-Disk) auf. Dazu wird der Superblock (logischer Block 1), die 'i-list' mit den 'i-node'-Verweisen (logische Blöcke 2 bis n) und die Freispeicherliste des Dateisystems angelegt (Abschnitt 2.1.1.). Die maximal mögliche Anzahl von Dateien wird entsprechend der vorgegebenen Dateisystemgröße (512-Byte-Blockanzahl) über einen festen Teiler im Programm 'mkfs' berechnet.

restor

Das Standalone-Programm 'restor' liest eine auf dem Back-Up-Speichermedium mit dem UNIX-Systemprogramm 'dump' angelegte komplette Kopie des Betriebssystems mit allen System- und Anwenderprogrammen auf ein vorher mit dem 'format' und 'mkfs' präpariertes Master-Massendatenspeichermedium ein.
Es ist zu beachten, daß die Standalone-Version des Programmes 'restor' sowohl das Root- als auch das Anwenderdateisystem wiederherstellt. Das entsprechende UNIX-Systemprogramm 'restor' kann, im Gegensatz dazu, das Root-Dateisystem nicht rückspeichern.

3.4. UNIX-Anfangsladeprozedur

Die Anfangsladeprozedur des Betriebssystems UNIX basiert auf zwei Programmen, dem 'primary bootsrapper' und dem 'secondary bootstrapper'.
Der 'primary bootstrapper' befindet sich immer resident im physisch ersten 512-Byte-Block auf dem externen Massenspeicher des UNIX-Betriebssystems. Er wird entweder automatisch oder auf manuelle Dialoganforderung hin nach dem RESET-Systemstart des Computers in den Hauptspeicher eingelesen. Seine auf das Einlesen folgende Abarbeitung hat das Suchen des als reguläre UNIX-Datei im ersten UNIX-Dateisystem auf der Systemplatte oder auf dem Back-Up-Speichermedium befindlichen 'secondary bootstrappers' zur Folge. Wird dieser eigentliche UNIX-Anfangslader - ein Programm mit einer Größe von bis zu 10 KByte - gefunden, wird es vom 'primary bootstrapper' eingelesen und zur Abarbeitung gebracht.
Der 'secondary bootstrapper' stellt ein autonom arbeitendes kleines Betriebssystem dar, unter dessen Steuerung jedes der im vorhergehenden Abschnitt behandelten Standalone-Programme (von einer eingerichteten Systemplatte oder von einem Back-Up-Speichermedium aus) abgearbeitet werden kann. Ist ein arbeitsfähiger externer Systemspeicher mit allen zum

UNIX-Systemstart notwendigen Dateien vorhanden, erfolgt vom 'secondary bootstrapper' das Einlesen und danach der Start des UNIX-Kerns, eine Prozedur, die, je nach Rechner, einige Sekunden dauern kann.
Der UNIX-Systemkern übernimmt die Formierung des gesamten Betriebssystems - den Aufbau der Systemtabellen, das Setzen von Anfangswerten sowie vieles andere mehr. Ist dieser Prozeß abgeschlossen, erfolgt durch die Ausgabe eines Prompt-Eingabebereitschaftszeichens vom UNIX-Kern die Fertigmeldung und der Übergang in den Single-User-Mode. Der am Systemterminal arbeitende Operator hat jetzt alle Ressourcen des UNIX-Systems zu seiner Verfügung und kann entweder den Übergang in den Multi-User-Mode einleiten ('login'-Meldung auf allen angeschlossenen Terminals) oder das System als einzelner Anwender nutzen.
Der Übergang vom Single-User-Mode in den Multi-User-Mode wie der gesamte Anfangsladevorgang läuft bei den meisten UNIX-Computersystemen weitestgehend automatisch ab. Dabei sind verschiedene Systemüberprüfungen eingebaut, die durch Operatorabfragen gegebenenfalls übersprungen werden können.
Das UNIX-Systemprogramm 'fsck' übernimmt dabei zum Beispiel die Überprüfung des erstellten (oder eines defekten) UNIX-Dateisystems auf innere Integrität und, bis zu einem bestimmten Maße, die Beseitigung (repair) von Fehlern.
Die Überprüfung des Dateisystems umfaßt sowohl die Überprüfung der Freispeicherliste, der 'i-list', der 'i-node'-Blöcke, jeder einzelnen Directory bzw. Datei, der Mehrfachdateiverweise ('links') und vieler anderer das Dateisystem kennzeichnender Informationen.
In Tafel 3.1 ist ein Beispiel für eine UNIX-Systeminitialisierung wiedergegeben.

Tafel 3.1. UNIX-Systeminitialisierung (Beispiel)

Rechneraktion und **Bedieneraktion**	Bemerkung
	Netzeinschalten und RESET.
U8000 Monitor Vers. 2.0	Meldung des EPROM-Monitors.
Press START to load System: **boot**	Eingabe des Boot-Kommandos.
	Einlesen (von Floppy-Disk) und Start 'secondary boot'.
Boot	'secondary boot' Promptzeichen ':'.
:format	Aufruf des Programmes 'format'.
UNIX Disk Formatter	
Drive (0-3):**0**	Eingabe der Disklaufwerksnummer.
Last chance before formatting unit 0	
Continue ? y	Eingabe (y) 'yes' oder (n)
Now formatting unit 0	
Format complete	
Drive (0-3):**(RETURN)(RETURN)**	Formatierung weiterer Laufwerke? Nein!
Boot	
:mkboot	Aufruf des Programmes 'mkboot'. Schreiben 'primary bootstrapper' auf die Hard-Disk.
Boot	
:mkfs	Aufruf des Programmes 'mkfs'.
file system size: **5000**	Eingabe der maximalen Blockanzahl (5000).
file system: **hd(0,13200)**	Eingabe des logischen Gerätenamens (hd), der physischen Platteneinheit (0) und eines Versatzes (13200 512-Byte-Blöcke) bezogen auf den Block 0 der Platteneinheit.
i-size 1600	Ausgabe der maximalen Dateianzahl des eingerichteten Dateisystems (1600 Dateien).
m/n = 12 72	Ausgabe des Versatzes von aufeinander folgenden 512-Byte-Blöcken auf einer Plattenspur (12) und der Anzahl der Sektoren auf einer Plattenspur (72).
Boot	
:restor	Aufruf des Programmes 'restor'.
file system: **hd(0,13200)**	Eingabe des Plattenbereiches.
Last chance before restore on disk	
Continue ? y	Eingabe (y) 'yes' oder (n)
Now restoring disk hd(0,13200)	
Restore complete !	

```
Boot
:
```

Starttastenbetätigung für automatischen Systemstart.

```
Welcome to UNIX Version 7.3

Do you want to check the file system?
Entry y for yes or n for no:y

checking /dev/hd0
        checking i-node and directory entries.
    ... checking tree structure
    problems were discovered
        repairing i-node and directory entries
    ... checking free list
    free list is ok
62398 total blocks in file system
0 bad blocks (0 percent)
53982 free blocks (86%)
20000 free i-node (96%)


UNIX shows the current date and time

Wed Nov 14  15:05:02  1986  PST
Please enter correct date or press RETURN: 15-nov-86 07:44

login: root
password:
#
```

4. Standardoperatorprozeduren

Der vierte Abschnitt ist eine Zusammenstellung von Standardoperatorprozeduren für die Arbeit am UNIX-Terminal. Beginnend mit dem 'login' (Nutzerzuschaltung) und 'logout' (Nutzerabschaltung) werden die wichtigsten Kommandofolgen für die Directory- und Dateimanipulation, für die Nutzung der On-line-Handbücher u.a.m. behandelt. Dieser Abschnitt ist besonders für die schnelle Einarbeitung in das Betriebssystemkonzept an einem lauffähigen UNIX-Computer vorgesehen. Dabei wird auf einige in den Abschnitten 5. und 6. (UNIX-Standardkommandos und Shellkommandointerpreter) noch ausführlich zu behandelnde Befehlsabläufe vorgegriffen.

4.1. Ein-/Ausschaltprozedur des UNIX-Betriebssystems

Die über eine asynchrone serielle V24-Schnittstelle an einem bereits laufenden UNIX-Rechner angeschlossenen Nutzerterminals bringen nach dem Netzeinschalten, spätestens jedoch nach dem ersten RETURN-Zeichen, die die Eingabebereitschaft anzeigende 'login'-Meldung des Betriebssystems.
Nach der Eingabe des Nutzernamens - dieser muß vorher einmalig vom Systemprogrammierer dem Betriebssystem bekannt gemacht worden sein - oder nach der Eingabe einer Sammelnutzerbezeichnung, wie 'user' oder 'guest' antwortet das Terminal entweder sofort mit dem Eingabebereitschaftszeichen (Promptzeichen) von UNIX

```
$   Multi-User-Promptzeichen von UNIX
#   Single-User- und Superuser-Promptzeichen von UNIX
```

oder es wird, falls das vorher von dem Nutzer einmal so festgelegt wurde, über die Ausgabe 'password:' ein spezielles Nutzerkodewort angefordert. Nach der Password-Eingabe (sie erscheint zum Schutz vor einem unberechtigt Zuschauenden nicht auf dem Bildschirm des Terminals) kann dann die Arbeit am Rechner beginnen. Das Password stellt eine beliebige ASCII-Zeichenfolge dar. Es dient zum Schutz der Dateien und Directories eines Nutzers vor unberechtigtem Zugriff.
Alle Password-Folgen eines Systems werden im verschlüsselten Zustand in der Systemdatei 'etc/passwd' abgespeichert. Sie sollten dort nur dem Superuser des jeweiligen UNIX-Rechners zugänglich (z.B. zum Löschen) sein, da ihre Entschlüsselung nicht prinzipiell unmöglich ist. Das Setzen oder Verändern des Schlüsselwortes erfolgt mit dem UNIX-Standardkommando 'passwd'.
Nach der Eingabe des Nutzernamens und des Schlüsselwortes wird vom Betriebssystem - wenn vorhanden - eine anwenderspezifische Initialisierungsdatei ('.profile') abgearbeitet. Diese Datei besteht aus einer frei wählbaren Folge von Standardkommandos des UNIX-Betriebssystems. Sie dient der schnellen Herstellung eines vom jeweiligen Nutzer als wünschenswert empfundenen Anfangszustandes für seine Arbeit am Terminal. Die '.profile'-Datei kann vom Nutzer mit einem Editor erstellt werden. Sie muß sich im jeweiligen 'login directory' des Anwenders befinden.

Nach dem 'login', der Password-Eingabe und der Abarbeitung der .profile'-Datei ist es mögich, daß, vom Systemprogrammierer (Superuser) veranlaßt, bestimmte Nutzerhinweise auf dem Bildschirm als 'message of the day' erscheinen (Tafel 4.1).
Ferner ist die Meldung 'you have mail' denkbar, ein Hinweis auf Post von anderen Nutzern des Systems über die Inter-User-Kommunikationsmittel des UNIX-Betriebssystems.
Sind alle Initialisierungsabläufe abgeschlossen, meldet sich UNIX standardmäßig mit dem Prompteingabezeichen ('$'), um die Bereitschaft zur Annahme von Kommandos anzuzeigen.
UNIX-Kommandos sind nichts anderes als Namen von Dateien, die ausführbare Programme oder Shellprozeduren enthalten. Kommandoparameter werden grundsätzlich, durch mindestens ein Leerzeichen vom Kommandonamen getrennt, auf der gleichen Eingabezeile angegeben, auf der sich der Kommandoname selbst befindet. Abgeschlossen wird eine Kommandoeingabe durch ein CR-Wagenrücklaufzeichen (auf dem Terminal meist als RETURN-Taste bezeichnet!). Auf die Eingabe dieses Zeichens folgt dann unmittelbar die Bearbeitung der Kommandoeingabe durch das Betriebssystem. Als erstes muß dabei vom Kommandointerpreter (shell) die angegebene Kommando- oder Shellprozedurdatei im UNIX-Dateibaum gesucht und zur Abarbeitung gebracht werden. Die Reihenfolge, in der die verschiedenen Directories eines konkreten Systems nach dem Dateinamen durchgemustert werden, ist standardmäßig festgelegt:

1. momentanes Arbeitsdirectory / 'current directory'
2. '/bin'-Directory
3. /usr/bin'-Directory.

Eine Beeinflussung dieses Standardsuchablaufes kann über die Shellvariable 'path' erfolgen.
Wird die Kommandodatei nicht gefunden, erfolgt eine Fehlermeldung und der Abbruch der Eingabebearbeitung.
Wurde bei der Kommandosuche ein ausführbares Programm gefunden, startet der Shellkommandointerpreter das Programm und geht danach sofort wieder dazu über, auf Benutzereingaben zu warten, die allerdings erst nach dem Ende der Programmabarbeitung bearbeitet werden.
Zu beachten ist deshalb, daß trotz einer eventuell momentan noch laufenden Kommandoabarbeitung mit Bildschirmausgaben - also schon, bevor das nächste Eingabebereitschaftszeichen erscheint - der Anwender bereits das nächste (oder die nächsten) Kommando(s) eingeben kann. Diese als 'type ahead' bezeichnte Eigenschaft des Betriebssystems UNIX hilft Zeit zu sparen, in dem in den Abarbeitungsphasen eines Systemkommandos oder eines Anwenderprogramms die nächsten Aktionen bereits eingeleitet werden können. Daß die Bildschirmausschriften dann manchmal etwas durcheinandergeraten können, sollte nicht stören.
Außerdem ist es möglich, mehrere Kommandos auf einer Befehlszeile, jeweils getrennt durch ein Semikolonzeichen (';'), anzugeben. Das abschließende RETURN-Zeichen startet dann die aufeinanderfolgende Abarbeitung nicht nur eines, sondern aller Kommandos.

Tafel 4.1. Ein-/Ausschaltdialog an einem UNIX-Nutzerterminal (Beispiel)

UNIX-Bedienerdialog	Bemerkung
login:**mayer1** password:**(ehrenfried)** --- Mitteilung an alle Nutzer --- Der Spooler-Drucker des Systems ist heute von 6-12 Uhr nicht in Betrieb ! --- Am System arbeiten: mueller tty01 nov 2 09:22 mayer2 tty03 nov 2 11:34 schulze tty09 nov 2 12:30 **$date** Mon Nov 3. 13:27:10 GMT 1986 $(CTRL-D) logout	Netzeinschalten am Terminal Name des 'login directory'. Schlüsselworteingabe (Erscheint nicht auf dem Bildschirm!). Ausgabe einer Mitteilung des Systemprogrammierers. Ausschrift, die durch die Abarbeitung der '.profil'-Datei entsteht (echo Am System arbeiten:; who). Meldung der UNIX-Eingabebereitschaft ('$'). Anforderung der Datumsausgabe. Kommandoabarbeitung! Eingabe des Abschaltkommandos. Netzausschalten am Terminal.

Erwähnt werden muß an dieser Stelle auch die Möglichkeit der Hintergrundverarbeitung von Programmen. Durch ein dem Namen eines Kommandos und eventuell folgenden Parametern nachgestelltes '&'-Sonderzeichen wird ein für die Hintergrundbearbeitung vorgesehenes Kommando gekennzeichnet. Dadurch können vom Anwender am Terminal z.B. ein rechenintensives Kommando im Hintergrund und ein ein-/ausgabeintensives Kommando im Vordergrund gleichzeitig zur (Multi-Task-) Abarbeitung gebracht werden (siehe auch Abschnitt 6.).

Will der Nutzer die Arbeit am Terminal nach einer UNIX-Sitzung beenden, muß das ASCII-Zeichen 'CTRL-D' als logout-Kommando eingegeben werden. Erst danach darf der Netzschalter des Terminals betätigt werden. Wird das 'CTRL-D' vergessen, kann das bei manchen UNIX-Implementationen eventuell zu späteren Problemen führen. (CTRL-D = CONTROL-Taste am Terminal drücken und **gleichzeitig** den Buchstaben 'D' eingeben.)

4.2. Zeichensatz bei der Arbeit mit dem UNIX-Betriebssystem

Die Arbeit mit dem UNIX-Betriebssystem geht grundsätzlich vom ASCII-Zeichensatz - auch als ISO-7-Bit-Kode bezeichnet - aus (Tafel 4.2). Insgesamt stehen hier dem Anwender 128 Zeichen (Kleinbuchstaben, Großbuchstaben und Sonderzeichen) zur Verfügung. Jedes der 128 Zeichen hat seine eigene Bedeutung. Doppelbelegungen existieren nicht auch nicht zwischen Klein- und Großbuchstaben. Einigen Sonderzeichen wurden spezielle Steuerfunktionen zugeordnet, die allerdings nicht immer ganz einheitlich bei allen UNIX-Rechnern Verwendung finden.
Bei der Arbeit am UNIX-Terminal sollten insbesondere die folgenden Hinweise beachtet werden.
Bei Schreibfehlern kann (solange nicht die RETURN-Taste betätigt wurde) mit den zwei Korrekturzeichen gearbeitet werden:

Löschen des letzten eingegebenen Zeichens:	BACKSPACE-Taste oder CTRL-H (CTRL-Taste am Terminal drücken und gleichzeitig den Buchstaben 'H' eingeben).
Löschen der letzten eingegebenen Zeile:	Delete-Line-Taste oder CTRL-U.

Wichtig ist außerdem zu wissen, daß die Abarbeitung jedes unter UNIX laufenden Programmes durch die Eingabe von CTRL-C abgebrochen werden kann. Terminalausgaben lassen sich mit CTRL-S stoppen (nicht abbrechen!) und mit CTRL-Q wieder fortsetzen.

Anmerkung:
In der Tafel 4.2 sind in der jeweils ersten Spalte die dezimalen und in der jeweils zweiten Spalte die hexadezimalen Werte des ASCII-Zeichens angegeben.
Für die Steuerzeichen (Hex-Kode 00 bis 1F) wurden neben der Steuerzeichenkennung wie 'NUL', 'SOH', 'STX' usw. auch die zugehörige CONTROL-Zeichenkodierung wie '^@', '^A', '^B' usw. angegeben. Dabei bedeutet '^A' daß die CONTROL-Taste auf dem Terminal gedrückt wird und gleichzeitig die Buchstabentaste 'A' betätigt wird. Das ist besonders bei den Terminals zu beachten, die keine oder nur unvollständige Steuerzeichentasten besitzen.

Tafel 4.2. ASCII-Zeichensatz des UNIX-Betriebssystems (ISO-7-Bit-Kode)

0	00	NUL	^@	32	20	SPC	64	40	@	96	60	
1	01	SOH	^A	33	21	!	65	41	A	97	61	a
2	02	STX	^B	34	22	"	66	42	B	98	62	b
3	03	ETX	^C	35	23	#	67	43	C	99	63	c
4	04	EOT	^D	36	24	$	68	44	D	100	64	d
5	05	ENQ	^E	37	25	%	69	45	E	101	65	e
6	06	ACK	^F	38	26	&	70	46	F	102	66	f
7	07	BEL	^G	39	27	'	71	47	G	103	67	g
8	08	BS	^H	40	28	(	72	48	H	104	68	h
9	09	HT	^I	41	29	)	73	49	I	105	69	i
10	0A	LF	^J	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	^K	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	^L	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	^M	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	^N	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	^O	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	^P	48	30	0	80	50	P	112	70	p
17	11	DC1	^Q	49	31	1	81	51	Q	113	71	q
18	12	DC2	^R	50	32	2	82	52	R	114	72	r
19	13	DC3	^S	51	33	3	83	53	S	115	73	s
20	14	DC4	^T	52	34	4	84	54	T	116	74	t
21	15	NAK	^U	53	35	5	85	55	U	117	75	u
22	16	SYN	^V	54	36	6	86	56	V	118	76	v
23	17	ETB	^W	55	37	7	87	57	W	119	77	w
24	18	CAN	^X	56	38	8	88	58	X	120	78	x
25	19	EM	^Y	57	39	9	89	59	Y	121	79	y
26	1A	SUB	^Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	^[	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	^\	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	^]	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	^^	62	3E	>	94	5E	^	126	7E	~
31	1F	US	^_	63	3F	?	95	5F	_	127	7F	DEL

Dabei bedeuten:

NUL	Null
SOH	Anfang des Kopfes (start of heading)
STX	Anfang des Textes (start of text)
ETX	Ende des Textes (end of text)
EOT	Ende der Übertragung (end of transmission)
ENQ	Antwortanforderung (enquiry)
ACK	positive Rückmeldung (acknowledge)
BEL	Klingel (bell)
BS	Rückwärtsschritt (backspace)
HT	Horizontaltabulator (horizontal tabulation)

LF	Zeilenvorschub (line feed)
VT	Vertikaltabulator (vertical tabulator)
FF	Formularvorschub (form feed)
CR	Wagenrücklauf (carriage return)
SO	Umschaltung (shift out)
SI	Rückschaltung (shift in)
DLE	Datenübertragung (data link escape)
DC	Gerätesteuerung (device control)
NAK	negative Rückmeldung (negative acknowledge)
SYN	Synchronisierung (synchronous idle)
ETB	Ende des Datenblocks (end of transmission block)
CAN	ungültig (cancel)
EM	Ende der Aufzeichnung (end of medium)
SUB	Zeichensubstitution (substitute character)
ESC	Umschaltung (escape)
FS	Dateitrennzeichen (file separator)
GS	Gruppentrennzeichen (group separator)
RS	Satzendezeichen (record separator)
US	Gerätetrennzeichen (unit separator)

4.3. Arbeit am UNIX-Terminal

Besonders einfach und unkompliziert gestaltet sich die Einarbeitung in den UNIX-Standardkommandosatz durch die Arbeit an einem lauffähigen UNIX-Nutzerterminal. Dieser Abschnitt ist deshalb so aufgebaut, daß der Leser - wenn realisierbar, durch gleichzeitiges Ausprobieren möglichst schnell den Zugang zu einigen besonders wichtigen Befehlsabläufen dieses Betriebssystems findet.
Die ersten, wichtigsten Kommandos, mit denen ein neuer Nutzer des UNIX-Systems in Berührung kommt, betreffen die Dateiarbeit. Im folgenden deshalb zunächst einige wichtige Befehlsabläufe für die Directory- und Dateimanipulation:

Erzeugen eines Directories
Löschen eines Directories ohne/Dateien bzw. mit Dateien
Ausgabe des Namens des momentanen Arbeitsdirectories
Ausgabe des Inhaltes eines Directories
Wechsel des Arbeitsdirectories

Erzeugen einer Datei
Umbenennen einer Datei
Löschen einer Datei
Ausgabe einer Datei - Ein-/Ausgaberedirektion
Duplizierung einer Datei
Erzeugen eines Dateimehrfachverweises
Durchmustern einer Datei - Pipekonzept
Compileraufruf einer Datei.

Wichtig für den Nutzer eines UNIX-Arbeitsplatzes sind außerdem Informationen über den momentanen Systemstatus und einige Kentnisse über die Kommunikation mit anderen Nutzern des gleichen Rechnersystems über die "elektronische Post":

Ermittlung der momentanen Systemnutzer
Ermittlung von Datum und Uhrzeit
Ermittlung der UNIX-Systemaktivitäten
Absenden von Post
Empfangen von Post.

Auf die ausführliche Beschreibung der Kommandos, insbesondere auf die Beschreibung der existierenden vielfältigen optionalen Kommandospezifikationen, wird aus Gründen der Einfachheit und Übersichtlichkeit verzichtet. Die ausführliche Kommandobeschreibung befindet sich im fünften Abschnitt.

4.3.1. Erzeugen eines Directories

Das Erzeugen eines neuen, zunächst leeren Directories erfolgt mit dem UNIX-Kommando 'mkdir' (make directory). Die auf den Kommandonamen folgende ASCII-Zeichenkette stellt den Namen des neuen Dateiinhaltsverzeichnisses dar.
In unserem Beispiel wird im momentanen Arbeitsdirectory ein Subdirectory mit dem Namen 'doc' angelegt.

```
$mkdir doc
$
```

Wird anstelle eines einzelnen Directories ein voller oder ein relativer Pfadname (z.B.: mkdir anwender/peter/doc) angegeben, wird das neue Dateiinhaltsverzeichnis nicht in der momentanen Arbeitsdirectory, sondern in dem durch den Pfadnamen spezifizierten Dateibaumzweig angelegt.

4.3.2. Löschen eines Directories

Das Löschen eines leeren Directories, also eines Dateiinhaltsverzeichnisses, in dem sich keine Dateien befinden, erfolgt mit dem Kommando 'rmdir' (remove directory). Die auf den Kommandonamen folgende ASCII-Zeichenkette - im Beispiel 'alle.texte' - stellt den Namen des zu löschenden Dateiverzeichnisses dar.

```
$rmdir alle.texte
$
```

Gibt man anstelle eines einzelnen Directories einen vollen oder relativen Pfadnamen (mkdir /anwender/peter/doc) an, wird das Dateiinhaltsverzeichnis nicht in der momentanen Arbeitsdirectory, sondern in dem durch den Pfadnamen spezifizierten Dateibaumzweig gelöscht.
Falls das zu löschende Directory nicht leer ist, also noch Dateien oder Subdirectories enthält, erfolgt eine Fehlerausschrift (cannot remove directory not empty), und der Löschversuch wird abgebrochen.

Soll ein Directory mit allen in ihm enthaltenen Dateien oder Subdirectories gelöscht werden, kann das durch die Verwendung des Kommandos 'rm' (remove) geschehen.

```
$rm -ri alle.texte
alle.texte/text1:y
alle.texte/text2:y
alle.texte/morefiles:y
alle.texte/morefiles/sub.text1:y
alle.texte/morefiles/sub.text2:y
alle.texte:y
$
```

Die Kommandoparameter und spezifizieren das '-Kommando zweifacher Weise.
Der Buchstabe ist die Rekursivanforderung, die dem Anwender die Eingabe jedes einzelnen Datei- und Subdirectorynamens bei der Löschanforderung erspart.
Der Buchstabe 'i' bewirkt, daß einzeln - für jede Datei, für jedes Subdirectory und zuletzt für das zu löschende Directory selbst - die Löschfreigabe angefordert wird.
Der den beiden Buchstaben 'r' und vorgestellte Bindestrich wird bei vielen UNIX-Systemkommandos zur Kennzeichnung der Kommandooptionen verwendet (Unterscheidung Parameterfeld von Datei- oder Directorybezeichnung).

An dieser Stelle ein Hinweis auf die 'gefürchtete' und in vielen Veröffentlichungen über UNIX beschriebene Variante des 'rm'-Kommandos, die ganze Dateisysteme mit einem Federstrich (oder besser mit einigen Tastenbewegungen) ohne irgendwelche Rückfragen oder Warnungen versinken läßt.

```
$rm -r directoryname
$
```

Der UNIX-Anwender hat hier natürlich sehr sorgfältig und überlegt zugehen. Ist das der Fall, dann besteht auch bei der Anwendung des Kommandos kein Grund zur Panik.

4.3.3. Ausgabe des Namens des momentanen Arbeitsdirectories

Die Ausgabe des Namens des momentanen Arbeitsdirectories erfolgt mit dem Kommando 'pwd' (print working directory').

```
$pwd
/anwender/peter/doc
$
```

Das 'pwd'-Kommando gibt immer den vollen Pfadnamen des momentanen Arbeitsdirectories, ausgehend von dem Root-Directory, auf dem Terminal aus.

4.3.4. Ausgabe des Inhaltes eines Directories

Die Ausgabe des Inhaltes eines Dateiinhaltsverzeichnisses - also die Ausgabe aller Namen der in ihm enthaltenen Dateien und Subdirectories erfolgt mit dem Kommando 'ls' (list).

```
$ls
text1          text2          morefiles
$
```

Wird kein zusätzlicher Kommandoparameter angegeben, erfolgt die Ausgabe des Directoryinhaltes in kurzer Form. Wird das 'ls'-Kommando durch den Parameter '-l' spezifiziert, erfolgt die Ausgabe in ausführlicher (langer) Form.

```
$ls -l
total 5
-rwxrwxr-x  1  mueller1    124   aug  20    11:54  morefiles
-rw-rw-rw-  1  mueller1   2970   sep   9    16:20  text1
-rw-rw-rw-  1  mueller1    197   sep   9    10:12  text2
$
```

Im langen Ausgabeformat wird der Dateimode (z.B. bei 'morfiles': read r, write w, execute x - user rwx, group rwx other r-x), die Anzahl der 'link'-Mehrfachverweise (1) der jeweiligen Datei, der Dateieigentümer (mueller1), die Dateigröße in Bytes (124), das Datum (20. August) sowie die Zeit der letzten Dateimodifikation (11:54) und schließlich der Datei- oder Directoryname (morefiles) selbst angegeben.
Das Kommando 'ls' kann, außer durch '1', durch viele weitere Parameterkombinationen spezifiziert werden, um Gruppen von Dateien, Zusatzinformationen u.v.a.m. zur Ausgabe zu bringen (Abschnitt 5).

4.3.5. Wechsel des Arbeitsdirectories

Der Wechsel des Arbeitsdirectories erfolgt mit dem Kommando 'cd' (change working directory). Die auf das 'cd'-Kommando folgende ASCII-Zeichenkette stellt den Namen des neuen Dateiinhaltsverzeichnisses dar.

```
$pwd
/anwender/peter
$cd bin
$pwd
/anwender/peter/bin
$
```

Der Wechsel des Arbeitsdirectories ist selbstverständlich der Zugriffserlaubnis des jeweiligen Nutzers entsprechend möglich.
Gibt man anstelle eines einzelnen Directorynamens im momentanen Arbeitsdirectory einen vollen oder relativen Pfadnamen an, wird der Wechsel des Arbeitsdirectories von dieser Spezifizierung ausgehend vorgenommen.

4.3.6. Erzeugen einer Datei

Das Erzeugen einer Datei kann auf sehr verschiedene Weise erfolgen. Eine von vielen Möglichkeiten ist die Benutzung eines der unter UNIX verfügbaren Texteditoren. In unserem Beispiel wird der Standardeditor 'ed' benutzt.

```
$ed datei1                      Aufruf des Editors 'ed' mit Angabe des
?datei1                         Namens der zu editierenden Datei.
a                               a-Kommando von 'ed' / Texteingabe.
Heute ist ein schoener Tag.     Eingabe des Textes der Datei 'datei1'.
Dies ist eine UNIX-Datei.
                                Punkt-Kommando von 'ed' / Textende.
w                               w-Kommando von 'ed' / Textschreiben.
54                              Zeichenanzahlausgabe des Editors.
q                               q-Kommando von 'ed' / Quit.
$
```

Die eingegebenen Zeichen werden beim 'ed'-Editor zunächst in einen internen Pufferspeicherbereich eingeschrieben und erst nach dem w-Kommando als Datei unter dem angegebenen Dateinamen ('datei1') im aktuellen Arbeitsdirectory angelegt.

4.3.7. Umbenennen einer Datei

Das Umbenennen einer bereits vorhandenen Datei erfolgt unter UNIX mit dem Kommando (move).

```
$mv  alter.name  neuer.name
$
```

Die auf die Eingabe des 'mv'-Kommandos folgende Notation des alten und des neuen Dateinamens wird, wie in allen andern Kommandos in solchen Fällen auch, jeweils durch mindestens ein Leerzeichen voneinander getrennt.
Durch das 'mv'-Kommando wird der Dateiinhalt in keiner Weise verändert. Es wird ausschließlich der Dateiname im Directory verändert. Eventuell vorhandene Dateimehrfachverweise (links) bleiben beim Umbenennen unverändert erhalten.

4.3.8. Löschen einer Datei

Das Löschen einer (oder mehrerer) Datei(en) in einem Directory erfolgt schon erwähnt mit dem Kommando (remove).

```
$rm  alle.texte*
$
```

In unserem Beispiel werden alle Dateien im momentanen Arbeitsdirectory, deren Namen mit der Zeichenkette 'alle.texte' beginnt, gelöscht. Durch die Angabe des '*'-Sonderzeichens nach der Zeichenkette 'alle.texte' wird spezifiziert, daß die dieser Zeichenkette folgenden Bezeichnungen im Dateinamen ignoriert werden können, so daß alle Dateien, die mit der

Zeichenkette 'alle.texte' beginnen, durch das 'rm'-Kommando zum Löschen ausgewählt werden.
Durch diese (und weitere unter UNIX verfügbare) nicht nur im Kommando zum Löschen einer Datei, sondern ganz allgemein bei allen Datei- und Directorynamen verwendbaren Abkürzungen lassen sich Dateigruppen auf besonders einfache Weise spezifizieren (siehe auch Abschnitt 6. Shellkommandointerpreter).

4.3.9. Ausgabe einer Datei-Ein-/Ausgaberedirektion

Die Ausgabe einer Datei auf dem Bildschirm des Terminals erfolgt mit dem Kommando 'cat' (concatenate and print)

```
$cat datei1
Heute ist ein schoener Tag.
Dies ist eine UNIX-Datei.
$
```

Die Datei mit dem angegebenen Pfadnamen wird bei Verwendung des 'cat'-Kommandos vollkommen unbearbeitet, so wie sie auf dem Datenträger abgespeichert ist, auf dem Bildschirm ausgegeben.
Da das 'cat'-Kommando die Angabe mehrerer Dateien zuläßt (dann sind jeweils die Pfadnamen durch ein Leerzeichen getrennt), ist auch die aufeinanderfolgende Ausgabe mehrerer Dateien möglich.

Am Beispiel des 'cat'-Kommandos lassen sich sehr einfach die im Betriebssystem UNIX implementierten Möglichkeiten der Ein-/Ausgaberedirektion andeuten.
Gibt man hinter dem angegebenen Dateinamen der auszugebenden Datei das für die Ein-/Ausgaberedirektion eines Datenstroms vorgesehene '-Zeichen und danach einen weiteren Dateinamen an, wird der beim 'cat'-Kommando normalerweise auf dem Bildschirm erscheinende Dateiinhalt in die durch den zweiten Dateinamen spezifizierte Datei umgeleitet.

```
$cat datei1 > datei2
$
```

Nach Abarbeitung der Kommandozeile im angegebenen Beispiel ist der Inhalt der beiden Dateien 'datei1' und 'datei2' identisch. Die Datei 'datei2' wird entweder automatisch neu angelegt oder ihr alter Inhalt geht bei dieser Prozedur verloren.
Bei der Ein-/Ausgaberedirektion von Datenströmen existieren unter UNIX vielfältige Möglichkeiten. So kann durch ein einfaches Ausgabekommando auf diese Weise eine Sammeldatei aus mehreren Teildateien zusammengestellt werden, es kann ein für ein bestimmtes Ein-/Ausgabegerät vorgesehener Datenstrom auf ein anderes Ein-/Ausgabegerät umgelegt werden und vieles andere mehr.

4.3.10. Duplizieren einer Datei

Natürlich existiert unter UNIX nicht nur die eben beschriebene Möglichkeit, eine Datei mit Hilfe der Ein-/Ausgaberedirektion zu duplizieren. Standardkommando für das Duplizieren von Dateien ist das Kommando 'cp' (copy).

```
$cp datei1 datei2
$
```

Der erste angegebene Dateipfadname im 'cp'-Kommando ist immer der Quelldatei, der zweite immer der Zieldatei vorbehalten. Durch Abarbeitung des 'cp'-Kommandos wird auch physisch eine Kopie der Quelldatei auf dem UNIX-Dateisystemdatenträger angelegt.
Existiert bereits eine Datei mit dem Dateinemen 'datei2' wird deren Inhalt bei der Ausführung des 'cp'-Kommandos überschrieben.

4.3.11. Erzeugen eines Dateimehrfachverweises

Das Erzeugen eines Mehrfachverweises auf eine im selben oder in einem anderen Directory des Dateisystems bereits existierende Datei erfolgt mit dem Kommando 'ln' (link).

```
$ln /anwender/ulli/programm1 /anwender/peter/doc/help
$
```

Der erste angegebene Dateipfadname (/anwender/ulli/programm1) spezifiziert beim 'ln'-Kommando eine bereits vorhandene Datei, der zweite (/anwender/peter/doc/help) stellt die neue Datei dar.
Existiert bereits eine Datei mit dem Dateinamen '/anwender/peter/doc/help' wird deren Inhalt bei der Ausführung des 'ln'-Kommandos überschrieben.
Durch die Erzeugung eines Dateimehrfachverweises wird eine Datei, die physisch nur ein einziges Mal auf dem externen Datenträger existiert, in mehrere Directories aufgenommen.

4.3.12. Durchmustern einer Datei – Pipekonzept

Das Durchmustern einer Datei nach einer definierten Zeichenkette erfolgt mit dem Kommando 'grep' (search a file for pattern).

```
$cat datei1
Peter sagt: Ja.
Ulli  sagt: Nein.
Heinz sagt: Hallo.
$grep Hallo datei1
Heinz sagt: Hallo.
```

Das Ergebnis der Abarbeitung des 'grep'-Kommandos ist ein standardmäßig auf dem Bildschirm des Terminals ausgegebener Zeichenstrom mit allen Zeilen der durchzumusternden Datei, die die spezifizierte Zeichenkette enthalten. In unserem Beispiel ist 'datei1' die Datei, die nach einer Zeichenfolge durchsucht werden soll 'Hallo' ist die zu suchende Zeichen-

kette. Die standardmäßig auf dem Bildschirm erscheinende Zeile ist also 'Heinz sagt: Hallo.'.

An diesem 'grep'-Kommandobeispiel läßt sich ein erster Einblick die Möglichkeiten des Pipekonzeptes geben.
Durch die auf die Notation des 'grep'-Kommandos folgende Angabe des Zeichens '|' wird ein Pipeinterprozeßkanal eröffnet, der dafür sorgt, daß der Datenausgabestrom, der bei der Abarbeitung des 'grep'-Kommandos entsteht, als Dateneingabestrom für die Abarbeitung eines unmittelbar folgenden Kommandos - bei uns für das 'wc'-Kommando - bereitgestellt wird.

```
$grep Hallo datei1 | wc
     1
$
```

Das Kommando 'wc' (word count) ist ein sehr einfaches Kommando. Es zählt bei der fehlenden Angabe einer Option die Zeilen einer Datei (oder eines Datenstromes) und gibt die ermittelte Zeilenanzahl auf dem Bildschirm des UNIX-Terminals aus. Im Beispiel eine '1'.
Ein Vorteil (von vielen weiteren) bei der Abarbeitung von Programmkommandofolgen mit Hilfe solcher Pipeinterprozesskanäle ist zum Beispiel das Einsparen von später nicht benötigten Zwischen- oder Hilfsdateien.

4.3.13. Compileraufruf einer Datei

Der Übersetzungsaufruf einer Datei in unserem Beispiel durch den C-Compiler des UNIX-Betriebssystems - erfolgt mit dem C-Compileraufruf
Das Übersetzungsprogramm, sei es nun ein Assembler, ein C-Compiler, ein FORTRAN-Compiler oder ein anderer Übersetzer, erzeugt je nach angegebenen Optionen oder auch mehrere neue Dateien (Objektprogramm, Listing usw.).

```
$cat prog3.c
main ()
{
printf ("\n\n");
printf ("\tHeute ist ein schoener Tag.");
printf ("\n");
printf ("\tDies ist eine Zeichenkettenausgabe.");
}
$ls
prog3.c
$cc prog3.c
$ls
prog3.c     a.out
$a.out

Heute ist ein schoener Tag.
Dies ist eine Zeichenkettenausgabe.

$
```

Wird der C-Compiler, wie angegeben, ohne Optionen aufgerufen, entsteht eine Datei mit dem Standardnamen 'a.out', die das übersetzte und gelinkte, sofort abarbeitungsfähige Maschinenprogramm der C-Quellprogrammdatei 'prog3.c' enthält.

4.3.14. Ermittlung der momentanen Systemnutzer

Die Ermittlung der momentanen Systemnutzer eines UNIX-Rechnersystems folgt mit dem Kommando 'who' (who is on system).

```
$who
mueller1  tty1  jul  24 08:04
mayer2    tty7  jul  24 06:07
schulze   tty8  jul  24 09:24
$
```

In unserem Beispiel sind im Moment der Abarbeitung des 'who'-Kommandos drei Nutzer des Systems an ihren Terminals tätig: Müller1, Mayer2 und Schulze an den Terminalarbeitsplätzen tty1, tty7 und tty8.

4.3.15. Ermittlung von Datum und Uhrzeit

Ermittlung von Datum und Uhrzeit (Systemdatum und Systemuhrzeit) erfolgt mit dem Kommando 'date' (print date).

```
$date
Mon  Apr 21  17:42:12  GMT 1986
$
```

Es handelt sich in unserem Beispiel um Montag, den 21.4.86. Die Uhrzeit wird in Stunden, Minuten und Sekunden angegeben. Die Zeitzonenangabe 'GMT' ist insbesondere bei sehr großen UNIX-Rechnernetzen von Bedeutung.
Das Setzen von Systemdatum und Systemuhrzeit ist dem Systemprogrammierer (Superuser) vorbehalten.

4.3.16. Ermittlung der UNIX-Systemaktivitäten

Die Ermittlung der momentanen Systemaktivitäten des UNIX-Rechners erfolgt mit dem Kommando 'ps' (print status).

```
$ps -a
PID  TTY          TIME      CMD
  0  ?          155:52    swapper
  1               0:03    /etc/init
 41  tty0         0:06    /etc/update
138  tty0         0:08    - sh
139  tty0         0:07    ps -
```

Das 'ps'-Kommando ermöglicht die Ermittlung des Status aller momentan laufenden Kommandoabarbeitungsaktivitäten. Dabei ist zu beachten, daß die Ausgabe des Systemstatus immer nur einen Augenblickszustand widerspiegelt,

der sich schon im Laufe der Terminalausgabe wieder ändern kann. Der 'ps'-Kommandoausdruck enthält die jeweilige Prozeßidentifikationsnummer (PID), die Terminalnummer (TTY), die den Prozeß auslöste, die kumulierte Prozeßabarbeitungszeit (TIME) und die Kommandobezeichnung (CMD).

4.3.17. Absenden von Post – Empfangen von Post

Das Absenden und das Empfangen von Post an einen bzw. von einem anderen Nutzer des gleichen UNIX-Rechnersystems erfolgt mit dem Kommando 'mail'

```
$mail username
Achtung Paul!
Wir haben die Datei 'file27' neu editiert.
(CTRL-D)
$
```

Die im Postsendekommando auf den Kommandonamen folgende ASCII-Zeichenkette 'username' stellt den 'login'-Namen des Nutzers dar, für den die Post bestimmt ist.
Die so mit der "elektronischen Post" abgesandte Nachricht wird zunächst so lange im UNIX-Systemspeicher zwischengespeichert, bis der Nutzer, für den die Post bestimmt ist, das nächste 'login' am UNIX-System abarbeitet. Bei der Abarbeitung des 'login'-Kommandos erkennt das Betriebssystem das eventuelle Vorhandensein von Post und gibt, wenn das der Fall ist, die Meldung 'you have mail' auf dem Terminal als Hinweis auf die Entgegennahme mit dem 'mail'-Kommando aus.

```
you have mail
$mail
Achtung Paul!
Wir haben die Datei file27 neu editiert.
$
```

Das Empfangen von Post erfolgt mit dem gleichen 'mail'-Kommando wie das Senden, nur daß kein Empfängername angegeben wird.

4.3.18. Schreiben an ein UNIX-Terminal

Das direkte Schreiben an ein anderes (momentan eingeschaltetes) UNIX-Terminal erfolgt mit dem Kommandoprogramm 'write'.

```
$write tty3
Achtung Paul!
Wir haben die Datei 'file27' neu editiert.
(CTRL-D)
$
```

Der Unterschied zum mail-Kommandoprogramm besteht darin, daß bei 'write' keine Zwischenspeicherung in Systemdateien erfolgt. Die Nachricht wird sofort auf dem Bildschirm des Empfängers (zeilenweise) ausgegeben.

5. UNIX-Standardkommandosatz

Die Behandlung des Standardkommandosatzes des Betriebssystems UNIX steht im Mittelpunkt des fünften Abschnitts. Der Kommandosatzumfang ergibt sich aus den Releasestufen UNIX Vers. 7 bzw. UNIX System III. Die für den Anwender wichtigen Kommandos werden in voller Breite mit allen Optionen dargestellt. Weniger wichtige Anwenderkommandos werden überblicksmäßig behandelt, und es wird auf weiterführende Literatur verwiesen. Die Kommandoprogramme des UNIX-Systemprogrammierers (Superuser) werden nur in den Übersichtstafeln berücksichtigt.

5.1. Auswahl des UNIX-Standardkommandosatzes

Eine ausführliche Behandlung aller Kommandoprogramme der UNIX-Standardversionen ist aus Gründen des Umfangs an dieser Stelle prinzipiell nicht möglich. Deshalb wurde eine Auswahl getroffen, die die für den Anwender normalerweise wichtigen Standardkommandoprogramme enthält. Die dem Systemprogrammierer vorbehaltenen Superuser-Kommandoprogramme wurden nur in die Übersichtstafeln aufgenommen, auf ihre Beschreibung kann hier verzichtet werden, da der allgemeine UNIX-Nutzer sich dieser Kommandos in einem Multi-User-System ohnehin nicht bedienen kann.
Einige Kommandos, wie zum Beispiel der adb-Debugger, die awk-Sprache, der bas-Compiler usw., stellen komplette eigenständige Programmiersysteme dar. Sie haben eine umfangreiche Eingabesyntax, so daß hier (auch aus Platzgründen) auf weiterführende Literatur verwiesen werden muß.
Der Shellkommandointerpreter und der C-Compiler werden wegen ihrer besonderen Bedeutung innerhalb des UNIX-Systems in den Abschnitten 6. und 7. ausführlich behandelt.
Der Leser hat bei der Beschreibung des UNIX-Systemkommandos zu beachten, daß, trotz der weitgehenden Standardisierung dieses Betriebssystems, bei unterschiedlichen Implementationen mehr oder weniger voneinander abweichende Varianten von Eingabesyntax, Optionen, Fehlermeldungen u.a.m. existieren können.
An dieser Stelle sei auch nochmals darauf hingewiesen, daß jedes UNIX-Kommando ein ganz normales unter Steuerung dieses Betriebssystems laufendes Programm darstellt, das auf die Systemressourcen über die allen zur Verfügung stehenden UNIX-Systemcalls zugreift. Der UNIX-Kommandosatz ist also offen und kann sehr freizügig erweitert oder verringert werden abhängig von den jeweiligen Anwendererfordernissen.

5.2. UNIX-Kommandoübersicht

Die in der Tafel 5.1 wiedergegebene alphabetische und aufgabenbezogene Kommandoübersicht des Betriebssystems UNIX bezieht sich auf den Standardkommandosatz der Releasestufen UNIX Vers. 7 bzw. UNIX System III.

Die Superuser-Systemkommandoprogramme wurden in der Tafel 5.1 mit einem 'S' gekennzeichnet. Mit 'W' gekennzeichnete Kommandoprogramme werden nur überblicksmäßig vorgestellt, und es wird auf weiterführende Literatur verwiesen. Kommandos, die weder mit einem 'S' noch mit einem 'W' gekennzeichnet sind, werden in den nachfolgenden Abschnitten ausführlich behandelt.
Beim Nachschlagen im "UNIX Programmers Manual" /1/ /2/ /3/, dem Standardwerk des UNIX-Nutzers, ist wichtig zu wissen, daß in allen Ausgaben eine einheitliche Einteilung in neun Sektionen existiert:

Sektion 0	Introduction (0)
Sektion 1	User Commands (1)
Sektion 2	System Calls (2)
Sektion 3	Subroutines (3)
Sektion 4	Special Files (4)
Sektion 5	Files and Conventionen (5)
Sektion 6	Games (6)
Sektion 7	Program Support Files / Macros and Conventionen (7)
Sektion 8	System Administrator Commands / Maintenance (8)

Die angegebenen Sektionsnummern (0), (1) ... (8) befinden sich auch auf der Kopfzeile jeder Seite dieses Manuals hinter dem auf der Seite beschriebenen Kommando, Systemaufruf oder Unterprogramm. Innerhalb der Sektionen gilt eine alphabetische Reihenfolge. Durch dieses System kann jede gewünschte Information recht schnell gefunden werden. Die Erläuterungen im "UNIX Programmers Manual" sind allerdings trotz der nahezu 1000 Seiten einiger Ausgaben dieses Werkes meist maximal komprimiert und manchmal auch etwas akademisch. Deshalb wird von den Autoren des vorliegenden Buches bei der Behandlung der nicht vollständig beschriebenen UNIX-Kommandoprogramme auf einige deutsch- und englischsprachige Veröffentlichungen hingewiesen, die ihnen, bezogen auf die Darstellung des jeweiligen Kommandoprogrammes, besonders geeignet erscheinen.
Zu beachten ist außerdem, daß bei der Arbeit am UNIX-Terminal zwei nützliche Hilfsmittel dem Anwender zur Verfügung stehen, das rechnergestützte Lernprogramm 'learn' und die sog. On-line-Dokumentation 'man'.
Durch den Aufruf des Kommandoprogramms 'learn', verbunden mit einer Reihe auswählbarer Schlüsselwörter, kann eine rechnergestützte Unterweisung für verschiedene UNIX-Themenkomplexe angefordert werden.
Mit Hilfe des Kommandos 'man' ist der Abruf der rechnerintern gespeicherten Dokumentation jedes einzelnen UNIX-Kommandos möglich. Die Ausgabe erfolgt standardmäßig auf dem Bildschirm, kann nötigenfalls aber auch auf den Drucker umgeleitet werden.

Tafel 5.1. Alphabetische und aufgabenbezogene UNIX-Kommandoübersicht

ac	S	login accounting	Nutzerprotokollierung/-abrechnung
adb	W	debugger	interaktiver Programmtestdebugger
ar		archive	Archiv- und Bibliotheksverwaltung
as		assembler	Assemblerprogrammübersetzer
at		later command execution	zeitgesteuerte Kommandoabarbeitung
awk	W	pattern processing	Textfilterprogramm
bc	W	C-interface calculator	Arithmetik-Tischrechnerprogramm
cal		print calendar	Kalenderausgabe
calendar		automatic reminder	automatischer Terminkalender
cat		concatenate and print	Dateiausgabe
cb		C program beautifier	C-Programm-Formatierungshilfe
cc		C compiler	C-Programmübersetzer
cd		change working directory	Wechsel des Arbeitsdirectories
chmod		change mode	Setzen Dateizugriffserlaubnisbits
chown		change owner	Wechsel der Besitzerzugehöhrigkeit
chgrp		or group	oder der Gruppenzugehörigkeit
clri	S	clear i-node	Löschen einer i-node
cmp		compare two files	Vergleich zweier Dateien
col	W	filter reverse LF	Spaltendrucktextfilter
comm		select or reject lines	Suchen gleicher Zeilen in Dateien
cp		copy	Kopieren von Dateien
crypt		decode and encode	Verschlüsseln/Entschlüsseln
csh	W	Cshell	Cshell Kommandointerpreter
cu	W	call UNIX	UNIX-Rechnerkopplung
date		print date	Ausgabe von Datum und Uhrzeit
dc	W	desk calculator	Tischrechnerprogramm
dcheck	S	file directoty chek	Konsistenzüberprüfung von Directories
dd		convert and copy	Konvertieren/Umkopieren
deroff	W	remove nroff constructs	Entfernen aller Formatierungsbefehle
df	S	disk free	Ausgabe der freien Dateisystemblöcke
diff		file comperator	Ermittlung von Dateiunterschieden
du		summarize disk usage	Ermittlung Plattenspeicherbelegung
dump	S	file system dump	Herstellen eines Dateisystemabzuges
echo		echo argument	Echoausgabe
ed		text editor	Texteditor
enroll		secret mail	Schlüsselwortvergabe,
xsend			Senden und Empfangen
xget			geheimer elektronischer Post
eqn	W	typset mathematics	Ausgabe mathematischer Formeln
expr		evaluate expression	Berechnung arithmetischer Ausdrücke
f77	W	FORTRAN77 compiler	FORTRAN77-Programmübersetzer
file		determine file typ	Ermittlung des Dateityps
find		find files	Suchen von Dateien

grep		search file for pattern	Durchmustern von Dateien
egrep		extend grep	erweitertes Durchmustern
fgrep		fast grep	beschleunigtes Durchmustern
icheck	S	file system chek	Dateisystemkonsistenzüberprüfung
iostat	S	report i/o statistics	Ausgeben einer Ein-/Ausgabestatistik
join		database operator	identifikationszeichenbedingte Ausgabe
kill		terminate process	Prozeßbearbeitungsabbruch
ld	W	loader	UNIX-Lader
learn		comuter aided learn	UNIX-Einführungslernprogramm
lex	W	lexical analysis	Programmgenerator für lex. Analyse
lint		C program verifier	C-Syntaxprüfer
ln		make a link	Herstellen einer Dateilinkverbindung
login		UNIX system login	UNIX-Nutzerzuschaltung
look		find lines	Suchen von Dateizeilen
lorder		find relationen	Ermitteln von Objektprogrammbezügen
lpr		line printer spooler	Druckerausgabeprogramm
ls		list directory contents	Ausgabe des Directorieinhaltes
m4	W	macro processor	Makroprozessor
mail		send or receive mail	Senden und Empfangen von Post
make	W	maintain program group	Programmerzeugungshilfe
man		print manual	Ausgabe einer Kommandobeschreibung
mesg		permit or deny messages	Nachrichtenempfangssperre
mkconf	S	generate conf tables	Generierungstabellenein-/-ausgabe
mkdir		make a directory	Anlegen eines Directories
mkfs	S	make a file system	Anlegen eines UNIX-Dateisystems
mknod	S	make a special file	Anlegen einer Ein-/Ausgabedatei
mount	S	mount/dismount	Mount/Dismount UNIX-Dateisystem
mv		move or rename files	Verschieben / Umbenennen von Dateien
ncheck	S	generate i-number names	i-number-Pfadnamenbestimmung
newgrp		log in a new group	Ändern der Gruppenzugehörigkeit
nice		run at low priority	Wechsel der Abarbeitungspriorität
nohup			
nm		print name list	Symboltabellenausgabe
od		octal dump	Dumpausgabe
passwd		change login passwd	Schlüsselwortvergabe
pr		print file	formatierte Dateiausgabe
prof		display profile	Ausgabe Programmabarbeitungsprofil
ps		process status	Prozeßstatusermittlung
pstat	S	print system facts	Ausgabe von UNIX-Systemtabellen
ptx		print index	Wortindexerstellung
pwd		print working directory	Ausgabe des Arbeitsdirectories
quot	S	file system ownership	Ausgabe der Dateisystemnutzerblöcke

ratfor	W	rational FORTRAN	FORTRAN-Preprocessor
restor	S	file system restor	Rückspeichern eines Dateisystems
rm rmdir		remove files	Löschen von Dateien und Directories
sa	S	system accounting	Systemabrechnung
sed	W	stream editor	Datenstromeditor
sh		shell command language	Shellkommandointerpreter
size		size of object file	Ausgabe einer Objektprogrammgröße
sleep		suspend execution	Abarbeitungspause
sort		sort or merge files	Sortierprogramm
spell		find errors	Suchen von Rechtschreibfehlern
split		split a file	Aufsplitten einer Datei
strip		remove symbols	Entfernen der Symboltabelle
struct	W	structure FORTRAN	Strukturieren von FORTRAN-Programmen
stty		set terminal options	Setzen von Terminalparametern
su		substitute user id	Erweitern der Benutzerzugriffsrechte
sum		sum and count blocks	Bestimmung Blockanzahl/Prüfsumme
sync	S	update super block	Aktualisieren des Superblocks
tabs		set terminal tabs	Setzen Terminaltabulatoreinstellung
tail		deliver part of a file	Übergehen von Dateiteilen
tar		tape archiver	Magnetbandarchivar
tbl	W	format tables	Tabellenformatierungsprogramm
tee		pipe fitting	Ein-/Ausgabepipeverzweigung
test		condition command	Testkommando
time		time a command	Bestimmung einer Ausführungszeit
touch		update file	Aktualisierung Dateidatum/-zeit
tr		translate characters	Zeichenkettenkonvertierung
troff nroff	W	text formatting	Textformatierung
true		provide truth values	erzwungener Exitstatus TRUE/FALSE
tty		get terminal name	Ermittlung des Terminalnamens
uniq		report repeated lines	Löschen doppelter Dateizeilen
units		conversion program	Einheitenumrechnungsprogramm
uucp	W	UNIX to UNIX copy	UNIX-UNIX Dateikopierprogramm
uux	W	UNIX to UNIX execution	UNIX-UNIX Kommandoausführung
wait		await of process	Warten auf einen Hintergrundprozeß
wall	S	write to all users	Schreiben an alle UNIX-Nutzer
wc		word count	Wortzählungsprogramm
who		who is on system	Wer arbeitet am System ?
write		write another user	Schreiben an andere Nutzer
yacc	W	compiler-compiler	Compiler-Compiler

Systemzugriffskontrolle		
login	UNIX system login	UNIX-Nutzerzuschaltung
newgrp	log in a new group	Ändern der Gruppenzugehörigkeit
passwd	change login passwd	Schlüsselwortvergabe
su	substitute user id	Erweitern der Benutzerzugriffsrechte
sync S	update super block	Aktualisieren des Superblocks
Systemverwaltung/Statusinformation		
ac S	login accounting	Nutzerprotokollierung/-abrechnung
date	print date	Ausgabe von Datum und Uhrzeit
dd	convert and copy	Konvertieren/Umkopieren
du	summarize disk usage	Ermittlung der Plattenspeicherbelegung
dump S	dump file system	Herstellen Dateisystemabzug
icheck S	file system check	Dateisystemkonsistenzüberprüfung
iostat S	report i/o statistics	Ausgabe der Ein-/Ausgabestatistik
learn	computer aided learn	UNIX-Einführungslernprogramm
man	print manual	Ausgabe einer Kommandobeschreibung
mkconf S	generate conf tables	Ein-/Ausgabe Generierungstabelle
mkfs S	make a file system	Anlegen eines UNIX-Dateisystems
mknod S	make a special file	Anlegen einer Ein-/Ausgabedatei
mount S	mount/dismount	Mount/Dismount eines Dateisystems
ps	process status	Prozeßstatusermittlung
pstat S	print system facts	Ausgabe von UNIX-Systemtabellen
quot S	file system ownership	Ausgabe der Dateisystemnutzerblöcke
restor S	file system restore	Rückspeichern eines Dateisystems
sa S	system accounting	Systemabrechnung
Terminal- und Druckerarbeit		
echo	echo argument	Echoausgabe
lpr	line printer spooler	Druckerausgabeprogramm
stty	set terminal options	Setzen von Terminalparametern
tabs	set terminal tabs	Setzen Terminaltabulatoreinstellung
tty	get terminal name	Ermittlung des Terminalnamens
who	who is on system	Wer arbeitet am System ?
Kommandointerpreter		
csh W	C shell	C-Shellkommandointerpreter
sh W	shell command language	Shellkommandointerpreter

Dateiverwaltung		
cat	concatenate and print	Dateiausgabe
cd	change working directory	Wechsel des Arbeitsdirectories
chmod	change mode	Setzen Dateizugriffserlaubnisbits
chown	change owner	Wechsel der Besitzerzugehörigkeit
chgrp	or group	oder der Gruppenzugehörigkeit
clri S	clear i-node	Löschen einer i-node
cmp	compare two files	Vergleich zweier Dateien
comm	select or reject lines	Suchen gleicher Zeilen in Dateien
cp	copy	Kopieren von Dateien
crypt	decode and encode	Verschlüsseln/Entschlüsseln
dcheck S	file directory check	Konsistenzprüfung von Directories
df	disk free	Ausgabe freier Dateisystemblöcke
diff	file comperator	Ermittlung von Dateiunterschieden
file	determine file typ	Ermittlung eines Dateityps
find	find files	Suchen von Dateien
ln	make a link	Herstellen einer Dateilinkverbindung
ls	list directory contents	Ausgabe des Directorieinhaltes
mkdir	make a directory	Anlegen eines Directories
mv	move or rename files	Verschieben / Umbenennen von Dateien
ncheck S	generate i-number names	i-number-Pfadnamenbestimmung
od	octal dump	Dumpausgabe
pwd	print working directory	Ausgabe des Arbeitsdirectories
rm	remove files	Löschen von Dateien und
rmdir		Directories
split	split a file	Aufsplitten einer Datei
sum	sum and count blocks	Bestimmung Blockanzahl/Prüfsumme
tail	deliver part of a file	Übergehen von Dateiteilen
touch	update file	Aktualisierung Dateidatum/-zeit
Kommunikation		
cu W	call UNIX	UNIX-Rechnerkopplung
enroll	secret mail	Schlüsselwortvergabe,
xsend		Senden und Empfangen
xget		geheimer Post
mail	send or receive mail	Senden und Empfangen von Post
mesg	permit or deny messages	Nachrichtenempfangssperre
uucp W	UNIX to UNIX copy	UNIX-UNIX Dateikopierprogramm
uux W	UNIX to UNIX execution	UNIX-UNIX Kommandoausführung
wall S	write to all users	Schreiben an alle UNIX-Nutzer
write	write another user	Schreiben an andere Nutzer

Softwareprojektunterstützung		
ar	archive	Archiv- und Bibliotheksverwaltung
cal	print calendar	Kalenderausgabe
calendar	automatic reminder	automatischer Terminkalender
make W	maintain program group	Programmerzeugungshilfe
tar	tape archiver	Magnetbandarchivar
Programmabarbeitungsunterstützung		
at	later command execution	zeitgesteuerte Kommandoabarbeitung
expr	evaluate expression	Berechnung arithmetischer Ausdrücke
kill	terminate process	Prozeßabarbeitungsabbruch
nice	run at low priority	Wechsel der Abarbeitungspriorität
nohup		
prof	display profile	Ausgabe Programmabarbeitungsprofil
sleep	suspend execution	Abarbeitungspause
tee	pipe fitting	Ein-/Ausgabepipeverzweigung
test	condition command	Testkommando
time	time a command	Bestimmung einer Ausführungszeit
true	provide truth values	erzwungener Exitstatus TRUE/FALSE
units	conversion program	Einheitenumrechnungsprogramm
wait	await of process	Warten auf einen Hintergrundprozeß
Programmiersprachen		
adb W	debugger	interaktiver Programmtestdebugger
as	assembler	Assemblerprogrammübersetzer
bc W	C-interface calculator	Arithmetik-Tischrechnerprogramm
cb	C program beautifier	C-Programm-Formatierungshilfe
cc	C compiler	C-Programmübersetzer
dc W	desk calculator	Tischrechnerprogramm
f77 W	FORTRAN77 compiler	FORTRAN77-Programmübersetzer
ld W	loader	UNIX-Lader
lex W	lexical analysis	Programmgenerator für lex. Analyse
lint	C program verifier	C-Syntaxprüfer
lorder	find relationen	Ermittlung von Objektprogrammbezügen
m4 W	macro processor	Makroprozessor
nm	print name list	Symboltabellenausgabe
ratfor W	rational FORTRAN	FORTRAN-Preprocessor
size	size of object file	Ausgabe einer Objektprogrammgröße
strip	remove symbols	Entfernen der Symboltabelle
struct W	structure FORTRAN	Strukturieren von FORTRAN-Programmen
yacc W	compiler-compiler	Compiler-Compiler

Textverarbeitung			
awk	W	pattern processing	Textfilterprogramm
col	W	filter reverse LF	Spaltendrucktextfilter
deroff	W	remove nroff commands	Entfernen aller Formatierungsbefehle
ed		text editor	Texteditor
eqn	W	typset mathematics	Ausgabe mathematischer Formeln
grep		search file for pattern	Durchmustern von Dateien
egrep		extend grep	erweitertes Durchmustern
fgrep		fast grep	beschleunigtes Durchmustern
join		database operator	identifikationszeichenbedingte Ausgabe
look		find lines	Suchen von Dateizeilen
pr		print file	formatierte Dateiausgabe
ptx		print index	Wortindexerstellung
sed	W	stream editor	Datenstromeditor
sort		sort or merge files	Sortierprogramm
spell		find errors	Suchen von Rechtschreibfehlern
tbl	W	format tables	Tabellenformatierungsprogramm
tr		translate characters	Zeichenkettenkonvertierung
troff nroff	W	text formatting	Textformatierung
uniq		report repeated lines	Löschen doppelter Dateizeilen
wc		word count	Wortzählungsprogramm

5.3. Beschreibung der UNIX-Systemkommandoprogramme

Bei der Behandlung der wichtigsten UNIX-Systemkommandoprogramme werden die folgenden Konventionen in der Aufrufbeschreibung verwendet:

- Die fett gedruckte Zeichenfolge des Kommandonamens ist exakt (wie abgedruckt) einzugeben (z.B. 'adb', 'ar' usw.).
- Fett gedruckte Zeichenfolgen bei Kommandoparameterangaben weisen auf die Eingabepflicht dieser Parameter hin (z.B. 'key' und 'afile' beim ar-Kommandoprogramm).
  Vom Anwender sind diese Kommandoparameterangaben beim Kommandoaufruf, seinen Bedürfnissen entsprechend, zu aktualisieren.
- Optionale Kommandoparameterangaben werden in Klammern '[ ]' eingeschlossen (z.B. '-w' 'objfil' und 'corfil' beim ar-Kommandoprogramm).
  Vom Anwender sind diese optionalen Kommandoparameterangaben beim Kommandoaufruf zu aktualisieren.
- Auf Kommandoparameterangaben folgende Punkte ('...') kennzeichnen gegebenenfalls folgende gleichartige Parametereingaben (z.B. 'name ...' beim ar-Kommandoprogramm).
- Bei einigen Kommandoprogrammen (z.B. 'as', 'cb', 'cc' usw.) werden optionale oder nichtoptionale Kommandoparameterangaben durch eine (aus einem Punkt '.' und einem Buchstaben bestehende) nachgestellte Zeichenkombination abgeschlossen. Diese nachgestellte Zeichenkombination weist auf einen bestimmten Dateiinhalt hin und ist bei der Aktualisierung der Kommandoparameterangaben durch den Anwender unverändert zu übernehmen.

Kommando: adb interaktiver Programmtestdebugger (debugger)

Aufruf: **adb** [-w][objfil [corfil]]

Beschreibung:
Der adb-Debugger ist ein besonders für in Assembler und C geschriebene Anwenderprogramme vorgesehenes interaktiv arbeitendes Programmtesthilfswerkzeug. Er ermöglicht die gesteuerte Abarbeitung von Anwenderprogrammen in einer definierten Softwareumgebung. Dabei werden der symbolische Zugriff zu lokalen oder globalen Programmvariablen, die Einzelschrittabarbeitung, Disassemblerfunktionen und vieles andere mehr unterstützt.
Das Testobjekt ist eine ausführbare Programmdatei ('objfil' - Standard: 'a.out'), eine Speicherabzugsdatei ('corfil' - Standard: 'core') oder beides zusammengenommen. Durch eine spezielle Option '-w' kann eine eventuell beim adb-Programmtest notwendig werdende Modifikation von 'objfil' und 'corfil' erlaubt oder - wenn nicht vorhanden - verboten werden.
Voraussetzung für die effektive Anwendung des adb-Debuggers ist die Beherrschung der rechnerspezifischen Maschinensprache.

in [1] J.F. Maranano, S.R. Bourne: A Tutorial Introduction to ADB

Kommando: Archiv- und Bibliotheksverwaltungsprogramm (archive and library maintainer)

Aufruf: **key** [posname] **afile** [name ...]

Beschreibung:
Das Archiv- und Bibliotheksverwaltungsprogramm 'ar' ermöglicht die Behandlung von mehreren Einzeldateien ('name ...') in einer zusammenfassenden Archivdatei ('afile'), die, außer für Archivierungsaufgaben, auch als Bibliotheksdatei von dem UNIX-Lader 'ld' verarbeitet werden kann. Die Archivdatei wird, wenn nicht vorhanden, beim ersten Replaceaufruf erstellt und kann dann fortgeschrieben (replace) und wieder aufgelöst (extract) werden.
Folgender Kommandoschlüssel 'key'=(uabivcl)(rdxtpmq) wird benutzt:

r replace — Ersetzen benannter Datei(en) in der Archivdatei.
Mit r kann stehen:
- u — Es werden nur die Dateien ersetzt, die ein jüngeres Erstellungsdatum haben als die entsprechende Datei in der Archivdatei.
- b, i — Die Dateien werden nach (a - after) oder vor (b und i - before) der Position der Datei mit den Namen 'posname' in der Archivdatei angeordnet. Standardmäßig finden ersetzte Dateien am Ende des Archivs Platz.

d delete — Löschen der benannten Datei(en) aus der Archivdatei.

x extract — Rückspeichern der Einzeldateien aus der Archivdatei in das UNIX-Dateisystem. Gibt man im Kommandoaufruf keine Einzeldateinamen an, wird der gesamte Einzeldateibestand der Archivdatei rückgespeichert. Die Archivdatei bleibt unverändert.

t table — Ausgabe des Archivinhaltsverzeichnisses. Werden einzelne Dateinamen im Kommando angegeben, erscheinen nur die angegebenen Dateien im Archivinhaltsverzeichnis. Die Schlüsselbuchstabenkombination 'vt' erzeugt ein alle Dateiinformationen umfassendes Ausgabeprotokoll.

m — Verschieben der angegebenen Datei(en) innerhalb der Archivdatei an das Archivdateiende oder an eine wie bei (replace) zu spezifizierende Position in der Archivdatei ('posname' - a, b, i).

p print — Ausgeben des Inhaltes der Archivdatei auf der Standardausgabe. Werden einzelne Dateinamen im Kommando angegeben, werden nur die angegebenen Dateien ausgegeben.

q quick — Aufnehmen der angegebenen Datei(en) in das Archiv ohne Überprüfung, ob die Datei(en) eventuell schon im Archiv vorhanden ist (anfügen an das Archivdateiende).

v verbose — Protokollierung aller Aktivitäten.

c create — Unterdrückung einer speziellen create-Meldung beim replace-Aufruf einer neuen Archivdatei.

l local — Die benötigten temporären Dateien ('/tmp') werden im momentanen Arbeitsdirectory angelegt.

Kommando: Assemblerprogrammübersetzer (assembler)

Aufruf: [option] **file.s**

Beschreibung:
Das Kommandoprogramm 'as' ist der rechnerspezifische Assemblerprogrammübersetzer. Der Übersetzungsprozeß der zu übersetzenden Datei(en) 'file.s' kann durch folgende Optionen spezifiziert werden:

-l	Erzeugen einer Listingdatei 'file.l'.
-o objfile	Das übersetzte Assemblerprogramm wird nach dem impliziten Aufruf des UNIX-Laders 'ld' nicht mit dem Standardnamen für ein abarbeitungsbereites Maschinenprogramm 'a.out' versehen, sondern unter dem angegebenen Namen 'objfile' abgelegt.
-p	Das Listing des übersetzten Programmes wird beim Übersetzen auf dem Standardausgabegerät ausgegeben.
-u	Unaufgelöste Symbole im zu übersetzenden Programm werden wie externe Variablen behandelt.

Kommando: at zeitgesteuerte Kommandoabarbeitung (execute commands at a later time)

Aufruf: **at time** [day] [file]

Beschreibung:
Das Kommandoprogramm 'at' ermöglicht die vorprogrammierte Abarbeitung einer Datei mit einem oder mehreren Kommandos bzw. mit kompletten Shellskripten zu einer bestimmten Uhrzeit an einem bestimmten Tag der Woche oder eines Monats. Gibt man keinen Dateinamen an, wird vom Kommandoprogramm 'at' der Standardeingabezeichenstrom verarbeitet.

time	Uhrzeit der Kommandoabarbeitung (hhmm '1632' für 16 Uhr 32).
day	Monat und Tag oder nur Tag der Woche ('feb 23' oder 'fr week').
file	Datei mit einer oder mehreren UNIX-Kommandozeilen.

Anmerkung:
Die Genauigkeit der zeitgesteuerten Abarbeitung hängt von der Quantisierung der UNIX-Abfragesteuerung in der Datei '/usr/lib/crontab' ab. Durch Editieren dieser Datei kann der Abfragezyklus verändert werden.

Kommando: awk zeichenkettenorientiertes Such- und Verarbeitungsprogramm (pattern scanning and processing language)

Aufruf: **awk** [-Fa] [-f prog] [file ...]
awk [-Fa] [statement] [file ...]

Beschreibung:
Das zeichenkettenorientierte Such- und Verarbeitungsprogramm 'awk' ist

primär für die Bearbeitung von Textdateien vorgesehen. Es übernimmt das zeilenorientierte Durchmustern bzw. die zeilenorientierte Verarbeitung einer oder mehrerer Dateien ('file ...') nach einer durch spezielle awk-Anweisungen ('prog' bzw. 'statement') festgelegten Weise. Der umfangreiche Anweisungssatz von 'awk' ist in einer C ähnlichen Sprachnotation abgefaßt. Das awk-Kommandoprogramm wird als Textfilter bezeichnet, kann aber auch als eigenständiges Programmiersystem Verwendung finden. Gibt man im awk-Aufruf keine zu bearbeitende Datei an, wird der Standardeingabezeichenstrom vom Kommandoprogramm 'awk' verarbeitet.

-Fa	Das dem Großbuchstaben 'F' folgende ASCII-Zeichen wird als Trennzeichen gewertet.
-f	Der folgende Dateiname ('prog') spezifiziert eine Datei mit awk-Anweisungen.
statement	awk-Anweisungsfolge.
file	Durch wk' zu bearbeitende Datei.

in [1] A.V. Aho, B.W. Kernighan, P.J. Weinberger: Awk - a pattern scanning and processing language
[11] S.R. Bourne: The UNIX System
[12] Z. Stanka, S. Lösch: UNIX - Führer durch das System

Kommando: bc Tischrechnerprogramm (C-interface desk calculator)

Aufruf: **bc** [-c] [-l] [file ...]

Beschreibung:
Das Kommandoprogramm 'bc' ist ein interaktiv arbeitendes Arithmetikrechenprogramm. Es verwandelt das UNIX-Terminal in einen Tischrechner. Die Syntax der Eingabesprache entspricht weitgehend einer C ähnlichen Sprachnotation. Verarbeitet wird vom bc-Tischrechnerprogramm grundsätzlich der Standardeingabezeichenstrom (Tastatur) - die berechneten Ergebnisse erscheinen auf der Standardausgabe (Bildschirm).

-c	Aufruf des bc-Tischrechnerprogramms mit der 'compile only' Option.
-l	Verwendung der UNIX-Standardbibliothek ('/usr/lib/bib.b' sinus, cosinus, arcustangens, exponential, logarithmus, bessel).
file	Bibliothek mit vordefinierten Makrofunktionen.

in [1] L.L. Cherry, R. Morris: BC - An arbitrary precision desk calculator language

Kommando: cal Kalenderausgabe (print calendar)

Aufruf: **cal** [Monat] **Jahr**

Beschreibung:
Das Kommandoprogramm 'cal' gibt einen Kalender des angegebenen Monats

eines Jahres ('January 1986') bzw. des angegebenen Jahres ('1952') auf dem Standardausgabegerät aus.

Kommando: calendar automatischer Terminkalender (automatic reminder)

Aufruf: **calendar**

Beschreibung:
Das Kommandoprogramm 'calendar' dient zur Führung eines tagesbezogenen Terminkalenders. Es arbeitet zusammen mit einer vom Anwender anzulegenden Datei 'calendar' mit den gespeicherten Terminaktivitäten im login-Directory des Nutzers.
Das calendar-Programm mustert alle Zeilen der Datei 'calendar' Nutzer-directory auf Datumsangaben der folgenden Form durch:

7/31	für 31. Juli
12/24/83	für 24.12.83
Jan 1	für 1. Januar.

Wird eine Übereinstimmung mit dem aktuellen Tagesdatum festgestellt, erfolgt die Ausgabe der entsprechenden Dateizeile auf dem Bildschirm. Das gleiche gilt für den auf das aktuelle Tagesdatum folgenden Tag (am Wochenende - Montag).

Kommando: cat Dateiausgabe (concatenate and print)

Aufruf: **cat** [-u] **file**

Beschreibung:
Ausgabe des Inhaltes einer oder mehrerer Dateien auf dem Standardausgabegerät (Bildschirm) oder mit Hilfe der Ein-/Ausgaberedirektion auf einen Drucker ('/dev/lp'). Die Angabe der Option bewirkt die ungepufferte Ausgabe.

Kommando: cb C-Programm Formatierungshilfe (C program beautifier)

Aufruf: **cb** [file.c]

Beschreibung:
Das Kommandoprogramm 'cb' dient der Formatierung von C-Programmen (Einrückungen, Leerzeichen ...), so daß eine der C-Syntax entsprechende Programmstruktur entsteht.
Das Kommandoprogramm 'cb' kann sowohl auf Dateien ('file') als auch auf den Standardeingabezeichenstrom angewendet werden.

Kommando: C-Programmübersetzer (C compiler)

Aufruf: [option] **file.c**

Beschreibung:
Das Kommandoprogramm 'cc' ist der auf den jeweiligen Rechner bezogene C-Standardcompiler des UNIX-Betriebssystems. In der Regel übersetzt der C-Compiler das Programm zunächst in die rechnerspezifische Assemblersprache (Mnemoniks), um dann durch einen impliziten Assembleraufruf ('as') diesen Zwischenkode in einen Objektkode umzuwandeln, der vom UNIX-Lader ('ld') durch Auflösung externer Referenzen und durch die Nutzung von Standardbibliotheksprogrammen in ein abarbeitungsfähiges Maschinenprogramm 'a.out' umgewandelt wird.
Es gelten folgende Standardnamen:

file.c	C-Programm
file.a	Assemblerprogramm
file.o	Objektprogramm
a.out	abarbeitungsfähiges Maschinenprogramm

Es existieren folgende Optionen ('option') des C-Compilers:

-E	Der Übersetzungsprozeß wird nach der Abarbeitung des Makro-Preprozessors abgebrochen. Das Ergebnis der Abarbeitung wird auf dem Standardausgabegerät ausgegeben.
-S	Der Übersetzungsprozeß wird vor dem Aufruf des Assemblers abgebrochen. Das Ergebnis der Programmübersetzung - ein Mnemonik-Assemblerprogramm - wird als Datei 'file.s' abgelegt.
-O	Das übersetzte C-Programm wird durch den Aufruf eines Optimierungsprogramms ('optimizer') hinsichtlich der Länge des erzeugten Programmkodes optimiert.
-c	Der letzte Schritt beim Übersetzungsprozeß, die Erzeugung einer abarbeitungsfähigen Datei 'a.out' durch den UNIX-Lader ('ld'), wird unterdrückt. Das übersetzte Programm wird im Objektkodeformat als 'file.o' abgelegt.
-g	Es erfogt die Generierung einer Symboltabelle.
-w	Warnungen beim Übersetzungslauf werden unterdrückt.
-o output	Das Ergebnis des Übersetzungsprozesses, das abarbeitungsfähige Maschinenprogramm, bekommt nicht den Standardnamen 'a.out', sondern den für 'output' eingetragenen Dateinamen.

Abhängig von der speziellen UNIX- und C-Rechnerimplementation existieren ggf. weitere Optionen des C-Compilers mit unterschiedlicher Bedeutung. Optionen für den impliziten Assembleraufruf können in der Befehlszeile des cc-Kommandos mit angegeben werden.

/5/,/6/ B.W. Kernighan, D.M. Ritchie: The 'C' Programming Language
in /1/ D.M. Ritchie: The C Programming Language - Reference Manual
in /1/ D.M. Ritchie: A Tour through the UNIX C Compiler
in /1/ S.C. Johnson: A Tour through the Portable C Compiler

Kommando: cd Wechsel des Arbeitsdirectories (change working directory)

Aufruf: **cd** [directory]

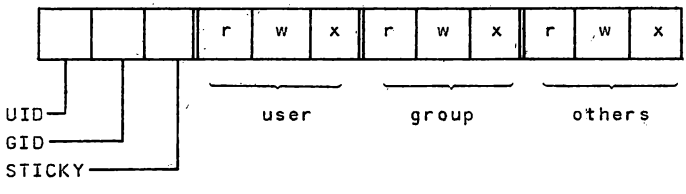
Beschreibung:
Mit dem cd-Kommando kann ein Wechsel vom momentanen Arbeitsdirectory in ein anderes Arbeitsdirectory erfolgen. Der Name des Directories, das zum neuen Arbeitsdirectory werden soll, kann als relativer oder absoluter Pfadname angegeben werden. Wird kein Directory angegeben, erfolgt der Wechsel in das login-Directory des Nutzers. Werden statt eines Directories zwei Punkte ('..') angegeben, erfolgt der Wechsel in das vorhergehende Arbeitsdirectory.

Anmerkung:
Das Kommando 'cd' gehört zum Basisbefehlsumfang des Shell- oder Cshell-kommandointerpreters. Es existiert also kein mit 'cd' bezeichnetes Kommandoprogramm im Dateisystem.

Kommando: chmod Setzen der Dateizugriffserlaubnisbits (change mode)

Aufruf: **chmod mode file**

Beschreibung:
Mit dem Kommandoprogramm 'chmod' erfolgt des Setzen der Zugriffserlaubnisbits einer oder mehrerer Dateien ('file ...'). Das Zugriffserlaubnisargument ('mode') kann entweder als Zahl oder als symbolischer Ausdruck angegeben werden.
Angabe des Zugriffserlaubnisargumentes 'mode' in Form einer Oktalzahl (eine ODER-Verknüpfung mehrerer dieser Zugriffserlaubnisargumentzahlen ist zulässig):



7000	6000	5000		4000	Benutzer-ID-Nummer wird bei Ausführung gesetzt.
7000	6000		3000	2000	Gruppen-ID-Nummer wird bei Ausführung gesetzt.
7000		5000	3000	1000	Stickybit wird gesetzt (Programmtextsegment bleibt nach Ausführung geladen).
0700	0600	0500		0400	Leseerlaubnis Besitzer (user).
0700	0600		0300	0200	Schreiberlaubnis Besitzer (user).
0700		0500	0300	0100	Ausführungserlaubnis Besitzer (user).

```
0070 0060 0050       0040   Leseerlaubnis Gruppe (group).
0070 0060      0030  0020   Schreiberlaubnis Gruppe (group).
0070      0050 0030  0010   Ausführungserlaubnis Gruppe (group).
0007 0006 0005       0004   Leseerlaubnis für andere (others).
0007 0006      0003  0002   Schreiberlaubnis für andere (others).
0007      0005 0003  0001   Ausführungserlaubnis für andere (others).

Angabe des Zugriffserlaubnisargumentes 'mode' in symbolischer Form
(Beispiel       oder    '):

+       Hinzufügen einer Erlaubnis.
-       Streichen einer Erlaubnis.
=       Zuordnung einer absoluten Erlaubnis.

u       Besitzer (login-Inhaber/user).
g       Gruppe (group).
o       alle anderen (others).
a       u, g und o zusammengenommen.

s       Benutzer-ID oder Gruppen-ID.
t       Stickybit.

r       Leseerlaubnis (read).
w       Schreiberlaubnis (write).
x       Ausführungserlaubnis (execute).
```

Kommando: chown / chgrp Wechsel der Besitzer- oder der Gruppenzugehörigkeit (change owner or group)

Aufruf:
```
chown owner file
chgrp group file
```

Beschreibung:
Mit den Kommandoprogrammen 'chown' und 'chgrp' kann die Besitzer- (owner) oder Gruppenzugehörigkeit (group) einer oder mehrerer Dateien ('file ...') verändert werden.

Kommando: cmp Vergleich zweier Dateien (compare two files)

Aufruf:
```
cmp [-l] [-s] file1 file2
```

Beschreibung:
Das Kommandoprogramm 'cmp' ermöglicht den byteweisen Vergleich zweier Dateien. Sind die beiden Dateien 'file1' und 'file2' identisch, erfolgt keine Ausschrift. Wird beim Vergleich ein Unterschied festgestellt, erfolgt eine Mitteilung (Bytenummer und Zeilennummer, bei der der Unterschied auftrat).

Gibt man keine der Optionen '-l' oder '-s' in der Befehlszeile an, wird danach der Vergleich abgebrochen.

-l Der Vergleich der beiden Dateien wird auch nach dem Auftreten eines Unterschiedes weitergeführt. Alle unterschiedlichen Bytes werden signalisiert (Bytenummer dezimal / Bytes oktal).

-s Bis auf einen Rückkehrkode erfolgen keinerlei Ausgaben (identisch 0: unterschiedlich 1: fehlerhafte Eingabeargumente 2).

Kommando: col Spaltendrucktextfilter (filter reverse line feeds)

Aufruf: col [-bfx]

Beschreibung:
Das Kommandoprogramm 'col' gehört zur Familie der Textverarbeitungsprogramme. Es ermöglicht die Ausgabe eines Textes, der, in mehrere Spalten pro Seite aufgeteilt, auf einem Drucker ausgegeben werden soll. Verarbeitet wird von 'col' der Standardeingabezeichenstrom - die in Spalten aufgeteilte Druckseite wird auf dem Standardausgabegerät ausgegeben.

in /1/ UNIX Programmer's Manual - col(1)

Kommando: comm Suchen gleicher Zeilen in zwei Dateien (select or reject lines common two sorted files)

Aufruf: **comm** [-123] **file1 file2**

Beschreibung:
Das Kommandoprogramm 'comm' vergleicht die Zeilen der beiden Dateien ('file1' / 'file2') und gibt eine in drei Spalten strukturierte Liste aus. Die erste Spalte (1) enthält alle Zeilen, die nur in der Datei 'file1' vorkommen, die zweite Spalte (2) enthält die der ersten Spalte entsprechenden Zeilen der Datei 'file2' und die dritte Spalte (3) enthält alle die Zeilen die sowohl in der Datei 'file1' als auch in der Datei 'file2' vorkommen. Die Spalten lassen sich voneinander jeweils durch eine Tabulatoreinrückung unterscheiden.
Durch die Angabe einer ein- oder zweistelligen Optionszahl (1, 2, 3, 12, 23, 13) können einzelne Spalten bei der Ausgabe ausgeblendet werden.
Gibt man anstelle eines Dateinamens einen Strich im Aufruf '-' ein, wird eine der beiden Dateien durch den Standardeingabezeichenstrom ersetzt.

Kommando: cp Kopieren von Dateien (copy)

Aufruf: **cp file1 file2**
cp file directory

Beschreibung:
Das Kommandoprogramm 'cp' erstellt eine Kopie einer Datei 'file1' und

speichert diese Kopie unter dem Dateinamen 'file2' ab.
Eine zweite Aufrufform des cp-Kommandos ermöglicht außerdem das Kopieren einer oder mehrerer Dateien 'file ...' unter dem gleichen Namen in ein angegebenes Directory ('directory').
Zu beachten ist in beiden Fällen, daß vor dem Kopieren keine Datei mit einem identischen Dateinamen existiert, da sonst deren Inhalt verlorengeht.

Kommando: crypt Verschlüsseln/Entschlüsseln (decode and encode)

Aufruf: **crypt** [password]

Beschreibung:
Das Kommandoprogramm 'crypt' dient der Verschlüsselung oder der Entschlüsselung von Datenbeständen. Verarbeitet wird zur Ver- oder Entschlüsselung der Standardeingabezeichenstrom. Das Ergebnis der crypt-Verarbeitung wird auf dem Standardausgabegerät ausgegeben. Das crypt-Kommando ist also ein sog. Filterprogramm. Die Erkennung, ob es sich um eine Verschlüsselung oder um eine Entschlüsselung handelt, erfolgt automatisch. Durch die Angabe eines Schlüsselwortes ('password') kann ein spezieller Kodierungsschlüssel (zum Beispiel zur Verschlüsselung einer Datei) vorgegeben werden er ist dann auch bei der Entschlüsselung anzugeben, sonst sind die Originaldaten nicht mehr zugänglich. Gibt man bei der Verschlüsselung kein spezielles Schlüsselwort vor, wird das jeweilige login-Schlüsselwort (s. a. Kommando: password) verwendet.

Anmerkung:
Ein Hinweis auf Sicherheitsfragen bei der Nutzung von UNIX-Systemdiensten findet sich u. a. in [1] [12] [13] und [14].

Kommando: csh C-Shellkommandointerpreter (cshell command processing)

Aufruf: **csh** [-option] [file]

Beschreibung:
Das Kommandoprogramm 'csh' ist ein Kommandointerpreter, der dem UNIX-Standardkommandointerpreter 'sh' (s. a. Abschnitt 6.) in vielfältiger Weise ähnlich ist. Substantielle Unterschiede bestehen bei den Funktionen 'history' (Abarbeitung schon einmal abgearbeiteter Kommandos) und 'alias' (Definition eigener Kommandos bestehend aus UNIX-Standardsystemkommandos). Der csh-Kommandointerpreteraufruf erfolgt entweder ohne weitere Spezifikation - in diesem Fall wird die Standardeingabe (Terminal) von der Tastatur behandelt oder er erfolgt in Verbindung mit einer durch ihn abzuarbeitenden Datei.

[12] Z. Stanka, S. Lösch: UNIX - Ein Führer durch das System

Kommando: cu UNIX-Rechnerkopplung (call UNIX)

Aufruf: **cu telno** [-t] [-s speed] [-a acu] [-l line]

Beschreibung:
Das Kommandoprogramm 'cu' ermöglicht die Zusammenarbeit von wei UNIX-Rechnern über den seriellen Datenkanal eines Terminals. Es kann für eine Verbindung zweier UNIX-Systeme zur interaktiven Arbeit oder zur Übertragung von ASCII-Dateien zwischen beiden Computern genutzt werden.

Anmerkung:
Eine recht ausführliche Beschreibung einer speziellen Implementationsvariante des cu-Kommandoprogramms zur UNIX-Rechnerkopplung befindet sich in [12] (siehe auch Kommandoprogramm 'uucp' und 'uux').

Kommando: date Ausgabe von Datum und Uhrzeit (print date)

Aufruf: **date**

Beschreibung:
Das Kommandoprogramm 'date' ermöglicht die Ausgabe des Datums, des Wochentages und der momentanen Uhrzeit, mit der das UNIX-System arbeitet.
Dem Systemprogrammierer (UNIX-Superuser) ist es außerdem möglich, mit Hilfe des date-Kommandos das Datum und die Uhrzeit, mit der das Betriebssystem arbeitet, neu festzulegen.

Kommando: dc Tischrechnerprogramm (desk calculator)

Aufruf: **dc** [file]

Beschreibung:
Das Kommandoprogramm 'dc' ist zur interaktiven Nutzung eines UNIX-Terminals als Tischrechner vorgesehen. Es arbeitet mit umgekehrter polnischer Notation und besitzt sehr leistungsfähige Charakteristiken.

in [1] R. Morris, L. Cherry: DC - An Interactive Desk Calculator

---

Kommando: dd Konvertieren/Umkopieren (convert and copy)

Aufruf: **dd** [option=value]

Beschreibung:
Das Kommandoprogramm 'dd' ermöglicht die Konvertierung und das Umkopieren des Standardeingabezeichenstroms entsprechend bestimmter Formate auf den Standardausgabezeichenstrom. Es ist besonders bei der Ein-/Ausgabekonvertierung unentbehrlich.

Für 'option=value ...' kann in der Kommandoaufrufzeile des dd-Kommandos stehen:

if='in.file'	Der Name der Eingabedatei ist 'in.file' (z.B. E/A-Gerät).
of='out.file'	Der Name der Ausgabedatei ist 'out.file' (z.B. E/A-Gerät).
ibs=n	Eingabeblockformat in Bytes (Standard 512 Byte).
obs=n	Ausgabeblockformat in Bytes (Standard 512 Byte).
bs=n	Setzen eines gemeinsamen Ein- und Ausgabeblockformates in Bytes.
cbs=n	dd-interne Konvertierungspuffergröße in Bytes.
skip=n	Übergehen von n Eingabesätzen vor dem Umkopieren/Konvertieren.
files=n	Übergehen von n Eingabedateien vor dem Umkopieren/Konvertieren.
seek=n	Übergehen von n Ausgabesätzen vor dem Umkopieren/Konvertieren.
count=n	Nur n Eingabesätze werden umkopiert.
conv=ascii	EBCDIC in ASCII Konvertieren.
conv=ebcdic	ASCII in EBCDIC Konvertieren.
conv=block	Umkopieren von Sätzen variabler zu Sätzen fester Länge.
conv=ublock	Umkopieren von Sätzen fester zu Sätzen variabler Länge.
conv=lcase	Alle Buchstaben werden zu Kleinbuchstaben.
conv=ucase	Alle Buchstaben werden zu Großbuchstaben.
conv=swab	Vertauschen der Bytereihenfolge in jeweils Bytepärchen.
conv=noerror	Übergehen aller Fehlermeldungen.
conv=pad	Vergrößern aller Eingabesätze auf den Wert von ibs.

Kommando: deroff Entfernen aller Formatierungsbefehle (remove nroff, troff, tbl and eqn constructs)

Aufruf: **deroff** [-w] [file ...]

Beschreibung:
Das Kommandoprogramm 'deroff' gehört zu der Familie der troff/nroff-Textverarbeitungsprogramme. Es entfernt aus einer oder aus mehreren Dateien ('file ...') alle Formatierungsbefehle (Kommandozeilen, Backslash-Konstruktionen, Makrodefinitionen usw.) der Textverarbeitungsprogramme 'nroff', 'troff', 'tbl' und 'eqn'.
Gibt man keinen Dateinamen im deroff-Aufruf an, wird der Standardeingabezeichenstrom verarbeitet. Der von 'deroff' erzeugte Zeichenstrom wird in jedem Fall auf dem Standardausgabegerät ausgegeben.
Die Option '-w' bewirkt, daß der Ausgabezeichenstrom einer Wortliste (mit jeweils einem Wort pro Zeile) gleicht, die nur Ketten von Buchstaben, Zahlen und Apostrophzeichen in direkter Verbindung mit Buchstaben beinhaltet. Alle anderen Zeichen werden übergangen.

Kommando: diff Ermittlung der Unterschiede zwischen zw Dateien (differential file comparator)

Aufruf: **diff** [-efbh] **file1 file2**

Beschreibung:
Das Kommandoprogramm 'diff' dient dem Vergleich zweier Dateien 'file1' und 'file2' zur Ermittlung der Dateiunterschiede. Unterschiedliche Zeilen beider Dateien werden (zusammen mit den Editorbefehlen, die notwendig sind um die Dateien in Übereinstimmung zu bringen) auf dem Standardausgabegerät ausgegeben. Gibt man anstelle eines der beiden Dateinamen ('file1' oder 'file2') einen Strich ein, wird eine der beiden Dateien durch den Standardeingabezeichenstrom ersetzt.
Folgende Optionen werden vom diff-Kommando verarbeitet:

-e Es wird beim Vergleich ein Befehlsskript des ed-Editors erstellt, der, auf die erste Datei angewendet, die erste Datei der zweiten Datei identisch macht.
-f Es wird ein der Option '-e' ähnlicher Shellbefehlsskript erstellt - nur in umgekehrter Richtung.
-b Leerstellen und Tabulatoren werden beim Vergleich ausgeklammert.
-h Es ist ein Vergleich von Dateien unbegrenzter Länge möglich (ohne Option '-e' und '-f').

Kommando: du Ermittlung der Plattenspeicherbelegung (disk usage)

Aufruf: **du** [-s] [-a] [name ...]

Beschreibung:
Das Kommandoprogramm 'du' dient der Ermittlung der für eine oder mehrere Dateien ('name ...') bzw. für eine oder mehrere Directories ('name ...') benötigten Plattenspeicherbelegung. Die ausführliche Übersicht, wieviel 512-Byte-Blöcke für die angegebenen Dateien und Directories benötigt werden, wird auf dem Standardausgabegerät ausgegeben. Bei der Angabe von Directories werden bei dieser Übersicht auch rekursiv alle in ihr enthaltenen Dateien und Subdirectories berücksichtigt.
Wird kein Datei- oder Directoryname angegeben, wird die Plattenspeicherbelegung des momentanen Arbeitsdirectories ausgegeben.
Folgende Optionen werden vom du-Kommando verarbeitet:

-s Nur die Gesamtsumme aller benötigten Plattenspeicherblöcke für Dateien und Directories wird ausgegeben.
-a Für jede Datei werden die benötigten Plattenspeicherblöcke ausgegeben (standardmäßig erfolgt die Ausgabe bezogen auf ein Directory).

Kommando: echo Echoausgabe (echo arguments)

Aufruf: **echo** [-n] [argument ...]

Beschreibung:
Das Kommandoprogramm 'echo' wiederholt die in der Programmaufrufzeile notierte Argumentzeichenkette ('argument') auf dem Standardausgabegerät. Mehrere Argumente werden durch Leerzeichen voneinander getrennt und durch ein abschließendes Newline-Zeichen beendet.
Wird die Option '-n' im echo-Aufruf angegeben, entfällt das abschließende Newline-Zeichen.

Kommando: ed Texteditor (text editor)

Aufruf: **ed** [-x] [file]

Beschreibung:
Das Kommandoprogramm 'ed' ist ein interaktiv arbeitendes Hilfsmittel zur Erstellung oder Veränderung von Text- und Quellprogrammdateien. Der ed-Editor arbeitet zeilenorientiert, d.h., für jede Operation muß eine Zeile (oder mehrere) des zu bearbeitenden Textes spezifiziert werden.
Gibt man beim ed-Editoraufruf einen Dateinamen 'file' an, wird diese Datei zunächst in den internen Editorpufferspeicher eingelesen. Anderenfalls ist dieser interne Puffer zunächst leer.
Durch die Angabe der Option '-x' kann, ähnlich wie beim Kommandoprogramm 'crypt', die Bearbeitung von verschlüsselten Texten erfolgen.
Es gilt folgende, hier vereinfacht wiedergegebene Kommandoreferenzliste des ed-Texteditors (Achtung: Die Klammerzeichen '[' und ']' sind nicht mit einzugeben, sie dienen hier nur der Kennzeichnung der Feldbegrenzung) (s.a. /13/,/14/):

[line]**a**[...text...]	(a append) Einfügen des Textes ...text...' nach der angegebenen Zeile 'line'. Übergang in den Textmode. (Der Text beginnt immer mit einem Newline-Zeichen.)
[line]**i**[...text...]	(i insert) Einfügen des Textes ...text...' vor der angegebenen Zeile 'line'. Übergang in den Textmode. (Der Text beginnt immer mit einem Newline-Zeichen.)
[b,e]**c**[...text...]	(c change) Austauschen des Textes der angegebenen Zeilen 'b' bis 'e' durch einen Text ...text...'. Übergang in den Textmode. (Der Text beginnt immer mit einem Newline-Zeichen.)
	Beenden Textmode. (Das Punktzeichen muß zum Beenden des Textmodes als erstes Zeichen am Anfang einer Zeile stehen.)
[b,e]**.d**	(d delete) Löschen der Zeilen 'b' bis 'e'.
[b,e]**s/str1/str2/**	(s substitute) Substituieren von 'str1' durch 'str2' in allen angegebenen Zeilen. Wird ein Buchstabe 'g' am Ende der substitut-Befehlszeile ein-

	gegeben, erfolgt der Austausch beliebig viele Male pro Zeile. Anderenfalls erfolgt der Austausch nur beim ersten Auftreten von 'str1'.
u	(u undo) Rückgängigmachen des zuletzt eingegebenen Befehls.
[b,e] m [d]	(m move) Verschieben der angegebenen Zeilen 'b' bis ' hinter die Zeile 'd'.
[b,e] j	(j join) Verbinden der angegebenen Zeilen 'b' bis 'e' zu einer Zeile.
[b,e] l	(l list) Ausgeben der angegebenen Zeilen 'b' bis 'e' (einschließlich Sonderzeichen).
[b,e] p	(p print) Ausgeben der angegebenen Zeilen 'b' bis 'e'.
e[file]	(e edit) Editieren einer neuen Datei 'file'. Löschen des alten Editorpufferinhaltes.
f	(f filename) Ausgabe des Dateinamens der Datei, die momentan bearbeitet wird.
[line] r [file]	(r read) Einlesen einer spezifizierten Datei 'file' aus dem UNIX-Dateisystem ab der angegebenen Zeile 'line'.
[b,e] w[file]	(w write) Schreiben des Editorpufferinhaltes von der Zeile 'b' bis ' als Datei 'file' das UNIX-Dateisystem.
P[character]	(p prompt) Setzen des ed-Promptzeichens auf 'character' (Standard: '*').
[line] Ka	(k mark) Kennzeichnung der Zeile 'line' mit dem Adressierungssymbol 'a'.
[$] =	Ausgeben der aktuellen bzw. der letzten Zeilennummer.
q	(q quit) Editor-Quitzeicheneingabe. Beenden der ed-Editorarbeit. (Achtung: Es erfolgt kein automatisches Abspeichern des ed-Editorpufferinhaltes.)

Der ed-Editor arbeitet mit einem Zeilenpointer, der auf die bearbeitete Zeile Bezug nimmt. Er wird bei der Kommandoeingabe automatisch aktualisiert. Soll bei einer ed-Kommandoeingabe auf diese aktuelle Zeile zugegriffen werden, kann die Angabe von 'line', 'b' oder/und 'e' entfallen. Außerdem gelten folgende Notationsvereinbarungen bei der Zeilenadressierung ('line' 'b', ' '):

Punktzeichen		momentan bearbeitete Textzeile.
Dollarzeichen	$	letzte Zeile im Editorpuffer.
	9	Zeile 9.
	21,42	Zeile 21 bis 41.
	a	Zeile, die mit dem Kommando 'Ka' markiert wurde.

in [1] B.W. Kernighan: Advanced editing on UNIX
in [1] B.W. Kernighan: A Tutorial Introduction to the ED Text Editor
[13] H. McGilton, R. Morgan: Einführung in das UNIX System
[14] M. Banahan, A. Rutter: UNIX - Lernen, verstehen, anwenden

Kommando: enroll, xsend, xget Senden/Empfangen geheimer Post (secret mail)

Aufruf:
```
enroll
xsend user
xget
```

Beschreibung:
Mit den drei Kommandos 'enroll', 'xsend' und 'xget' ist das Senden und Empfangen von geheimer elektronischer Post über die UNIX-Nutzerkommunikation möglich.
Mit dem enroll-Kommando wird ein Kodewort festgelegt, unter dem die geheime Post empfangen und gelesen werden kann.
Mit dem xsend-Kommando wird (analog dem Kommando 'mail' für nicht verschlüsselte Post) eine geheime Nachricht an einen bestimmten Empfänger ('user') abgesandt.
Mit 'xget' kann man nach Abfrage des Kodewortes eine geheime Nachricht abfordern.

Kommando: eqn Ausgabe mathematischer Formeln (typset mathematics)

Aufruf: eqn [file ...]

Beschreibung:
Das Kommandoprogramm 'eqn' gehört zur troff/nroff-Textverarbeitungsprogrammfamilie. Es dient primär der Preprozessorverarbeitung mathematischer Ausdrücke in einem grafischen Fotosatzsystem. Es kann jedoch auch bei der Ausgabe mathematischer Ausdrücke bei bestimmten Druckern angewendet werden.

in [1] J.F. Ossanna: NROFF/TROFF Users Manual
in [1] B.W. Kernighan, L.L. Cherry: Typesetting Mathematics - Users Guide

---

Kommando: expr Berechnung arithmetischer Ausdrücke (evaluate expression)

Aufruf: expr argument

Beschreibung:
Das Kommandoprogramm 'expr' behandelt die vorgegebenen Argumente 'argument ...' als mathematische Ausdrücke. Es berechnet diese Ausdrücke und gibt das Ergebnis der Berechnung auf der Standardausgabe aus. Eine der wichtigsten Eigenschaften des expr-Kommandoprogramms ist die Möglichkeit des Rechnens mit Shellvariablen.

Folgende arithmetischen Operationen können verwendet werden:

+	für die Addition.
-	für die Subtraktion.
*	für die Multiplikation.
/	für die Division.
%	für die Bestimmung des Restes einer Operation.

Außerdem existieren Operationen zur Bestimmung logischer Ausdrücke und für Zeichenkettenvergleiche (siehe auch in [1] - expr(1)).

Kommando: F77 FORTRAN77-Compiler (FORTRAN77 compiler)

Aufruf: **f77** [option] **file.f**

Beschreibung:
Bei dem Kommandoprogramm 'f77' handelt es sich um einen FORTRAN77-Compiler. Er ermöglicht die Übersetzung einer (mehrerer) durch ein nachgestelltes .f' gekennzeichneten Datei(en) ('file.f ...') mit FORTRAN77-Quellprogrammanweisungen in ein ladefähiges Objektprogramm 'file.o' oder in ein ausführbares Maschinenprogramm 'a.out'.
Für den F77-Compiler existieren im UNIX-Betriebssystem zw optimierende Precompiler 'EFL' und 'ratfor'.
Es gelten die folgenden Optionen des F77-Compilers:

-c	Erzeugen eines Objektprogrammes 'file. Unterdrücken des Laderaufrufes.
-g	Erzeugen einer Symboltabelle.
-w	Unterdrücken aller Warnungen des F77-Compilers.
-p	Vorbereiten des Objektprogrammes für eine Laufzeitprofilbestimmung.
-O	Anfordern eines Objektkodeoptimierungslaufes.
-S	Übersetzen von 'file.f' in ein Assemblerprogramm 'file.s'
-o output	Für das ausführbare Maschinenprogramm wird der anstelle von 'output' eingetragene Name verwendet.
-onetrip	DO-Schleifen werden so übersetzt, daß sie mindestens einmal ausgeführt werden.
-u	Abweichend von den üblichen Konventionen wird als Standardtyp von Variablen 'undefined' verwendet.
-C	Anforderung eines speziellen Übersetzungskodeüberprüfers.
-F	Anforderung des 'EFL und 'ratfor' Precompilers für alle mit und '.r' gekennzeichneten Dateien. Das Ergebnis des Precompilerlaufes sind mit .f' gekennzeichnete Dateien.
-Ex	' ist als EFL-Option zu werten.
-Rx	ist als ratfor-Option zu werten.

Optionen des UNIX-Assemblers können beim F77-Compiler in der gleichen Weise wie beim C-Compiler 'cc' und beim Assembler ' angegeben werden.

[1] S.I. Feldmann, P.J. Weinberger: A Portable FORTRAN 77 Compiler

Kommando: file Ermittlung des Dateityps (determine file type)

Aufruf: **file filename**

Beschreibung:
Das Kommandoprogramm 'file' untersucht eine oder mehrere Dateien ('filename ...') auf ihren Inhalt. Dabei wird vom file-Kommandoprogramm versucht, herauszufinden, in welcher Programmiersprache eine Datei erstellt wurde.

Kommando: find Suchen von Dateien (find files)

Aufruf: **find pathnamelist expression**

Beschreibung:
Das Kommandoprogramm 'find' mustert das (oder die) durch 'pathnamelist...' benannte(n) Verzeichnis(se) nach Dateien durch. Die Suchbedingung wird durch ein einzelnes oder durch mehrere Kriterien ('expression ...') vorgegeben.
Für den Ausdruck 'expression kann im Kommandoaufruf stehen:

-name	filename	Es wird die Datei mit dem Namen 'filename' gesucht.
-perm		Es wird nach Dateien mit der Dateizugriffserlaubnisbitbelegung 'onum' (Oktalzahl) gesucht.
-type		Es wird nach Dateien eines bestimmten Typs 'c' gesucht. Für 'c' kann stehen: b Blockgerätedatei, c Zeichengerätedatei, d Directory, f reguläre Datei.
-links		Es werden Dateien mit 'n' Linkverweisen gesucht.
-user		Es werden Dateien eines bestimmten Nutzers ('uname') gesucht.
-group	gname	Es werden Dateien einer bestimmten Nutzergruppe ('gname') gesucht.
-size		Es werden Dateien mit einer bestimmten Anzahl 'n' von 512-Byte-Blöcken gesucht.
-inum		Es werden Dateien mit einer bestimmten Anzahl 'n' von i-nodes gesucht.
-atime		Es werden Dateien gesucht, auf die in den letzten 'n' Tagen zugegriffen wurde.
-mtime		Es werden Dateien gesucht, die in den letzten 'n' Tagen modifiziert wurden.
-newer	file	Es werden Dateien gesucht, die ein jüngeres Erstellungsdatum besitzen als die angegebene Datei 'file'.
-exex	command	Das angegebene Kommando 'commmand' wird auf die Dateien, auf die die Suchbedingung zutrifft, angewandt.

-ok command	Das angegebene Kommando 'commmand' wird auf die Dateien, auf die die Suchbedingung zutrifft, mit nochmaliger Nutzerabfrage ('yes' oder 'no') vor der Kommandoausführung angewandt.
-print	Ausgabe des Pfadnamens der gefundenen Datei(en).

Bei der Angabe der Zahl 'n' in den Suchbedingungen bezieht sich eine ganze Zahl immer auf genau nur diese eine Dezimalzahl, bei der Angabe der Zahl 'n' in der Form '+n' auf alle Zahlen, die größer sind als 'n' und bei der Angabe der Zahl in der Form '-n' auf alle Zahlen, die kleiner sind als 'n'.
Für den Ausdruck 'expression' können auch in beliebiger Reihenfolge mehrere Suchbedingungen angegeben werden. Steht zwischen den mit einem Bindestrich beginnenden Suchbedingungen ein Leerzeichen, gilt implizit eine UND-Verknüpfung. Eine ODER-Verknüpfung kann durch die Angabe von '-o' zwischen zwei Bedingungen erreicht werden.

Kommando: grep Durchmustern einer Datei (search file for pattern)

Aufruf:
grep [option] expression [file ...]
egrep [option] [expression] [file ...]
fgrep [option] [string] [file ...]

Beschreibung:
Das Kommandoprogramm 'grep' ermöglicht, eine oder mehrere Dateien 'file...' nach Zeilen mit einem bestimmten Textmuster 'expression' bzw. 'string' durchzumustern. Die Zeilen der Dateien, in denen dieses Textmuster gefunden wird, werden auf dem Standardausgabegerät ausgegeben. Gibt man keine Dateinamen im Kommandoaufruf an, wird der Standardeingabezeichenstrom von 'grep' bearbeitet.
Folgende Optionen können im grep-Kommando verwendet werden:

-v	Es werden alle Zeilen ausgegeben, in denen die Suche erfolglos bleibt.
-c	Es wird nur die Anzahl der Zeilen ausgegeben, in der das Textmuster gefunden wurde.
-l	Es werden nur die Namen der Dateien ausgegeben, denen das Textmuster gefunden wurde.
-n	Es werden die Zeilen, in denen das Textmuster gefunden wurde zusammen mit ihrer zugehörigen Zeilennummer ausgegeben.
-s	Bis auf eventuelle Fehlermeldungen (error status) werden alle Ausgaben unterdrückt.
-h	Die Ausgabe des Dateikopfes wird unterdrückt.
-y	Groß- und Kleinbuchstaben werden beim Suchen nicht unterschieden.
-e exp	Angabe des zu suchenden Textmusters 'exp', wenn das Textmuster mit einem Bindestrich beginnt.

Neben dem Durchmustern nach vorgegebenen Textmustern können die Programme 'grep' und 'egrep' auch das Durchsuchen der Dateien nach Textmustern übernehmen, die einfachen ('grep') oder auch sehr komplexen ('egrep') Bedingungen - sogenannten 'regular expressions' - genügen (z.B.: Suchen

aller Zeilen mit Worten, die aus fünf Buchstaben bestehen und mit der Zeichenkette 'ad' beginnen usw.).
Das Kommandoprogramm 'fgrep' ist eine sehr schnell arbeitende Variante von 'grep' - ohne die Möglichkeit der Bearbeitung von 'regular expressions'.

Anmerkung:
Die Möglichkeiten der Bearbeitung von 'regular expressions' mit dem Kommandoprogramm 'grep' und 'egrep' werden recht ausführlich beschrieben in:

[13] H. McGilton, R. Morgan: Einführung in das UNIX System

Kommando: join identifikationszeichenbedingte Ausgabe (relational database operator)

Aufruf: **join** [option] **file1 file2**

Beschreibung:
Mit dem Kommandoprogramm 'join' können diejenigen Zeilen zweier Dateien zur Ausgabe gebracht werden, die in der gleichen Zeile ein gleiches erstes Zeichen - das sogenannte Identifikatorzeichen - besitzen. Die Identifikationszeichen müssen außerdem in aufsteigender Reihenfolge (entsprechend dem ASCII-Kode) geordnet sein. Zur Spezifikation der identifikationszeichenbedingten Ausgabe beim join-Kommando existiert eine Reihe von Optionen.

in [1] UNIX Programmers Guide - join(1)

Kommando: kill Prozeßbearbeitungsabbruch (terminate process)

Aufruf: **kill** [-signalno] **processid**

Beschreibung:
Mit dem Kommandoprogramm 'kill' ist der Abbruch der Bearbeitung eines Prozesses möglich. Als Parameter muß die Prozeßidentifikationsnummer ('processid') des jeweiligen Prozesses angegeben werden. Eine optionale Signalnummer ('signalno') gestattet den Abbruch eines Prozesses mit einem bestimmten Endesignal (Standard: -signalno=-15).

Kommando: ld Lader (loader)

Aufruf: **ld** [option] **file**

Beschreibung:
Das Kommandoprogramm 'ld' ist der Lader des UNIX-Betriebssystems. Der Lader erstellt aus einem oder mehreren Objektprogrammen ('file ...') ein abarbeitungsbereites Maschinenprogramm mit dem Standardnamen 'a.out'. Beim Ladeprozeß werden außerdem eventuell vorhandene Bibliotheksaufrufe einge-

bunden und externe Referenzen aufgelöst. Der Aufruf des Laders kann sowohl explizit als auch implizit in Verbindung mit einem Assembler- und/oder Compileraufruf erfolgen.
Die vom UNIX-Lader bearbeiteten Optionen sind stark von der jeweiligen Rechnerimplementation abhängig. Ihre Aufstellung und genaue Bedeutung ist den jeweiligen Rechnerunterlagen zu entnehmen.

in [1] UNIX Programmers Guide - ld(1)

Kommando: learn UNIX-Lernprogramm (computer aided instruction)

Aufruf: **learn** [-directory] [subject [lesson [speed]]]

Beschreibung:
Das Kommandoprogramm 'learn' ermöglicht die Anforderung einer rechnergestützten Unterweisung für mehrere durch 'subjekt' auswählbare Themenkomplexe: C, editor, macros, files und morefiles.
Jeder einzelne dieser Themenkomplexe ist in mehrere selektiv anwählbare Lektionen ('lesson') unterteilt, die bei Bedarf auch unter Angabe unterschiedlicher Geschwindigkeiten 'speed' zur Abarbeitung angefordert werden können.
Gibt man einen Directorynamen ('directory') an, werden Zwischendateien, die bei der Abarbeitung von 'learn' entstehen, in diesem Directory abgelegt.

Kommando: lex Programmgenerator für die lexikalische Analyse (lexical analysis)

Aufruf: **lex** [-tvf] [file ...]

Beschreibung:
Das Kommandoprogramm 'lex' gehört zur Familie der Compilerbauwerkzeuge (s.a. yacc-Kommandoprogramm). Es ermöglicht die Generierung von Programmen zur lexikalischen Bearbeitung beliebiger Zeichenkettendateien. Die Vorgabe der Bearbeitungsbedingungen erfolgt über die Datei(en) 'file ...'. Gibt man keinen Dateinamen im Aufruf an, wird von 'lex' der Standardeingabezeichenstrom verarbeitet.
Das Ergebnis der lex-Verarbeitung ist ein in der Programmiersprache C geschriebenes lexikalisches Analyseprogramm 'lex.yy.c', das, mit dem cc-Compiler übersetzt, eine Analyse von Zeichenkettendateien entsprechend den in 'file' spezifizierten Bedingungen ermöglicht.
Das Kommandoprogramm 'lex' verarbeitet die folgenden Optionen:

-t Die Ausgabe von 'lex' erfolgt auf dem Standardausgabegerät.
-v Es wird eine statistische Analyse der lex-Verarbeitung angefertigt.
-f Es erfolgt bei kleineren Dateien eine beschleunigte Abarbeitung.

in [1] M.E. Lesk, E. Schmidt: LEX - Lexial Analyzer Generator

Kommando: lint C-Syntaxprüfer (C program verifier)

Aufruf: **lint** [option] **file**

Beschreibung:
Das Kommandoprogramm 'lint' ist ein Überprüfungsprogramm für C-Programme. Es ermittelt Syntaxfehler, C-Programmportabilitätsprobleme, Parameterübermittlungsfehler, nicht verwendete Variablen und anderes mehr. Die Meldungen von 'lint' werden auf dem Standardausgabegerät ausgegeben. Gibt man keine Optionen an, werden alle Standardüberprüfungen von 'lint' ausgeführt.
Das Kommandoprogramm 'lint' verarbeitet die folgenden Optionen ('option'):

-c Es erfolgt ausschließlich die Portabilitätsüberprüfung sogenannten 'casts'.
-h Es erfolgt ausschließlich die Untersuchung der Programmstruktur der C-Programme.
-b Es erfolgt ausschließlich die Untersuchung der C-Programme auf ungenutzte break-Anweisungen.
-x Es erfolgt ausschließlich die Untersuchung der C-Programme auf ungenutzte externe Referenzen.
-v Es erfolgt die Unterdrückung von Meldungen über ungenutzte Argumente in Funktionen.
-u Es erfolgt die Unterdrückung von Meldungen über ungenutzte Funktionen und Variablen.
-n Es erfolgt die Unterdrückung von Meldungen zur Unverträglichkeit der untersuchten C-Programme mit der C-Standardbibliothek.
-a Es erfolgt ausschließlich die Meldung des Zugriffs von long-Variablen in den untersuchten C-Programmen auf int-Variable.

Das lint-Kommandoprogramm reagiert außerdem auf folgende Kommentare in den zu überprüfenden C-Programmen:

/*NOTREACHED*/ Meldungen über ungenutzte Programmteile werden unterdrückt.
/*VARARGSn*/ Die Datentypen der ersten 'n' Argumente werden überprüft - alle weiteren Prüfungen werden unterdrückt (Standard n=0).
/*NOSTRICT*/ Die Überprüfung des nächsten Ausdruckes wird unterdrückt.
/*ARGSUSED*/ Entspricht der Option '-v' für die nächste Funktion.
/*LINTLIBRARY*/ Meldungen über ungenutzte Funktionen werden unterdrückt.

in [1] S.C. Johnson: Lint, A C Program Checker

Kommando: ln Herstellen einer Dateilinkverbindung (make a link)

Aufruf: **ln file1** [file2]

Beschreibung:
Mit dem Kommandoprogramm 'ln' ist die Herstellung einer Dateilinkverbindung möglich, die die Zuordung eines zweiten (oder weiteren) Dateinamens 'file2' zu einer physisch nur einmal vorhandenen Datei 'file1' im UNIX-Dateisystem erlaubt.
Das ln-Kommando dient insbesondere der Aufnahme einer bereits vorhandenen Datei in ein zweites Directory. Wird der zweite Dateiname weggelassen, erfolgt die Aufnahme der Datei 'file1' unter dem gleichen Dateinamen in das momentane Arbeitsdirectory.
Das ln-Kommando kann nur innerhalb eines Dateisystems angewendet werden.

Kommando: login UNIX-Nutzerzuschaltung (UNIX system logging-in)

Aufruf: **login** [username]

Beschreibung:
Das Kommandoprogramm 'login' dient der Zuschaltung eines Nutzers 'username' zur Arbeit unter dem UNIX-Betriebssystem entsprechend der ihm zugeteilten Zugriffsrechte. Wird der Nutzername nicht im Aufruf angegeben, wird er anschließend bei der login-Kommandoabarbeitung abgefragt.

Kommando: look Suchen von Dateizeilen (find lines)

Aufruf: **look** [-df] **string** [file]

Beschreibung:
Das Kommandoprogramm 'look' mustert eine Datei 'file' nach Zeilen durch, die mit der spezifizierten Zeichenkette 'string' beginnen, und gibt die entsprechenden Zeilen über das Standardausgabegerät aus. Gibt man keinen Dateinamen an, wird die Datei '/usr/dict/words' von 'look' durchgemustert.
Das Kommandoprogramm 'look' kennt folgende Optionen:

-d Nur Buchstaben, Zahlen, Tabulatorzeichen und Leerzeichen werden beim Vergleich berücksichtigt.
-f Groß- und Kleinbuchstaben werden beim Durchmustern nicht unterschieden.

Kommando: lorder Ermittlung von Objektprogrammbezügen (find ordering relation for an object library)

Aufruf: **lorder file**

Beschreibung:
Das Kommandoprogramm 'lorder' verarbeitet eine oder mehrere Objekt- und/oder Bibliotheksdateien. Es ermittelt, ob zusammengehörige Dateien derart existieren, daß in der einen Datei externe Symbole verwendet werden, die in einer anderen Datei definiert worden sind. Die Ausgabe des lorder-Kommandoprogramms besteht aus den ermittelten Dateipaaren, zwischen denen solche Zusammenhänge existieren. Gibt man nur einen Dateinamen an, wird der Standardeingabezeichenstrom als zweite Objektprogrammdatei gewertet.

Kommando: lpr Druckerausgabe (line printer spooler)

Aufruf: **lpr** [-rcm] [file ...]

Beschreibung:
Das Kommandoprogramm 'lpr', der sog. Druckerspooler, ordnet die angegebenen Dateien ('file ...') in die Warteschlange des UNIX-Zeilendruckerprogramms ein. Gibt man keinen Dateinamen an, wird der Standardeingabezeichenstrom vom lpr-Kommandoprogramm verarbeitet.
Der Druckerspooler 'lpr' kennt die folgenden Optionen:

-r Löschen der Datei nach dem Ausdruck.
-c Von der Datei wird beim Einordnen in die Druckerwarteschlange zunächst eine temporäre Sicherheitskopie angefertigt.
-m Anforderung einer Rückmeldenachricht (über 'mail') nach erfolgtem Druck.

Kommando: ls Ausgabe des Inhalts eines Directories (list contents of directory)

Aufruf: **ls** [-ltasdruci] [name ...]

Beschreibung:
Das Inhaltsverzeichnis jedes der benannten Directories ('name ...') wird, entsprechend den angegebenen Optionen, ausgegeben.
Werden Dateinamen ('name ...') anstelle von Directorynamen im ls-Aufruf angegeben, erfolgt die Ausgabe des Dateinamens in Verbindung mit den durch die angegebenen Optionen spezifizierten Zusatzinformationen.
Wird kein Name angegeben, erfolgt die Ausgabe des Inhaltes des momentanen Arbeitsdirectories.
Die Optionen können beim ls-Kommandoprogramm in beliebiger Weise kombiniert werden. Die Ausgabe erfolgt standardmäßig, bezogen auf die Dateinamen, in alphabetisch sortierter Folge.

Die Optionen des ls-Kommandos haben folgende Bedeutung:

-l Die Ausgabe erfolgt im langen Format mit allen Zusatzinformationen.
-t Die Ausgabe erfolgt sortiert nach dem letzten Modifikationszeitpunkt.
-a Es erfolgt die Ausgabe aller Eintragungen in einem Directory - auch von Dateien, deren Namen mit einem Punkt '.' beginnen, die sonst nicht ausgegeben werden.
-s Es erfolgt zusätzlich die Ausgabe der Dateigröße in 512-Byte-Blöcken.
-d Von den im ausgegebenen Directory enthaltenen Subdirectories wird nur der Name, nicht der Inhalt, angegeben.
-r Die Ausgabe erfolgt gegenüber der Standardausgabe umgekehrter Reihenfolge.
-u Die Ausgabe erfolgt sortiert nach dem letzten Zugriffszeitpunkt.
-c Die Ausgabe erfolgt sortiert nach der letzten i-node-Modifikation.
-i Es erfolgt zusätzlich die Ausgabe der i-number.

Kommando: m4 Makroprozessor (macro processor)

Aufruf: **m4** [file ...]

Beschreibung:
Das Kommandoprogramm 'm4' ist ein Makroprozessor, der die Verarbeitung von selbst definierten Makros ('file ...') erlaubt. Er ist bei beliebigen Programmiersprachen, insbesondere aber in Verbindung mit der C-Sprache, anwendbar.
Wird der m4-Makroprozessor ohne Angabe eines Dateinamens aufgerufen, wird der Standardeingabezeichenstrom verarbeitet. Das Ergebnis der m4-Verarbeitung wird auf dem Standardausgabegerät ausgegeben.

in [1] B.W. Kernighan, D.M. Ritchie: The M4 Macro Processor

Kommando: mail Senden und Empfangen von Post (send receive mail)

Aufruf: Senden **mail user ...**
Empfangen **mail** [-rpq] [-f file]

Beschreibung:
Das Kommandoprogramm 'mail' dient dem Senden und Empfangen von elektronischer Post zwischen zwei UNIX-Nutzerterminals (bzw. an sich selbst). Die zu übermittelnden Nachichten werden nach dem Absenden und vor dem Empfangen in speziellen Systemdateien hinterlegt, so daß sie auch beim Ausschalten des Rechners erhalten bleiben. Beim Zuschalten eines Nutzers (login) wird das Vorliegen von mail-Nachrichten dem jeweiligen Anwender über eine Ausschrift angezeigt.

Senden: Wird das Kommandoprogramm 'mail' unter Angabe eines oder mehrerer Nutzernamen ('user ...') aufgerufen, wird die über den Standardeingabezeichenstrom eingehende, auf den Aufruf folgende Zeichenkette als zu übermittelnde Nachricht gewertet. Das Ende

der Nachricht wird mit einem CTRL-D-Zeichen oder einem einzeln stehenden Punktzeichen ('.') in einer Zeile angezeigt.

Empfangen: Wird das mail-Kommandoprogramm ohne Angabe eines Nutzernamens aufgerufen, werden nacheinander alle eingegangenen Nachrichten auf dem Standardausgabegerät ausgegeben. Nach der Ausgabe jeweils einer Nachricht existieren folgende Auswahlmöglichkeiten:

Newline-Zeichen	Ausgabe der nächsten Nachricht.
d	Löschen der Nachricht - Ausgabe der nächsten Nachricht.
p	Nochmalige Ausgabe der eben ausgegebeben Nachricht.
	Ausgabe der vorhergehenden Nachricht.
s [file ...]	Abspeichern der Nachricht in der (oder den) benannten Datei(en).
[file ...]	Abspeichern der Nachricht ohne Header in der (oder den) benannten Datei(en).
m [user ...]	Absenden der Nachricht an den (oder die) benannten Nutzer 'user
CTRL-D (oder EOT)	Abbrechen der Nachrichtenausgabe.
q	entspricht CTRL-D.

Beim Aufruf des mail-Kommandoprogramms zur Nachrichtenausgabe können außerdem folgende Optionen und Spezifikationen angegeben werden:

-r	Die Nachrichten werden in umgekehrter Reihenfolge ihres Eingangs ausgegeben (Standard: FIFO - first in first out).
-q	Das mail-Kommandoprogramm wird bei Eingehen eines Interrupts (Eingabe des Zeichens CTRL-C am Terminal) verlassen.
-p	Es erfolgt die aufeinanderfolgende Ausgabe aller Nachrichten ohne Zwischenrückfragen.
-f file	Die Datei 'file' wird so ausgegeben, als handele es sich um eine Nachrichtendatei.

Kommando: make Programmerzeugungshilfe (maintain program group)

Aufruf: **make** [-f makefile] [option] **file**

Beschreibung:
Das Kommandoprogramm 'make' ist ein Hilfsmittel für die Erstellung großer Programmsysteme unter Beachtung gegebener Abhängigkeiten zwischen einzelnen Programmodulen. Es unterstützt sowohl die automatische Überwachung der Einhaltung dieser Abhängigkeiten als auch die vordefinierte Ablaufverarbeitung zur Erzeugung des fertigen Programmsystems.

in [1] S.I. Feldman: Make - A Program for Maintaining Computer Programs
[13] H. McGilton, R. Morgan: Einführung in das UNIX-System

Kommando: Ausgabe einer Kommandobeschreibung (print manual)

Aufruf: **man command**

Beschreibung:
Das Kommandoprogramm 'man' dient der Ausgabe spezifizierter Teile der rechnerintern gespeicherten Dokumentation aller UNIX-Standardkommandos (on-line manual). Das ausgewählte Kommando ('command') wird in der Eingabezeile des man-Kommandos angegeben. Die Ausgabe der entsprechenden Dokumentation erfolgt auf dem Standardausgabegerät.

Kommando: mesg Nachrichtenempfangssperre (permit or deny messages)

Aufruf: **mesg** [-n] [-y]

Beschreibung:
Mit Hilfe des Kommandoprogramms 'mesg' kann der Empfang von write-Nachrichten (siehe auch Kommandoprogramm 'write') von anderen Nutzern des UNIX-Rechnersystems durch die Angabe von zwei unterschiedlichen Optionen erlaubt ('y') oder verboten ('-n') werden.

Kommando: mkdir Anlegen eines Directories (make a directory)

Aufruf: **mkdir dirname**

Beschreibung:
Mit dem Kommandoprogramm 'mkdir' kann jeder Nutzer, seinen speziellen Wünschen und Erfordernissen entsprechend, ein oder mehrere Subdirectories ('dirname ...') anlegen. Das neue Dateiinhaltsverzeichnis wird mit dem Zugriffsmode '0777' im momentanen Arbeitsdirectory angelegt. Voraussetzung für das Anlegen neuer Directories ist der Besitz der Schreiberlaubnis des Anwenders im momentanen Arbeitsdirectory.

Kommando: Verschieben oder Umbenennen von Dateien (move or rename files)

Aufruf: **mv file1 file 2**
mv file directory

Beschreibung:
Das Kommandoprogramm 'mv' ermöglicht das Umbenennen einer Datei mit einem alten Dateinamen ('file1') in eine Datei mit einem neuen Dateinamen ('file2').
Außerdem ermöglicht es das Verschieben einer oder mehrerer Dateien 'file von einem momentanen Directory in ein neues Directory ('directory').

Im Gegensatz zum Kommandoprogramm 'cp' (copy) wird beim Kommandoprogramm 'mv' der alte Dateiname ('file1') gelöscht. Beim Verschieben bleibt der originale Dateiname bis auf die sich neu ergebende Pfadspezifikation unverändert erhalten.

Kommando: newgrp Ändern der Gruppenzugehörigkeit (log new group)

Aufruf: **newgrp group**

Beschreibung:
Das Kommandoprogramm 'wgrp' ermöglicht einem Besitzer, seine momentane Gruppenzugehörigkeit zu erweitern, indem er sich in eine neue Arbeitsgruppe einschaltet. Er besitzt danach alle Zugriffsrechte der neuen Gruppe ('group'). Ist für die Gruppe 'group' ein Schlüsselwort (password) definiert, wird dies vor der Erteilung der neuen Gruppenzugriffsrechte abgefragt.

Kommando: nice/nohup Wechsel der Abarbeitungspriorität (run a command at low priority)

Aufruf: **nice** [-number] **command** [arguments]
nohup **command** [arguments]

Beschreibung:
Mit Hilfe des Kommandoprogramms 'nice' kann die Abarbeitungspriorität eines Programmes ('command') mit eventuell zugehörigen Parameterangaben ('arguments') verändert werden. Der UNIX-Anwender kann dabei die Priorität nur verringern, der UNIX-Superuser kann sie mit Hilfe von 'nice' auch erhöhen.
Im UNIX-Betriebssystem entspricht die Zahl 0 der höchsten, die Zahl 20 der niedrigsten Priorität. Vorgegebene Standardpriorität ist 10. Gibt man im nice-Kommando eine Zahl ('number') an, wird diese Zahl zur vorgegebenen Priorität addiert. Gibt man keine Zahl im Kommandoaufruf an, wird die Prioritätszahl um 7 erhöht.
Das Kommandoprogramm 'nohup' bewirkt, daß die Abarbeitung eines Programmmes ('command' 'argument') durch keinen externen Interrupt (Eingabe des Zeichens CTRL-C am Terminal) unterbrochen wird.

Kommando: Symboltabellenausgabe (print name list)

Aufruf: **nm** [-gnopru] [file ...]

Beschreibung:
Das Kommandoprogramm 'nm' ermöglicht die Ausgabe der Symboltabelle einer oder mehrerer Objektprogrammdateien ('file ...') zusammen mit dem jeweiligen hexadezimalen Wert des Symbolnamens und einem Kennzeichnungsbuchstaben.

Voraussetzung für die ordnungsgemäße Arbeit von 'nm' ist, daß für die angegebene Datei eine Symboltabelle generiert wurde. Gibt man keinen Dateinamen im Aufruf an, wird standardmäßig die Datei 'a.out' bearbeitet. Das nm-Kommandoprogramm kennt folgende Optionen:

-g Nur externe und globale Symbolnamen werden ausgegeben.
-n Die ausgegebenen Symbole werden vorher entsprechend ihren numerischen Werten sortiert.
-o Der Dateiname wird vor jedem Symbol ausgegeben. (Wichtig für die Symbolausgabe von Archivdateien.)
-p Die Ausgabe erfolgt unsortiert entsprechend der Symbolreihenfolge in der Objektprogrammdatei (Standard: alphabetisch sortiert).
-r Die Ausgabe erfolgt in umgekehrter Reihenfolge wie bei der Option '-p
-u Es werden nur undefinierte Symbolnamen ausgegeben.

Bei der Ausgabe der Symboltabelle finden die folgenden Kennzeichnungsbuchstaben Verwendung:

U Symbolname ist undefiniert.
A Absolutwertsymbolname.
T Textteilsymbolname.
D Datenteilsymbolname (initialisierte Daten).
B BSS-Segmentsymbolname (nichtinitialisierte Daten).
C allgemeiner Symbolname.

Wird der Kennzeichnungsbuchstabe als Kleinbuchstabe ausgegeben, handelt es sich um einen lokalen, anderenfalls um einen globalen Symbolnamen.

Kommando: od Dumpausgabe (octal dump)

Aufruf: **od** [-bcdox] [file] [[X]offset[.]] [b]

Beschreibung:
Das Kommandoprogramm 'od' ermöglicht die Ausgabe einer Datei ('file') oder - wenn kein Dateiname angegeben wird - des Standardeingabezeichenstroms als Dumpausdruck mit Adressenangabe. Durch fünf Optionen ist das Ausgabeformat frei wählbar:

-b byteweise Oktalformatausgabe.
-c byteweise Ausgabe als ASCII-Zeichen (wenn möglich).
-d wortweise Dezimalformatausgabe.
-o wortweise Oktalformatausgabe.
-x wortweise Hexadezimalformatausgabe.

Gibt man einen Versatzwert ('offset') in Form einer oktalen Zahl (ohne besondere Spezifikation), in Form einer hexadezimalen Zahl (mit X beginnend) oder in Form einer dezimalen Zahl (durch beendet) an, wird der Dump erst ab der angegebenen Adresse ausgegeben.
Die abschließende Notation des Buchstabens 'b' bewirkt, daß nur der erste 512 Byte lange Block bearbeitet wird.

Kommando: passwd Schlüsselwortvorgabe (change login password)

Aufruf: passwd [name]

Beschreibung:
Das Kommandoprogramm 'passwd' dient zum Installieren oder zum Verändern des login-Schlüsselwortes eines UNIX-Nutzers ('name'). Die Vorgabe des neuen oder des veränderten Schlüsselwortes erfolgt vom passwd-Kommando dialoggesteuert nach Abfrage eines eventuell vorhandenen alten Schlüsselwortes und nach zweimaliger Sicherheitsabfrage des neuen Schlüsselwortes. Vergißt ein UNIX-Nutzer ein einmal von ihm installiertes Schlüsselwort für seine login-Zugriffsrechte, kann nur der Systemprogrammierer (superuser) helfen.
Achtung: Die Schlüsselwörter werden zur Sicherheit vor unbefugt Zuschauenden in keinem Fall auf dem Bildschirm angezeigt.

Kommando: pr formatierte Dateiausgabe (print file)

Aufruf: pr [option] [file ...]

Beschreibung:
Das Kommandoprogramm 'pr' dient zur formatierten Ausgabe einer oder mehrerer Dateien ('file ...') oder, wenn kein Dateiname im Aufruf angegeben wird, des Standardeingabezeichenstroms. Das pr-Kommando bewirkt die in Seiten aufgeteilte Ausgabe der Datei mit einem fünfzeiligen Seitenkopf und einem fünfzeiligen Seitenfuß. Die Standardseite hat dabei 56 Nutzzeilen für den Dateitext. Der Seitenkopf enthält das Datum, den Dateinamen (oder eine andere durch die Option 'h' spezifizierte Headerinformation) und die Seitennummer.
Das Kommandoprogramm 'pr' kennt folgende Optionen:

-n	Die Datei wird in 'n' Spalten ausgegeben - jede Zeile wird deshalb auf '72/n'-Zeichen gekürzt (n positive ganze Zahl).
+n	Die Datei wird erst ab Seite 'n' ausgegeben.
-htext	Anstelle des Dateinamens wird die Zeichenkette 'text' als Headerinformation ausgegeben.
-wn	Die Anzahl der Zeichen in einer Zeile beträgt 'n' (Standard: 72).
-ln	Die Anzahl der Zeilen einer Seite beträgt 'n' (Standard: 66).
-t	Die Ausgabe der fünf Kopf- und Fußzeilen entfällt.
-sc	Die Spalten einer mehrspaltigen Dateiausgabe werden durch das Zeichen 'c' separiert (Standard: Tabulatorzeichen).
-m	Mehrere Dateien ('files ...') werden simultan, jede jeweils in einer Spalte, ausgegeben.

Der vom pr-Kommandoprogramm erzeugte Ausgabezeichenstrom wird auf dem Standardausgabegerät ausgegeben.

Kommando: prof Ausgabe eines Programmabarbeitungsprofils (display profile)

Aufruf: prof [-al] [file]

Beschreibung:
Das Kommandoprogramm 'prof' ermöglicht die Ausgabe eines Programmabarbeitungsprofils (welche Funktionsaufrufe werden verwendet, wie groß ist die Abarbeitungszeit usw.) bei einem Programmlauf auf das Standardausgabegerät. Voraussetzung für die Anwendung des prof-Kommandoprogrammes ist allerdings, daß im abzuarbeitenden Programm ('file') die Funktion 'monitor (...)' definiert wurde, so daß beim Programmlauf die Standarddatei 'mon.out' entsteht.
Gibt man keinen Dateinamen ('file') im Aufruf von 'prof' wird die Standarddatei 'a.out' vom prof-Kommandoprogramm bearbeitet.
Durch die Angabe der Option '-a' wird die Ausgabe aller externen und internen Symbole bei der Auswertung angefordert (Standard: externe Symbole).
Mit der Option '-l' wird die Ausgabe nach Symbolen geordnet angefordert (anderenfalls nach fallenden Prozentzahlen geordnet).

Kommando: ps Prozeßstatus (process status)

Aufruf: ps [-alx] [namelist]

Beschreibung:
Das Kommandoprogramm 'ps' dient zur Ausgabe des momentan aktuellen Status der laufenden Prozesse auf dem Standardausgabegerät.
Eine Reihe von Optionen ermöglicht die Spezifizierung dieser Ausgabeliste nach bestimmten Kriterien:

a Es wird nicht nur eine Liste der eigenen Prozesse (Standard), sondern auch eine komplette Liste aller weiteren laufenden UNIX-Prozesse erzeugt.
l Es wird eine lange Ausgabeliste der aktuellen Prozesse erzeugt, die viele zusätzliche Informationen enthält, die in der Standardliste nicht enthalten sind.
x Es werden auch alle die Prozesse zur Ausgabe gebracht, die keinem Nutzerterminal zugeordnet sind.

Kommando: ptx Wortindexerstellung (print index)

Aufruf: ptx [option] [input] [output]

Beschreibung:
Das Kommandoprogramm 'ptx' ermöglicht die Erstellung eines permutierten Wortindexverzeichnisses (Sachwörterverzeichnis), wie es im "UNIX Programmers Manual" [1] verwendet wird. Verarbeitet wird vom ptx-Kommando

normalerweise der Standardeingabezeichenstrom - der erarbeitete Index wird auf dem Standardausgabegerät ausgegeben. Die Angabe einer Eingabe- ('input') und/oder Ausgabedatei ('output') im Kommandoaufruf ist optional.

in [1] UNIX Programmers Manual - ptx(1)

Kommando: pwd Ausgabe des Arbeitsdirectories (print working directory)

Aufruf: pwd

Beschreibung:
Das Kommandoprogramm 'pwd' dient der Ausgabe des Namens des momentanen Arbeitsdirectories auf dem Standardausgabegerät.

Kommando: ratfor FORTRAN-Preprocessor (rational FORTRAN)

Aufruf: ratfor [option] [file.r ...]

Beschreibung:
Das Kommandoprogramm 'ratfor' stellt einen FORTRAN-Preprocessor dar, der die FORTRAN-Programmnotation etwas rationalisiert. Das ratfor-Kommandoprogramm übersetzt das in der ratfor-Syntax abgefaßte Programm in ein Programm, das den Syntaxregeln des FORTRAN77-Compilers 'f77' entspricht.

in [1] B.W. Kernighan: RATFOR - A Preprocessor for a Rational Fortran

---

Kommando: rm/rmdir Löschen von Dateien und Directories (remove)

Aufruf: rm [-fri] file
rmdir directory

Beschreibung:
Die Kommandoprogramme 'rm' und 'rmdir' dienen zum Löschen von einer oder von mehreren Dateien bzw. von einem oder von mehreren Directories. Beim Löschen von Directories wird vom rmdir-Kommandoprogramm vorausgesetzt, daß das angegebene Directory keine Dateien oder Subdirectories enthält.
Das rm-Kommandoprogramm kennt folgende Optionen:

-f Es erfolgt ein zwingendes Löschen von Dateien (auch von geschützten Dateien).
-r Es erfolgt ein rekursives Löschen eines Directories mit allen in ihm enthaltenen Dateien und Subdirectories (Achtung !).
-i Es erfolgt bei der Angabe der Option '-r' ein interaktives Löschen mit gesonderter Löschabfrage der einzelnen Dateien und Subdirectories.

Kommando: sed Datenstromeditor (stream editor)

Aufruf: **sed** [-n] [-e script] [-f sfile] [file ...]

Beschreibung:
Bei dem Kommandoprogramm 'sed' handelt es sich um einen sog. Datenstromeditor. Der wesentliche Unterschied zwischen dem ed-Editor und dem sed-Editor ist dabei in der nichtinteraktiven Arbeit von 'sed' zu suchen. Die Kommandosyntax beider Editoren ist nahezu identisch.
Beim sed-Editorprogramm muß zunächst eine durch die Option '-f' gekennzeichnete Datei ('sfile') angelegt werden, die alle Befehlseingaben für den sed-Editor enthält. Während der sed-Editorabarbeitung werden keine Eingaben verlangt, da das sed-Kommandoprogramm seine Befehle dann aus der Datei 'sfile' entnimmt und abarbeitet. Durch die Option '-e' kann die abzuarbeitende Befehlsfolge 'script' für den sed-Editor allerdings auch direkt in der sed-Aufrufzeile notiert werden.
Vom sed-Editor wird normalerweise der Standardeingabezeichenstrom verarbeitet. Es können aber auch eine oder mehrere zu verarbeitende Dateien ('file ...') benannt werden. Das Ergebnis der sed-Verarbeitung erscheint auf dem Standardausgabegerät.
Gibt man beim sed-Aufruf die Option '-n' an, werden nur die vom sed-Editor bearbeiteten Dateizeilen ausgegeben. Ohne Angabe dieser Option werden alle Dateizeilen, auch die, die unverändert übernommen werden, zur Ausgabe gebracht.
Die Verarbeitungsmöglichkeiten von 'sed' gehen über die reinen Editorbefehle von 'ed' an einigen Stellen hinaus. Zusätzliche Möglichkeiten existieren dabei insbesondere bei der Verarbeitung sogenannter 'regular expressions' (siehe auch 'grep').

in [1] L.E. McMahan: SED A Non-Interactive Text Editor
[12] Z. Stanka, S. Lösch: UNIX Führer durch das System
[13] H. McGilten, R. Morgan: Einführung in das UNIX System

Kommando: sh Shellkommandointerpreter (command language)

Aufruf: **sh** [-option] [arg ...]

Beschreibung:
Bei dem Kommandoprogramm 'sh' handelt es sich um den Standardkommandointerpreter des UNIX-Betriebssystems. Der sh-Kommandointerpreter wird normalerweise implizit beim Systemstart aufgerufen. Sein Aufruf kann allerdings auch, wie angegeben, explizit erfolgen. Der Shellkommandointerpreter des UNIX-Betriebssystems wird im sechsten Abschnitt dieses Buches behandelt (siehe auch 'csh').

Kommando: Ausgabe der Objektprogrammgröße (size of an object file)

Aufruf: [objectfile ...]

Beschreibung:
Mit dem Kommandoprogramm 'size' kann man die Größe einer oder mehrerer Objektprogrammdateien ('objectfile') ermitteln und auf dem Standardausgabegerät ausgeben. Gibt man keine Objektprogrammdatei im size-Aufruf an, wird die Größe der Datei 'a.out' ermittelt.

Kommando: sleep Abarbeitungspause (suspend execution)

Aufruf: **sleep time**

Beschreibung:
Das Kommandoprogramm 'sleep' ermöglicht die Spezifizierung einer Abarbeitungspause 'time' (in Sekunden), in der der Rechner wartet, bis er den nächsten Befehl ausführt.

Kommando: sort Sortierprogramm (sort or merge files)

Aufruf: **sort** [-mubdfinrtx] [+pos1] [-pos2] [-o oname] [-T directory] [iname ...]

Beschreibung:
Das Kommandoprogramm 'sort' ist ein sehr universelles Programm zum Sortieren und Mischen von Dateiinhalten. Das sort-Kommandoprogramm verarbeitet die benannte(n) Datei(en) ('iname ...') oder den Standareingabezeichenstrom (für 'iname' muß dann ein Bindestrich stehen). Das Ergebnis der sort-Verarbeitung wird auf dem Standardausgabegerät ausgegeben oder in der durch die Option '-o' gekennzeichneten Ausgabedatei ('oname') abgelegt. Bei der Bearbeitung anfallende temporäre Zwischendateien können bei Bedarf durch die Option '-T' in einem bestimmten Directory ('directory') abgelegt werden.
Gibt man mehrere Dateien ('iname ...') im sort-Kommandoaufruf vor, werden diese Dateien simultan behandelt. Es entsteht dann ein (entsprechend den vorgegebenen Kriterien sortiertes) zusammengemischtes Ergebnis.
Eine Schlüsselrolle bei der sort-Bearbeitung kommt der Zerlegung der zu verarbeitenden Eingabezeilen in einzelne, standardmäßig durch Leerzeilen oder Tabulatoren voneinander separierte Felder zu. Auf diese Felder wird der Sortierungsschlüssel des Kommandoprogrammes 'sort' angewendet.
Die Angabe der Notation '+pos1' und '-pos2' im sort-Kommandoaufruf zeigt an, daß die für das Sortieren signifikanten Felder einer Zeile bei 'pos1' beginnen und vor 'pos2' enden. Die Angabe von 'pos1' und 'pos2' erfolgt dabei grundsätzlich in der Form 'm.n' gegebenenfalls von einem oder mehreren Optionsbuchstaben ('bdfinr') gefolgt.

Dabei bedeuten:

m (ganze Zahl) - Steht für die Anzahl der zu überspringenden Felder vom Zeilenanfang gerechnet.
u (ganze Zahl) - Steht für die Anzahl der zu überspringenden Zeichen vom (durch die Angabe von gekennzeichneten) Feldanfang gerechnet.
b Führende Leerzeichen im spezifizierten Feld werden übergangen.
d Nur Buchstaben, Zahlen und Leerzeichen werden beim Sortieren berücksichtigt (entsprechend Wörterbuchregeln).
f Es wird beim Sortieren kein Unterschied zwischen Groß- und Kleinbuchstaben gemacht.
i Alle Zeichen außerhalb des ASCII-Kodes (040 bis 176 / oktal) werden beim Sortieren ignoriert.
n Die Sortierung erfolgt nach dem ersten auftretenden Zahlenfeld. Die Sortierung erfolgt aufsteigend, entsprechend dem arithmetischen Wert des Zahlenfeldes (Vorzeichen, führende Nullen und Dezimalpunkte sind im Zahlenfeld zugelassen).
r Die Reihenfolge des Sortiervorgangs wird umgekehrt.

Die in 'pos1' und in 'pos2' angegebenen Optionsbuchstaben überschreiben nötigenfalls die im sort-Aufruf angegebenen Optionen. Wird die pos-Angabe ('pos1' und 'pos2') weggelassen, gelten als Standardwerte der Anfang und das Ende jeder Zeile.
Neben den bei der Feldspezifikation angebbaren Optionen ('bdfinr') können beim sort-Kommandoprogramm die folgenden Optionen ('-mubdfinrtx') auch global spezifiziert werden:

m Die Eingabedateien (' ...') werden nur gemischt, nicht sortiert.
u Doppelte Zeilen werden beim Sortieren/Mischen gelöscht.
b s.o.
d
f
i
n s.o.
r s.o.
tx Das Feldtrennzeichen in den Eingabedateien ist kein Leerzeichen oder Tabulatorzeichen, sondern das spezifizierte Zeichen 'x' (bei Sonderzeichen muß vor dem Sonderzeichen dann zusätzlich ein Backslash-Zeichen stehen).

Kommando: spell Suchen Rechtschreibfehlern (find spelling errors)

Aufruf: **spell -option** [file ...]

Beschreibung:
Das Kommandoprogramm 'spell' sucht der (oder in den) Datei(en) ('file ...') nach Wörtern, die nicht in einer zum UNIX-Betriebssystem gehörenden Referenzwörterbuchdatei im Directory '/usr/dict' vorkommen. Die Referenzwörterbuchdatei muß dabei dem Standardwortschatz der verwendeten Sprache entsprechen. Gibt man keinen Eingabedateinamen 'file ...' an, wird

der Standardeingabezeichenstrom verarbeitet. Das Ergebnis der spell-Verarbeitung erscheint auf der Standardausgabe.
Folgende Optionen werden vom spell-Kommandoprogramm verarbeitet:

-v Anzeige aller nicht vorkommenden Wörter.
-b Im Englischen gilt die britische Schreibweise.
-x Anzeige des Wortstamms.

Kommando: split Aufsplitten einer Datei (split a file into pieces)

Aufruf: **split** [-n] [file [name]]

Beschreibung:
Das Kommandoprogramm 'split' teilt eine Datei ('file') in mehrere Dateien mit einer festen Anzahl ('-n') von Zeilen auf.
Der Name der Ausgabedateien wird aus den Buchstaben (Standard) oder aus einem angegebenen Namen ('name'), ergänzt durch die fortlaufende Angabe von zwei Buchstaben der Folge aa, ab, ...az, ba, bb, bc ...zz, gebildet.
Gibt man keine Eingabedatei an, wird der Standardeingabezeichenstrom verarbeitet. Wird keine Zeilenzahl für die aufgesplitteten Dateien angegeben, gilt implizit n=100.

Kommando: strip Entfernen der Symboltabelle und der Relokationsbits (remove symbols and relocation bits)

Aufruf: **strip objectfile**

Beschreibung:
Das Kommandoprogramm 'strip' entfernt aus einer (oder aus mehreren) Objektprogrammdatei(en) die angehängte Symboltabelle und die zum Laden notwendigen Relokationbits, um notwendigenfalls Speicherplatz einzusparen.

Kommando: struct Strukturieren von FORTRAN-Programmen (structure FORTRAN programs)

Aufruf: **struct** [option] **file**

Beschreibung:
Das Kommandoprogramm 'struct' dient zur Überführung eines FORTRAN-Programms in eine für den FORTRAN-Preprocessor 'ratfor' verarbeitbare Form. Dabei werden neben verschiedenen kosmetischen Veränderungen insbesondere Steuerstrukturen, Statementnumerierungen und Zeichenkettenvereinbarungen umgewandelt (siehe auch FORTRAN77-Compiler).

in [17] B.W. Kernighan: RATFOR - A Preprocessor for a Rational Fortran

Kommando: stty Setzen Terminalparametern (set terminal options)

Aufruf: **stty** [option]

Beschreibung:
Mit Hilfe des Kommandoprogrammes 'stty' kann im Terminaltreiber des UNIX-Betriebssystems eine Fülle von Parametern ('option') eingestellt werden, die der Anpassung an die Eigenschaften des angeschlossenen Terminals dienen (Baudrate, Zeilenvorschubmode usw.).
Wird das stty-Kommando ohne Angabe einer Option aufgerufen, werden die momentan eingestellten Terminaloptionen ausgegeben.
Die wichtigsten Optionen des stty-Kommandoprogramms sind:

even	Sperren gerader Parität.
-even	Freigeben gerader Parität.
odd	Sperren ungerader Parität.
-odd	Freigeben ungerader Parität.
cbreak	Zeichenweise Eingabe.
-cbreak	Zeilenweise Eingabe.
nl	Zeilenabschluß durch ein Newline-Zeichen.
-nl	Zeilenabschluß durch ein Newline- und ein Linefeed-Zeichen.
echo	Echoausgabe der eingegebenen Zeichen.
-echo	Keine Echoausgabe der eingegebenen Zeichen.
lcase	Großbuchstaben werden zu Kleinbuchstaben umgewandelt.
-lcase	Großbuchstaben werden nicht zu Kleinbuchstaben umgewandelt.
tabs	Tabulatorzeichen bleiben erhalten.
-tabs	Tabulatorzeichen werden zu Leerzeichen umgewandelt.
erase x	Das Zeichen 'x' wird zum Erase-Zeichen.
50, 75, 110, 150, 200, 300, 600, 1200, 2400, 4800, 9600	Setzen der Terminalbaudrate.

Kommando: Erweitern der Benutzerzugriffsrechte (substitute user id temporarily)

Aufruf: **su** [userid]

Beschreibung:
Das Kommandoprogramm 'su' ermöglicht die Erweiterung der Benutzerzugriffsrechte auf die durch 'userid' spezifizierten Dateien. Voraussetzung ist dabei die Kenntnis des zugehörigen Schlüsselwortes ('passwd').
Gibt man keine Benutzeridentifikation ein, wird durch 'su' eine Erweiterung auf die Zugriffsrechte des UNIX-Superusers angefordert.

Kommando: Bestimmung der Blockanzahl und der Prüfsumme einer Datei (sum and count blocks in a file)

Aufruf: **sum file**

Beschreibung:
Das Kommandoprogramm 'sum' ermittelt eine 16-Bit-Prüfsum der benannten Datei und zählt die Anzahl der von ihr belegten 512-Byte-Blöcke. Beides wird auf das Standardausgabegerät ausgegeben.

Kommando: tabs Setzen der Tabulatoreinstellung (set terminal tabs)

Aufruf: **tabs terminal**

Beschreibung:
Das Kommandoprogramm 'tabs' ermöglicht das Setzen der Tabulatoreinstellung des angeschlossenen Terminals entsprechend bestimmter vorgegebenen Terminaltypen ('terminal').

in /1/ UNIX Programmer's Manual term(7)

Kommando: tail Übergeben eines bestimmten Teils einer Datei an das Standardausgabegerät (deliver the last part of a file)

Aufruf: **tail** [+/-number [-lbc]] [file]

Beschreibung:
Das Kommandoprogramm 'tail' ermöglicht das Übergeben eines bestimmten Teils einer Datei ('file') an das Standardausgabegerät. Gibt man keine Datei ('file') im Aufruf an, wird der Standardeingabezeichenstrom verarbeitet.
Der zu übergebende Teil beginnt nach '+number' Einheiten vom Dateianfang oder bei '-number' Einheiten vor dem Dateiende. Beendet wird die Dateiübergabe immer erst mit dem Dateiende. Gezählt werden als Einheiten dabei entweder die Zeilen einer Datei ('-l'), die 512-Byte-Blöcke einer Datei ('-b') oder die Zeichen einer Datei ('-c'). Standardmäßig gilt die Angabe '+/-number' immer für Dateizeilen. Wird die Angabe '+/-number' weggelassen, werden die letzten 10 Zeilen einer Datei an das Standardausgabegerät übergeben.

Kommando: tar Magnetbandarchivar (tape archiver)

Aufruf: tar [key] [name ...]

Beschreibung:
Das Kommandoprogramm 'tar' ermöglicht das selektive Ein- und Auslesen von Dateien ('name ...') oder Directories ('name ...') auf das jeweilige Back-Up-Medium Magnetband oder Floppy-Disk des UNIX-Rechnersystems. Die auf dem externen Datenträger aufgespielten Dateien werden im Gegensatz zu dem Kommandoprogrammen 'dump' und 'restor' im sogenannten tar-Format abgelegt, das auch den gezielten Abruf einer einzelnen Datei ermöglicht.
Die Arbeit des Kommandoprogrammes 'tar' wird durch Schlüsselbuchstaben 'key' gesteuert:

r Die benannten Dateien oder/und Directories werden am Ende des tar-Archivs abgelegt.

x Es erfolgt ein Rückschreiben der benannten Dateien oder/und Directories aus dem tar-Archiv in das UNIX-Dateisystem. Gibt man keine Dateinamen oder Directorynamen an, wird das komplette tar-Archiv zurückgeschrieben.
Im UNIX-Dateisystem eventuell vorhandene Dateien oder Directories mit gleichen Namen werden überschrieben.

t Es wird ein Inhaltsverzeichnis des tar-Archivs auf dem Standardausgabegerät ausgegeben.

u Es werden nur noch nicht auf dem tar-Archiv befindliche Dateien oder Directories bzw. solche, die seit dem letzten Aufspielen modifiziert wurden, in das tar-Archiv beim Aufspielen übernommen.

c Es wird ein neues tar-Archiv angelegt. Ein alter tar-Archivinhalt wird dabei (auch abhängig von der Magnetbandpositionierung) ggf. überschrieben.

v Alle Aktionen von 'tar' werden auf dem Standardausgabegerät angezeigt.

Weitere tar-Parameterangaben, Laufwerksselektnummern, Blockungsfaktoren, Gerätenamensvorgaben u.a.m. sind beim tar-Kommandoprogramm abhängig von der jeweiligen UNIX-Rechnerimplementation.

Kommando: tbl Tabellenformatierungsprogramm (format tables for nroff or troff)

Aufruf: tbl [files ...]

Beschreibung:
Das Kommandoprogramm 'tbl' ist ein Preprozes zur Formatierung von Tabellen bei der nroff- und troff-Textverarbeitung. Die zu formatierenden Tabellen werden in der spezifizierten Datei ('file ') oder im Standardeingabezeichenstrom für die tbl-Preprozessorverarbeitung speziell markiert ('.TS' und .TE').

Mit Hilfe einer Reihe von Formatierungsbefehlen kann dann innerhalb der markierten Tabelle die gewünschte Tabellenform eingestellt werden.

in [17] M.E. Lesk: TBL - A Program to Format Tables

Kommando: tee Ein-/Ausgabepipevermittlungsprogramm (pipe fitting)

Aufruf: **tee** [-i] [-a] [file]

Beschreibung:
Das Kommandoprogramm 'tee' vermittelt den Standardeingabezeichenstrom ohne Veränderung an das Standardausgabegerät und legt bei Angabe eines Dateinamens ('file') gleichzeitig eine Kopie der vermittelten Daten in einer Datei ab.
Das Kommandoprogramm 'tee' kennt zwei Optionen:

-i Ignorieren von Interruptzeichen (CTRL-C) im Datenstrom.
-a Anfügen der Daten an die bereits bestehende Datei 'file'.

Kommando: test Testkommando (condition command)

Aufruf: **test expr**

Beschreibung:
Das Kommandoprogramm 'test' dient zur Realisierung bedingungsabhängiger Aktionen. Es ermittelt den Status von 'expr' und gibt einen Ausgangswert '0' (TRUE) oder 'ungleich 0' (FALSE) zurück.
Für den Wert 'expr' können folgende Eingaben stehen:

-r file	TRUE wenn 'file' existiert und lesbar.
-w file	TRUE, wenn 'file' existiert und schreibbar.
-f file	TRUE, wenn 'file' existiert und kein Directory.
-d file	TRUE, wenn 'file' existiert und ein Directory.
-s file	TRUE, wenn 'file' existiert und mehr als 0 Bytes enthält.
-t (number)	TRUE, wenn die eröffnete Datei, deren 'file descriptor number' der Zahl 'number' entspricht, mit einem Terminal verbunden ist (wird 'number' nicht angegeben, gilt number=1).
-z s1	TRUE, wenn die Zeichenkette 's1' die Länge 0 hat (nicht existiert).
-n s1	TRUE, wenn die Zeichenkette 's1' eine Länge ungleich 0 hat (existiert).
s1=	TRUE, wenn die Zeichenketten 's1' und 's2' gleich sind.
s1!=s2	TRUE, wenn die Zeichenketten 's1' und 's2' nicht gleich sind.
s1	TRUE, wenn die Zeichenkette 's1' kein Nullstring ist.
n1 -eq n2	TRUE, wenn die Integerzahlen 'n1' und 'n2' gleich sind.
n1 -ne n2	TRUE, wenn die Integerzahlen 'n1' und 'n2' ungleich sind.
n1 -gt n2	TRUE, wenn die Integerzahl 'n1' größer als die Integerzahl 'n2' ist.

n1 -ge n2 TRUE, wenn die Integerzahl 'n1' größer oder gleich der Integerzahl 'n2' ist.
n1 -lt n2 TRUE, wenn die Integerzahl 'n1' kleiner als die Integerzahl 'n2' ist.
n1 -le n2 TRUE, wenn die Integerzahl 'n1' kleiner oder gleich der Integerzahl 'n2' ist.

Die angegebenen Ausdrücke können auch negiert oder logisch verknüpft verwendet werden:

! Negationsoperatorzeichen.
-a logisches UND.
-o logisches ODER.

Außerdem können durch die Angabe von Klammern Gruppierungen gebildet werden.

Kommando: time Bestimmung der Kommandoausführungszeit (time a command)

Aufruf: **time command**

Beschreibung:
Das Kommandoprogramm 'time' ermöglicht die Bestimmung der Ausführungszeit eines vorgegebenen Kommandos ('command').
Ausgegeben wird die Anwenderprogrammausführungszeit (user), die Systemprogrammausführungszeit (system) und die Summe aus beiden (alles in Sekunden). Die Ausgabe selbst erfolgt auf dem Diagnosedatenausgabezeichenstrom (Terminal).

Kommando: touch Aktualisierung Modifikatiosdatum und -zeit einer Datei (update date last modified)

Aufruf: **touch file**

Beschreibung:
Das Kommandoprogramm 'touch' ermöglicht die Aktualisierung des Datums und der Uhrzeit der letzten Modifikation einer oder mehrerer Dateien auf das momentane UNIX-Systemdatum und auf die momentane UNIX-Systemzeit.

Kommando: tr Zeichenkettenkonvertierung (translate characters)

Aufruf: **tr** [-cds] **string1 string2**

Beschreibung:
Das Kommandoprogramm 'tr' ist ein Zeichenkettenfilter. Es übersetzt den Standardeingabezeichenstrom in einen Standardausgabezeichenstrom in der Weise, daß jedem Zeichen der Standardeingabe, das im 'string1' vorkommt,

durch das an der gleichen Position im 'string2' stehende Zeichen ersetzt wird.
Gibt man den 'string2' kürzer als den 'string1' vor, wird der Rest durch Vervielfachung des letzten Zeichens automatisch aufgefüllt.
Das Kommandoprogramm 'tr' kennt drei Optionen:

-c Die Werte der Zeichenfolge 'string1' werden komplementiert.
-d Alle in 'string1' vorkommenden Zeichen des Eingabezeichenstroms werden gelöscht.
-s Sich wiederholende Zeichen des Eingabezeichenstroms werden unterdrückt.

Die Notation der Zeichenfolgen 'string1' und 'string2' kennt folgende Besonderheiten:

a-b	Alle Zeichen zwischen dem Zeichen und dem Zeichen 'b'.
Backslash-Zeichenxxx	Ein Zeichen mit dem oktalen Wert ('xxx' Zahlwert).
Backslash-Zeicheny	Steht für ein Backslash-Zeichen ('y' irgendein Zeichen, ausgenommen eine Zahl).

Kommando: troff/nroff Textformatierungsprogramm (format text)

Aufruf: **troff -mr** [option] **file**
nroff -mr [option] **file**

Beschreibung:
Die Kommandoprogramme 'troff' und 'nroff' gehören zur Familie der UNIX-Textverarbeitungsprogramme. Sie ermöglichen die Formatierung von Texten durch Einfügung von speziellen nroff/troff-Befehlen in die Textdateien. Bei der Textaufbereitung werden diese Befehle dann durch 'nroff' und 'troff' entsprechend ihrer Bedeutung interpretiert und ausgeführt.
Das Kommandoprogramm 'troff' ist speziell der Fotosatzausgabe vorbehalten, während das Kommandoprogramm 'nroff' eine oder mehrere Dateien ('file...') auf dem Standardausgabegerät ausgibt. Ein Anwendungsbeispiel für 'nroff' sind die im UNIX-Rechner gespeicherten Manuals der On-line-Dokumentation ('man'), die bei der Ausgabe mit nroff-Befehlen formatiert werden.

in [1] J.F. Ossana: NROFF/TROFF Users Manual
in [1] B.W. Kernighan: A TROFF Tutorial
[13] H. McGilton, R. Morgan: Einführung in das UNIX System

Kommando: true, false Ausgabe eines erzwungenen Exitstatus TRUE oder FALSE (provide truth values)

Aufruf: **true**
false

Beschreibung:
Die Kommandos 'true' und 'false' werden verwendet, um einen Exitstatus '0' (TRUE) oder 'ungleich 0' (FALSE) zu erzwingen, der in der Folge (meist in Shellskripten) verwendet wird, um einen bestimmten Abarbeitungsablauf zu erreichen.

Kommando: tty Ermittlung des Terminalnamens (get terminal name)

Aufruf: **tty**

Beschreibung:
Das Kommandoprogramm 'tty' dient zur Ermittlung des Namens des Terminals, an dem dieses Kommando eingegeben wird.

Kommando: uniq Löschen doppelter Dateizeilen (report repeated lines in a file)

Aufruf: **uniq** [udc] [+n [-n] [inputfile[outputfile]]]

Beschreibung:
Das Kommandoprogramm 'uniq' untersucht den Eingabezeichenstrom oder eine Eingabedatei ('inputfile') auf benachbarte, identische Zeilen. Werden solche Zeilen ermittelt, werden sie so gelöscht, daß nur eine der beiden identischen Zeilen zusammen mit den nur einmal vorkommenden Eingabezeilen an den Ausgabezeichenstrom oder an die Ausgabedatei ('outputfile') übergeben werden.
Das Kommandoprogramm 'uniq' kennt folgende Optionen:

-u Sich nicht wiederholende Eingabezeilen werden ausgegeben.
-d Jeweils eine der sich wiederholenden Eingabezeilen wird ausgegeben.
-c Jeder Zeile wird die Anzahl vorangestellt (wie oft kommt sie vor).
+n Die ersten Zeichen jeder Zeile werden übergangen.
-n Die ersten Felder jeder Zeile werden übergangen. (Ein Feld ist definiert als eine Kette von Zeichen, das keine Space- oder Tabulatorzeichen enthält.)

Kommando: units Einheitenumrechnungsprogramm (conversion program)

Aufruf: **units**

Beschreibung:
Das Kommandoprogramm 'units' dient der Umrechnung von Maßeinheiten (Zoll in Meter, Inch in Zentimeter, Pfund in Kilogramm usw.).
Die Ein-/Ausgabe der umzurechnenden Einheiten erfolgt dialoggesteuert. Eine komplette Liste der Maßeinheiten, die units-Kommandoprogramm umgerechnet werden können, befindet sich in der Datei '/usr/lib/units' des jeweiligen UNIX-Systems.

Kommando: uucp UNIX-UNIX Dateikopierprogramm (unix to unix copy)

Aufruf: **uucp** [option] **sourcefile** **destinationfile**

Beschreibung:
Das Kommandoprogramm 'uucp' ist ein komplexes Programmpaket zur Softwareunterstützung der Kopplung von zwei UNIX-Rechnern über einen Terminalkanal. Es gestattet den Austausch von Dateien von einem UNIX-Rechner zu einem anderen UNIX-Rechner. Die spezifizierten Dateinamen ('sourcefile' und 'destinationfile') werden dabei durch einen vorgestellten Systemnamensteil ('systemname!pathname') ergänzt, um den ausgewählten Rechner zu definieren. Die möglichen Systemnamen müssen dem uucp-Kommandoprogramm bekannt sein.

in [1] D.A. Nowitz: UUCP Implementation Description

Kommando: uux UNIX - UNIX Kommandoausführung (unix to unix command execution)

Aufruf: **commandstring**

Beschreibung:
Das Kommandoprogramm 'uux' dient, ähnlich wie 'uucp', der Softwareunterstützung von UNIX-Rechnerkopplungen über einen Terminalkanal. Im Gegensatz zu 'uucp' das den Dateitransfer organisiert, übernimmt 'uux' die Organisationsarbeit, die notwendig ist, um auf einem an UNIX-Rechner angeschlossenen zweiten UNIX-Rechner ein Kommando ausführen zu können. Die für eine solche abgesetzte Kommandoausführung zugelassenen Kommandos sind begrenzt (z.B.: 'mail' 'lpr' 'diff' u.a.).
Die Notation des uux-Kommandoaufrufes ('commandstring') muß in einer speziellen uux-Kommandosyntax erfolgen.

in [1] D.A. Nowitz: Uucp Implementation Description
in [1] D.A. Nowitz, M.E. Lesk: A Dial-Up Network of UNIX System

Kommando: wait Warten auf den Abschluß der Hintergrundprogrammbearbeitung (await completion of process)

Aufruf: **wait**

Beschreibung:
Das Kommandoprogramm 'wait' wartet so lange, bis alle in der Hintergrundprogrammbearbeitung befindlichen Prozesse des jeweiligen Nutzers beendet sind. Ist das der Fall, erfolgt die bis dahin verhinderte Bereitmeldung des UNIX-Systems.

Kommando: Wortzählungsprogramm (word count)

Aufruf: **wc** [-lwc] [name ...]

Beschreibung:
Das Kommandoprogramm dient der Zählung der Zeichen ('-c' characters), der Worte ('-w' words) oder/und der Zeilen ('-l' lines) einer oder mehrerer Dateien ('name ...') bzw. des Standardeingabezeichenstroms.
Wird mehr als ein Dateiname im wc-Aufruf angegeben, wird außerdem noch die totale Summe aller ermittelten Werte ausgegeben.

Kommando: who Wer arbeitet am System (who is on the system ?)

Aufruf: **who**

Beschreibung:
Das Kommandoprogramm 'who' dient der Information des aufrufenden Nutzers, wer momentan am UNIX-System (an welchem Terminal und seit wann) eingeschaltet ist.

Kommando: write Schreiben an anderen Nutzer (write to another user)

Aufruf: **write user** [ttyname]

Beschreibung:
Das Kommandoprogramm 'write' ermöglicht das Schreiben eines Textes an einen anderen momentan arbeitenden Nutzer ('user' oder 'ttyname') des gleichen UNIX-Systems. Der zu übertragende Text wird sofort nach dem write-Kommandoaufruf geschrieben. Die Übertragung erfolgt zeilenweise. Das Ende des zu übertragenden Textes wird mit einem EOT-Zeichen ('CTRL-D') angezeigt.

Kommando: yacc Compiler-Compiler (yet another compiler compiler)

Aufruf: **yacc** -vd grammer

Beschreibung:
Das Kommandoprogramm 'yacc' ist ein Programm, das mit Hilfe von anwenderdefinierten Syntaxregeln (Syntaxbaum) einen Parser erstellt, der Dateien einer anwenderorientierten Programmiersprache auf Syntaxfehler untersucht.
Das Kommandoprogramm 'yacc' arbeitet bei der Schaffung Sprachübersetzungswerkzeugen eng mit dem Kommandoprogramm 'lex' Generierung lexikalischer Analyseprogramme zusammen.

in [1] S.C. Johnson: YACC: Yet Another Compiler-Compiler
[1] A.V. Aho, S.C. Johns LR Parsing, Computer Surveys, Juni 1974

6. Kommandosprache Shell

Die Kommandosprache Shell stellt die Standardschnittstelle von UNIX Benutzer dar ("Schale" des Betriebssystems). Die bei Shell vorhandenen Möglichkeiten zur interaktiven Arbeit und beim Aufbau von Kommandoprozeduren werden in diesem Abschnitt dargelegt. Sie ermöglichen in Verbindung mit den UNIX-Kommandos für den Anwender eine effektive Erarbeitung von Problemlösungen.

6.1. Konzept und Möglichkeiten

Shell bildet das Bindeglied zwischen Benutzer und dem Betriebssystem UNIX. Sie ist sowohl eine Kommando- als auch Programmiersprache. Shell organisiert die interaktive Arbeit (Kommandoaufruf und -abarbeitung) und erlaubt den Aufbau von Kommandoprozeduren. Dadurch können oft zu wiederholende Kommandofolgen zu einem Kommando (Shell-Prozedur oder -Skript genannt) zusammengefaßt werden. Der Anwender kann mit den Möglichkeiten von Shell komplizierte Abläufe so aufbereiten, daß auch mit dem Betriebssystem weniger vertraute Personen die Bedienung übernehmen können. Dies erfolgt, indem verschiedene existierende Bausteine (UNIX-Kommandos oder eigene Programme) mit den Mitteln einer höheren Programmiersprache zu Problemlösungen verknüpft werden. Kleine Beispiele dazu werden im Abschn. 6.3. angegeben.

Die hier beschriebene Standardversion von Shell wird oft als "Bourne-Shell" (nach dem Autor S.R. Bourne) bezeichnet. Daneben existieren weitere Shellversionen, die je nach Anwendungsgebiet vom Nutzer ausgewählt werden können. Eine verbreitete Version ist C-Shell, die zusätzliche erweiterte Möglichkeiten für die interaktive Arbeit bietet.
Im Zusammenhang mit der stärkeren kommerziellen Nutzung (Büroautomatisierung) von UNIX-Systemen werden z.Z. für diesen Bereich verbesserte Nutzerschnittstellen, wie "Menü-Shell" oder "Visual-Shell" angeboten.

Die Kommandosprache Shell und UNIX-Kommandos können in beiden Richtungen miteinander kommunizieren. Es können Zeichenkettenparameter (z.B.: Dateinamen oder Optionen) an Kommandos übergeben werden. Durch die Kommandos wird ein Returnkode gesetzt, der zur Ablaufsteuerung verwendet werden kann. Die Standardausgabe eines Kommandos kann als Shelleingabe genutzt werden.
Innerhalb von Kommandoprozeduren besteht die Möglichkeit der Verwendung von Ablaufsteuerungen (z.B.: while, if..then..else, case u.a.). UNIX-Kommandos werden durch Suchen in Directories gefunden. Sie können ihre Eingaben entweder vom Terminal oder von einer Datei erhalten.
Durch Shell wird die Kommandoumgebung modifiziert. Die Ein-/Ausgabe kann z.B. Dateien zugewiesen werden. Weiterhin können Prozesse erzeugt werden, die über einen sogenannten Pipekanal miteinander kommunizieren.
Nach diesem Überblick nun zu den Möglichkeiten der interaktiven Arbeit von Shell.

6.2. Interaktive Arbeit

Nach dem Zuschalten eines Nutzers (Login-Prozedur) wird Shell gestartet. Es meldet sich mit einem sog. Promptzeichen (normalerweise "$") und wartet die Eingabe von Kommandos. Jede Eingabe ist mit einem Newline-zeichen (Returntaste) abzuschließen. Shell liest dann die Eingabezeile und organisiert die Abarbeitung.

Kommandos

Eine zusammengefaßte Beschreibung der wichtigsten UNIX-Kommandos wurde im Abschnitt 5. gegeben. Kommandos bestehen aus dem Kommandonamen (Dateiname des ausführbaren Programms) und optional gefolgt von Argumenten, die durch Leerzeichen zu trennen sind. Argumente können Dateinamen oder Optionen sein. Sie kennzeichnen die Objekte, mit denen das Programm arbeitet, und spezifizieren die Arbeitsweise des Programms. Nicht alle Kommandos sind ausführbare UNIX-Dateien. Es gibt auch Kommandos, die Bestandteil von Shell sind (z.B. 'cd'), d.h., es existiert keine Datei mit dem Namen 'cd'. Kommandooptionen werden i.allg. durch ein führendes Minuszeichen gekenn-zeichnet (implizite UNIX-Vereinbarung). So gibt z.B. das Kommando

```
cat bsp.c
```

den Inhalt der Datei bsp.c über die Standardausgabe aus. Das Kommando

```
ls -l
```

gibt eine alphabetisch sortierte Liste der Namen aller in dem aktuellen Directory enthaltenen Dateien aus. Das Argument -l teilt dem Kommando ls mit, daß die Statusinformationen: Zugriffserlaubnis, Eigner, Größe in Bytes, Anzahl der Verbindungen (links) zu dieser Datei und Datum des letzten Dateizugriffs für jede Datei mit auszugeben sind. Ohne diese Option werden nur die Dateinamen ausgegeben (siehe Abschn. 5.). Für Kommandos existieren meist mehrere Optionen, die auch Kombination untereinander angegeben werden können. So gibt

```
ls -tls
```

eine zeitlich sortierte Liste (nach dem Datum des letzten Dateizugriffs) der Dateinamen des aktuellen Directories im langen Format aus. Die Datei-größe wird hierbei in Blöcken (512 Byte) angegeben (Option s).
In einer Zeile können mehrere Kommandos angegeben werden. Sie sind durch Semikolons zu trennen.

Um ein Kommando auszuführen, erzeugt Shell einen neuen Prozeß und wartet normalerweise auf das Ende der Abarbeitung. Ein Kommando kann aber auch im Hintergrund laufen. Dies wird durch Angabe des Zeichens "&" zum Schluß des Kommandos erreicht. So startet z.B. das Kommando

```
cp datei1 datei2 datei3 dirxyz&
```

eine Kopieroperation, die die angegebenen Dateien in das als letztes Argument angegebene Directory kopiert. Die Dateinamen bleiben unverändert.

Shell ist danach wieder zum Dialog bereit, ohne auf das Ende des Kopiervorgangs zu warten. Diese Arbeitsweise ist vor allem für umfangreiche zeitintensive Operationen empfehlenswert.
Um einem solchermaßen gestarteten Prozeß auf der Spur bleiben zu können, gibt Shell nach der Erzeugung seine Prozeßnummer aus. Eine Liste aller augenblicklich aktiven Prozesse kann über das Kommando ps erzeugt werden.

Ein-/Ausgabezuweisung

Die meisten Kommandos erzeugen einen Ausgabestrom auf dem Standardausgabegerät, das normalerweise mit dem Terminal verbunden ist. Diese Ausgabe kann aber auch als Datei abgelegt werden. Dann ist folgende Schreibweise zu verwenden:

```
ls -l > dateiname
```

Die Notation "> dateiname" wird durch Shell interpretiert und ist kein Argument, das an ls übergeben wird. Falls die angegebene Datei noch nicht existiert, so erzeugt Shell sie. Andernfalls wird der alte Inhalt der Datei überschrieben. Der Ausgabestrom kann aber auch an den bisherigen Inhalt angehängt werden. Dies wird durch folgende Schreibweise erreicht:

```
ls -l >> dateiname
```

Für Kommandos, die eine Eingabe vom Standardeingabegerät (normalerweise ebenfalls das Terminal) erwarten, kann diese Eingabe auch von einer Datei kommen. Dies wird durch folgende Notation erreicht:

```
wc < dateiname
```

Das Kommando liest von der Standardeingabe, in diesem Fall der angegeben Datei zugewiesen, und gibt die Anzahl der Zeilen, Wörter und Zeichen im Eingabestrom aus. Falls nur einer dieser Werte gewünscht wird, so muß dies über eine Option spezifiziert werden. So ermittelt z.B.

```
wc -l < dateiname
```

die Zeilenzahl der angegebenen Datei.

Die Symbole < und > repräsentieren normalerweise die Datenströme für die Standardeingabe und -ausgabe (Dateideskriptoren 0 und 1 siehe Abschn. 7.5.2.). Durch Voranstellen einer Ziffer kann eine Neuzuweisung des Datenstroms erreicht werden, z.B. wird durch

```
2 >errdat
```

der Ausgabestrom der Standardfehlerausgabe (Dateideskriptor 2) als Datei 'errdat' abgelegt. Durch die folgende Notation

```
>&n      bzw.      <&n          (n Dateideskriptor)
```

wird erreicht, daß der Dateideskriptor n dupliziert wird (Systemaufruf dup, siehe Abschn. 2.2.) und der Datenstrom dem Dateideskriptor n zuge-

ordnet wird. So werden z.B. durch

```
bsp.c >file 2>&1
```

die Standardausgabe und die Standardfehlerausgabe zu einem Datenstrom verschmolzen, der als Datei 'file' abgelegt wird. Durch die Angabe von

```
<&-        bzw.        >&-
```

werden die Standardeingabe bzw. -ausgabe abgeschlossen.

Pipes und Filter

Die Standardausgabe eines Kommandos kann direkt mit der Standardeingabe eines anderen Kommandos durch Angabe des sogenannten Pipeoperators ("|"), wie

```
ls -l |
```

verbunden werden. Die Kommandos werden auf diese Weise durch den Aufbau eines Pipekanals verbunden. Die Wirkung ist die gleiche wie in

```
ls -l > datei; wc < datei
```

mit der Ausnahme, daß keine Dateien benötigt werden. Statt dessen werden zwei parallel laufende Prozesse durch einen Systemaufruf pipe miteinander verbunden. Die Synchronisation erfolgt automatisch. Dabei wird der einschreibende Prozeß (in diesem Fall ls) angehalten, wenn der Pipekanal voll ist, und der auslesende Prozeß 'wc' wird gestoppt, wenn keine Zeichen mehr im Pipekanal vorhanden sind. Die Größe des Pipekanals beträgt meist 5000 Zeichen.

Filter wird ein Programm genannt, das einen Eingabestrom von der Standardeingabe liest, ihn auf irgendeine Weise verarbeitet (z.B.: alle Kleinbuchstaben in Großbuchstaben umwandelt) und das Ergebnis an die Standardausgabe ausgibt. Oft läßt sich eine Problemlösung durch das Erstellen von speziellen Filterprogrammen und durch geschickte Kombination mit existierenden Kommandos relativ schnell erreichen. Beispiele von einfachen Filterprogrammen werden im Abschn. 7. angegeben.
Ein Beispiel für ein Filterprogramm ist das UNIX-Kommando grep, das aus dem Eingabestrom die Zeilen heraussucht, die eine als Argument angegebene Zeichenkette enthalten. So ermittelt

```
ls | grep projekt1
```

nur die Ausgabezeilen von ls, die die Zeichenkette projekt1 enthalten. Ein anderes nützliches Filterprogramm ist sort, das verschiedene Sortieroperationen (je nach spezifizierter Option) ausführen kann. So gibt

```
who | sort
```

eine alphabetisch sortierte Liste aller augenblicklich zugeschalteten Nutzer aus.

Es ist ebenfalls möglich, mehrere Pipeoperatoren anzugeben. So gibt

```
ls | grep projekt1 | wc -l
```

die Anzahl der Dateinamen in dem aktuellen Directory aus, die die Zeichenkette 'projekt1' enthalten.

Dateinamensbildung

Für viele Kommandos sind Dateinamen Argumente. So gibt

```
ls -l bsp.c
```

Informationen über die Datei 'bsp.

Im Shell existiert die Möglichkeit, durch Angabe von speziellen Symbolen eine Liste von Dateinamen zu erzeugen. So generiert

```
ls -l
```

als Argumente für ls alle Dateinamen des aktuellen Directories, die mit ".c" enden. Das Zeichen "*" steht dabei für eine beliebige Zeichenkette, einschließlich der leeren Zeichenkette. Es kann an jeder Position stehen. Folgende spezielle Symbole sind für eine abkürzende Angabe von Dateinamen verwendbar:

*	steht für eine beliebige Zeichenkette (außer "/"), einschließlich der leeren Zeichenkette
?	steht für genau ein beliebiges Zeichen
[..]	steht für genau ein Zeichen aus den eingeschlossenen Zeichen, ein durch ein Minuszeichen getrenntes Paar von zwei Zeichen steht für alle Zeichen, die lexikalisch zwischen den Zeichen liegen

Dieser Mechanismus hilft dem Anwender, Eingaben zu minimieren. So steht

```
[a-z]*
```

für alle Dateinamen des aktuellen Directories, die mit den Kleinbuchstaben a bis z beginnen.

Beispiele für Dateinamen in Shell-Kommandos:

`rm *`	streicht alle Dateien des aktuellen Directories
`rm *kap1*`	streicht alle Dateien, die die Zeichenkette kap1 an beliebiger Stelle im Dateinamen enthalten
`ls ?`	listet alle Dateien aus, deren Name aus nur einem Buchstaben besteht
`ls projekt??`	listet alle Dateien aus, deren Name aus 9 Zeichen besteht und mit projekt beginnt
`ls -l kap[123459]*`	listet alle Dateien auf, deren Name mit kap1 bis kap5 und kap9 beginnt

`ls -l kap[1-59]*` andere Schreibweise dafür (dabei ist zu beachten, daß es sich nicht um die zweistellige Zahl 59 handelt!)

`echo /usr/hugo/*/test`
ermittelt alle Dateien test in Subdirectories von /usr/hugo und gibt die vollständigen Pfadnamen aus (die Ausführung dieses Kommandos kann längere Zeit beanspruchen, da alle Subdirectories durchsucht werden müssen!)

Es gibt eine Ausnahme bei der Angabe eines Dateinamensmusters. Ein Punkt (".") zum Anfang eines Dateinamens muß explizit angegeben werden. So gibt

```
echo
```

alle Dateinamen des aktuellen Directories aus, die nicht mit einem Punkt beginnen. Dagegen gibt

```
echo
```

alle Dateinamen aus, die mit einem Punkt beginnen. Dies vermeidet die Ausgabe der Dateinamen (directory) und ".." (parent directory). Das Kommando ls unterdrückt implizit Informationen über Dateien, die mit einem Punkt beginnen. Der Aufruf

```
ls
```

listet alle Dateien, einschließlich die mit einem Punkt beginnen,

Spezielle Symbole

Zeichen mit einer speziellen Bedeutung für Shell,

```
< >     |  &
```

werden Metazeichen genannt. Ihre spezielle Rolle verlieren sie, wenn ihnen das Zeichen "\" (backslash) vorangestellt wird. So gibt

```
echo \ ?
```

nur das Fragezeichen aus und

```
echo \\
```

gibt das einzelne Zeichen "\" aus.
Um Zeichenketten zu erlauben, die länger als eine Zeile sind, wird die Folge \Newline (Returntaste) von Shell ignoriert. Das ermöglicht, sehr lange Kommandofolgen vom Terminal aus einzugeben. Das so eingegebene Newline (Returntaste) ist dann lediglich ein Trennzeichen und veranlaßt noch keinen Kommandostart.
Das Backslashzeichen dient zur Kennzeichnung einzelner Metasymbole. Eine Zeichenkette von speziellen Symbolen kann durch Einschließen in Hochkommas ihre besondere Bedeutung verlieren. So gibt

```
echo xx'
```

die Zeichenkette

aus. Die so gekennzeichnete Zeichenkette darf Newlinezeichen (sie sind dann ebenfalls nur Trennzeichen), aber kein Apostroph enthalten.
Daneben existiert noch ein dritter Kennzeichnungsmechanismus, der Anführungsstriche benutzt. Dieser wird im Abschn. 6.3.1. beschrieben.

Promptzeichen und Login

Bei der interaktiven Arbeit gibt Shell vor dem Lesen eines Kommandos ein sog. Promptzeichen aus. Implizit ist dies ein Dollarzeichen ("$"). Es kann aber durch das Kommando PS1 verändert werden. So weist

```
PS1=Kommando:
```

dem Promptzeichen die Zeichenkette Kommando: zu.
Wenn ein Newline eingegeben wird, obwohl noch weitere Eingaben benötigt werden, so gibt Shell ein zweites Promptzeichen (implizit ">") aus. Dies kann auch durch falsche Verwendung der eben beschriebenen Kennzeichnung von speziellen Symbolen auftreten. Über einen Interrupt (Eingabe von CTRL-C) kann Shell wieder in den Kommandoeingabemode gebracht werden. Das zweite Promptzeichen kann über das Kommando PS2 verändert werden, z.B.

```
PS2=weiter:
```

Nach der Login-Prozedur wird Shell aufgerufen, um vom Terminal Kommandos zu lesen und auszuführen. Wenn das Login-Directory des Benutzers eine Datei mit dem Namen .profile enthält, dann wird vorausgesetzt, daß sie Kommandos enthält. Diese werden von Shell vor der Dialoganforderung ausgeführt, d.h., die in .profile abgelegte Kommandofolge wird bei jeder Login-Prozedur automatisch ausgeführt. Der Anwender kann so von ihm ständig benötigte Bedingungen oder Abläufe fest verankern.

6.3. Shell-Prozeduren

Shell-Prozeduren sind Programme, die in einer Sprache vorliegen, die von Shell verstanden wird. In den folgenden Punkten wird eine Beschreibung dieser Programmiersprache und ihrer Anwendung gegeben. Es ist keine vollständige Definition der Sprache. Anhand von Erläuterungen und Beispielen sollen dem Leser die vielfältigen Möglichkeiten verdeutlicht und eine Unterstützung für eigene Anwendungen gegeben werden.

Da Shell selber ein Programm ist, kann auch sein Eingabestrom von Datei aus erfolgen:

```
sh < datei
```

Shell liest Kommandos aus der angegebenen Datei, die als Shell-Prozedur

(oder Shell-Skript) bezeichnet wird. Bei dieser Form können keine Argumente übergeben werden. Shell benötigt aber das Symbol "<" nicht, es kann direkt

```
sh datei Argumente...
```

geschrieben werden, wobei dem Dateinamen optional Argumente folgen können. Auf die mit dem Aufruf übergebenen Argumente kann mit Hilfe von Positionsparametern (Bezeichnung: "$1", "$2",...) innerhalb der Kommandodatei zugegriffen werden.
Wenn z.B. die Shell-Prozedur 'dc' folgende Kommandos enthält:

```
cd $1
mkdir $2
cp *.c $2
```

so ist

```
sh dc projekt1 beispiele
```

äquivalent zu

```
cd projekt1
mkdir beispiele
cp *.c beispiele
```

Es werden alle Dateien, des Directories 'projekt1', die mit enden, das neu zu erstellende Directory 'beispiele' kopiert.

UNIX-Dateien besitzen drei unabhängige Attribute: read, write und execute. Das Kommando chmod kann dazu benutzt werden, eine Datei als ausführbar (execute) zu kennzeichnen. Durch

```
chmod +x dc
```

erhält die Datei 'dc' einen ausführbaren Status. Danach kann anstelle von

```
sh dc projekt1 beispiele
```

auch

```
dc projekt1 beispiele
```

angegeben werden. Dies gestattet es, Shell-Prozeduren wie Programme benutzen.

Zusätzlich zur Übermittlung der Argumente ist die Anzahl der Positionsparameter in der Shell-Prozedur über die Variable "$#" und der Name der gerade auszuführenden Datei über "$0" verfügbar. Die Variable "$*" kennzeichnet alle Positionsparameter, außer "$0". Wenn z.B. die Shell-Prozedur 'dc1' das Kommando

```
cp $* /usr/peter/bsp
```

enthält und mittels

```
dc1 bsp1.c bsp1 text script
```

aufgerufen wird, werden die angegebenen Dateien das Directory /usr/peter/bsp kopiert.

6.3.1. Variablen

Shell bietet die Möglichkeit der Verarbeitung von Zeichenkettenvariablen. Sie können z.B. für folgende Aktivitäten benutzt werden:

- Bestimmung der Anzahl und der Werte von Kommandozeilenargumenten
- Eingabe von Daten in Shell-Prozeduren
- Dateinamenserzeugung
- Abkürzung von Directory- und Pfadnamen
- Bestimmung, ob ein von Shell aufgerufenes Programm fehlerfrei abgearbeitet wurde
  Festlegung von Directories, in denen Shell nach Kommandos sucht
  Setzen von Informationen über die Programmierumgebung des Anwenders (z.B.: Terminaltyp .)

Namen und Wertzuweisung

Variablennamen müssen mit einem Buchstaben (einschließlich Unterstrich) beginnen und können aus Buchstaben, Ziffern und dem Unterstrich bestehen. Durch Anweisungen können Variablen Werte zugewiesen werden. Durch

```
nutzer=peter  text_1=hallo  dir=/usr/hugo/tmp
```

werden .B. den Variablen nutzer, text_1 und dir Werte zugewiesen. Auf den Wert einer Variablen kann zugegriffen werden, indem vor den Namen ein $ gesetzt wird. Variablen sind als Abkürzung für häufig benutzte Zeichenketten nützlich. So ist z.B.:

```
c1=/usr/peter/buch/kapitel1
cd $c1
```

äquivalent zu

```
cd /usr/peter/buch/kapitel1
```

Wenn bei der Variablensubstitution dem Namen Buchstaben oder Ziffern folgen, so muß der Variablenname in geschweifte Klammern eingeschlossen werden. Dadurch wird er von den folgenden Zeichen getrennt. Das folgende Beispiel verdeutlicht die Anwendung der geschweiften Klammern:

```
tmpa=/usr/tmp/tempdatei
tmp=/tmp/temp

cp bsp $tmpa          kopiert bsp in /usr/tmp/tempdatei
cp bsp ${tmp}a        kopiert bsp in /tmp/tempa
```

Variablen können einmal durch Verwendung des Gleichheitszeichens oder durch eine read-Anweisung Werte zugewiesen werden.

```
user=peter
dir=/usr/$user/buch/kapitel_1
null=                          (leere Zeichenkette)
text='diese Zeile enthaelt Sonderzeichen: *     ? newline
      und [ ]
text1="diese Zeichenkette enthaelt ein

read a b
                (Eingabe von:  peter /usr/peter/projektA),
echo $a         gibt peter aus,
echo $b         gibt /usr/peter/projektA aus, siehe Abschn. 6.3.2.)
```

Zeichenketten und Variablen können auch untereinander verkettet werden. Wenn die Variable PATH den Wert ":/bin:/usr/bin:" hat und die Zeichenkette "/usr/frank/bin" hinzugefügt werden soll, dann ist das über

```
PATH=/bin/frank/bin$PATH      oder
PATH=$PATH/bin/frank/bin
```

möglich.

Die Standardausgabe eines Kommandos kann ebenfalls Variablen zugewiesen werden. Dies wird durch Einschließen des Kommandos die Zeichen erreicht. Wenn das augenblickliche Directory z.B. /usr/name/projektA/teil1 ist, so weist

```
dir=`pwd`
```

der Variablen dir die Zeichenkette /usr/name/projektA/teil1 zu. Dieser Mechanismus gestattet es, Kommandos, die Zeichenketten verarbeiten, innerhalb von Shell-Prozeduren zu benutzen.

Weitere Beispiele:

```
for i    `ls -t`
do
(Die Variable i erhält den Dateinamen des aktuellen Directories)

set `date`
echo $6 $2 $3, $4
(Das Kommando set setzt die Positionsparameter ($i) auf die
 Zeichenketten des Ausgabestroms von date. Das Kommando echo
 gibt einen Teil dieser Zeichenketten in neuer Reihenfolge aus:
 1985 Aug 20, 10:50:25)
```

Implizite Variablenwerte

Einer Shell-Variablen können implizite Werte zugewiesen werden. Normalerweise ist der Wert einer nicht gesetzten Variablen die leere Zeichenkette.

Wenn der Variablen d kein Wert zugewiesen wurde, dann gibt

```
echo $d        oder
echo ${d}
```

nichts aus.
Ein impliziter Wert kann unter Verwendung der folgenden Notation mit angegeben werden.

```
echo ${d-ab}
```

In diesem Fall wird der Wert der Variablen 'd' ausgegeben, wenn sie gesetzt ist. Wurde der Variablen 'd' kein Wert zugewiesen, so wird die Zeichenkette "ab" ausgegeben. Die impliziten Werte werden unter Berücksichtigung der Kennzeichnung der speziellen Symbole ermittelt.
Die folgende Zeile

```
flag=${d-'*'}
```

weist der Variablen 'flag' die Zeichenkette 'd' nicht gesetzt wurde. Ähnlich kann durch

```
echo ${d-$1}
```

der Wert des ersten Positionsparameters ausgegeben werden, falls 'd' kein Wert zugewiesen wurde.
Einer Variablen kann durch folgende Notation ein impliziter Wert zugewiesen werden:

```
echo ${d=ab}
```

Diese Anweisung substituiert die gleiche Zeichenkette, wie in

```
echo ${d-ab}
```

und weist gleichzeitig der Variablen 'd', wenn sie nicht schon vorher gesetzt wurde, die Zeichenkette "ab" zu. Positionsparameter können über die Zuweisungsnotation ${..=..} nicht verändert werden.

Die Notation

```
tmp=${d?"Variable d hat keinen Wert!"}
```

bewirkt, daß der Variablen 'tmp' der Wert der Variablen 'd' zugewiesen wird, falls diese vorher gesetzt wurde. Andernfalls wird der Variablenname und die nach dem Fragezeichen angegebene Zeichenkette ausgegeben. Falls nach dem Fragezeichen keine Zeichenkette folgt, so wird die Standardnachricht

```
Variablenname: parameter not set
```

ausgegeben. Danach wird die Shell-Prozedur verlassen.

Schlüsselwortparameter

Shell-Variablen erhalten Werte durch Zuweisung oder durch den Eintritt in eine Shell-Prozedur. Ein Argument der Form Name=Wert, das dem Prozedurnamen vorangestellt ist, bewirkt eine Wertzuweisung, bevor die Shell-Prozedur gestartet wird. Der zugewiesene Wert wird durch die Shell-Prozedur nicht verändert. Die Zeile

```
user=hugo command dir1 dir2
```

führt die Shell-Prozedur command mit den Argumenten dir1 dir2 aus, wobei vorher die Variable 'user' den Wert 'hugo' erhält. Argumente der Form Name=Wert können auch an beliebiger Stelle der Argumentliste stehen, wenn davor das Flag -k angegeben wird (-k steht für Schlüsselwortparameter key). Andere verbleibende Argumente sind innerhalb einer Shell-Prozedur über die Positionsparameter $1, $2 usw. erreichbar.

```
command dir1 -k user=peter dir2
```

Das spezielle Shell-Kommando 'set' kann innerhalb einer Shell-Prozedur zum Setzen der Positionsparameter benutzt werden. Die nach dem Kommando folgenden Argumente werden entsprechend ihrer Reihenfolge $1, $2 usw. zugewiesen. So weist

```
set
```

$1 den ersten Dateinamen des aktuellen Directories zu, $2 den nächsten, usw.
Zusätzlich zum Setzen von Positions- und Schlüsselwortparametern beim Aufruf einer Shell-Prozedur können Shell-Variablen zu einer beliebigen Zeit (z.B. in der Datei .profile) exportiert werden. Sie sind dann für alle folgenden Prozeduren verfügbar. Das Kommando

```
export term
```

markiert die Variable term für den Export. Beim Aufruf einer Shell-Prozedur werden Kopien von allen exportierten Variablen für die Benutzung innerhalb der Prozedur angefertigt. Durchgeführte Modifikationen dieser Variablen innerhalb von Prozeduren ändern nicht den Wert der Variablen für Shell. Variablen, deren Wert sich auch innerhalb einer Prozedur nicht verändern darf, können als readonly vereinbart werden. Die Form ist dieselbe wie beim Kommando export.

```
readonly term
```

Nachfolgende Versuche, solche Variablen zu verändern, sind nicht gestattet.

Reservierte Variablen

Eine Reihe von Shell-Variablen werden bei jeder Login-Prozedur gesetzt. Diese Variablen spezifizieren bestimmte Shell-Funktionen. Ihre Namen sind reserviert.

$HOME Legt das implizite Argument für das Kommando cd fest (Home- oder Login-Directory). Das Kommando cd ohne Argumente entspricht cd $HOME.
Der implizite Wert dieser Variablen wird für jeden Nutzer durch das 6. Feld in der Datei /etc/passwd festgelegt. (Felder sind durch Doppelpunkte getrennt.)

$IFS Legt den Satz von Zeichen fest, die als Leer- oder Trennzeichen interpretiert werden. Normalerweise sind Leerzeichen, Tabulator und Newline (ASCII) dieser Variablen zugewiesen.

$MAIL Beim interaktiven Arbeiten überprüft Shell, bevor ein Promptzeichen ausgegeben wird, die durch diese Variable gekennzeichnete Datei. Wenn diese Datei seit der letzten Überprüfung modifiziert wurde, dann erfolgt die Ausgabe der Nachricht: you have mail.
Diese Variable wird normalerweise in der Datei .profile gesetzt, z.B.: MAIL=/usr/mail/peter,

$PATH Enthält eine Liste von Directorynamen, in denen Shell nach ausführbaren Kommandos sucht (Kommandosuchpfad). Enthält implizit die Zeichenkette :/bin:/usr/bin
Der Doppelpunkt trennt die Directorynamen. Falls die Zeichenkette mit einem Doppelpunkt beginnt, so wird mit der aktuellen Directory begonnen.

$PS1 Legt die Shell-Promptzeichen fest.
$PS2 Implizit ist $PS1 gleich $ und $PS2 gleich >. (siehe Abschn. 6.2.)

Die folgenden Shell-Variablen werden bei jeder Kommandoausführung durch Shell aktualisiert.

$? Kennzeichnet den Austrittsstatus (Returnkode) des zuletzt ausgeführten Kommandos als dezimale Zeichenkette. Die meisten Kommandos geben bei fehlerfreier Abarbeitung eine Null zurück. Die Auswertung dieses Wertes kann durch eine if-, until- oder while-Anweisung erfolgen.

$$ Ist die Prozeßnummer (dezimal) augenblicklich ausgeführten Shells. Da die Zuordnung der Prozeßnummern zu existierenden Prozessen eindeutig ist, kann diese Zeichenkette günstig zur Generierung von temporären Dateinamen verwendet werden, z.B.:
tempdatei=/tmp/name.$$

$! Ist die Prozeßnummer (dezimal) des zuletzt im Hintergrund gelaufenen Prozesses.

$- Die aktuellen Shell-Flags, wie -x und -v (siehe Abschn. 6.5.).

Im Zusammenhang mit der Übergabe von Kommandozeilenargumenten stehen folgende Variablen zur Verfügung:

$#	Enthält die Anzahl (dezimal) der Positionsparameter.
$0	Der Name des Programms (Kommando).
$1, $2,	Enthalten die dem Programm übergebenen Argumente (Positionsparameter).

Die Verwendung der o.g. Variablen wird in mehreren Beispielen dieses Abschnitts illustriert.

Die Parameter $* und $@ stehen für die Folge aller Positionsparameter (ab $1). Ein Unterschied besteht beim Einschließen in Anführungszeichen.

"$*"	Alle Argumente eines Programms als eine Zeichenkette. Ist äquivalent zu: "$1 $2 $n"
"$@"	Alle Kommandozeilenargumente, ohne Trennzeichen. Ist äquivalent zu: "$1" "$2" "$n"

Ein Beispiel dazu:

Vorausgesetzt die Shell-Prozedur 'delete' enthält die folgende Zeile

```
"$*"
```

Dann führt Shell bei der Eingabe

```
$delete p1.o p2.o
```

das Kommando

```
"p1.o p2.o"
```

aus. Es wird versucht die eine Datei "p1.o p2.o" streichen. Um Kommandozeilenargumente an ein weiteres Kommando weiterzureichen, ist die "$@"-Notation zu verwenden (oder $* ohne Anführungszeichen). Das Beispiel 'delete' zeigt, warum. Wenn 'delete' die Zeile

```
"$@"
```

enthält, dann erfolgt beim Aufruf von

```
$delete p1.o p2.o
```

das Streichen der zwei Dateien. Weiterhin ist es damit auch möglich, Dateien zu streichen, deren Name ein Leerzeichen enthält.

```
$delete p1.o p2.o "proj abc"
```

Durch diese Eingabe führt Shell das Kommando

```
rm p1.o p2.o "proj abc"
```

(Das ist mit der $*-Notation nicht möglich.)

Neben dem bereits beschriebenen Kennzeichnungsmechanismus für spezielle Shell-Symbole (Metazeichen) durch die Zeichen \ und gibt es noch eine dritte Möglichkeit. Diese verwendet die Anführungsstriche. Innerhalb von Anführungszeichen wird die Parameter- und Kommandosubstitution durchgeführt, es erfolgt aber keine Dateinamenserzeugung und Leer- bzw. Trennzeicheninterpretation.

```
echo "$PATH `ls`"
```

übergibt an echo den Wert der Variablen PATH und die Ausgabe des Kommandos ls als ein Kommandozeilenargument. Dagegen übergibt

```
echo
```

das Zeichen als Argument an das Kommando echo.

6.3.2. Anweisungen

Anweisungen steuern den Programmablauf innerhalb einer Shell-Prozedur. In Shell-Prozeduren besteht die Möglichkeit der Verwendung von bedingten Anweisungen (if und case), von Schleifenanweisungen (for, while und until), von Eingabeanweisungen (read) und von Kommentaren.

Kommentare

Der Doppelpunkt ist ein spezielles Shell-Kommando. Es kann für Kommentarzeilen innerhalb einer Shell-Prozedur benutzt werden.

```
: Name der Prozedur
: Was sie leistet
: Beschreibung und Handhabung ect.
```

Dabei ist zu beachten, daß nach dem Doppelpunkt ein Leerzeichen folgt und das jede Kommentarzeile mit einem Doppelpunkt beginnt.

if-Anweisung

Die allgemeine Form der Einfachverzweigung ist

```
if  Kommandoliste_1
then  Kommandoliste_2
else  Kommandoliste_3
fi
```

Der else-Zweig ist dabei optional. Ein Mehrfachtest der Form

```
if
then
else if
     then
     else if
          ...
          fi
     fi
fi
```

kann unter Ausnutzung der erweiterten if-Notation auch wie folgt geschrieben werden

```
if
then
elif
  then
  elif

fi
```

In allen Fällen wird das letzte Kommando der nach 'if' folgenden Kommandoliste zur Verzweigung ausgewertet. Ist sein Austrittsstatus gleich Null, so wird die nach 'then' folgende Kommandoliste ausgeführt. Andernfalls werden die nach 'else' folgenden Kommandos zur Abarbeitung gebracht (falls ein solcher Zweig überhaupt existiert). Das reservierte Wort 'fi' beendet die if-Anweisung.

Die spezielle Folge

```
if Kommandoliste_1
then Kommandoliste_2
fi
```

ist äquivalent zu folgender Schreibweise

```
Kommandoliste_1  &&  Kommandoliste_2
```

Umgedreht wird bei

```
Kommandoliste_1   ||  Kommandoliste_2
```

die zweite Kommandoliste nur ausgeführt, wenn der Austrittsstatus des letzten Kommandos der ersten Kommandoliste ungleich Null ist. (Entspricht den logischen Operatoren UND und ODER der Sprache C, siehe Abschn. 7.)

Kommandos true und false

Die Kommandos 'true' und 'false', selber nicht Bestandteil Shell, übergeben immer einen festen Wert.

```
true      "wahr" (Austrittsstatus gleich Null)
false     "falsch" (Austrittsstatus ungleich Null)
```

Da Shell selbst keine logischen Berechnungen ausführen kann, wird Shell-Prozeduren das UNIX-Kommando 'test' in Verbindung mit Verzweigungsanweisungen benutzt.

Kommando test

Das Kommando 'test', ebenfalls nicht Bestandteil von Shell, wird in Shell-Prozeduren zur Ermittlung von bestimmten logischen Bedingungen benutzt. Die zu testende logische Bedingung wird dem Kommando 'test' in Form von Argumenten übergeben. So können z.B. bestimmte Datei- und Zeichenkettenbedingungen getestet werden.

```
test -r Dateiname       wahr, wenn Datei existiert und lesbar ist
test -f Dateiname       wahr, wenn Datei existiert und kein
                        Directory ist
test string1=string2    wahr, wenn beide Zeichenketten gleich sind
u.v.a.m.
```

Eine komplette Beschreibung des Kommandos 'test' ist im Abschnitt 5. angegeben. Das folgende Beispiel demonstriert die Anwendung des Kommandos 'test' in Verbindung mit der if-Anweisung. Die Datei 'abc' wird ausgegeben, wenn sie existiert und lesbar ist. In anderen Fällen werden entsprechende Texte ausgegeben.

```
if test -r abc -a -f abc
then cat abc
elif test -f abc
  then echo "Die Datei abc ist nicht lesbar!"
  elif test -d abc
    then echo "Die Datei abc ist eine Directory!"
    else echo "Die Datei abc existiert nicht!"
fi
```

case-Anweisung

Die case-Anweisung gestattet eine Mehrfachverzweigung. Ihre allgemeine Form ist:

```
    Variable in
  Musterliste_1 ) Kommandoliste_1
  Musterliste_2 ) Kommandoliste_2

esac
```

Die case-Anweisung berechnet den Wert der nach case folgenden Variable und vergleicht ihn entsprechend der Reihenfolge mit den angegebenen Mustern. Die Muster bestehen aus einer Liste von Zeichen, wobei die Wirkung der Shell-Metazeichen zu berücksichtigen ist. Bei einer Übereinstimmung wird dann die entsprechende Kommandoliste abgearbeitet.
Eine Musterliste kann aus einem Muster oder mehreren durch das Zeichen | (ODER) zu trennenden Mustern bestehen. Sie wird durch eine schließende

runde Klammer abgeschlossen. Da das Shell-Metazeichen * für ein beliebiges Muster steht, kann die folgende Zeile

```
*) Kommandoliste
```

in einer case-Anweisung für einen Zweig verwendet werden, der abzuarbeiten ist, wenn keine andere Übereinstimmung festgestellt wird ("otherwise-Zweig") Es erfolgt keine Kontrolle, die absichert, daß nur ein Muster mit dem case-Argument übereinstimmt. Die erste gefundene Übereinstimmung definiert die abzuarbeitende Kommandofolge. Deshalb muß ein sog. otherwise-Zweig auch der letzte Zweig innerhalb einer case-Anweisung sein.

Ein Beispiel für die Anwendung einer Mehrfachverzweigung wird in der folgenden Shell-Prozedur (append) gegeben.

```
case $# in
  1) cat >> $1;;
  2) cat >> $2 < $1;;
  *) echo 'Syntax: append [from] to';;
esac
```

Bei Aufruf mit einem Argument wird der Standardeingabestrom an das Ende der als Argument angegebenen Datei angehängt. Bei zwei Argumenten wird der Inhalt der zweiten Datei an den Inhalt der ersten Datei angehängt. Falls mehr als zwei Argumente angegeben werden, dann erfolgt eine Ausgabe richtigen Gebrauch des Kommandos.
Ein anderes Beispiel für die Anwendung der case-Anweisung ist die Unterscheidung zwischen verschiedenen Formen von Argumenten.

```
case $i in
  -[ocs])
  -*) echo 'ungueltiges Flag $i'
  *.c) ...
  *) echo 'ungueltiges Argument $i'
```

Im ersten case-Zweig wird die Übereinstimmung mit den Flags , -c und -s getestet. Der zweite case-Zweig erkennt alle anderen Zeichenketten, die mit einem Minuszeichen beginnen. Im dritten case-Zweig werden alle Zeichenketten, die mit ".c" enden, erkannt. Der letzte case-Zweig erkennt alle restlichen Zeichenketten (otherwise-Zweig).
Das nächste Beispiel soll die Kennzeichnung von Metazeichen durch das Backslashzeichen (\) verdeutlichen.

```
case $i in
  \?) Kommandoliste_1;;
  ?|-?) Kommandoliste_2;;
  *) Kommandoliste_3;;
```

Die erste Kommandoliste wird abgearbeitet, wenn der Wert der Variablen das Fragezeichen ist. Die zweite Kommandozeile wird abgearbeitet, wenn die Variable 'i' aus genau einem Zeichen besteht (außer dem Fragezeichen) oder aus einem Minuszeichen gefolgt von einem beliebigen Zeichen besteht. In

jedem anderen Fall wird die dritte Kommandoliste abgearbeitet.

Innerhalb von Shell-Prozeduren werden häufig Schleifen in Abhängigkeit von den Argumenten ($1, $2, ...) aufgebaut, um Kommandofolgen für jedes Argument auszuführen. Zum Aufbau von Schleifen dienen die Wiederholungsanweisungen for, while und until.

for-Anweisung

Die allgemeine Form der for-Anweisung ist:

```
for Steuervariable in Wert_1 Wert_2
do
    Kommandoliste
done
```

Es werden der nach 'for' folgenden Steuervariablen die nach dem Schlüsselwort 'in' angegebenen Werte zugewiesen. Nach jeder Zuweisung werden die nach 'do' folgenden Kommandos abgearbeitet. Dieser Prozeß dauert so lange, bis alle Werte zugewiesen wurden.
Das reservierte Wort 'in' mit der nachfolgenden Liste Werten ist optional, d.h., dieser Teil kann weggelassen werden. In diesem Fall werden der Steuervariablen nacheinander alle Kommandozeilenargumente (entspricht: in "$@") zugewiesen. Folgende Regeln sind bei der Anwendung der for-Anweisung zu beachten:

Falls das reservierte Wort 'in' in der for-Anweisung erscheint, so muß dem Wort 'do' ein Newline oder ein Semikolon vorangestellt sein.
- Dem reservierten Wort 'done' muß ein Newline oder Semikolon vorangestellt sein.
- Der aktuelle Wert der Steuervariablen ist innerhalb des for-Körpers über die Notation $Steuervariable erreichbar.
- Die Kommandoliste ist eine Folge von einem Kommando oder mehreren einfachen Kommandos, die durch Newline oder Semikolon zu trennen und abzuschließen sind.

Ein Beispiel für die Anwendung von 'for' ist das create-Kommando, das folgenden Inhalt hat:

```
for i do >$i; done
```

Der Aufruf

```
create alpha beta
```

gewährleistet, daß die Dateien 'alpha' und 'beta' existieren und leer sind. (Die Notation ">Datei" bewirkt, daß die Datei erzeugt wird oder daß der Inhalt einer bereits existierenden Datei gelöscht wird.)
Das folgende Beispiel verändert für alle Dateien des aktuellen Directories die Zugriffserlaubnis.

```
for i in *
do
    chmod 744 $i
done
```

Eine andere dazu äquivalente Form ist:

```
for i in `ls`
do
    chmod 744 $i
done
```

Die for-Anweisung kann auch für iterative Prozesse, wie die n-malige Wiederholung von Kommandos, verwendet werden.

```
for i in 5 4 3 2 1
do
    write peter < Nachricht
    sleep 60
done
```

Dieses Beispiel gibt den Inhalt der Datei 'Nachricht' an den eingeschalteten Nutzer 'peter' aus. Dies erfolgt fünfmal im Abstand von einer Minute.

while-Anweisung

Die allgemeine Form der while-Anweisung ist:

```
while Kommandoliste_1
do
    Kommandoliste_2
done
```

Es wird der Austrittsstatus des letzten Kommandos der ersten Kommandoliste ermittelt. Falls dieser Null ist, so wird die zweite Kommandoliste abgearbeitet. Dieser Prozeß wird so lange wiederholt, bis der Austrittsstatus des letzten Kommandos der ersten Kommandoliste ungleich Null ist. Die Schleife wird dann beendet.
Im folgenden Beispiel wird die Anzahl der Schleifendurchläufe durch die Zahl der Kommandozeilenargumente bestimmt.

```
while test $# -gt 0
do
    ...
    shift
done
```

Das Kommando test $# -gt 0 entspricht dem Vergleich $# > 0. shift ist ein Shell-Kommando, das die Positionsparameter $2,$3,... in $1,$2,... umbenennt und $# um 1 dekrementiert.

until-Anweisung

Sie hat die allgemeine Form

```
until Kommandoliste_1
do
    Kommandoliste_2
done
```

Die until-Anweisung ermittelt zuerst den Austrittsstatus des letzten Kommandos der ersten Kommandoliste. Die zweite Kommandoliste wird so lange wiederholt, bis der Austrittsstatus dieses Kommandos ungleich Null ist. Dabei ist zu beachten, daß der Körper (zwischen do und done) der until-Anweisung nur abgearbeitet wird, wenn das letzte Kommando der ersten Kommandoliste einen Austrittsstatus ungleich Null hat! (Im Gegensatz zur repeat/until-Anweisung von PASCAL, wo er immer mindestens einmal ausgeführt wird.)
Eine Anwendung der until-Anweisung ist das Warten auf das Eintreten eines externen Ereignisses.

```
until test -f start
do
    sleep 300
done
```

In diesem Beispiel werden die nach done folgenden Kommandos erst abgearbeitet, wenn die Datei 'start' existiert. Der Test, ob diese Datei existiert, erfolgt alle fünf Minuten (sleep 300).

read-Anweisung

Mit Hilfe der read-Anweisung können in Shell-Prozeduren Eingaben angefordert werden. Die read-Anweisung liest eine Zeile von der Standardeingabe und weist den eingegebenen Text einer oder mehreren Variablen zu.

Das folgende Beispiel verdeutlicht die Arbeitsweise 'read'.

```
read a
          Eingabe:    diese Zeile wurde eingegeben(CR)
echo $a
          Ausgabe:    diese Zeile wurde eingegeben

read a b c
          Eingabe:    dies ist eine zu lesende Zeile(CR)
echo $a
          Ausgabe:    dies
echo $b
          Ausgabe:    ist
echo $c
          Ausgabe:    eine zu lesende Zeile
```

Eine Schleifenanweisung kann im Zusammenwirken mit einer case-Anweisung

dazu benutzt werden, eine bestimmte Eingabe abzuwarten. Der folgende Programmteil wartet z.B. auf die Eingabe von j oder n (groß oder klein).

```
while
     echo -n "Bitte geben Sie j oder n ein:
     read x
     case $x in
         [jJnN]*) false;;
         *) true;;

do
     echo dies ist eine falsche Eingabe!
done
```

Ein anderes Beispiel ist die Zuweisung von Zeilen einer Datei Shell-Variablen.

```
cat Datei |
  while read a
  do
     Kommandoliste
  done
```

Jede Zeile der Datei steht der folgenden Kommandoliste als Wert der Variablen 'a' zur Verfügung. Die Schleife wird beendet, nachdem die letzte Zeile der Datei zugewiesen wurde.
Zur Ausgabe dient das allgemeine echo-Kommando. In den Beispielen wurde die Anwendung von echo illustriert. Shell-Metazeichen können mit echo ausgegeben werden, wenn die verschiedenen Kennzeichnungsmechanismen benutzt werden.

```
echo 'dies sind ein Stern (*) und ein Fragezeichen (?)'
```

Interne Daten

Shell-Proceduren können Daten (Texte) durch Eingaben oder Dateien erhalten. Daneben existiert noch die Möglichkeit, sie als festen Bestandteil der Prozedur zu integrieren. Solche Daten sind dann zwischen << ! und ! einzuschließen (das Zeichen nach << kann auch ein anderes sein). Abgeschlossen werden die Daten durch eine Zeile, die nur das nach << folgende Zeichen enthält. Im nachfolgenden Beispiel werden die Zeilen zwischen << ! und ! als Standardeingabe für das Kommando grep genommen.

```
for i
do grep $i << !
     ...
     peter wrx0123
     heinz wrx1670

   !
done
```

Bevor die Daten dem Kommando zur Verfügung gestellt werden, erfolgt falls erforderlich eine Parametersubstitution. Dies wird durch die folgende Prozedur 'edg' illustriert.

```
ed $3 << %
    g/$1/s//$2/g
    w
    %
```

Der Aufruf

```
edg Zeichenkette_1 Zeichenkette_2 Datei
```

ersetzt dann jede Zeichenkette_1 in der angegebenen Datei durch die Zeichenkette_2 (siehe dazu Beschreibung des Editors ed).
Eine Parametersubstitution kann verhindert werden, wenn das Zeichen \ zur Kennzeichnung von speziellen Symbolen verwendet wird. So ist

```
ed $3 <<+
    1,\$s/$1/$2/g
    w
    +
```

äquivalent zum obigen Beispiel (diesmal aber mit einer anderen Editor-Kommandofolge). Eine Parametersubstitution in den internen Daten einer Shell-Prozedur kann vollständig verhindert werden, wenn das erste nach << folgende Zeichen durch \ gekennzeichnet wird.

```
grep $i << \#

#
```

Kommandogruppierung

Shell-Kommandos können in zwei Arten gruppiert werden:

```
{ Kommandoliste }
```

und

```
( Kommandoliste )
```

Bei der ersten Form wird die Kommandoliste einfach abgearbeitet. Die zweite Form veranlaßt die Abarbeitung der Kommandofolge als separaten Prozeß. So führt z.B.:

```
(cd xy; rm abc)
```

das Kommando rm abc in dem Directory xy aus, ohne das aktuelle Directory zu verändern. Die Kommandos

```
cd xy; rm abc
```

haben denselben Effekt, danach ist das aktuelle Directory aber 'xy'.

Verschachtelte Shell-Prozeduren

In Shell-Prozeduren existiert die Möglichkeit, weitere Shell-Prozeduren (auch rekursiv) aufzurufen. Den aufgerufenen Prozeduren können wiederum Argumente übergeben werden.
Wenn die Shell-Prozedur 'ausgabe' (im Directory /usr/hugo) die Kommandos

```
cd $1
echo Directory $1
for i in `ls`
do
   if test -d $i
   then /usr/hugo/ausgabe $i
   else echo $i
   fi
done
```

enthält, so veranlaßt der Aufruf

```
ausgabe dir1
```

die Ausgabe aller Dateinamen des bei 'dir1' beginnenden Dateibaums.

Kommandozusammenfassung

Die folgende Übersicht enthält eine Zusammenfassung der internen Kommandos von Shell. Sie sind alphabetisch geordnet und ihre Wirkung wird kurz erläutert. Optionale Teile werden bei der Angabe der Kommandosyntax in eckige Klammern eingeschlossen.

break [n]	Austritt aus einer for-, while- oder until-Schleife. Falls n angegeben wird, so erfolgt Verlassen der n-ten Schleife.
case Variablenname **in** [Muster[|Muster]...) Kommandoliste;;]... **esac**	Shell-Mehrfachverzweigung (siehe Abschn. 6.3.2., Seite 142)
cd [neues_Directory]	Wechselt das aktuelle Arbeitsdirectory (siehe Abschn. 5.)
continue [n]	Veranlaßt Sprung zum Beginn einer for-, while- oder until-Schleife. Falls n angegeben wird, so erfolgt Neustart der n-ten Schleife.
esac	Beendet die case-Anweisung.
eval [Shell-Kommandos]	Führt die angegebenen Kommandos innerhalb des aktuellen Shells aus.

exec [Kommando]

Führt das angegebene Kommando anstelle des aktuellen Shells

exit [n] Verlassen des aktuellen Shells. Die Austrittsvariable ($?) wird auf den Wert n gesetzt, falls er spezifiziert wurde, oder auf den Austrittsstatus des letzten ausgeführten Kommandos.

export [Variablenliste]

Kennzeichnet die angegebenen Shell-Variablen als globale Variablen. Falls keine Variablenliste spezifiziert wird, so erfolgt die Ausgabe aller exportierten Variablen.

for Variablenname [**in** Wertliste;] **do** Kommandoliste; **done**

Shell-for-Anweisung (siehe Abschn. 6.3.2., Seite 144)

login [Benutzername]

Das aktuelle Shell wird ersetzt durch das login-Shell für den angegebenen Nutzernamen.

newgrp Gruppenname

Änderung der Gruppenzugehörigkeit (siehe Abschn. 5.).

read [Variablenliste]

Einlesen einer Zeile von der Standardeingabe und Zuweisen der Wörter zu den angegebenen Variablen.

readonly [Variablenliste]

Kennzeichnet die angegebenen Shell-Variablen als nicht veränderbar. Falls keine Variablenliste spezifiziert wird, so erfolgt die Ausgabe aller readonly-Variablen.

set [**-ceiknpstuvx** [Positionsparameter]]

Setzen von speziellen Optionen oder der Positionsparameter für das aktuelle Shell. Falls keine Positionsparameter oder Optionen angegeben werden, so erfolgt die Ausgabe der Werte von gesetzten Shell-Variablen.

kurze Definition der Shell-Parameter (Optionen):

-c Zeichenkette

Kommandos werden von der Zeichenkette gelesen.

Bewirkt das Verlassen von Shell, falls ein Kommando einen Austrittsstatus ungleich Null hat.

-i	Kennzeichnet Shell als interaktiv arbeitend. In diesem Fall wird das Signal 'terminate' (kill 0) ignoriert, und das Signal 'interrupt' (Eingabe CTRL-C) wird eingefangen und ignoriert.
-k	Kennzeichnung für Schlüsselwortparameter. Erlaubt die Plazierung von Variablenwertzuweisungen beliebiger Stelle der Argumentliste.
	Bewirkt nur das Einlesen von Kommandos. Die Ausführung wird erst nach Eingabe von CTRL-D veranlaßt.
	Kommandos werden von der Standardeingabe geles Die Ausgaben von Shell erfolgen an den Dateideskriptor 2 (Standard-Fehlerausgabe).
-t	Liest ein Kommando und führt es aus. Danach wird Shell verlassen.
	Setzt die Austrittsstatusvariable ($?) auf 1, falls auf eine nicht gesetzte Shell-Variable zugegriffen wird.
	Veranlaßt die Ausgabe von Shell-Eingabezeilen nach ihrem Einlesen.
-x	Veranlaßt die Ausgabe von Kommandos und deren aktueller Parameter.
	Schaltet die -v und -x Optionen aus.

sh [**-ceiknpstuvx**] [Programmname] [Programmparameter]	
	Führt das angegebene Programm mit seinen Parametern in einem (evtl. durch Optionen spezifizierten) Subshell aus. (Die Optionen sind analog zum set-Kommando.)
shift	Umbenennung der Positionsparameter $2, $3,... in $1, $2,... und Dekrementieren von $# um 1.
times	Ausgabe der akkumulierten Benutzer- und Systemzeiten für von Shell gestartete Prozesse.
trap [Kommandoliste] [Signal_1] [Signal_2]...	
	Shell-trap-Kommando (siehe Abschn. 6.4.1., Seite 153)
umask [n]	Legt den impliziten Dateierstellungsmodus (read/write/execute) in oktaler Darstellung fest. Falls kein Wert angegeben wird, erfolgt die Ausgabe des augenblicklichen Modes. Dabei ist zu beachten, daß die Reihenfolge umgedreht zu der von 'chmod' (siehe Abschn. 5.) ist.

until Steuerliste; **do** Kommandoliste; **done** Shell-until-Anweisung (siehe Abschn. 6.3.2., Seite 146)
wait [Prozeßnummern] Warten auf das Ende eines oder mehrerer Prozesse. Falls keine Prozeßnummern angegeben werden, erfolgt die Ausgabe aller aktuellen Tochterprozesse, auf deren Ende gewartet wird.
while Steuerliste; **do** Kommandoliste; **done** Shell-while-Anweisung (siehe Abschn. 6.3.2., Seite 145)

6.4. Fehler- und Signalbehandlung

Die Art der Fehlerbehandlung hängt von der Fehlerart und von der Arbeitsweise von Shell (interaktiv oder nicht) ab. Die Kommandoausführung kann aus einem der folgenden Gründe mißlingen:

- Eine Ein-/Ausgabezuweisung gelingt nicht (Dateien existieren nicht oder können nicht erzeugt werden).
- Das Kommando selber existiert nicht, oder kann nicht ausgeführt werden.
  Das Kommando wird nicht ordnungsgemäß beendet (Auftreten Fehlers).
- Das Kommando wird normal beendet, liefert aber einen Rückgabewert ungleich Null.

Beim Auftreten der eben genannten Fehler arbeitet Shell weiter und veranlaßt die Ausführung des nächsten Kommandos. Mit Ausnahme des letzten Falls wird eine Fehlernachricht generiert.
Die folgenden Fehlerarten führen zum Abbruch einer Kommandoprozedur:

- Syntaxfehler (z.B.: if...then...done)
- Das Auftreten eines Signals (z.B. Interrupt). Nach Ausführung des augenblicklichen Kommandos wird die Kommandoprozedur verlassen oder zum Terminaldialog (bei interaktiver Arbeit) zurückgekehrt.
  Fehlerhafte Verwendung eines internen Shell-Kommandos.

Ein interaktiv arbeitendes Shell kehrt in den Eingabemodus zurück.
Das Shell-Flag -e (es kann beim Aufruf von Shell oder durch das Kommando set gesetzt werden) veranlaßt, daß ein nicht interaktiv arbeitendes Shell bei Auftreten eines jeden Fehlers abgebrochen wird.

Zur Unterstützung des Tests von Kommandoprozeduren bietet Shell zwei Mechanismen. Der erste kann durch die Anweisung

```
set -v    (v steht für verbose, im Sinne von zusätzlichen
          Informationen)
```

innerhalb einer Kommandoprozedur oder durch Angabe in der Aufrufzeile

```
sh -v Prozedurname
```

eingeschaltet werden. Er veranlaßt die Ausgabe der Kommandozeilen, wenn sie gelesen werden. Dadurch wird das Finden von einzelnen syntaktischen Fehlern unterstützt. Zusätzlich kann durch das n-Flag

```
set -n
```

die Ausführung von gelesenen Kommandos verhindert werden. Sie werden dann erst nach Eingabe von CTRL-D (EOF) ausgeführt.
Durch Setzen des x-Flags

```
set          oder      sh -x Prozedurname
```

wird die Ausgabe der Kommandos mit ihren aktuellen Parametern (nach Substitution) bei der Ausführung veranlaßt. Der aktuelle Stand der Shell-Flags ist über die Variable $- erreichbar. Zum Beispiel durch

```
echo $-
```

Signale und Traps

Signale kennzeichen externe Ereignisse (Hardwarefehler, bestimmte Eingaben u.a.). UNIX-Signale können auf unterschiedliche Weise behandelt werden:

Normalerweise führen sie zur Beendigung des Prozesses, ohne daß noch weitere Aktivitäten ablaufen.
- Sie können ignoriert werden, dann erfolgt kein Abbruch des Prozesses.
  Sie können eingefangen werden. In diesem Fall muß der Prozeß entscheiden, welche Aktionen beim Auftreten der Signale ausgeführt werden sollen.

Weitere detailliertere Informationen sind in der UNIX-Dokumentation unter signal(2) und kill(1) zu finden. Hier eine Auswahl der für Shell relevanten Signale.

Signalnummer	Name
1	hangup
2	interrupt
3	quit
9	unstoppable kill
14	alarm clock
15	software termination (von kill(1))

Normalerweise wird bei Auftreten eines Signals zum Terminaldialog zurückgekehrt. Mit Hilfe der internen Shell-Anweisung trap können in Kommandoprozeduren bestimmte Aktivitäten bei Eintreten von Signalen veranlaßt werden. Die trap-Anweisung ist für drei Situationen anwendbar.

(1) trap " Kommandoliste " Signalnr. Signalnr.
(2) trap "" Signalnr. Signalnr.
(3) trap Signalnr. Signalnr.

In der ersten Form legt die Kommandoliste die Aktionen fest, die bei Eintreten der angegebenen Signalnummern (1 bis 15) ausgeführt werden.
Wenn die Kommandoliste die leere Zeichenkette ist, so wie in der zweiten Form, dann bedeutet dies, daß die angegebenen Signale ignoriert werden.
In der dritten Form werden die vorher durch eine trap-Anweisung gesetzten Signale wieder in ihren Initialzustand gebracht.
Somit kann eine Unterbrechung von Kommandoprozeduren durch Auftreten von Signalen (z.B.: 2 - interrupt: Eingabe von CTRL-C; 1 - hangup: Ausschalten des Terminals oder Eingabe von CTRL-D) verhindert werden.

```
trap    1 2
```

Oft arbeiten Kommandoprozeduren mit temporären Dateien. In diesen Fällen kann über eine trap-Anweisung veranlaßt werden, daß vor Abbruch noch alle temporären Dateien gelöscht werden.

```
trap     -f /tmp/ps$$; exit" 1 2 3
```

Das Kommando exit ist ein weiteres internes Kommando, das die Ausführung einer Shell-Prozedur beendet. Es wird benötigt, da andernfalls nach der trap-Behandlung an der Stelle fortgesetzt wird, wo die Prozedur unterbrochen wurde.

Zum Abschluß noch ein Beispiel. Es aktualisiert die Zeit des letzten Dateizugriffs für eine als Kommandozeilenargumente zu übergebende Liste von Dateien.

```
flag=
trap     -f junk$$; exit" 1 2 3 15
for i
do
  case $i in
     -c) flag=N
      *) if test -f $i
         then cp $i junk$$; rm junk$$
         elif test $flag
           then > $i
           else echo "Datei $i existiert nicht!"
         fi

done
```

Der Variablen 'flag' wird zuerst die leere Zeichenkette zugewiesen. Sie kennzeichnet in der Shell-Prozedur, ob eine nicht existierende Datei erzeugt werden soll.
Die trap-Anweisung sichert, daß bei Eintreffen der angegebenen Signale die temporäre Datei 'junk$$' vor Verlassen der Shell-Prozedur (exit) gestrichen wird. In der folgenden for-Anweisung werden nacheinander alle Kom-

mandozeilenargumente der Variablen 'i' zugewiesen. Die case-Anweisung setzt bei Erkennen des Flags -c die Variable 'flag'. Alle anderen Argumente werden als Dateinamen interpretiert.
Falls die Datei existiert, wird durch das Kopieren des Dateiinhalts die Zeit des letzten Dateizugriffs aktualisiert. Durch Angabe des Flags -c wird erreicht, daß die Datei erzeugt wird (>$i), wenn sie noch nicht existiert. Sonst erfolgt eine Fehlermeldung.

7. Programmiersprache C

Mit dem Betriebssystem UNIX gewann die Programmiersprache C wesentlich an Bedeutung. Sie füllt eine Lücke zwischen höheren Sprachen wie PASCAL, COBOL und BASIC und den Assemblersprachen. C ist für ein weites Anwenderspektrum einsetzbar und bietet günstige Voraussetzungen zur Realisierung von portablen Programmen. Derzeit sind für alle populären Betriebssysteme eine Reihe von C-Compilern verfügbar. In diesem Abschnitt wird in relativ kompakter Form eine Übersicht über die Sprachelemente von C gegeben. Daneben wird eine Auswahl von Standardbibliotheksfunktionen und die Handhabung des C-Compilers unter UNIX beschrieben.

7.1. Charakteristik und Bedeutung

Die Programmiersprache C entstand aus dem Bedarf nach einer Sprache, in der das 1970 in Assembler geschriebene Betriebssystem UNIX neu geschrieben werden sollte. Betriebssysteme müssen schnell, kompakt und maschinennah sein (also Assemblersprache). Allerdings müssen sie auch zuverlässig, portabel (d.h. auf einen anderen Rechner übertragbar), lesbar und wartungsfreundlich sein (also höhere Programmiersprache).
Die heute benutzte C-Sprache verbindet die beiden eben genannten Aspekte. Sie wurde von B.W. Kernighan und D.M. Ritchie entworfen und realisiert. Sie ist das Produkt einer mehrjährigen Entwicklung. Die meisten wichtigen Ideen stammen von BCPL, die von M. Richards entwickelt wurde. Der Einfluß von BCPL vollzog sich indirekt durch die Sprache B, die von K. Thompson im Jahr 1970 für das erste UNIX-System erarbeitet wurde. C wurde ursprünglich für PDP-11-Rechner unter UNIX entwickelt. Sie enthält aber keine Abhängigkeit von spezieller Hardware oder von einem Betriebssystem (/5/).
C ist eine allgemein anwendbare Programmiersprache, die nicht auf ein spezielles Anwendungsgebiet zugeschnitten ist. Obwohl man sie als "Systemprogrammiersprache" bezeichnet, weil sie sich zur Implementation von Betriebssystemen gut eignet, wird sie ebenso für numerische Berechnungen, zur Textverarbeitung und für die Arbeit mit Datenbanken verwendet.
C ist eine relativ maschinennahe Programmiersprache. Entworfen, um Programme portabel, schnell und kompakt zu machen, füllt C eine Lück zwischen höheren Sprachen wie BASIC, COBOL und PASCAL und dem Sprachniveau von Assemblersprachen. Deshalb wird C auch oft als "mittelhohe" (medium level) Programmiersprache bezeichnet.
Die Programmiersprache C hat durch ihre Effizienz die Assemblerprogrammierung unter UNIX vollständig verdrängt. (Selbst Entwickler von UNIX kannten die Assemblersprache der PDP-11-Rechner nicht!) Sie stellt quasi einen Assemblerersatz dar.
International benutzen die meisten großen Mikrocomputerhersteller und Softwarefirmen C, um Systemprogramme, Betriebssysteme, Sprachübersetzer und Anwenderprogrammpakete zu erstellen. Beispiele dafür sind: Digital Research schreibt alle neuen Produkte in C (z.B. dBase III, WordStar 2000, CP/M-68K). Microsoft und Visicorp setzen C sehr breit ein (z.B. für Multiplan, XENIX, Visiword und VisiOn). Daneben wird eine Vielzahl von Grafiksoftware in C erstellt (/25/).
Warum ist C nun so populär? Ein Grund liegt darin, daß C-Programme relativ leicht von einem Rechner oder Betriebssystem auf andere übertragen werden können (Portabilität). C ist vom 8-Bit-Mikroprozessor über 16-Bit-Personal-Computer und Kleinrechner bis zum Großrechner verfügbar. Sie wird

deshalb auch oft als "portable Assemblersprache" bezeichnet. C gestattet eine maschinennahe Programmierung, z.B. besteht die Möglichkeit, häufig benutzte Variablen in Registern zu halten. Dadurch wird die Effektivität des erzeugten Kodes verbessert. C enthält daneben aber auch eine Reihe von Elementen der strukturierten Programmierung, wie sie z.B. in Sprachen wie PASCAL zu finden sind. C ist eine kurze knappe Sprache, die einen kleinen Kern von reservierten Wörtern und Sprachelementen besitzt. Deshalb kann sie auch relativ schnell erlernt werden. Die Sprachdefinition von C enthält keine Funktionen für die Ein- und Ausgabe. Diese sind in einer Bibliothek von Standardfunktionen untergebracht. Eine getrennte Übersetzbarkeit von Programmodulen ist bei C möglich. C bietet für den Programmierer ein geeignetes Werkzeug zur Implementation von größeren Programmsystemen der unterschiedlichsten Anwendungsfälle.

Einige negative Aspekte von C sollen nicht unerwähnt bleiben. C stellt erhöhte Anforderungen an den Programmierer, die sehr weit gehenden Freiheiten der Sprache mit Vorsicht und Erfahrung zu verwenden. Die sehr kurzschriftartige Schreibweise (resultierend aus dem Ziel, maximal viele Informationen auf einer Bildschirmseite unterzubringen) kann zu schwer lesbaren Programmen führen.

Zusammenfassend können drei Gründe für die Popularität C genannt werden:

(1) Der erste Grund liegt in der Effizienz.
C füllt eine Lücke zwischen höheren Sprachen und dem Assemblersprachniveau. C-Programme sind kompakt und schnell. Bei der Anwendung von C entstehen gegenüber der Assemblerprogrammierung keine wesentlichen Effektivitätsnachteile. C-Programme bieten durch ihre Struktur aber bessere Voraussetzungen zur Programmpflege und -wartung.

(2) Der zweite Grund liegt in der Portabilität.
Aus dem Konzept der Maschinenunabhängigkeit von C ergibt sich eine gute Übertragbarkeit von C-Compilern und damit C-Programmen auf verschiedene Rechner und unterschiedliche Betriebssysteme. So sind heute C-Compiler vom 8-Bit-Prozessor bis zum Großrechner verfügbar.

(3) Der dritte Grund liegt im Bezug zum Betriebssystem UNIX.
C als "Muttersprache" von UNIX wird ganz automatisch eine weite Verbreitung und Bedeutung erlangen, da UNIX und davon abgeleitete Betriebssysteme international eine ganz wesentliche Rolle spielen.

Der derzeitige Erfolg von UNIX (98% in C implementiert) und einer Vielzahl von in C geschriebener Anwendersoftware läßt keinen Zweifel an der Effizienz und Zuverlässigkeit der Programmiersprache C aufkommen.

Die nachfolgende Sprachbeschreibung beginnt mit einem einführenden Beispiel, das die Struktur von C-Programmen und wesentliche Sprachelemente von C verdeutlicht. Die Beschäftigung mit diesem Beispiel soll den Einstieg in die Erläuterung der Sprache erleichtern und zum Experimentieren mit C anregen. Denn der einzige Weg, eine Programmiersprache zu erlernen, ist, in ihr Programme zu schreiben.

Für die Sprachbeschreibung wird vorausgesetzt, daß der Leser mit den grundlegenden Elementen der Programmierung vertraut ist.

7.2. Einführendes Beispiel

Als einführendes Beispiel wurde ein kleines Vertragsbearbeitungsprogramm ausgewählt. Mit ihm ist es möglich, Vertragsdaten einzugeben und Listen von Verträgen zusammenzustellen, deren Übergabetermine in einem bestimmten Quartal liegen. Dabei erfolgt zusätzlich eine Aufsummierung der Preise. Ein Vertragseintrag besteht aus folgenden Daten:

- Vertragsnummer
- Objektbezeichnung (Zeichenkette aus max. 20 Zeichen)
- Übergabetermin (Monat und Jahr)
- Preis

Die Auswahl der bestehenden Möglichkeiten erfolgt über ein kleines Menü zu Beginn des Programms. Das Bild 7.1 zeigt das C-Programm einschließlich Zeilennumerierung.

```
 1  /*
 2   * einfuehrendes Beispiel
 3   */
 4
 5  #include <stdio.h>
 6  #define ANZV 50 /* max. Anzahl von Vertraegen */
 7
 8  struct termin {
 9              int mon;    /* Monat: 1-12  */
10              int jahr;   /* Jahr:  0-99 */
11          };
12
13  struct vertrag {
14              int nr;                 /* Objektnummer */
15              char bez[20];           /* Objektbezeichnung */
16              struct termin ueb;      /* Uebergabe (Monat/Jahr) */
17              long preis;             /* Preis der Vertragsleistung */
18          } ltv[ANZV];
19
20  liste(quartal, jahr)
21  int quartal, jahr;
22  {
23      struct vertrag *vp;
24      int ug, og;
25      long gp; gp=0;
26      printf("\nVertraege mit Uebergabetermin im %d. Quartal 19%2d", quartal, jahr);
27      printf("\n  Nr.\tBezeichnung\t\tMonat\t\tPreis\n");
28      og=quartal*3; ug=og-2;
29      for (vp=ltv; vp->nr != 0; vp++) {
30          if (vp->ueb.mon <= og && vp->ueb.mon >= ug) {
31              printf("\n%5d  %-20s", vp->nr, vp->bez);
32              printf("\t%2d\t%10ld,- M", vp->ueb.mon, vp->preis);
33              gp += vp->preis;
34          }
35      }
36      printf("\n\n\t\t\t  Gesamtwert:   %10ld,- M\n", gp);
37  }
```

Bild 7.1. Programm 'efbsp.

```
38
39  eingabe()
40  {
41      struct vertrag *vp;
42      for (vp=ltv; vp->nr!=0; vp++);
43      printf("\n...Objektnr., Bezeichnung, ");
44      printf("Uebergabetermin (Monat Jahr) und Preis eingeben!\n");
45      while(scanf("%d%s%d%d%ld",&vp->nr,vp->bez,&vp->ueb.mon,&vp->ueb.jahr,&vp->preis)!=EOF)
46        vp++;
47  }
48
49  main()
50  {
51      int c, q, j;
52      printf("\n ---- VERTRAGSPROGRAMM ----\n");
53      do {
54          printf("\n Auswahlmoeglichkeiten:\n");
55          printf(" Moechten Sie ...\n");
56          printf("\t... Vertraege eingeben\t\t(Eingabe von 1)\n");
57          printf("\t... nach Quartal sortieren\t(Eingabe von 2)\n");
58          printf("\t... ins OS zurueckkehren\t(Eingabe von 0)\n");
59          scanf("%d", &c);
60          switch (c) {
61            case 0: break;
62            case 1: eingabe(); break;
63            case 2: printf("\n\t...fuer welches Quartal und Jahr? ");
64                    scanf("%d %d", &q, &j);
65                    liste(q,j); break;
66            default: continue;
67          }
68      } while (c);
69  }
70
```

Bild 7.1. (Fortsetzung)

Erläuterungen zum Programm:

- Der Quelltext beginnt mit einem Kommentar (Zeile 1 bis 3). Kommentare in C werden durch die Zeichenpaare /* und */ begrenzt.

- Durch die der Zeile 5 stehende Anweisung werden die in der Datei 'stdio.h' enthaltenen Vereinbarungen (z.B. die in Zeile 45 verwendete Konstante 'EOF') zur Ein-/Augabearbeit ins Programm integriert.

- Die Zeile 6 definiert eine Konstante (ANZV) mit dem Wert 50. Danach erfolgt die Vereinbarung von zwei Strukturtypen. Einmal 'termin' mit den Elementen 'mon' und 'jahr' (beides Integergrößen) und zum anderen 'vertrag'. Dieser Strukturtyp besteht aus den Elementen: 'nr' (Integergröße), einem Zeichenfeld 'bez[20]', einer Struktur vom Typ 'termin' (d.h. Monat und Jahr) und einem Preis (Typ 'long').

- In der Zeile 18 wird ein Feld aus 50 Elementen des Typs 'vertrag' mit dem Namen 'ltv' angelegt.

- Nach diesen globalen Vereinbarungen folgen drei Funktionen ('liste', 'eingabe' und 'main'). Die Funktion 'main' ist der Eintrittspunkt des Programms. Der Funktionskörper wird in C durch geschweifte Klammern

begrenzt. Die Funktion 'main' hat keine formalen Parameter. Sie arbeitet mit drei lokalen Integervariablen (Vereinbarung in Zeile 51). Über die Standardausgabefunktion 'printf' erfolgt die Ausgabe von Texten. Die Zeichen '\n' und '\t' sind eine spezielle Notation für das Newline- und Tabulatorzeichen.

In der Zeile 59 wird durch Aufruf der Standardeingabefunktion 'scanf' eine Eingabe erwartet. Der eingegebene Wert wird als Dezimalzahl interpretiert (Format: "%d") und der Variablen zugewiesen. Die Notation &c bedeutet "Adresse der Variablen c".

Im Anschluß an die Eingabe erfolgt eine Mehrfachverzweigung in Abhängigkeit vom Wert der Variablen 'c'. Im Fall 0 wird die case-Anweisung sofort verlassen (break-Anweisung). Im Fall 1 wird die Funktion 'eingabe' aufgerufen und erst dann zum Ende der case-Anweisung (Zeile 67) gesprungen. Im Fall 2 wird die Eingabe von Quartal und Jahr angefordert (Werte werden durch 'scanf' den Variablen 'q' und 'j' zugewiesen) und dann die Funktion 'liste' mit diesen Werten als aktuellen Parametern aufgerufen. In allen anderen Fällen (default-Zweig) wird durch die continue-Anweisung zum Beginn der das Menü umgebenden do-Schleife (Zeile 53) gesprungen. Diese Schleife wird so lange abgearbeitet, bis der Wert von ' Null ist (Zeile 68).

- Die Funktion 'eingabe' enthält im Funktionskörper als erstes eine Vereinbarung eines Zeigers 'vp' (struct vertrag *vp;), der auf Strukturen vom Typ 'vertrag' verweist. Dieser Zeiger wird durch den ersten Teil der nachfolgenden for-Anweisung auf den Beginn des Feldes 'ltv' (d.h. auf den ersten Vertragseintrag) gesetzt. Der zweite Teil der for-Anweisung ist die Testbedingung (solange die Nummer eines Vertragseintrages ungleich Null ist, so lange führe die nach 'for' folgenden Anweisungen in diesem Fall die leere Anweisung - aus). Der dritte Teil ist die Iterationsanweisung, die im Anschluß an den Körper der for-Anweisung ausgeführt wird. Durch die Notation 'vp++' wird der Zeiger inkrementiert, d.h., er zeigt auf den nächsten Vertragseintrag im Feld 'ltv'

- Die while-Anweisung in Zeile 45 ruft die Eingabefunktion 'scanf' auf, um einen Vertragseintrag zu füllen. Die Zuweisungen erfolgen über den aktuellen Wert des Zeigers 'vp' (&vp->ueb.mon bedeutet: Adresse des Monatselements, des Eintrages, auf den 'vp' zeigt). Nach jedem Füllen eines Vertragseintrages wird der Zeiger 'vp' inkrementiert, d.h., er zeigt auf den nächsten Eintrag. Die Eingabe von CTRL-D (EOF) beendet die Eingabe von Vertragsdaten.

- Die Funktion 'liste' besitzt zwei formale Parameter ('quartal' und 'jahr'). Die Vereinbarung der Typen erfolgt vor dem Funktionskörper in Zeile 21. Es wird wieder mit einem Zeiger auf Vertragseinträge gearbeitet. Zusätzlich werden die Variablen 'ug' und 'og' als interne Monatsgrenzen und 'gp' als Gesamtpreis verwendet. Die aktuellen Parameterwerte werden durch die printf-Anweisung (Zeile 26) in die Überschrift der Tabelle integriert (Format %d bzw. %2d, d.h. als Dezimalzahl bzw. als zweistellige Dezimalzahl). Durch die folgende for-Anweisung (Zeile 29 bis 35) werden alle Vertragseinträge der Reihe nach durchgemustert. Für jeden Eintrag wird auf Übereinstimmung des Jahres getestet und ob der Übergabemonat im entsprechenden Quartal liegt. Der Operator '&&' bedeu-

tet ein logisches UND. Im Fall der Übereinstimmung werden die Vertragsdaten formatiert ausgegeben. Die Vertragsnummer als fünfstellige Dezimalzahl (%5d), die Objektbezeichnung als Zeichenkette aus 20 Zeichen (%-20s). Das Minuszeichen in der Formatangabe bewirkt eine Linksausrichtung des Textes im Ausgabefeld.

- Zusätzlich wird der Gesamtpreis um den aktuellen Vertragspreis erhöht (Zeile 33). Dabei ist die Notation 'gp += vp->preis' gleichbedeutend mit der bekannten Schreibweise 'gp gp + vp->preis'.

- Nach Abschluß des Durchsuchens aller Vertragseinträge wird noch der Gesamtwert aller aufgelisteten Verträge ausgegeben.

Das Bild 7.2 zeigt das Menü und Beispiele von zwei erzeugten Ausgaben.

```
---- VERTRAGSPROGRAMM ----

Auswahlmoeglichkeiten:
Moechten Sie ...
          Vertraege eingeben          (Eingabe von 1)
          nach Quartal sortieren      (Eingabe von 2)
          ins OS zurueckkehren        (Eingabe von 0)

Vertraege mit Uebergabetermin im 3. Quartal 1986
  Nr.   Bezeichnung              Monat          Preis

 3542  Kammerspiele               8          23500,- M
 2864  Friedrichstadtpalast       9         300250,- M
 4724  Komische-Oper              7           2550,- M
 3145  Volksbuehne                8          64200,- M

                          Gesamtwert:        390500,- M

Vertraege mit Uebergabetermin im 4. Quartal 1986
  Nr.   Bezeichnung              Monat          Preis

 1487  Schauspielhaus            11          57600,- M
 3476  PdR/TiP                   12           1250,- M
 6421  Deutsches-Theater         10          43210,- M

                          Gesamtwert:        102060,- M
```

Bild 7.2. Ausgaben des Programms 'efbsp.c'

7.3. Sprachbeschreibung

Dem Charakter dieses Handbuches entsprechend, soll C hier in relativ kurzer Form dargestellt werden. Die eingefügten Beispiele sollen zum eigenen Experimentieren anregen, denn nur so erlangt man die nötige Sicherheit im Umgang mit der Programmiersprache C. In diesem Zusammenhang sei auf das Standardwerk zu C ([5] bzw. [6]) und mehrere Lehrbücher ([19] [20] [21] und [22]) verwiesen.

7.3.1. Lexikalische Betrachtungen

Für C existieren sechs Klassen von Wörtern: Trennzeichen, Namen, vierte Wörter, Konstanten, Zeichenketten und Operatoren.

Trennzeichen

Leerzeichen, Tabulatorzeichen, Newlinezeichen (CR) und Kommentare (oft als "white spaces" bezeichnet) trennen benachbarte Wörter. Sie werden von C ignoriert. Kommentare müssen in die Zeichenpaare "/*" und "*/" eingeschlossen werden (z.B.: /* dies ist ein Kommentar */). Sie dürfen nicht geschachtelt sein.

Namen

Namen bezeichnen Variablen, Funktionen u.a. Sie bestehen aus einer Folge von Buchstaben (einschließlich dem Unterstrich "_") und Ziffern, wobei das erste Zeichen ein Buchstabe sein muß. Groß- und Kleinbuchstaben werden (von den meisten Compilern) unterschieden. Im allgemeinen sind nur die ersten acht Zeichen eines Namens signifikant. Externe Namen, die von verschiedenen Assemblern und Ladern benutzt werden, sind meist stärker eingeschränkt.

Reservierte Wörter

Eigene, vom Anwender ausgewählte Namen dürfen nicht mit den reservierten Wörtern von C übereinstimmen. Sie sind in der folgenden Übersicht zusammengefaßt.

auto	else	int	switch
break	entry	long	typedef
case	enum	register	union
char	extern	return	unsigned
continue	float	short	while
default	for	sizeof	
do	goto	static	
double	if	struct	

Einige Implementationen reservieren auch die Wörter 'asm', 'fortran' und 'void'.

Konstanten

Es werden numerische Konstanten und Zeichenkonstanten unterschieden. Bei den numerischen Konstanten wird zwischen ganzzahligen Konstanten und Gleitkommakonstanten unterschieden.

Eine ganzzahlige Konstante besteht aus einer Ziffernfolge. Diese wird als Dezimalwert interpretiert, falls sie nicht mit einer führenden Null beginnt. Eine Ziffernfolge, die mit einer führenden Null beginnt, wird als Oktalzahl interpretiert. Konstanten werden als Hexadezimalzahlen aufgefaßt, wenn einer Folge hexadezimaler Ziffern (0...9, A...F oder ...f) die Zeichen "0x" oder "0X" vorangestellt sind.
Eine ganzzahlige Konstante (dezimal, oktal oder hexadezimal) kann durch die Endung "l" oder "L" als Konstante vom Typ 'long' (siehe Abschn. 7.3.2.) gekennzeichnet werden.
Für Gleitkommakonstanten ist die Dezimalpunkt- oder E-Notation anzuwenden.

Beispiele für numerische Konstanten:

```
15              dezimal
15l             dezimal long
017             oktal
0x0f            hexadezimal
15.0            Gleitkomma
1.5000e+1       Gleitkomma
```

Eine Zeichenkonstante besteht aus einem einzelnen in Apostrophe eingeschlossenen Zeichen (z.B.: 'y'). Daneben existiert für einige Sonderzeichen eine spezielle Schreibweise.

		ASCII
\0	NUL	(0x00)
\b	Backspace	(0x08)
\t	Tabulator	(0x09)
\n	Linefeed	(0x0A)
\f	Formfeed	(0x0C)
\r	CR	(0x0D)
\'	Hochkomma	(0x27)
\\	Backslash	(0x5C)

Außerdem ist es möglich, ein beliebiges Zeichen durch Angabe des oktalen Wertes anzugeben. Es ist dann folgende Schreibweise anzuwenden:

```
\ddd            (ddd steht für drei Oktalziffern)
```

Zeichenkettenkonstanten

Eine Zeichenkettenkonstante besteht aus einer in Anführungszeichen eingeschlossenen Folge von Zeichen (z.B. "dies ist eine Zeichenkette"). Der Compiler hängt an das Ende der Zeichenkette das Zeichen '\0' an. Dadurch kann ein Programm jederzeit das Ende einer Zeichenkette erkennen.

Innerhalb einer Zeichenkette kann die bei den Zeichenkonstanten beschriebene spezielle Schreibweise für Sonderzeichen verwendet werden (z.B.: "\nNr.\tName\t\tArtikel\n").

Die Zeichenkonstante 'y' besteht nur aus dem Zeichen y. Die Zeichenkettenkonstante "y" dagegen wird aus zwei Zeichen gebildet, dem y und der abschließenden Null (\0).

7.3.2. Datentypen und Vereinbarungen

In C ist die Interpretation eines Namens von zwei Attributen des Namens abhängig: der Speicherklasse und dem Typ. Die Speicherklasse legt den Ort und die Lebensdauer des mit dem Namen verbundenen Speicherbereichs fest. Der Typ bestimmt die Interpretation der Werte, die sich in diesem Speicherbereich befinden.

Durch eine Vereinbarung werden für einen oder mehrere Namen die Attribute Typ und Speicherklasse festgelegt. Die allgemeine Form einer Vereinbarung ist

```
[Speicherklasse] Datentyp Name,...
```

Die Angabe der Speicherklasse ist optional. Es können vier Speicherklassen vereinbart werden: 'automatic' 'static', 'extern' und 'register'. Variablen der Speicherklasse 'automatic' sind lokal zu einem Block (siehe Abschn. 7.3.9. "Programmstruktur und Speicherklassen und Bild 7.3). Nach Verlassen des Blocks sind sie nicht mehr zugänglich. Variablen, die als 'static' vereinbart wurden, sind auch lokal zu einem Block, sie behalten aber ihren Wert nach Verlassen des Blocks. Alle als 'extern' vereinbarten Variablen behalten ihre Werte während der Ausführung des gesamten Programms. Sie dienen zur Verbindung von separat übersetzten Programmodulen (siehe Abschn. 7.3.9.). Die Vereinbarung 'register' ist ein Hinweis an den Compiler, die so vereinbarten Variablen in CPU-Registern zu halten (soweit das möglich ist). Dies ist für häufig benutzte Variablen empfehlenswert. Solche Variablen sind ebenfalls lokal zu einem Block, und ihr Wert geht bei Verlassen des Blocks verloren.

C stellt einige wenige Grunddatentypen zur Verfügung. Zur Speicherung von Zeichen aus dem Zeichenvorrat der Implementierung findet der Typ

```
char
```

Verwendung. Für ganzzahlige Werte (integer) existieren drei Typen

```
short int
int
long int
```

Integergrößen können auch den Typ 'unsigned' erhalten. Sie werden dann als natürliche Zahlen (ohne Vorzeichen) interpretiert. Dabei gelten die üblichen Regeln der Arithmetik Modulo 2 hoch n, wobei n die Zahl der Bits der internen Rechnerdarstellung ist.

```
unsigned int
unsigned short int
unsigned long int
```

Das Wort 'int' kann bei Kombination mit 'short', 'long' oder 'unsigned' auch weggelassen werden.
Die interne Darstellung ist rechnerabhängig. Die Anzahl der von einer Variablen eines bestimmten Typs belegten Bytes kann mit Hilfe des sizeof-Operators (siehe Abschn. 7.3.3.) ermittelt werden. Es gilt

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
```

z.B. 1 Byte 2 Byte 2 Byte 4 Byte

Für Gleitkommawerte existieren zwei Typen:

float	einfache Genauigkeit (z.B. 4 Byte)
double	doppelte Genauigkeit (z.B. 8 Byte)

Neben diesen Grundtypen existieren noch daraus abgeleitete Typen:

Funktion	liefert Objekte eines bestimmten Typs
Feld	enthält Objekte der meisten Typen
Zeiger	verweist auf Objekte eines bestimmten Typs
Struktur	stellt eine Zusammenfassung mehrerer Objekte unterschiedlicher Typen dar
Union	bezeichnet einen Speicherplatz, der Objekte unterschiedlicher Typen enthalten kann
Aufzählung	deren Wertliste wird explizit definiert

Der Aufzählungstyp von C (enum) ist ähnlich dem Skalartyp von PASCAL. Im folgenden Beispiel wird ein Typ 'farbe' vereinbart. Variablen dieses Typs können die in den geschweiften Klammern angegebenen Werte annehmen.

```
enum farbe {rot, blau, gelb, braun}
```

Variablen eines solchen Typs können durch

```
enum farbe linie_f, inhalt_f;
```

vereinbart werden. Dabei ist 'farbe' der Aufzählungstypname und die Variablennamen sind 'linie_f' und 'inhalt_f'. Typ- und Variablenvereinbarung können auch zusammengefaßt werden.

```
enum farbe {rot, blau, gelb, braun}
     linie_f, inhalt_f;
```

Da der Aufzählungstyp nicht grundsätzlich zum Sprachumfang von C gehört, wird auf ihn im folgenden nicht weiter eingegangen.

Der Anwender hat die Möglichkeit, neben den von C vorgegebenen Typnamen auch eigene Typnamen festzulegen. Die Vereinbarung

```
typedef int LAENGE;
```

ermöglicht es, das Wort 'LAENGE' als Synonym für 'int' in weiteren Typvereinbarungen oder Ausdrücken zu verwenden. Dadurch werden keine neuen Typen eingeführt, es werden nur für schon existierende Typen neue Namen vergeben. Dadurch ist die Möglichkeit gegeben, die Programmdokumentation zu unterstützen (vor allem bei komplizierten strukturierten Typen), denn es können dem Anwenderproblem angepaßte Typnamen vergeben werden.

Bei der Berechnung von Ausdrücken (arithm. Operatoren, Zuweisungen, Funktionsaufrufen) werden von C implizite Typkonvertierungen durchgeführt. Der Programmierer "merkt" davon nichts, sollte die Regeln aber kennen.

In einem Ausdruck werden alle Objekte vom Typ 'char' in den Typ 'int' konvertiert.

Bei der Berechnung von arithmetischen Ausdrücken werden folgende Typkonvertierungen ausgeführt:

Operanden vom Typ 'char' oder 'short' werden in int-Werte umgewandelt. Das Ergebnis ist vom Typ 'int'.

Operanden vom Typ 'float' werden in double-Werte umgewandelt. Das Erbebnis ist vom Typ 'double'. (Gleitkommaoperationen in C finden mit doppelter Genauigkeit statt.)

Wenn ein Operand vom Typ 'double' ist, so wird der andere auch in 'double' konvertiert. Das Resultat ist vom Typ 'double'. Analoges gilt, wenn einer der Operanden vom Typ 'long' oder 'unsigned' ist.

Falls keiner dieser Fälle vorliegt, so sind beide Operanden vom Typ 'int' und das Ergebnis ist ebenfalls ein int-Wert.

Bei Vergleichsausdrücken und logischen Ausdrücken erfolgt eine automatische Typumwandlung in der Form, daß sie den Wert 1 erhalten, wenn der Ausdruck wahr ist, und 0 bei falsch.

Bei Funktionsaufrufen werden die Typen 'char' und 'short' zu 'int' umgewandelt. Der Typ 'float' wird zu 'double' und Feldnamen werden zu Zeigern konvertiert. Es ist zu beachten, daß bei der Parameterübergabe keine Typprüfung erfolgt! Deshalb muß oft zur Typanpassung der aktuellen Parameter der nachfolgend erläuterte cast-Operator angewendet werden.

Neben diesen automatischen Typkonvertierungen in Ausdrücken kann eine Typumwandlung auch erzwungen werden. Dazu dient der sogenannte cast-Operator (siehe Abschn. 7.3.3.). Der gewünschte Typ wird dabei in runde Klammern eingeschlossen und vor den Ausdruck gesetzt.

z.B.: `(double)x`

Die Variable x wird dadurch in den Typ 'double' konvertiert. Der cast-Operator ist insbesondere bei der Nutzung von Standard-Bibliotheksfunktionen zur Anpassung der Argumenttypen oder des Ergebnistyps von Bedeutung.

Die internen Typumwandlungen, die bei Zuweisungen implizit durchgeführt werden, sind im Abschn. 7.3.4. aufgeführt.

Alle Variablen müssen vor ihrem Gebrauch vereinbart werden. Durch die Vereinbarung wird ein Typ für eine Liste von Variablen spezifiziert. Optional kann davor noch eine Speicherklasse angegebenen werden.

Beispiele für Variablenvereinbarungen:

```
int  anzahl, max,
char c, text[64];
float delta;
```

die erste Vereinbarungszeile ist äquivalent zu

```
int anzahl;     /* Anzahl der Iterationen */
int max;
int nr;
```

Diese Form ist zur Kommentierung der Variablenverwendung und für spätere Modifikationen besser geeignet.

Variablen können bei ihrer Vereinbarung auch initialisiert werden (auf einige Restriktionen wird später eingegegangen). Man spricht dann von einer Definition. Wenn dem Namen ein Gleichheitszeichen und eine Konstante folgen, so wird diese Konstante als Initialwert verwendet, wie in

```
char       'y';
int  anzahl  100;
double  delta = 2.0e-6;
```

Sowohl bei der Vereinbarung als auch bei der Definition ist auf das abschließende Semikolon zu achten!
Nichtinitialisierte Variablen in den Speicherklassen 'static' und 'extern' werden mit 0 initialisiert. Automatische Variablen, für die keine Initialisierung durchgeführt wird, haben undefinierte Werte. Bei der Vorstellung der abgeleiteten Datentypen wird auf deren Initialisierung weiter eingegangen.

7.3.3. Operatoren und Ausdrücke

In diesem Abschnitt werden die Operatoren, die zur Bildung von Ausdrücken Verwendung finden können, erläutert.
Bei der Angabe der einfachen Ausdrücke und der unären Operatoren haben die sog. lvalues (left-values) eine besondere Bedeutung. Ein lvalue ist ein Ausdruck, der auf ein Objekt verweist. Das einfachste Beispiel für einen lvalue ist ein Variablenname. Elemente von Feldern, Strukturen, Unions und Bitfelder sind auch lvalues. Zeigervariablen und ganze Strukturvariablen gehören ebenfalls zur Klasse der lvalues, ganze Felder dagegen nicht.
Es gibt Operatoren, die einen lvalue liefern. Wenn A ein Ausdruck ist, der einen Zeigerwert liefert, dann ist *A ("indirekt", Inhalt des Feldes, auf das A verweist) ein lvalue, der das Objekt bezeichnet, auf das A zeigt.

In der folgenden Übersicht der Operatoren von C wird jeweils angegeben, ob ein Operator einen lvalue als Operanden erwartet und ob er einen lvalue als Resultat liefert.

Ausdrücke in C sind einfache Ausdrücke oder durch Operatoren verknüpfte Ausdrücke. Jeder Ausdruck besitzt einen Wert.
Einfache Ausdrücke können sein:

Namen	
Konstanten	
Zeichenketten	
(Ausdruck)	Verändern der Rangordnung von Operatoren
Name(Argumentliste)	Funktionsaufruf mit optionaler Parameterliste (Ausdrücke durch Kommata getrennt)
Name[Ausdruck]	Feldkomponenten
lvalue.Name	Verweise auf Teilstrukturen
Zeiger->Name	(siehe Abschn. 7.3.7.)

Die bei den einfachen Ausdrücken verwendeten Operatoren

() [] -> (Minuszeichen gefolgt von >)

werden auch als primäre Operatoren bezeichnet. Sie haben gegenüber allen anderen Operatoren den höchsten Vorrang.

Zur Verknüpfung von Ausdrücken bietet C sehr vielfältige Operatoren. Sie sind in der folgenden Tabelle zusammengefaßt. Beispiele und Hinw zu einzelnen Operatoren werden im Anschluß an die Tabelle angegeben.

Unäre Operatoren:		
Arithm. Operatoren:		Umkehrung des Vorzeichens (ein unäres + existiert nicht)
	--lvalue	Verminderung des Wertes des durch den lvalue gekennzeichneten Objekts um 1 vor der Verwendung des Wertes
	lvalue--	dto. nach Verwendung des Wertes
	++lvalue	Erhöhung des Wertes des durch den lvalue gekennzeichneten Objekts um 1 vor der Verwendung des Wertes
	lvalue++	dto. nach Verwendung des Wertes
Logischer Operator:		logische Negation (Umkehrung des Wahrheitswertes
Bitoperator:	~	Bildung des bitweisen Komplements
cast-Operator:	(typ)	(typ)A ist die Darstellung des Wertes des Ausdrucks A im Typ 'typ'

Adreßoperatoren:	*	*P ist der Inhalt des Feldes, auf das der Zeiger P zeigt
	&	&lvalue ist die Adresse des durch den lvalue gekennzeichneten Objekts
	(siehe Abschn. 7.3.6.)	
Binäre Operatoren:		
Arithm. Operatoren:		Multiplikation
	/	Division
	%	Berechnung des Rests bei Division ganzer Zahlen (Modulo)
	+	Addition
	-	Subtraktion
Vergleichsoperatoren:		
	>	größer als
	<	kleiner als
	>=	größer als oder gleich
	<=	kleiner als oder gleich
		gleich
		ungleich
Bitoperatoren:	<<	Verschieben aller Bits nach links
	>>	Verschieben aller Bits nach rechts
	&	Bitweise UND-Verknüpfung
	|	Bitweise inklusive ODER-Verknüpfung
	↑	Bitweise exklusive ODER-Verknüpfung
Zuweisungsoperatoren:		
		einfache Wertzuweisung
	op=	a op= b entspricht a = a op (b) folgende Operatoren sind für op möglich: +, -, *, /, %, >>, <<, &, | und ↑
Logische Operatoren:		
	&&	logisches UND zweier Wahrheitswerte
	||	logisches ODER zweier Wahrheitswerte
Bedingungsoperator:		A?B:C hat den Wert des Ausdrucks B, falls der Ausdruck A einen von Null verschiedenen Wert hat, sonst den Wert des Ausdrucks C
Kommaoperator:		A,B hat als Wert den Wert des Ausdrucks B, der nach Auswertung des Ausdrucks A ermittelt wird

Arithmetische und Zuweisungsoperatoren

Der wohl am häufigsten verwendete Operator ist der Zuweisungsoperator (=), mit dessen Hilfe Variablen Werte zugewiesen werden können (z.B. weist y=5; der numerischen Variable y den Wert 5 zu). Ein Zuweisungsausdruck besitzt einen Wert (den der linken Seite). Mehrfachzuweisungen in einer Zeile (z.B.: a=b=c=7;) sind möglich, da ein Ausdruck der Form 'c=7' einen Wert besitzt.

Die binären arithmetischen Operatoren +, * und / sind in ihrer Wirkung eindeutig. Zum Divisionsoperator (/) sei angemerkt, daß das Ergebnis einer Division ganzer Zahlen wieder eine ganze Zahl ist! So wird durch

```
5/3;
```

der Variablen a der Wert 1 zugewiesen, auch wenn sie als Gleitkommavariable (float oder double) vereinbart wurde!
Der Modulo-Operator (%) ist nur auf ganze Zahlen anwendbar. Er dient Berechnung des bei der Division entstehenden Rests. So weist

```
5%3;
```

der Variablen a den Wert 2 zu.
Hervorzuheben ist die bei C durch die Inkrement- und Dekrement-Operatoren ++ und -- mögliche kompakte Schreibweise. Je nach Stellung bewirken sie eine Veränderung des Wertes der betreffenden Variable vor bzw. nach seiner Verwendung. So ist

```
b--;
```

gleichbedeutend mit

```
a = b;
b   b-1;
```

während

```
  = --b;
```

identisch mit

```
b   b-1;
    b;
```

ist. Ebenso typisch für C-Programme sind Ausdrücke der Form

```
A op= B
```

Diese Form ist gleichbedeutend mit

```
A = A op (B)
```

Es ist auf die Klammer um den zweiten Ausdruck zu achten. So ist

```
b+1;
```

identisch mit

```
a * (b+1);
```

und nicht mit

```
a = a * b + 1;
```

Vor allem bei komplizierten Ausdrücken sind diese Schreibweisen von Vorteil.

Vergleichsoperatoren

Vergleichsoperatoren testen eine Vergleichsaussage und liefern als Ergebnis einen Wahrheitswert (0 für falsch, 1 für wahr). In Abweichung zu anderen Sprachen stehen zwei Gleichheitszeichen == für Gleich und die zwei Zeichen != für Ungleich.

Bitoperatoren

C unterstützt einige Operatoren zur Bitmanipulation. Diese sind nicht für die Typen 'float' und 'double' erlaubt. Die Verschiebeoperatoren << und >> ermöglichen eine Links- bzw. Rechtsverschiebung des linken Operanden um die Anzahl von Bitpositionen, die durch den rechten Operanden gegeben wird. Leer werdende Bitpositionen werden mit Nullen aufgefüllt. (Eine Ausnahme kann bei einigen Compilern das Rechtsverschieben einer vorzeichenbehafteten Größe bilden.)

Logische Bitoperatoren:

~x	1	1	1	1	0	0	0	0
y	1	0	1	0	1	0	1	0
x	0	0	0	0	1	1	1	1
x&y	1	0	1	0	0	0	0	0
x|y	1	1	1	1	1	0	1	0
x↑y	0	1	0	1	1	0	1	0

Wenn 'a' eine Variable des Typs 'char' ist und den Wert 00110001 besitzt, so repräsentiert

```
~a          11001110
a&0x0f      00000001
a|0x0f      00111111
a↑0x0f      00111110
a>>1        00011000
a<<2        11000100
```

Zu den ODER-Verknüpfungen sei angemerkt, daß die exklusive ODER-Verknüp-

fung (↑) im Gegensatz zur inklusiven ODER-Verknüpfung (|) zwei 1er Bits zum Wert 0 verknüpft. Der UND-Operator (&) wird oft zur Maskierung mit einer Bitfolge benutzt, und der ODER-Operator (|) kann zum Setzen von Bits angewendet werden.

Logische Operatoren

Logische Operatoren werden auf Wahrheitswerte angewendet und haben einen Wahrheitswert als Ergebnis. Dabei wird jeder von Null verschiedene Wert als 1 (wahr) interpretiert. Ausdrücke die durch && und || verbunden sind, werden von links nach rechts abgearbeitet. Dabei ist zu beachten, daß die Auswertung eines solchen Ausdrucks beendet wird, sobald der logische Wert des Ergebnisses bekannt ist! Diese Eigenschaft muß beim Schreiben von Programmen beachtet werden.

Bedingungsoperator

Der Bedingungsoperator bildet eine elegante Alternative zur Darstellung eines Ausdrucks, der von einer Bedingung abhängt. Im Ausdruck

```
A1 ? A2   A3
```

wird zuerst der Ausdruck A1 berechnet. Wenn er ungleich Null ist (d.h. wahr), dann wird der Ausdruck A2 berechnet, und der ermittelte Wert ist der Gesamtwert. Sonst wird der Ausdruck A3 berechnet und als Gesamtwert genommen. Es wird nur einer der beiden Ausdrücke A2 oder A3 berechnet. So kann z.B. die Ermittlung des Maximums zweier Zahlen a und b auch folgendermaßen programmiert werden:

```
(a>b) ? a   b;
```

Kommaoperator

Durch den Kommaoperator werden zwei Ausdrücke miteinander verknüpft.

```
A1, A2
```

Der Ausdruck A1 wird ausgewertet, hat aber keinen Einfluß auf den Wert des Gesamtausdrucks. Dieser wird allein durch den zweiten Ausdruck bestimmt. Verwendet wird der Kommaoperator dann, wenn A2 von einem durch Auswertung von A1 veränderten Wert abhängt, syntaktisch aber ein einzelner Ausdruck verlangt wird. Er findet weiterhin Anwendung in Schleifen, bei denen zwei Indizes parallel verarbeitet werden sollen.

Vorrang und Assoziativität

Die nachfolgende Tabelle faßt die Regeln des Vorrangs und der Assoziativität der einzelnen Operatoren zusammen. Operatoren derselben Zeile haben den gleichen Vorrang. Der Vorrang nimmt von oben nach unten ab. So bindet der Operator > stärker als der Operator so daß bei Auswertung des

Ausdrucks

```
b > 20
```

zunächst der Wert des Teilausdrucks 'b > 20' ermittelt wird und dann geprüft wird, ob a mit diesem Ergebnis übereinstimmt. Die meisten Operatoren sind linksassoziativ, d.h., sie werden von links beginnend ausgewertet.

```
a/b/c
```

wird als

```
(a/b)/c
```

interpretiert

Operator	Assoziativität
() [] ->	links nach rechts
! ~ ++ -- (typ) & sizeof	rechts nach links
* / %	links nach rechts
+ -	links nach rechts
<< >>	links nach rechts
< <= > >=	links nach rechts
== !=	links nach rechts
&	links nach rechts
|	links nach rechts
↑	links nach rechts
&&	links nach rechts
| |	links nach rechts
?:	rechts nach links
= /= %= >>= <<= &= |= ↑=	rechts nach links
, (Kommaoperator)	links nach rechts

Anmerkung: Bei den einzelnen Zeichen * und & in Zeile 2 handelt es sich um die Adreßoperatoren.

Bei der Auswertung von Ausdrücken ist zu beachten, daß C nicht die Reihenfolge spezifiziert, in der die Operanden berechnet werden. Gleichzeitig ist die Reihenfolge, in der Funktionsargumente berechnet werden, nicht festgelegt. Deshalb sollten Programme stets so gestaltet werden, daß sie unabhängig von irgendwelchen Berechnungsreihenfolgen sind.

7.3.4. Anweisungen

Die Anweisungen einer Sprache steuern den Programmablauf. C enthält die grundlegenden Programmsteueranweisungen, die für gut strukturierte Programme erforderlich sind: Anweisungsverbund, Verzweigungen (if, switch), Schleifen mit Abbruchtest am Anfang (while, for) oder am Ende (do).
In C wird aus einem Ausdruck eine Anweisung, indem ein Semikolon angehängt wird. Das Semikolon steht für das Ende einer Anweisung und ist kein Trennzeichen wie in anderen Sprachen!

Beispiele für Anweisungen:

```
a *= 0.71;
--k;
up1(a,b);
```

Zusammengesetzte Anweisungen (Block)

Mehrere Anweisungen können durch Einschließen in geschweifte Klammern zu einer Verbundanweisung (oder Block) zusammengefaßt werden. (Dies entspricht dem begin...end aus anderen Sprachen, z.B. PASCAL.) Der schließenden Klammer, die einen Block beendet, folgt kein Semikolon. Ein so gebildeter Block entspricht syntaktisch einer einzelnen Anweisung. Angewendet werden Blöcke z.B. für mehrere Anweisungen nach if, else, while oder for.
Darüber hinaus können in jedem Block lokale Variablen vereinbart werden. Der Gültigkeitsbereich solcher Variablen ist dann der betrachtete Block, einschließlich aller eingeschachtelten Blöcke (sofern dort nicht weitere lokale Variablen mit denselben Namen vereinbart werden). Ein Block in C kann keine weiteren Funktionen enthalten. Weiteres dazu im Abschn. 7.3.9. Programmstruktur und Gültigkeitsbereiche.

Leere Anweisung

Die leere Anweisung besteht nur aus einem Semikolon. Sie findet dort Anwendung, wo syntaktisch eine Anweisung verlangt wird, vom Programm her aber keine Aktion ausgeführt werden soll.

Zuweisung

Aus einem Zuweisungsausdruck (z.B. a=b+5) wird durch Anhängen Semikolons eine Zuweisungs-Anweisung.

```
a=b+5;
```

Eine Zuweisung in C besitzt einen Wert. Es ist der Wert, der an die links stehende Variable übergeben wird. Dies kann zu ungewohnten, aber gültigen, Anweisungen, wie

```
    (b   2) + 3;
while ((c=getchar())    EOF)
```

führen. Da die Typkonvertierung bei Zuweisungen nach anderen Regeln abläuft als bei der Ausdrucksbildung, ist es notwendig, alle möglichen Konvertierungsfälle anzugeben. Die folgende Tabelle dazu wurde /20/ entnommen. Dabei entspricht die Angabe von "signed int" den Typen 'char' 'int' und 'long', und die Angabe von "integer" entspricht sämtlichen Integertypen einschließlich 'char'.

Variable (lvalue)	Ausdruck	Weitere Bedingungen	Konvertierung
signed int	signed int	Variablentyp ist kleiner als der Ausdruckstyp	die höherwertigen Bits werden abgeschnitten, d.h., der Wert wird eventuell verändert
signed int	signed int	Variablentyp ist größer als der Ausdruckstyp	das Vorzeichenbit wird vervielfacht, d.h. korrekte Wertübernahme
unsigned	signed int	Variablentyp ist kleiner als der Ausdruckstyp	die höherwertigen Bits werden abgeschnitten, d.h., der Wert wird eventuell verändert
unsigned	signed int	Variablentyp ist größer als der Ausdruckstyp	die höherwertigen Bits werden mit Nullen gefüllt, d.h., der Wert wird eventuell verändert
signed int	unsigned	Variablentyp ist kleiner als der Ausdruckstyp	die höherwertigen Bits werden abgeschnitten, d.h., der Wert wird eventuell verändert
signed int	unsigned	Variablentyp ist größer als der Ausdruckstyp	die höherwertigen Bits werden mit Nullen gefüllt, d.h. korrekte Wertübernahme

Variable (lvalue)	Ausdruck	Konvertierung
float, double	integer	als Nachkommastellen werden Nullen angenommen, d.h., der Wert wird korrekt übernommen
integer	float, double	die Nachkommastellen (bzgl. Festkommadarstellung) werden abgeschnitten, d.h., der Wert wird eventuell verändert
float	double	der Wert wird gerundet, so daß sieben signifikante Ziffern korrekt bleiben
double	float	die niederwertigen Bits der Mantisse werden mit Nullen aufgefüllt, d.h., der Wert wird korrekt übernommen

Die Verwendung von unterschiedlichen Typen bei der Zuweisung muß deshalb mit besonderer Vorsicht erfolgen.

Nun zu den Programmsteueranweisungen. An Verzweigungen bietet C die fache Verzweigung (if...else) und die Mehrfachverzweigung (switch).

if-Anweisung

Die allgemeine Form der if-Anweisung ist

```
if (Ausdruck)
     Anweisung_1
else
     Anweisung_2
```

Der else-Zweig ist dabei optional. Es wird der nach if folgende (in runde Klammern eingeschlossene) Ausdruck ausgewertet. Ist sein Wert ungleich Null (d.h. "wahr"), so wird die Anweisung_1 ausgeführt. Falls ein else-Zweig existiert und der Wert des Ausdrucks gleich Null (d.h. "falsch") ist, so wird die Anweisung_2 ausgeführt. Ansonsten wird mit der nächstfolgenden Anweisung fortgesetzt. Die Anweisungen können natürlich auch Verbundanweisungen sein.
In einer geschachtelten if-Folge gehört ein else-Zweig immer zu dem unmittelbar vorhergehenden if. So ist

```
if (n>0)
if (a<b)
  z=
else
  z=b;
```

gleichbedeutend mit

```
if (n>0) {
     if (a<b)
       z=a;
     else
       z=b;
     }
```

und nicht mit

```
if (n>0) {
     if (a<b)
       z=a;
     }
else
  z=b;
```

Im Gegensatz zu anderen Sprachen fehlt das trennende Schlüsselwort 'then'. Zu beachten ist das abschließende Semikolon nach z=a (z=a ist ein Ausdruck und z=a; ist eine Anweisung!).

switch-Anweisung

Die switch-Anweisung erlaubt eine Mehrfachverzweigung in Abhängigkeit vom Wert eines ganzzahligen Ausdrucks. Ihre allgemeine Form ist

```
switch (Ausdruck){
    case Fallkonstante_1   Anweisung(en)_1


    case Fallkonstante_n   Anweisung(en)_n
    default: Anweisung(en)

   }
```

Falls zu einem case-Zweig mehrere Anweisungen gehören sollen, so brauchen sie nicht als Verbundanweisung geschrieben zu werden. Der default-Zweig ist optional. Eine switch-Anweisung muß mindestens zwei Fälle enthalten. Der Wert des Ausdrucks (ganze Zahl einschließlich Textzeichen gestattet!) wird nacheinander mit den nach case folgenden Fallkonstanten (ganzzahlige Konstante oder Zeichenkonstante) verglichen. Bei einer Übereinstimmung werden alle (!) nachfolgenden Anweisungen ausgeführt. Falls keine Übereinstimmung festgestellt wird, so werden nur die nach default folgenden Anweisungen ausgeführt. Dadurch ist es möglich, mehrere Fallkonstanten für ein und dieselbe Aktion anzugeben. Die Reihenfolge der Anordnung der case-Zweige einschließlich des default-Zweiges ist unwesentlich.
z.B.:

```
     '\t':
case '\n'
Anweisungen
```

Die Vorgehensweise, daß nach einer festgestellten Übereinstimmung alle folgenden Anweisungen ausgeführt werden, stört im allgemeinen. Deshalb kann durch Angabe der break-Anweisung ein Verlassen der switch-Anweisung erreicht werden. (Anstelle von break kann auch return verwendet werden. In diesem Fall wird zusätzlich die Funktion verlassen.) Die folgenden wei Beispiele sollen die unterschiedliche Arbeitsweise demonstrieren.

```
int k,a;
k=2; a=1;
switch (k) {
  case 0:
  case 1: a++;
  case 2: a++;
  case 3: a++;
  case 4: a++;
  default:
}
```

Nach der switch-Anweisung hat a den Wert 3.

```
int k,a;
k=2; a=1;
switch (k) {
  case 0:
  case 1:       break;
  case 2:       break;
  case 3:       break;
  case 4:       break;
  default:
}
```

Nach dieser switch-Anweisung hat a den Wert 2.

Zur Bildung von Programmschleifen bietet C drei Anweisungen an: 'while', 'for' und 'do'.

while-Anweisung

Bei der while-Anweisung erfolgt der Test der Abbruchbedingung am Schleifenanfang. Ihre allgemeine Form ist

```
while (Ausdruck)
   Anweisung
```

Es wird der Wert des Ausdrucks berechnet. Bei einem Wert ungleich Null (d.h. "wahr") wird die Anweisung ausgeführt und danach der Ausdruck wiederum berechnet. Dieser Zyklus wird so lange durchgeführt, bis der Wert des Ausdrucks Null ist. Dann wird mit der nach while folgenden Anweisung fortgesetzt. Für Anweisung stehen natürlich meist mehrere Anweisungen in Form einer Verbundanweisung.
z.B.:

```
while (k<=10) {
    flags[k]=0;
    k+=DELTA;

}
```

Oft wird bei einer while-Anweisung die ganze Arbeit auch innerhalb des Ausdrucks getan; ohne explizit einen Schleifenkörper anzugeben.

```
while ((c=getchar())     '\n');
```

Diese Schleife wird so lange ausgeführt, bis über die Bibliotheksfunktion getchar() (Einzelzeicheneingabe) ein Newlinezeichen übergeben wird.

for-Anweisung

Die allgemeine Form der for-Anweisung ist

```
for (Ausdr_1; Ausdr_2; Ausdr_3)
    Anweisung
```

Sie stellt im Grunde eine Abkürzung für die Anweisungsfolge

```
Ausdr_1;
while (Ausdr_2){
    Anweisung
    Ausdr_3;
}
```

dar. Jeder der drei Ausdrücke kann weggelassen werden. Die Semikolons müssen dann aber trotzdem geschrieben werden. Fehlt der zweite Ausdruck, dann wird er als permanent wahr genommen. So ist

```
for (;;) {

}
```

eine unendliche Schleife, die durch andere Mittel (break- oder return-Anweisung) verlassen wird. Der erste Ausdruck einer for-Anweisung stellt eine Initialisierung dar, der zweite ist die Abbruchbedingung und der dritte Ausdruck kann als Zählanweisung bezeichnet werden. Dies wird deutlich in

```
for (i=0; i<N; i++) {

}
```

Durch Verwendung des Kommaoperators im ersten und dritten Ausdruck lassen sich elegant Mehrfachinitialisierungen und unterschiedliche Schrittweiten realisieren. So beginnt die Schleife

```
for (i=0, k=1; i<64;      k*=2) {

}
```

mit i=0 und k=1, läuft, solange i kleiner 64 ist, wobei bei jedem Durchlauf i um 1 erhöht und k verdoppelt wird.

do-Anweisung

Bei der while- und for-Schleife erfolgt der Test der Abbruchbedingung Anfang der Schleife. Die do-Schleife testet am Ende nach jedem Durchlauf des Schleifenkörpers auf Wiederholung. Somit wird der Körper wenigstens einmal ausgeführt. Die allgemeine Form der do-Anweisung ist

```
do
    Anweisung
while (Ausdruck);
```

Es wird die Anweisung (meist eine Verbundanweisung) ausgeführt. Danach wird der Ausdruck ausgewertet. Bei einem wahren Wert (d.h. ungleich Null) wird die Schleifenanweisung erneut abgearbeitet. Bei einem Ausdruckswert gleich Null wird mit der nächstfolgenden Anweisung fortgefahren.
So ist

```
do {
    ...
} while (n=up());
```

eine Schleife, die so lange abgearbeitet (mindestens einmal) wird, bis von der aufgerufenen Funktion up() der Wert 0 zurückgegeben wird.

Oft ist es notwendig, eine Schleife an einer anderen Stelle als am Anfang oder Ende zu verlassen oder auch einen Iterationsschritt vorzeitig erneut beginnen zu lassen. Dazu bietet C die break- und continue-Anweisung.

break-Anweisung

Die break-Anweisung besteht aus dem Wort break gefolgt von einem Semikolon. Sie bewirkt das sofortige Verlassen der innersten Schleife (oder einer switch-Anweisung). Sie ist ihrer Funktion nach ein Sprung hinter das Schleifenende. Mit ihrer Hilfe können z.B. unendliche Schleifen verlassen werden, wie in

```
while (1) {
    ...
    if (k>=10) break;

}
```

continue-Anweisung

Die continue-Anweisung besteht aus dem Wort continue gefolgt von einem Semikolon. Sie bewirkt, daß die nächste Iteration der umgebenden Schleife (for, while oder do) begonnen wird. Bei der while- und do-Anweisung heißt das, daß der Abbruchtest sofort ausgeführt wird. Bei der for-Anweisung wird die Steuerung an den Reinitialisierungsschritt übergeben. Eine continue-Anweisung innerhalb von switch, das selbst in einer Schleife auftritt, bewirkt die nächste Schleifeniteration (d.h. continue wirkt nur in Schleifen, nicht in switch!).

```
for (i=0; i<N; i++) {
    ...
    if (a[i]<0)      /* ueberspringe negative Elemente */
      continue;
                     /* Aktionen fuer positive Elemente */
}
```

Sprunganweisung und Marken

C stellt eine unbedingte Sprunganweisung zu Marken zur Verfügung. Formal sind solche Sprünge nicht notwendig. Sie können aber durch sparsame und sinnvolle Verwendung zur Programmklarheit beitragen.
Eine Marke in C hat die gleiche Form wie ein Variablenname, ihr muß ein Doppelpunkt folgen. Sie kann jede Anweisung kennzeichnen. Die Sprunganweisung besteht aus dem Wort goto gefolgt von einem Trennzeichen, einem Markennamen und einem Semikolon,
z.B.

```
goto fehler;
```

Die angegebene Marke muß in derselben Funktion liegen, in der sich auch die Sprunganweisung befindet. Sprünge in andere Funktionen hinein mittels goto sind in C nicht gestattet!
Gewöhnlich wird ein unbedingter Sprung verwendet, um die Verarbeitung in tiefer geschachtelten Anweisungen zu verlassen. Zum Beispiel, um zwei Schleifen auf einmal zu verlassen. Die break-Anweisung kann dazu nicht verwendet werden, da sie nur die innerste Schleife verläßt. Dies wird häufig für Sprünge zu einer Anlaufstelle für die Fehlerbehandlung eingesetzt.

```
for (...)
    for (...) {
    ...
    if (Ausdruck, der auf Fehler testet)
        goto fehler;
    ...
    }
...
fehler:         /* entsprechende Aktionen im Fehlerfall */
```

7.3.5. Funktionen und Argumente

Innerhalb von Programmabläufen ist es oft günstig, bestimmte Aktionen oder Berechnungen als eine einzelne Einheit auszulagern (Unterprogrammtechnik). Dies dient der Modularisierung und der Vereinfachung von komplexen Programmen. In der Programmiersprache C kann diese Aufgabe mit Hilfe der Funktionen gelöst werden. Die Übergabe von Werten an Funktionen kann über Parameter oder externe Variablen erfolgen. Jede Funktion in C liefert einen Wert. Der aufrufenden Funktion ist es jedoch freigestellt, ob sie diesen Wert auswertet. Damit erfüllen in C Funktionen sowohl die Aufgabe der Wertermittlung als auch die Aufgabe, die in PASCAL z.B. durch Prozeduren erledigt wird. Die Beschränkung auf nur eine Strukturkategorie (Funktionen) vereinfacht zwar die Programmierung, fordert aber auch eine erhöhte Aufmerksamkeit bei der Angabe und Verwendung von Werten.

Eine Funktion in C hat die allgemeine Form:

```
Ergebnistyp  Funktionsname ( formale Parameterliste )
     Vereinbarung der formalen Parameter
     {
          Funktionskörper
     }
```

Der Ergebnistyp, die formale Parameterliste, die Vereinbarung der formalen Parameter und der Funktionskörper sind dabei optional. So ist

```
dummy ()
  {}
```

eine Funktion ohne Parameter, die nichts leistet.
Wenn der Ergebnistyp (Rückgabewert) der Funktion ungleich 'int' ist, dann muß der Typ des Funktionwertes bei der Funktionsdefinition explizit angegeben werden.
Ein C-Programm besteht aus einer Reihe von Funktionsdefinitionen, die einer beliebigen Reihenfolge angeordnet sein können. Die Funktionen eines Programms können auf mehrere Quelltextdateien (Module) verteilt sein, die getrennt übersetzt werden können. Jedes Programm muß eine Funktion main() enthalten, bei der die Ausführung beginnt (entspricht dem Hauptprogramm). Auf weitere Details der Programmstruktur wird im Abschn. 7.3.9. eingegangen. Innerhalb von Funktionen dürfen keine weiteren Funktionen definiert werden!
Beispiel einer Funktionsdefinition:

```
double kreisfl (radius)  /* Kreisflaeche berechnen */
     double radius;
     {
          return (radius*radius*3.14159);
     }
```

Die Funktion berechnet den Inhalt eines Kreises. Sie benötigt als Argument den Radius (als Wert vom Typ 'double') und übergibt an die aufrufende Funktion den Kreisinhalt (ebenfalls als Wert vom Typ 'double').

Der Aufruf einer Funktion erfolgt durch Angabe des Funktionsnamens gefolgt von der in runden Klammern eingeschlossenen aktuellen Parameterliste. Die einzelnen Argumente sind dabei durch Komma zu trennen (es handelt sich dabei nicht um einen Kommaoperator). Als aktuelle Parameter können beliebige Ausdrücke erscheinen. Falls keine Parameter übergeben werden, so sind hinter dem Funktionsnamen nur die runden Klammern anzugeben.
Der Funktionsaufruf kann auch in einem Ausdruck erfolgen, wenn der Funktionswert weiter verwendet wird. Bei Anwendung in einer allein stehenden Anweisung ist eine Auswertung des Rückgabewertes nicht möglich.
Beispiele für Funktionsaufrufe:

in Ausdrücken:

```
fl=kreisfl(rad);
if (kreisfl(rad)*hoehe < 123.45) ...
inhalt=volumen (hoehe, kreisfl(rad));
while ((c=getchar())    EOF)
```

als Anweisung:

```
move(a,b);
leben();
```

Durch den Aufruf einer Funktion geht der Programmablauf an die Anweisungen des Funktionskörpers über. Die Rückkehr zur aufrufenden Funktion kann auf drei Wegen erfolgen:

- das Ende des Funktionskörpers (schließende geschweifte Klammer) wird erreicht
- Abarbeitung der Anweisung return;
- Abarbeitung der Anweisung return(Ausdruck);

in den ersten beiden Fällen ist der Funktionswert undefiniert. Im letzten Fall wird der Wert des Ausdrucks in der return-Anweisung als Funktionswert übergeben.
Als Typen von Funktionswerten sind zulässig:

- alle Grunddatentypen ('char' 'int', 'unsigned', 'float' und 'double' einschließlich der Kennung durch 'short' und 'long')
- Zeiger auf beliebige Objekte einschließlich Zeiger auf Felder, Strukturen und Funktionen
- Strukturen, bei früheren Compilern nicht möglich

Die Übergabe von Parametern an die aufgerufene Funktion erfolgt als Wertübergabe, d.h., die Werte der aktuellen Parameter werden in von der Funktion bereitgestellte Speicherstellen kopiert ("call by value"). Manipulationen mit den Parametern innerhalb der Funktion werden mit diesen "privaten" temporären Größen durchgeführt, so daß die Variablenwerte in der aufrufenden Funktion unverändert bleiben. Innerhalb einer Funktion ist jedes Argument also eine lokale Größe, die mit dem Wert des entsprechenden aktuellen Parameters initialisiert wird.
Zu dieser Verfahrensweise existieren Ausnahmen. Wird ein Feldname als Argument einer Funktion übergeben, so wird nur die Anfangsadresse des Feldes übergeben. Die Elemente des Feldes werden nicht kopiert. Die aufgerufene Funktion kann demzufolge die Feldelemente verändern. Im nächsten Punkt wird die Verwendung von Zeigern erläutert. Wird ein Zeiger als Argument einer Funktion übergeben, so kann die aufgerufene Funktion ebenfalls das Objekt verändern, auf das der Zeiger zeigt. Man spricht hierbei von einer Variablenübergabe durch Referenz ("call by reference").

Beim Funktionsaufruf prüft der Compiler nicht, ob die Typen der aktuellen Parameter mit denen der formalen Parameter übereinstimmen. Ebenso erfolgt keine Prüfung, ob die Anzahl der aktuellen Argumente der Parameteranzahl der Funktionsdefinition entspricht. Damit beim Funktionsaufruf keine undefinierten Werte übergeben werden, ist auf diese beiden Punkte besonders zu achten! (Das C-Prüfprogramm 'lint' erkennt solche Fehler.)
Wie schon im Abschn. 7.3.2. angeführt, erfolgt beim Funktionsaufruf eine implizite Argumenttypkonvertierung ('char', 'short' zu 'int' und 'float' zu 'double'). Dies reicht zur Typanpassung natürlich nicht immer aus. Deshalb muß zur Angleichung der Typen der aktuellen Parameter oft der cast-Operator eingesetzt werden, z.B. wenn beim Aufruf der Funktion 'sqrt' der Typ des Arguments 'int' oder 'float' ist.

```
sqrt((double)xyz);
```

Funktionen in C können rekursiv aufgerufen werden. Als Beispiel dazu kann die Fibonacci-Funktion für ganze Zahlen dienen, die folgendermaßen definiert ist:

```
f(x)=1     für x<=2
f(x)=f(x-1)+f(x-2)  für x>2
```

Die C-Funktion dazu kann folgendermaßen aussehen:

```
unsigned fib(x)
     int x;
     {
          if (x>2) return(fib(x-1)+fib(x-2));
                         /* rekursiver Aufruf */
          else return(1);
     }
```

7.3.6. Felder und Zeiger

Felder stellen eine Zusammenfassung von Objekten eines gleichen Typs dar. Auf die Elemente eines Feldes (array) kann mittels Indizierung zugegriffen werden. Ebenso wie Variablen müssen Felder vor ihrem Gebrauch vereinbart werden. Die allgemeine Form einer Feldvereinbarung lautet

```
Typ  Name[Elementanzahl];
```

Optional kann noch eine Speicherklassenangabe vorangestellt sein. Die Anzahl der Feldelemente wird bei der Vereinbarung durch einen konstanten Ausdruck vom Typ 'int' angegeben.

Beispiele für Feldvereinbarungen:

```
int w[5];       /* 5 Integergroessen: w[0]...w[4] */
double za[10]; /* Feld von 10 Gleitkommazahlen: za[0]...za[9] */
char text[64]; /* Feld von 64 Textzeichen */
```

In C beginnt die Indizierung immer bei 0 (im Gegensatz anderen Sprachen, z.B. Fortran oder PL/I, wo sie bei 1 beginnt), daß die einzelnen Elemente über

```
text[0]      text[63]
```

erreichbar sind. Als Index kann jeder Integerausdruck verwendet werden, natürlich auch Variablen vom Typ 'int' und Integerkonstanten.

Mehrdimensionale Felder sind auch möglich.

```
char namen[5][10];  /* zweidimensionales Feld aus Zeichen */
int tage[2][13];
float matrix[10][10][20];
```

In C ist ein zweidimensionales Feld tatsächlich ein eindimensionales Feld, dessen Elemente selbst Felder sind. Deshalb werden die Indizes in der Form

```
x[i][j]
```

geschrieben, anstatt

```
x[i,j]
```

wie in den meisten anderen Programmiersprachen. Die Elemente mehrdimensionaler Felder werden zeilenweise gespeichert. Das oben vereinbarte Feld 'tage' wird in der Reihenfolge

```
tage[0][0]  tage[0][12], tage[1][0]  tage[1][12]
```

gespeichert.
Analog zu Variablen können Felder bei ihrer Vereinbarung initialisiert werden. Dies ist allerdings nur bei globalen Feldern (außerhalb von Funktionen) sowie bei statischen lokalen Feldern (innerhalb von Funktionen und Speicherklasse 'static') erlaubt.
Die Initialwerte für die Feldelemente werden dabei durch Kommata getrennt und in geschweifte Klammern eingeschlossen. Bei mehrdimensionalen Feldern werden die Initialisierungen jeder einzelnen Dimension zusätzlich in geschweifte Klammern eingeschlossen. Erfolgt eine Initialisierung innerhalb der Feldvereinbarung, so können die Angaben zur Felddimension wegfallen. In einem solchen Fall wird die Dimension aus der Anzahl der Initialwerte abgeleitet.

```
static float z[ ] = {.0123, 1.234e-5, 12.34E+5};
static int tage[][]= {
          {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
          {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};
static int a[7]   {1, 2, 3};
static int b[ ]   {1, 2, 3};
```

Das Feld 'a' besitzt 7 Elemente (a[3] bis a[6] werden mit Null initialisiert). Das Feld 'b' dagegen besteht aus nur drei Elementen.
Zeichenfelder können zusätzlich über Zeichenkettenkonstanten initialisiert werden.

```
static text1[] = {'h', 'a', 'l', 'l'  'o'};
static text2[] = "hallo";
```

Die letzten beiden Vereinbarungen definieren Felder von Zeichen. Dabei ist zu beachten, daß das Feld 'text1' aus 5 Zeichen besteht und das Feld 'text2' aus 6 Zeichen (abschließendes Zeichen \0)!
Die Angabe der Speicherklasse 'static' kann entfallen, das Feld lobal, d.h. außerhalb von allen Funktionen definiert wird.

Zeiger

Ein Zeiger (pointer) ist ein Datentyp, der Adressen von Objekten eines bestimmten Datentyps aufnehmen kann. Es können sowohl Adressen von einfachen als auch zusammengesetzten Datenelementen (auch Funktionen) gebildet werden. Auch von Zeigern selbst können wiederum die Adressen in Zeigern abgelegt werden. Mit diesem Datentyp ist es möglich, auf Objekte über Zeiger "indirekt" zuzugreifen. Zeiger spielen in C eine zentrale Rolle. Sie können im Gegensatz zu anderen Programmiersprachen sehr freizügig manipuliert werden. Dies bringt in vielen Fällen eine Effektivitätssteigerung, erfordert aber auch einen sehr disziplinierten Umgang mit Zeigern. Zeigervariablen dürfen nur auf Objekte eines bestimmten Typs zeigen. Sie müssen vor ihrer Verwendung vereinbart werden. Bei der Vereinbarung von Zeigervariablen wird der Typ angegeben, auf den sie verweisen können. Zur Unterscheidung von normalen Variablen dient in der Vereinbarung der Stern (*) vor dem Namen. Zeigervariablen, die wiederum auf einen Zeiger verweisen, werden durch zwei Sterne (**) gekennzeichnet.

```
int *z;          /* Zeiger auf Integergroesse */
char *cp;        /* Zeiger auf Zeichen */
double *f[5];    /* Feld von Zeigern auf 'double'-Werte */
int **ip;        /* Zeiger auf Zeiger, die auf 'int'-Werte verw.*/
float a,b,*pf;   /* Vereinbarung von zwei 'float'-Variablen: a,b*/
                 /* und eines Zeigers 'pf' der auf 'float' verw.*/
struct pers *p;      /* Zeiger auf Struktur vom Typ 'pers' */
                     /* Strukturen siehe Abschn. 7.3.7. */
```

Die unären Operatoren * ("Inhalt von") und & ("Adresse von") kommen Zusammenhang mit Zeigern zur Anwendung. So kann durch

```
int a, b, *ptr;
ptr   &a;
```

der Zeigervariablen 'ptr' die Adresse von zugewiesen werden. Danach kann auf die Variable 'a' auch indirekt zugegriffen werden. Die beiden folgenden Anweisungen sind äquivalent.

```
b   a;
b   *ptr;
```

Der &-Operator ist nur für Variablen und Feldelemente zulässig. Der Adreßoperator (&) darf nicht auf "register"-Variablen angewendet werden. Der *-Operator darf nur auf Zeigervariablen angewendet werden. Der Ausdruck *ptr kann wie eine normale Variablenbezeichnung gehandhabt werden. Er kann überall dort vorkommen, wo die Variablenbezeichnung selber stehen kann, also auch links vom Zuweisungszeichen (=).

```
*ptr   15;
*ptr    7;
```

In C besteht eine enge Beziehung zwischen Zeigern und Feldern. Durch die Vereinbarung

```
int a[10]
```

wird ein Feld von 10 Integergrößen (a[0] bis a[9]) definiert. Wenn 'pa' ein Zeiger auf Integerwerte ist, der als

```
int *pa;
```

vereinbart ist, dann ergibt sich aus der Zuweisung

```
pa  &a[0];
```

daß 'pa' auf das erste Element des Feldes zeigt. Durch

```
*pa;
```

wird der Variablen 'x' der Inhalt von a[0] zugewiesen. Wenn 'pa' auf ein einzelnes Element des Feldes zeigt, so zeigt 'pa+1' auf das nächste Element und 'pa-1' verweist auf das vorhergehende Feldelement. Allgemein gilt: 'pa-i' zeigt auf das i-te Element vor 'pa' und 'pa+i' zeigt auf das i-te Element nach 'pa'. Demzufolge ist

```
*(pa+1)
```

der Inhalt von a[1], wenn 'pa' auf a[0] zeigt. All dies gilt unabhängig vom Typ der Feldvariablen. Die Angabe "Addiere i zu einem Zeiger" und darüber hinaus die gesamte Zeigerarithmetik basieren darauf, daß der Inkrement- oder Integerwert entsprechend der Länge der Objekte transformiert wird.
Die Verbindung zwischen Indizierung und Zeigerarithmetik ist sehr eng. Durch den Compiler wird ein Verweis auf ein Feld in einen Zeiger auf den Anfang des Feldes konvertiert. Dementsprechend kann die Zuweisung

```
pa  &a[0];
```

auch in der Form

```
pa  a;
```

geschrieben werden. Der Feldname kann aber auch selbst als Zeiger verwendet werden. Folgende Schreibweisen sind äquivalent:

```
a[i]    und    *(a+i)
&a[i]   und
```

Feldnamen sind keine Variablen, sie sind Adreßkonstanten, und deshalb dürfen Feldnamen in Konstrukten wie

```
&a     oder
a=...  oder
a++
```

nicht erscheinen!
Der Zeigeroperator kommt häufig in Ausdrücken zusammen mit Inkrement- und Dekrementoperatoren vor. Da die Operatoren *, ++ und -- den gleichen Vorrang haben, muß in diesen Fällen der Assoziativität besondere Aufmerk-

samkeit geschenkt werden. So wird bei

```
b   *++pa;
```

erst der Zeiger 'pa' inkrementiert und danach der Inhalt ermittelt und der Variablen 'b' zugewiesen. Bei

```
b   *pa++;
```

wird der Variablen 'b' erst der Inhalt des Objektes, auf das 'pa' zeigt, zugewiesen. Danach wird der Zeiger 'pa' inkrementiert, d.h., er zeigt nun auf das nächste Element.

Die Regeln der Arithmetik mit Zeigervariablen können wie folgt zusammengefaßt werden:

- Werte von Zeigervariablen dürfen nur anderen Zeigervariablen des gleichen Typs zugewiesen werden. Undefinierte Werte von Zeigervariablen sollten durch den Wert NULL gekennzeichnet werden.

- Zeigervariablen können dekrementiert und inkrementiert werden. Ebenfalls dürfen Zeiger um Integerwerte erhöht oder verringert werden. Die Arithmetik erfolgt dabei grundsätzlich in Speichereinheiten des Typs, auf den der Zeiger zeigt. ("skalierte Zeigerarithmetik").

- Wenn zwei Zeiger auf Elemente des gleichen Feldes zeigen, dann können sie miteinander verglichen werden. (Operatoren: !=, <, >, <=, >=) Der Ausdruck 'p1 > p2' ist wahr, wenn der Zeiger 'p2' auf ein Element zeigt, das sich im Feld vor demjenigen befindet, auf das 'p1' zeigt. Jeder Zeiger kann auf Gleichheit oder Ungleichheit mit dem Wert NULL getestet werden.

- Wenn zwei Zeiger auf Elemente des gleichen Feldes zeigen, so ergibt die Zeigersubtraktion 'p1 - p2' einen Integerwert, der die Anzahl der Elemente zwischen 'p1' und 'p2' angibt. Das gilt ebenfalls unabhängig vom Typ der Feldelemente.

Wie schon im Abschnitt zu Funktionen erwähnt, können Zeiger auch als Funktionsargumente übergeben werden. Als Beispiel hierzu soll die Funktion 'strcpy' dienen, die eine Zeichenkette 'a' zur Zeichenkette 'b' umspeichert.

```
strcpy(a, b)   /* kopiert Zeichenkette a in b */
char *a, *b;
{
     while ((*b++   *a++)    '\0');
}
```

Der Vergleich auf das abschließende '\0' ist hierbei sogar noch redundant. Dadurch, daß Feldnamen in Funktionsaufrufen als Zeiger auf das erste Element übergeben werden, ist es möglich, sie innerhalb der Funktion als Zeigervariablen zu verwenden. Dies setzt voraus, daß sie in der Funktion als Zeigervariablen und nicht als Felder vereinbart werden. Wäre im obigen

Beispiel z.B. durch

```
char a[];
```

vereinbart, so wären die angegeben Operationen mit nicht zulässig.

In C ist es gestattet, Zeiger auf Funktionen zu vereinbaren. Diese können z.B. dazu benutzt werden, um Funktionsadressen als Parameter an Funktionen zu übergeben. Die allgemeine Form der Vereinbarung eines Zeigers auf Funktionen ist

```
Typ (*Name)();
```

Dadurch wird eine Zeigervariable ('Name') vereinbart, die als Werte die Adressen von Funktionen annehmen kann, deren Funktionswert vom angegebenen Typ ist. Nachdem einer solchen Zeigervariablen ein Wert zugewiesen wurde, z.B.

```
int (*pf)();

pf   fkta;     /* fkta() ist vom Typ 'int' */
```

kann die Funktion indirekt aufgerufen werdem.

```
(*pf)(aktuelle Parameter);     ist äquivalent zu
fkta(aktuelle Parameter);
```

Da Zeiger selbst Variablen sind, können sie auch in Feldern angeordnet werden. Zeigerfelder finden bei der Manipulation von variabel langen Datenobjekten Anwendung. Die Vereinbarung von Zeigerfeldern erfolgt durch

```
Typ *Name[Anzahl der Zeiger];
```

Die Elemente eines solchen Feldes ('Name') zeigen auf Objekte des angegebenen Typs.

Kommandozeilenargumente

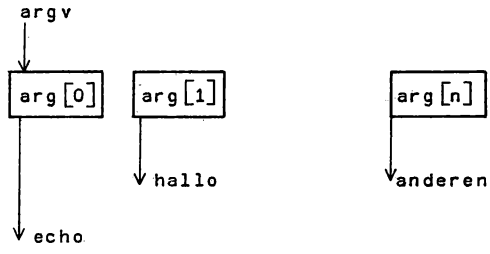
Beim Programmieren in C besteht die Möglichkeit, Argumente der Kommandozeile als Parameter einem Programm bei seinem Start zu übergeben. Der erste (meist argc genannt) ist die Anzahl der Kommandozeilenargumente. Der zweite (argv) ist ein Zeiger auf ein Feld von Zeichenketten, das die Argumente enthält, ein Argument pro Zeichenkette.
Ein einfaches Beispiel zur Verdeutlichung der notwendigen Vereinbarungen und der Benutzung dieser Parameter ist das Programm 'echo'. Es gibt seine Kommandozeilenargumente auf einer Zeile, getrennt durch Leerzeichen aus. So würde der Aufruf

```
echo hallo peter und alle anderen
```

die Ausgabe von

```
hallo peter und alle anderen
```

bewirken. 'argc' muß als Integergröße vereinbart sein, ihr Wert ist die Anzahl der Argumente (einschließlich des Programmnamens). In unserem Beispiel also 6. 'argv' muß als ein Feld von Zeigern vereinbart sein, die auf Zeichen zeigen (char *argv[];). Dabei zeigt arg[0] auf den Programmnamen selber.



Nun eine mögliche Version des Programms 'echo'.

```
main(argc, argv)    /* echo-Programm */
int argc;
char *argv[];
{
     while (--argc > 0)
          printf ((argc > 1) ? "%s     "%s\n", *++argv);
}
```

Da 'argv' ein Zeigerfeld ist, wird durch ++argv der erste echte Parameter (argv[1] anstatt argv[0]) adressiert. Durch jedes Inkrementieren wird auf das nächste Argument übergegangen. Die Ausgabefunktion 'printf' wird mit ihren Möglichkeiten im Abschn. 7.4. näher erläutert. "%s" ist eine Formatsteueranweisung von 'printf' für die Ausgabe von Zeichenketten. Die Ausgabe erfolgt so lange, bis keine Argumente mehr da sind (Dekrementieren von 'argc').

7.3.7. Strukturen und Unions

In einer Struktur in C können mehrere Objekte unterschiedlichen Typs zusammengefaßt werden. Sie ist mit dem record-Typ von Pascal vergleichbar Strukturen der Sprache C werden durch das Schlüsselwort 'struct' gekennzeichnet. Im Anschluß an 'struct' kann optional ein Name folgen, der den nachfolgend in geschweiften Klammern anzugebenden Strukturtyp kennzeichnet. So wird z.B. durch

```
struct datum {
     int tag;
     char monat[8];
     int jahr;
};   /* Strukturtypvereinbarung */
```

ein Strukturtyp mit dem Namen 'datum' vereinbart, der als Strukturkomponenten drei Variablenvereinbarungen enthält. Folgen der den Strukturtyp abschließenden geschweiften Klammer keine Variablennamen, so wird nur der

Strukturtyp vereinbart. Es wird dadurch kein Speicherplatz reserviert. Konkrete Datenelemente (Strukturen) können im Anschluß daran durch

```
...
struct datum geb_dat, beruf_begin;
```

vereinbart und erzeugt werden. Dabei kennzeichnet 'datum' den Strukturtyp, 'geb_dat' und 'beruf_begin' sind dagegen Bezeichner (d.h. Variablennamen) von zwei konkreten Objekten des Strukturtyps 'datum'. Als Komponenten einer Struktur können beliebige Datentypen auftreten, also auch Strukturen selber. Es besteht die Einschränkung, daß eine Struktur sich nicht selber als Struktur enthalten kann. Eine Strukturtypvereinbarung und die Vereinbarung von Variablen dieses Typs kann auch gleichzeitig erfolgen.

```
struct person {
     int pers_nr;
     char name[15];
     char vorname[15];
     int kostenstelle;
     struct datum einstellung;
} meister, fachdir, abt_ltr;
```

Dies kommt oft vor, wenn explizit kein Strukturtypname vergeben werden soll. In diesem Fall könnte also in der Vereinbarung das Wort 'person' auch wegfallen. Es ist syntaktisch nicht ausgeschlossen, daß für den Strukturtyp und für eine Variable der gleiche Name gewählt wird.

```
struct person meister, person;
```

Weiterhin können Namen von Strukturkomponenten mit anderen Variablennamen übereinstimmen. Für unterschiedliche Strukturen können auch gleiche Komponentbezeichnungen verwendet werden. Jeder Strukturtyp eröffnet ein eigenes Namensniveau. Die Namen der Komponenten einer einzigen Struktur müssen sich natürlich voneinander unterscheiden.

Felder von Strukturen können ebenso wie andere Felder vereinbart werden.

```
struct datum fam_tage[10], feiertage[10];
struct person abt_xyz[25], bereich_abc[100];
```

Die Elemente der so vereinbarten Felder sind Strukturen, und diese können weitere Strukturen als Komponenten enthalten (z.B. enthält Struktur 'person' die Struktur 'datum').

Eine Initialisierung von Strukturen ist nur möglich, wenn diese extern (d.h. außerhalb von Funktionen) vereinbart werden oder wenn sie innerhalb von Funktionen durch die Speicherklasse 'static' gekennzeichnet sind. Die Angabe der Initialwerte erfolgt anolog zur Feldinitialisierung, d.h. in geschweiften Klammern.

```
struct datum geb_tag   {7, "November", 1980};

struct person chef   {0815, "Meyer", "Hugo", 1234,
                        {1, "April", 1977}
                     };

struct datum fam_tage[] = {
                              {7, "November", 1980},
                              {16, "Juli", 1957},
                              {11, "April", 1952},
                           };
```

Der Zugriff auf Komponenten einer Struktur erfolgt durch Angabe des Strukturvariablennamens und des Komponentennamens. Beide Bezeichner werden dabei durch einen Punkt getrennt.

```
Strukturvariablenname.Komponentenname
```

Beispiele:

```
beruf_begin.monat
meister.name
chef.einstellung.jahr
hist_tage[2].tag     /* vom zweiten Eintrag im Feld der Tag */
abt_ltr.vorname[0]   /* der erste Buchstabe des Vornamens */
```

Auf Strukturen können die Operatoren zur Adreßbildung ('&') und Zugriff auf Komponenten ('.' und '->') angewendet werden.
Die Einschränkung, daß Strukturen als Ganzes nicht in Zuweisungen und als Funktionsargumente (oder als Resultat von Funktionen) zugelassen sind, ist in neueren C-Versionen beseitigt. Dies ist im Einzelfall an der zur Verfügung stehenden Compilerversion zu überprüfen. In den Fällen, wo diese Möglichkeiten nicht gegeben sind, kann man sich durch Zeiger auf Strukturen behelfen.
Zeiger auf Strukturen können auch als Strukturkomponenten auftreten. Damit ist es möglich, Baumhierarchien von Strukturen eines Typs aufzubauen. Die Vereinbarung eines Zeigers auf eine Struktur erfolgt durch

```
struct person *zp;  /* Zeiger auf Struktur 'person' */
struct datum  *zd;  /* Zeiger auf Struktur 'datum'  */
```

Beispiel einer Strukturvereinbarung, die Zeiger auf derartige Strukturen enthält:

```
struct knoten {
     int op;
     int typ;
     struct knoten *zlinks;
     struct knoten *zrechts;
};
```

Nach Zuweisung der Adresse einer gewünschten Struktur kann auf die Komponenten über die Notation

```
Strukturzeiger->Komponentenname
```

zugegriffen werden, z.B.

```
zp=&meister;

nr=zp->pers_nr;
```

Da '*zp' identisch mit 'meister' ist, kann auch die umständlichere Notation

```
(*zp).pers_nr
```

verwendet werden.
Da das Inkrementieren eines Zeigers grundsätzlich in Speichereinheiten des Objekttyps erfolgt, bedeutet das bei Zeigern auf Strukturen ein Inkrementieren um die Gesamtlänge der Struktur. Die Anwendung von Inkrement- und Dekrementoperatoren auf Strukturzeiger ist damit nur sinnvoll, wenn diese auf Felder von Strukturen zeigen.
Da der Vorrang des Operators '->' höher ist als der von '++' und '--', ist bei der gemeinsamen Verwendung dieser Operatoren der genauen Notation besonderes Augenmerk zu schenken. So entspricht

```
++zp->kostenstelle        ++(zp->kostenstelle)
```

d.h., die Kostenstellennummer wird um 1 erhöht. Um das nächste Strukturelement der Personaltabelle zu erreichen, muß

```
(++zp)->kostenstelle
```

geschrieben werden.
Der Vorrang des Operators liegt unter dem von '->'. Somit bezieht sich bei

```
struct knoten {
          int anz;
          int *pi;
       } a,b;

zp=&a;

nr=*zp->pi;
```

der Operator auf 'pi' (also der Inhalt dessen, worauf der Zeiger 'pi' zeigt) und nicht auf 'zp'. Bei

```
nr=*zp++->pi;  /* entspricht: nr=*zp->pi; zp++; */
```

bezieht sich das Inkrement auf den Zeiger 'zp', und bei

```
nr=++*zp->pi;  /* entspricht: ++*zp->pi; nr=*zp->pi; */
```

wird erst der Zeiger 'pi' inkrementiert und dann erfolgt die Zuweisung des Integerwertes.

Bitfelder

In Programmen werden oft Statusinformationen in einzelnen Bits gespeichert. C bietet für diese Arbeit eine spezielle Form von Strukturen, die Bitfelder, an. Mit ihnen ist es möglich, mehrere Bitvariablen in einem Maschinenwort (einer Integergröße) zu speichern. Die Vereinbarung von Bitfeldern ist einer Strukturvereinbarung ähnlich. Die Komponenten sind hierbei unsigned-Größen. Zusätzlich wird nach dem Komponentennamen ein Doppelpunkt und eine Zahl angegeben. Dieser Wert gibt die Bereichsgröße in Bits an. Durch die Vereinbarung

```
struct {
     unsigned err_code   3;
     unsigned busy       1;
     unsigned ready      1;
     unsigned error      1;
} flag;
```

wird eine Variable 'flag' vereinbart, die vier Felder enthält. Der Zugriff auf die so definierten Bitfelder erfolgt wie bei Strukturen, z.B.

```
flag.busy
```

Bitfelder können wie vorzeichenlose Zahlen behandelt werden.

Beispiele:

```
flag.error=flag.busy=0;  /* Loeschen von Bitfeldern */
...
flag.ready=1;            /* Setzen von Bits */
...
if(flag.busy)...         /* Test ob Bit gesetzt ist */

if(flag.err_code==0x80)  /* Abfrage Bitfeld */
```

Bei der Arbeit mit Bitfeldern sind folgende Besonderheiten zu beachten:

- Ein Bitfeld darf nicht über die Größe einer Integervariablen hinausgehen.
- Die Reihenfolge der Speicherung im Integerwort (ob von links nach rechts oder umgekehrt) ist rechnerabhängig.
- Falls kein Komponentenname angegeben wird, dient die Längenangabe Erzeugung einer Lücke.
- Einzelne Bitfelder besitzen keine Adresse, demzufolge darf der &-Operator nicht auf Bitfelder angewendet werden.

Unions

Unions sind eine spezielle Form des Strukturtyps. Sie bieten die Möglichkeit, daß sich Variablen unterschiedlicher Datentypen denselben Speicherplatz teilen, d.h., zu verschiedenen Zeiten wird der Inhalt der Union unterschiedlich interpretiert. Diese Datenstruktur entspricht dem

Varianten-record von Pascal und wird deshalb in [67] als Variante bezeichnet.
Für eine Union wird Speicherplatz entsprechend dem größten Datentyp reserviert. Die Vereinbarung erfolgt analog zur Strukturvereinbarung.

```
union variante {
          int a;
          float b;
          char c[3];
} uval;
```

Dadurch wird die Speicherstelle 'uval' vereinbart, die Variablen aller drei angegebenen Typen aufnehmen kann.
Es liegt in der Verantwortung des Anwenders, sich den gerade in der Union gespeicherten Typ zu merken. Dies kann über eine Variable oder ein Bitfeld erfolgen. Auf Elemente einer Union kann wie bei Strukturen zugegriffen werden.

```
Unionname.Komponentenname          oder
Unionzeiger->Komponentenname
```

Bei Unions sind nur der Zugriff auf Komponenten und die Ermittlung der Unionadresse gestattet. Sie können nicht an Funktionen übergeben werden oder von ihnen zurückgegeben werden. Zeiger auf Unions können genauso wie Zeiger auf Strukturen verwendet werden.

7.3.8. C-Preprocessor

Dem C-Übersetzungslauf ist die Abarbeitung des C-Preprocessors vorgelagert. Dieser durchsucht die Quelltextdatei nach Zeilen, die mit dem Zeichen '#' beginnen, wertet sie aus und führt entsprechende Aktionen in der Quelltextdatei aus. Mit Hilfe von Preprocessoranweisungen ist es z.B. möglich:

- weitere Quelldateien einzufügen
- symbolische Namen und Makros zu definieren
- Programmteile von der Übersetzung auszuschließen
  (bedingte Übersetzung)
  die Zeilennumerierung zu verändern

Preprocessoranweisungen können an beliebiger Stelle in der Quelldatei stehen. Ist eine Fortsetzung auf der Folgezeile notwendig, so muß die erste Zeile mit \ (CR) abgeschlossen werden. Der Preprocessor erzeugt als Ergebnis eine Quelldatei, in der die entsprechenden Aktionen (Ersetzung von symbolischen Namen, Makroauflösungen u.a.m.) ausgeführt wurden. Diese Datei kann der Anwender sich erzeugen, indem er beim cc-Kommando die Option '-P' angibt (siehe dazu Abschn. 7.7. unter cc-Kommando).

define-Anweisung

Mit Hilfe dieser Anweisung können symbolische Konstanten definiert werden, z.B.

```
#define ANZAHL 100
#define SCHRITTWEITE 0.002
#define MAX 30
```

Diese Anweisung bewirkt, daß in allen folgenden Quelltextzeilen der erste nach '#define' angegebene Name durch die darauffolgende Konstruktion ersetzt wird. Die Ersetzung erfolgt nicht, wenn der Name innerhalb einer Zeichenkettenkonstanten oder eines Kommentars auftritt.
Folgt dem Namen keine Konstruktion, z.B.

```
#define FLAG
```

so gilt der Name nur als definiert. Er kann dann zur Steuerung der bedingten Übersetzung eingesetzt werden (vergleiche auch Option -D beim cc-Kommando, Abschn. 7.7., Seite 226).
Die define-Anweisung wird i.allg. für Konstantendefinitionen verwendet. Sie kann aber auch für weitergehende Ersetzungen eingesetzt werden, z.B. Terminal- oder Druckersteuersequenzen:

```
#define CURSORUP putchar(\032)
#define KURSIV printf("\033[3m")
```

Dabei ist es in C übliche Praxis, Konstanten durch Großbuchstaben kennzeichnen, um sie vom übrigen Programmtext abzuheben.

Die define-Anweisung kann auch zur Makrodefinition angewendet werden. Nach dem Namen müssen dann, in runden Klammern eingeschlossen, die formalen Parameter folgen. Zwischen dem Makronamen und der öffnenden runden Klammer darf dabei kein Leerzeichen stehen!

Beispiel einer Makrodefinition:

```
#define max(A,B) ((A) > (B) ? (A)   (B))
```

Danach wird eine Quellzeile der Form

```
max(n+m,p);
```

durch

```
((n+m) > (p) ? (n+m)   (p));
```

ersetzt.
Bei der Defintion von Makros ist auf eine sorgfältige Klammersetzung zu achten, um die erforderliche Berechnungsreihenfolge bei der Ausdrucksbildung zu gewährleisten. Bei der Benutzung von Makros können Inkrement- und Dekrementoperatoren oder zusammengesetzte Zuweisungsoperatoren bei aktuellen Parametern zu unerwünschten Nebeneffekten führen! Makros führen im Gegensatz zu Funktionsaufrufen i.allg. zu einer Verbesserung des Laufzeitverhaltens, aber auf Kosten des Speicherplatzbedarfs. Beispiele für Makrodefintionen können der UNIX-Datei 'stdio.h' entnommen werden.

undef-Anweisung

Mit dieser Anweisung kann die Definition einer symbolischen Konstanten oder eines Makros vor dem Quelldateiende unwirksam gemacht werden. Dies erfolgt durch Angabe von

```
#undef Name
```

include-Anweisung

Durch Angabe von

```
#include <Dateiname>          oder
#include "Dateiname"
```

wird das Einfügen von kompletten Quelldateien (Dateien mit Standardvereinbarungen oder Standardmakros) veranlaßt. Die so eingefügten Dateien können weitere include-Anweisungen enthalten.
Wenn der Dateiname in spitze Klammern gesetzt wird, so erfolgt die Dateisuche in vordefinierten Directories. Wenn der Dateiname in Anführungsstriche eingeschlossen ist, dann wird zuerst das aktuelle Directory (in dem sich auch die Quelldatei befindet) durchsucht. Dabei sollte dann der vollständige oder relative Pfadname der include-Datei angegeben werden (vgl. auch Option -I des cc-Kommandos, Abschn. 7.7.).

if-Anweisung

Mit Hilfe der if-Anweisung ist es möglich, innerhalb einer Quelltextdatei bestimmte Programmteile (in Abhängigkeit von Ausdruckswerten oder definierten Konstanten) von der Übersetzung auszuschließen. Es sind drei Formen möglich:

(a)
```
#if konstanter_Ausdruck

          /* Programmteil 1 */

#else

          /* Programmteil 2 */

#endif
```

(b)
```
#ifdef Name

          /* Programmteil 1 */

#else

          /* Programmteil 2 */

#endif
```

(c)
```
#ifndef Name

        /* Programmteil 1 */

#else

        /* Programmteil 2 */

#endif
```

Der else-Zweig ist in allen drei Formen optional. Bei der Form (a) wird der konstante Ausdruck ausgewertet. Ist er wahr (d.h. ungleich Null), so werden nur die Anweisungen im Programmteil 1 übersetzt. Die Anweisungen im Prgrammteil 2 werden ignoriert. Sie werden (falls überhaupt vorhanden) nur dann übersetzt, wenn der Wert des konstanten Ausdrucks gleich Null (d.h. falsch) ist. In diesem Fall wird der Programmteil 1 von der Übersetzung ausgeschlossen.
In der zweiten und dritten Form wird getestet, ob der angegebene Name definiert ist (bei (a)) oder ob er nicht definiert ist (bei (b)). Entsprechend wird wiederum nur der Programmteil 1 oder 2 übersetzt. Die letzten beiden Formen finden hauptsächlich in Verbindung mit der Option -D des cc-Kommandos Anwendung, z.B. um spezielle Versionen zu generieren.

```
#if STANDALONE

        /* zusaetzliche Programmteile fuer Standalone-Vers. */

#endif
```

line-Anweisung

Sie hat die allgemeine Form

```
#line konstanter_Ausdruck
```

Optional kann dem Ausdruck noch ein Dateiname folgen.

```
#line 300 test.c
```

Durch eine line-Anweisung wird die Zeilennummer auf den Wert des konstanten Ausdrucks gesetzt. Die Angabe eines zusätzlichen Dateinamens bewirkt, daß der Name der aktuellen Quelldatei für den C-Compiler abgeändert wird (und zwar in den angegebenen Namen).
Diese Preprocessoranweisung wird hauptsächlich von Dienstprogrammen benutzt, die selber C-Quellprogramme (wie yacc und lex) erzeugen. In Anwenderprogrammen wird sie relativ selten eingesetzt.

7.3.9. Programmstruktur und Speicherklassen

Ein C-Programm besteht aus einer Anzahl von Funktionen und Variablen, die außerhalb von allen Funktionen vereinbart sind (den sog. externen Variablen - besser wäre die Bezeichnung globale Variablen).

Dabei ist eine Aufteilung des Quelltextes auf mehrere Dateien möglich, die getrennt übersetzt werden können. Bereits vorher übersetzte Funktionen können beim Verbinden aus Bibliotheken geladen werden. Das Bild 7.3 illustriert die Struktur von C-Programmen.

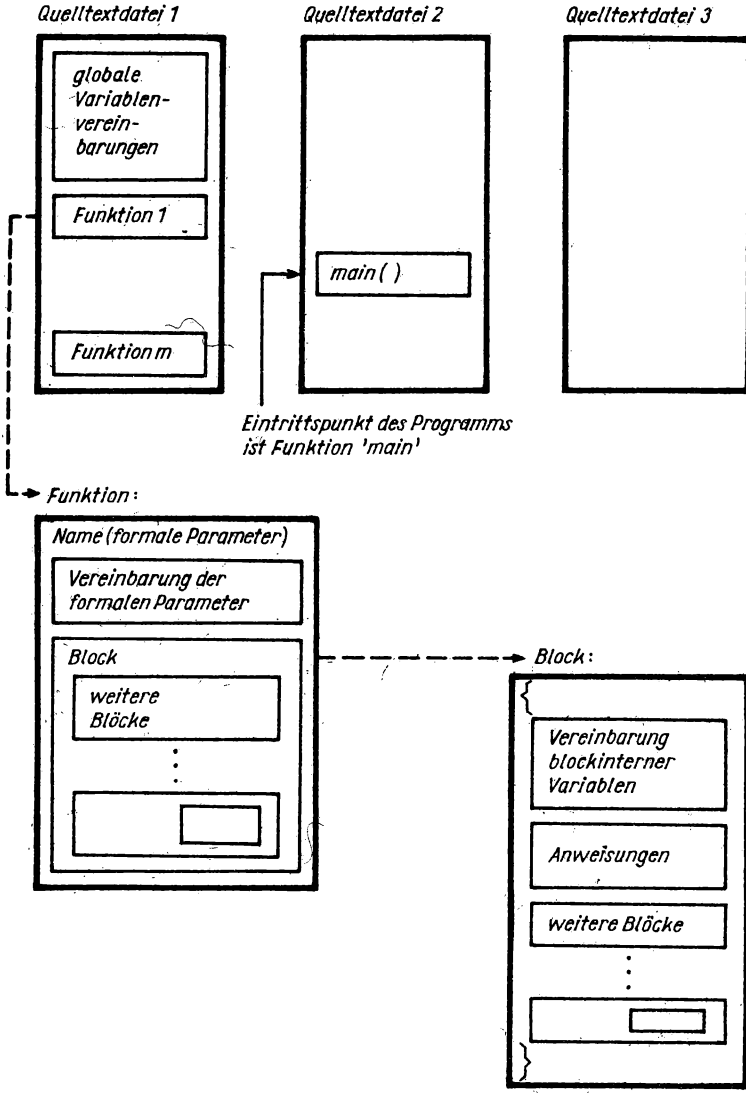


Bild 7.3. Struktur von C-Programmen

C ist im Sinne von PL/I oder Algol keine blockstrukturierte Sprache, da Funktionen nicht innerhalb anderer Funktionen (und damit Blöcken) definiert werden können. Andererseits können Variablen in einer blockstrukturierten Form vereinbart werden. Sie können der öffnenden geschweiften Klammer eines Blocks (Verbundanweisung) folgen, nicht nur derjenigen zu Beginn einer Funktion. Derartig vereinbarte Variablen heben jede identisch bezeichnete Variable in äußeren Blöcken auf.

```
if (i>0) {
     int i; /* Vereinbarung eines neuen 'i' */
     for (i=0; i<n; i++)

}
```

Der Gültigkeitsbereich der neuen Variablen 'i' ist der Block der "wahren" Verzweigung der if-Anweisung. Dieses steht in keiner Beziehung zu jedem anderen 'i' im Programm.
Die Blockstruktur bezieht sich auch auf externe Variablen, z.B. bezieht sich 'x' im folgenden Beispiel innerhalb der Funktion f' auf die interne double-Variable, außerhalb von 'f' dagegen auf die externe int-Variable.

```
int x;
...
f()
{
  double x;

}
```

Das gleiche gilt für Namen formaler Parameter.

```
int z;
...
f(z)
long z;
{

}
```

Innerhalb der Funktion 'f' bezieht sich auf den formalen Parameter und nicht auf die externe Variable.

Für Variablen und Funktionen können bei der Vereinbarung Speicherklassen festgelegt werden. Dabei ist immer nur die Angabe einer Speicherklasse möglich. Bei der Diskussion der Speicherklassen muß zwischen Variablen und Funktionen unterschieden werden.
Bei Variablen bestimmt die Speicherklasse die Lebensdauer, den Geltungsbereich und sie legt fest, ob Speicherplatz reserviert wird oder nicht. Folgende Speicherklassen sind möglich:

```
auto      register      static      extern
```

Die ersten beiden Speicherklassen 'auto' und 'register' sind nur auf intern vereinbarte Variablen (d.h. innerhalb von Funktionen und Blöcken) anwendbar.
Variablen der Speicherklasse 'auto' werden im Stack verwaltet, d.h., sie werden bei jedem Eintritt in die Funktion (bzw. Block) neu angelegt und müssen demzufolge neu initialisiert werden. Der Geltungsbereich und die Lebensdauer solcher Variablen sind auf die Funktion (bzw. Block) beschränkt. Die Speicherklasse 'register' ist eine Unterklasse der automatischen Variablen. Bezüglich Geltungsbereich und Lebensdauer gilt das gleiche wie bei der Speicherklasse 'auto'. Der Compiler versucht, aber als

Speicherplatz für die Variablen CPU-Register zu verwenden. Dem sind natürlich prozessorbedingte Grenzen durch den Rechner, auf dem man arbeitet, gesetzt.

Bei Variablen der Speicherklasse 'static' muß zwischen interner (d.h. innerhalb einer Funktion bzw. eines Blocks) und externer (d.h. außerhalb einer Funktion bzw. eines Blocks) Vereinbarung unterschieden werden.

(a) interne statische Variable*

Sie wird beim ersten Eintritt in die Funktion bzw. in den Block angelegt. Ihr Wert bleibt erhalten, auch wenn die Funktion verlassen wird, d.h., sie ist bei Wiedereintritt in die Funktion (bzw. Block) mit ihrem alten Wert verfügbar. Der Geltungsbereich einer solchen Variablen ist die Funktion bzw. der Block. Eine explizite Initialisierung in der Vereinbarung einer internen statischen Variablen wird nur beim ersten Durchlaufen der Funktion bzw. des Blocks durchgeführt.

(b) externe statische Variable

Sie wird außerhalb von allen Funktionen vereinbart und stellt eine globale Variable dar. Es wird Speicherplatz reserviert. Ihr Geltungsbereich ist aber auf die Quelltextdatei beschränkt, d.h., sie kann nicht von anderen Quelltextdateien importiert werden. In diesem Fall dient die Speicherklasse 'static' dazu, die Variable vor anderen Quelltextmodulen "geheim" zu halten.

Werden Variablen außerhalb von Funktionen vereinbart, so stehen sie global zur Verfügung, einmal allen Funktionen des Quelltextmoduls, in dem sie vereinbart wurden, und auch anderen Quelltextdateien (außer bei 'static').

Zugriff innerhalb eines Quelltextmoduls:

```
char a;
...
f()
{
  ...
  /* Verwendung der globalen Groesse a in der Funktion
     moeglich, eine extern-Vereinbarung der Form:
     extern char a;  ist redundant
     Geltungsbereich: Quelltextmodul */

}
```

Für den Zugriff von einer anderen Quelltextdatei aus, ist eine Vereinbarung der entsprechenden Variablen mit der Speicherklasse 'extern' notwendig.

```
extern char a;       /* es wird kein Speicherplatz reserviert */
extern double wert; /* es handelt sich um einen Verweis auf  */
                     /* Variablen, die in anderen Modulen     */
                     /* extern (nicht static) def. wurden     */
```

Beim Import externer Variablen gibt es zwei Möglichkeiten:

- den globalen Import
  (für die gesamte Quelltextdatei, extern-Vereinbarung muß
  außerhalb von Funktionen erfolgen,
  Geltungsbereich: Quelltextdatei)

- den lokalen Import
  (nur für eine Funktion (bzw. Block), extern-Vereinbarung
  erfolgt innerhalb einer Funktion (bzw. Block),
  Geltungsbereich: Funktion bzw. Block)

Zugriff von einem anderem Quelltextmodul aus:

global:

```
extern double wert;
...
fktabc()
{
  ...
  /* Verwendung von 'wert'
     Geltungsbereich: Quelltextmodul und Modul, wo 'wert'
                      definiert ist */

}
```

lokal:

```
...
fktabc()
{
  extern double wert;
  ...
  /* Verwendung von wert
     Geltungsbereich: Funktion fktabc und Modul, wo
                      wert definiert wurde */

}
```

Im Gegensatz zu Variablen der Speicherklässen 'auto' und 'register' erfolgt bei statischen Variablen (intern und extern) und bei globalen Variablen eine automatische Initialisierung (numerische Variablen werden mit Null und Zeichenvariablen mit '\0' initialisiert).
Interne Variablen ('auto' und 'register') müssen explizit initialisiert werden, damit sie bei Eintritt in die Funktion (bzw. Block) keine undefinierten Werte besitzen.
Fehlt bei internen Variablen eine Speicherklassenangabe, so wird implizit 'auto' angenommen.

Die eben für Variablen erläuterten Speicherklassenregeln treffen auch auf Funktionsnamen zu. Die Speicherklassen 'auto' und 'register' scheiden für Funktionen aus, da Funktionen in C nicht innerhalb von anderen Funktionen

(und damit Blöcken) definiert werden dürfen.
Funktionen sind implizit global (Speicherklasse 'extern'), d.h., sie können innerhalb und außerhalb der Quelltextdatei verwendet werden. Erfolgt die Anwendung einer Funktion, die einen Rückgabewert ungleich 'int' liefert, so muß der Funktionsname vereinbart werden.

Beispiele (innerhalb des gleichen Quelltextmoduls):

lokal:

```
abc()
{
  long random();     /* lokale Vereinbarung, Verwendung der
                        Funktion random in Funktion abc, es
                        kann auch: extern random();
                        geschrieben werden */

}

long random(a)
long a;
{

}
```

global:

```
long random();       /* globale Vereinbarung, Funktion random
                        kann in allen Funktionen der Quell-
                        textdatei verwendet werden */

abc()
{

}
...
def()
{

}
...
long random(a)
long a;
{

}
```

Bei Verwendung innerhalb einer anderen Quelltextdatei ist die Vereinbarung der Funktion als 'extern' notwendig.

global:

```
extern long random();     /* globaler Import einer Funktion aus
                             einem anderen Modul, Verwendung in
                             allen Funktionen moeglich */
...
f1()
{

}
...
f2()
{
  ...
}
```

lokal:

```
...
f1()
{
  extern long random();       /* lokaler Import einer Funktion
                                 aus einem anderen Modul, Ver-
                                 wendung nur in f1 moeglich */

}
```

Im Zusammenhang mit Funktionen bleibt noch die Speicherklasse 'static' übrig. Sie kann wie bei Variablen dazu verwendet werden, daß Funktionen für andere Quelltextdateien nicht verfügbar sind, sondern nur Funktionen aus dem gleichen Quelltextmodul.

```
abc()
{
  long random();

}

static random(a)          /* Funktion random ist nur in ihrer */
long a;                   /* Quelltextdatei bekannt, es be-   */
{                         /* steht keine Moeglichkeit, sie in */
                          /* anderen Modulen zu verwenden     */
}
```

7.4. Programmbeispiel

Nachdem die Beschreibung der Sprachelemente von C abgeschlossen ist und bevor auf die unter UNIX zur Verfügung stehenden Standardbibliotheksfunktionen eingegangen wird, nun zur Illustration ein Programmbeispiel. Es ist eine erste Umsetzung des bekannten Conway-Spiels (oft auch als Lebensspiel bezeichnet) in ein C-Programm.
Die Spielregeln sind schnell erläutert. Ausgehend von einem Spielfeld und

einer Anfangsbelegung mit Steinen ergibt sich Nachfolgebelegung wie folgt:

- hat ein Stein genau zwei Nachbarsteine, so bleibt er erhalten
- hat ein Stein mehr als drei oder weniger als zwei Nachbarsteine, so geht er verloren
- bei genau drei Nachbarsteinen wird auf einem leeren Feldpunkt ein Stein "geboren" (ein Stein mit drei Nachbarn bleibt erhalten)
- Jeder Feldpunkt bzw. Stein besitzt acht Nachbarpunkte:

```
1 2 3
4 * 5
6 7 8
```

Das Programm gibt die Entwicklung der Figuren auf dem Bildschirm aus. Dabei können interessante Erscheinungen, wie "Zellteilungen" und Fortbewegungen über das Spielfeld beobachtet werden. Als Endkonfigurationen treten feststehende oder pulsierende Elemente auf. Das Bild 7.4 zeigt das C-Quellprogramm.

```
/* Conway-Spiel */

#define ZEILEN 20
#define SPALTEN 60
#define STERN '#'
#define LEER '.'
#define CURSORUP '\032'

char b[ZEILEN][SPALTEN], z[ZEILEN][SPALTEN];

main()
{
    int n;
    start();
    do {
        ausgabe();
        leben();
    } while (n=move());
    for (n=1; n<ZEILEN-1; n++) putchar('\n');
    printf("Ende\n");
}

start()
{
        int c, i, j;
        /* Loeschen des Feldes b[][] */
        for (i=0; i<ZEILEN+1; i++)
            for (j=0; j<SPALTEN+1; j++) b[i][j]=LEER;
        printf("\f\n\tWillkommen zum CONWAY-Spiel!");
        printf("\n\tGeben Sie bitte NL oder eine Zahl zwischen 1 und 6 ein: ");
        switch (c=getchar()) {
            case '1':   /* Segler */
            b[1][8]=b[2][9]=b[3][7]=b[3][8]=b[3][9]=STERN; break;
            case '2':   /* Verkehrsampel */
            b[9][30]=b[10][29]=b[10][30]=b[10][31]=STERN; break;
```

Bild 7.4. Entwurf eines Programms zum Conway-Spiel

```
            case '3':   /* Kroete */
            b[9][29]=b[9][30]=b[9][31]=b[10][28]=b[10][29]=b[10][30]=STERN; break;
            case '4':   /* Pentomino */
            b[9][30]=b[9][31]=b[10][29]=b[10][30]=b[11][30]=STERN; break;
            case '5':   /* Uhr */
            b[8][29]=b[9][30]=b[9][31]=b[10][28]=b[10][29]=b[11][30]=STERN; break;
            case '6':   /* Pi */
            b[9][29]=b[9][30]=b[9][31]=b[10][29]=b[10][31]=b[11][29]=b[11][31]=STERN; break;
            default:    /* Eingabe Anfangsbelegung */
            do {
               printf("\tEingabe Anfangsbelegung - Zeile/Spalte: ");
                scanf("%d %d", &i, &j);
                while((c=getchar())!='\n');
                b[i][j]=STERN;
                ausgabe();
                putchar(CURSORUP);
                printf("Start?");
                if((c=getchar())!='\n') break;
                putchar(CURSORUP);
            } while(1);
            putchar(CURSORUP);
        };
        putchar('\n');
}

leben()
{
        int i,j,n;
        for (i=1; i<ZEILEN; i++) {
            for (j=1; j<SPALTEN; j++) {
                n=nachbar(i,j);
                if (n==2) {z[i][j]=b[i][j]; continue;}
                if (n==3) z[i][j]=STERN; else z[i][j]=LEER;
            }
        }
}

move()
{
        int i, j, v; v=0;
        for (i=1; i<ZEILEN; i++)
            for (j=1; j<SPALTEN; j++) {
                if (b[i][j] != z[i][j]) v++;
                b[i][j] = z[i][j];
            }
        return(v);
}

ausgabe()
{
        int i,j;
        for (i=1; i<ZEILEN; i++) {
            putchar('\t');
            for (j=1; j<SPALTEN; j++) putchar(b[i][j]);
            putchar ('\n');
        }
        for (i=0; i<ZEILEN-1; i++) putchar(CURSORUP);
}
```

Bild 7.4. (Fortsetzung)

```
nachbar(i,j)
int i,j;
{
        int n;
        n=0;
        if(b[i-1][j-1]==STERN) n++;
        if(b[i-1][j]==STERN)   n++;
        if(b[i-1][j+1]==STERN) n++;
        if(b[i][j-1]==STERN)   n++;
        if(b[i][j+1]==STERN)   n++;
        if(b[i+1][j-1]==STERN) n++;
        if(b[i+1][j]==STERN)   n++;
        if(b[i+1][j+1]==STERN) n++;
        return(n);
}
```

Bild 7.4. (Fortsetzung)

Erläuterungen zum Programm:

- Das Programm beginnt mit fünf define-Anweisungen, die Konstanten vereinbaren (Feldgröße, Belegungen). Die letzte define-Anweisung legt das Steuerzeichen für die Funktion "Cursor hoch" fest. Sie ist abhängig vom konkret verwendeten Terminaltyp.

- Danach erfolgt die Vereinbarung des Spielfeldes b [][]und einer Kopie z[][], in der immer die errechnete nächstfolgende Belegegung abgelegt wird. Diese Felder sind global, da ihre Vereinbarung außerhalb von Funktionen erfolgt.

- Der Algorithmus wurde in sechs Funktionen aufgeteilt. Das Hauptprogramm 'main' ruft die Startfunktion auf und geht dann in eine Schleife über, die so lange ausgeführt wird, wie auf dem Spielfeld noch Bewegungen ausgeführt werden (Rückgabewert von 'move' ungleich Null). Durch die Cursorsteuerung erfolgt die Ausgabe auf einem stehenden Bild.
  Die Startfunktion führt die Anfangsinitialisierungen (Löschen Spielfeld und Festlegen der Anfangsbelegung) aus. Dabei kann der Bediener mit einer festen Startbelegung (Eingabe 1 bis 6) beginnen oder er legt per Dialog selbst eine Startbelegung fest.
  Die Funktion 'leben' ermittelt für jeden Feldpunkt nach den Spielregeln seine Folgebelegung und trägt diese im Zwischenfeld z[][]ein. Der Rand des Spielfeldes bleibt immer leer! Sie benutzt die Funktion 'nachbar' die die Anzahl der Nachbarsteine für einen Feldpunkt zurückgibt.
- Die Funktion 'move' überträgt die neue Belegung aus dem Zwischenfeld z[][] in das Spielfeld b [][]und ermittelt gleichzeitig die Anzahl der Veränderungen.
  Die Ausgabefunktion 'ausgabe' gibt die Spielfeldbelegung auf dem Terminal aus. Bis auf die Funktion 'nachbar' arbeiten alle Funktionen ohne Parameter, da sie nur die globalen Spielfelder manipulieren.

Diese Version ist natürlich an vielen Stellen verbesserungswürdig. Ansatzpunkte für Verbesserungen und Erweiterungen könnten z.B. sein:

- Die Ausgabe sollte nur auf solche Felder beschränkt werden, Veränderungen auftreten.
- Bei der Manipulation der Feldinhalte könnten auch Zeiger eingesetzt werden.
- Die Cursorsteuerung könnte durch Auswertung der UNIX-Datei 'termacp' unabhängig von einem konkreten Terminal gestaltet werden (siehe dazu UNIX-Bibliotheksfunktion TERMLIB(3)).
- Bei der Gestaltung des Dialogs und des zeitlichen Ablaufs (evtl. Anhalten) sind Veränderungen denkbar. Bei dieser Version kann bei einem pulsierenden Schlußbild nur über ein Terminalinterrupt (CRTL-C oder DEL) abgebrochen werden.

Diese Hinweise sollen als Anregungen für eigene Experimente und Weiterentwicklungen bzw. Abwandlungen dienen.

7.5. Standardbibliotheken

Ein-/Ausagbefunktionen sind keine Bestandteile von C. Sie werden über Standardbibliotheksfunktionen realisiert. Es existiert ein fester Satz von Basisfunktionen zur Ein-/Ausgabe und zur Dateiarbeit. Diese einheitliche Schnittstelle wird bei jeder Implementation mit Hilfe der Ein-/Ausgabeschnittstelle des Betriebssystemkerns (Systemaufrufe) nachgebildet.
Der Anwender kann für die Ein-/Ausgabe natürlich auch diese Funktionen über Systemaufrufe nutzen. Die dort zur Verfügung stehenden Funktionen sind im UNIX-Programmers-Manual Teil 2 ([17]) zusammengefaßt (siehe auch Abschn. 2.2.). Diese Verfahrensweise geht aber auf Kosten der Portabilität der Anwenderprogramme.

Die Bibliotheksfunktionen sind in verschiedenen Bibliotheken enthalten, die im Directory "lib' abgelegt sind.

```
libc.a          Standardbibliothek
libm.a          mathematische Funktionen
```

Bei der Nutzung von Ein-/Ausgabefunktionen der Standardbibliothek muß Beginn der Quelldatei die Preprocessoranweisung

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

stehen. Dadurch werden die in der Datei 'stdio.h' enthaltenen Definitionen von Konstanten und Makros und Vereinbarungen von Ein-/Ausgabefunktionen ins Quellprogramm integriert. Diese Zeile kann bei der Nutzung von Grundfunktionen (z.B. printf) entfallen.

In den nachfolgenden Punkten werden die Bibliotheksfunktionen zur Ein-/Ausgabe, zur Dateiarbeit und der Satz der mathematischen Funktionen beschrieben. Daneben wird noch eine kleine Auswahl von nützlichen Funktionen, die der Erleichterung des Programmierens dienen, erläutert.

7.5.1. Ein-/Ausgabefunktionen

Die Ein-/Ausgabefunktionen arbeiten implizit mit der Standardein-/-ausgabe, die normalerweise mit dem Terminal verbunden ist. Umlenkungen dieser Datenströme wurden im Abschn. 6. (Shell) beschrieben.

Einzelzeichenein-/-ausgabe

Zur Einzelzeichenein-/-ausgabe existieren zwei Funktionen

```
int getchar()

putchar(c)
char c;
```

Die Funktion 'getchar' liefert als Wert das nächste Zeichen von der Standardeingabe (als int-Wert, wegen EOF). Die Konstante 'EOF' kennzeichnet dabei das Dateiende. Die Funktion 'putchar' gibt das als Argument angegebene Zeichen an die Standardausgabe aus.

Formatierte Ein-/Ausgabe

Für die formatierte Ein-/Ausgabe existieren die zwei Funktionen 'printf' und 'scanf'. Die Funktion 'printf' ist vordefiniert in der Form:

```
printf(format, Argumente ...)
char *format;
```

Die Funktion 'printf' konvertiert, formatiert und gibt die Argumentwerte unter Steuerung der Formatzeichenkette an die Standardausgabe aus. Die Steuerzeichenkette enthält zwei Objekttypen: einfache Zeichen, die in den Ausgabestrom kopiert werden, und Konvertierungsvereinbarungen, die die Konvertierung und Ausgabe des folgenden Arguments bewirken. Jede Konvertierungsvereinbarung beginnt mit einem Prozentzeichen (%), dem ein Konvertierungszeichen folgt. Als Konvertierungszeichen können auftreten:

d	Konvertierung des Arguments in Dezimalnotation
o	Konvertierung des Arguments in vorzeichenlose Oktalnotation (ohne führende Nullen)
	Konvertierung des Arguments in vorzeichenlose Hexadezimalnotation (ohne führendes 0x)
	Konvertierung des Arguments in vorzeichenlose Dezimalnotation
	Argument wird als einzelnes Zeichen genommen
	Argument wird als Zeichenkette genommen
	Argument wird als Gleitkommazahl interpretiert und in Exponentenschreibweise ausgegeben
f	Argument wird als Gleitkommazahl interpretiert und in Festkommadarstellung (ohne Exponent) ausgegeben
g	es wird %e oder %f genommen, jeweils die kürzere Darstellung

Zwischen dem Prozent- und Konvertierungszeichen können noch Längenangaben

und weitere Modifikationen auftreten.

- Ein Minuszeichen bewirkt die Linksausrichtung des Arguments in seinem Ausgabebereich.
- Eine Ziffernfolge legt den minimalen Ausgabebereich für den Ausgabebereich fest. Ist die Darstellung des Argumentwerts kürzer, so werden Leerzeichen vorangestellt. Beginnt die Zahl mit 0, so werden Nullen vorangestellt.
- Ein Punkt, der die Bereichsgröße von der nächsten Ziffernfolge (Genauigkeit) trennt.
- Eine Ziffernfolge (Genauigkeit), die die maximale Anzahl von auszugebenden Zeichen einer Zeichenkette oder die Anzahl der Stellen nach dem Komma (bei %e und %f) festlegt. Standardmäßig werden sechs Ziffern nach dem Komma ausgegeben.
- Das Zeichen 'l' zeigt an, daß das zugehörige Argument vom Typ 'long' ist.

Das Prozentzeichen kann innerhalb der Formatzeichenkette durch Angabe von '%%' ausgegeben werden.

Beispiele für die Anwendung von 'printf':

Um ein Datum in der Form "Dienstag, den 3.November, 15:20" auszugeben, kann folgende Anweisung benutzt werden. ('wochentag' und 'monat' sind Zeiger auf Zeichenketten)

```
printf("%s, den %d.%s, %02d:%02d", wochentag, tag, monat, h, min);
```

Um eine Gleitkommazahl mit sechs Stellen Genauigkeit auszugeben:

```
printf("y-Wert= %.6f", y);
```

Bei der Anwendung von 'printf' ist vor allem auf die Übereinstimmung der Anzahl und der Typen der Argumente mit den Angaben in der Steuerzeichenkette zu achten! Die Funktion 'printf' führt keine Typkonvertierung durch.

Die Funktion 'scanf' ist die zu 'printf' analoge Eingabefunktion, die Umwandlungsmöglichkeiten in umgekehrter Richtung bietet.

```
scanf(format, Zeiger ...)
char *format;
```

Sie liest Zeichen von der Standardeingabe, interpretiert sie entsprechend den Formatangaben in der Steuerzeichenkette und speichert die Ergebnisse in den dazugehörigen Objekten, auf die die Argumente verweisen.
Die Steuerzeichenkette enthält Konvertierungszeichen, die zur direkten Interpretation der Eingabefolgen benutzt werden. Die Steuerzeichenkette darf folgendes enthalten:

- Leer-, Tabulator- oder Newlinezeichen (sogenannte "white space characters"), werden ignoriert.
- Bestimmte Zeichen (nicht %), mit denen Zeichen des Eingabestroms übereinstimmen müssen, damit sie übernommen werden.

- Konvertierungsvereinbarung, aus dem Prozentzeichen und Konvertierungszeichen bestehend.

Eine Konvertierungsvereinbarung legt die Konvertierung des nächsten Eingabebereichs fest. Ein Eingabebereich ist definiert als eine Zeichenkette von "Nichtsteuerzeichen". Der Bereich geht bis zum nächsten Steuerzeichen oder bis zur Bereichsgrenze, wenn diese spezifiziert wurde. Das bedeutet auch, daß 'scanf' über Zeilengrenzen hinweg liest, um seine Eingabe zu finden, da Newline ein Steuerzeichen ist.

Als Konvertierungszeichen können auftreten:

d Eine Dezimalzahl wird erwartet. Das zugehörige Argument muß ein Zeiger auf eine integer-Größe sein.
Eine Oktalzahl (mit oder ohne führende Nullen) wird erwartet. Das zugehörige Argument muß ein Zeiger auf eine integer-Größe sein.
Eine Hexadezimalzahl (mit oder ohne führendes Ox) wird erwartet. Das zugehörige Argument muß ein Zeiger auf eine integer-Größe sein.
Ein einzelnes Zeichen wird erwartet. Das zugehörige Argument muß ein Zeichenzeiger sein. Das normale Übergehen der Trennzeichen ("white spaces") ist in diesem Fall unter- drückt. Um das nächste "Nichtsteuerzeichen" zu lesen, kann das Format '%1s' benutzt werden.
Eine Zeichenkette wird erwartet. Das zugehörige Argument muß ein Zeichenzeiger sein, der auf ein Feld von Zeichen zeigt, das groß genug sein muß, die Zeichenkette sowie das hinzugefügte abschließende '\0' aufzunehmen.

f oder e Eine Gleitkommazahl wird erwartet. Das zugehörige Argument muß ein Zeiger auf den Typ 'float' Das Eingabeformat besteht aus einem optionalen Vorzeichen, einer Folge von Ziffern, die einen Dezimalpunkt enthalten kann und möglicherweise aus einem Exponentenfeld (E oder e gefolgt von Integerzahl).

[...] Kennzeichnet eine Zeichenkette. Die in eckigen Klammern eingeschlossenen Zeichen legen einen Zeichensatz fest, aus dem eine Zeichenkette gebildet werden kann. Wenn als erstes Zeichen ein '↑' angegeben ist, so ist die Wirkung genau umgedreht, d.h., die Eingabe eines angegebenen Zeichens beendet die Eingabe. Das zugehörige Argument muß ein Zeiger auf ein Zeichenfeld sein.

Zwischen Prozent- und Konvertierungszeichen kann noch eine Längenangabe oder eine Kennzeichnung zur Zuweisungsunterdrückung auftreten.

- Eine Zahl legt die maximale Größe des Eingabebereichs fest.
- Ein Stern (*) kennzeichnet die Unterdrückung der Zuweisung, d.h., der Eingabebereich wird übergangen.
- Ein 'l' in Verbindung mit den Formaten 'd', und 'x' legt Variablen vom Typ 'long' fest. Bei und 'f' bedeutet ein 'l' den Typ 'double'.
- Ein 'h' in Verbindung mit den Formaten 'd', und legt Variablen vom Typ 'short int' fest.

Beispiele für die Anwendung von 'scanf'

Der Aufruf

```
float a; int b; char name[20];
scanf("%f %d %s", &a, &b, name);
```

mit der Eingabezeile

```
12.34E-5  15  hugo
```

weist der Variablen a den Wert 0.0001234, der Variablen 'b' den Wert 15 und der Zeichenkette name den Wert hugo\0 zu.
Der Aufruf

```
int a; float b; char name[20];
scanf("%3d %f %*d %[123456]", &a, &b, name);
```

mit der Eingabe

```
12345  6789  123ab45
```

weist den Wert 123 der Variablen 'a' , der Variablen 'b' den Wert 45.0, überspringt 6789 und weist 'name' die Zeichenkette 123\0 zu. Der nächste Aufruf von 'getchar' liefert das Zeichen

Die Funktion 'scanf' beendet die Eingabe, wenn die Steuerzeichenkette (format) abgearbeitet ist oder wenn die Eingabe nicht mit der Formatangabe harmoniert. Es sei nochmals wiederholt, daß die Argumente von 'scanf' Zeiger sein müssen. Es ist ein häufiger Fehler

```
scanf("%d", n);
```

anstelle von

```
scanf("%d", &n);
```

zu schreiben.

Interne Formatkonvertierung

Zu den Funktionen 'printf' und 'scanf' gibt es die analogen Funktionen 'sprintf' und 'sscanf', die die gleichen Konvertierungen durchführen, jedoch mit Zeichenketten anstelle der Standardein-/-ausgabe arbeiten.

```
sprintf(s, format, Argumente ...)
char *s, *format;

sscanf(s, format, Zeiger ...)
char *s, *format;
```

Die Funktion 'sprintf' konvertiert die Argumente entsprechend der Steuerzeichenkette 'format' und speichert das Ergebnis in die Zeichenkette ' ab. Diese muß natürlich groß genug sein, um das Ergebnis aufnehmen können.
Die Funktion 'sscanf' analysiert entsprechend der Steuerzeichenkette 'format' den Inhalt der Zeichenkette 's' und speichert die Ergebnisse in den Variablen, auf die die Argumente zeigen.
Diese Funktionen können zur Konvertierung von numerischen Werten in Zeichenketten (und umgekehrt) verwendet werden. Mit ihrer Hilfe lassen sich auch Zeichenketten initialisieren, zuweisen und verketten.

7.5.2. Funktionen zur Dateiarbeit

Die Philosophie der Dateiarbeit unter UNIX wurde bereits im Abschn. 2. dargelegt. Dateien können von Anwenderprogrammen aus über Systemaufrufe oder Bibliotheksfunktionen direkt manipuliert werden. In diesem Punkt werden die für die Dateiarbeit zur Verfügung stehenden Bibliotheksfunktionen beschrieben.

Bei der Anwendung des Schutzes von Teilen einer Datei gegen gleichzeitigen Zugriff (record locking) ist beim Zugriff auf verriegelte Teile zu beachten, daß der zugreifende Prozeß "schlafen" geht, so lange, bis der Schutz des entsprechenden Dateibereichs wieder aufgehoben wird.

Für die Arbeit mit Dateien sind sog. Dateizeiger (filepointer) notwendig, die eröffnete Dateien im Programm kennzeichnen. Dateizeiger verweisen auf eine Struktur 'FILE' (Dateikennsatz). Sie dienen in allen weiteren Dateioperationen zur Spezifizierung einer konkreten Datei (anstelle vom Dateinamen). Standardmäßig sind immer drei Dateien automatisch eröffnet, die folgende Dateizeiger haben und i.allg. mit dem Terminal verbunden sind.

Dateideskriptor	Dateizeiger	
0	stdin	Standardeingabe
1	stdout	Standardausgabe
2	stderr	Standardfehlerausgabe

Daneben existiert pro Datei ein interner Dateizeiger, der die Stelle für den nächsten Dateizugriff (Lesen oder Schreiben) markiert. Dieser Zeiger wird nach jeder Schreib-/Leseoperation automatisch weiter gesetzt. Er kann aber auch per Programm positioniert werden. Alle zur Dateiarbeit notwendigen Vereinbarungen (z.B. der Struktur 'FILE') werden durch die Preprocessoranweisung

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

ins Anwenderprogramm integriert.

In der folgenden Übersicht werden die Funktionen zur Dateiarbeit aufgeführt.

Eröffnen von Dateien

```
FILE *fopen(filename, type)
char *filename, *type
```

Die Funktion 'fopen' eröffnet eine Datei. Der Dateiname und die Art des Zugriffs werden als Zeichenkettenargumente übergeben. Der Typ kann sein:

	für Lesen (read)
	für Schreiben (write)
a	für Anhängen ans Dateiende (append)

Die Funktion gibt als Wert einen Dateizeiger zurück (NULL wenn Datei nicht eröffnet werden kann), der für alle weiteren Dateizugriffe benutzt werden muß.

Lesen und Schreiben von Dateiinhalten

Zeichenweise Ein-/Ausgabe und Zugriff auf Speicherworte:

int getc(fp) FILE *fp; int fgetc(fp) FILE *fp;	Eingabe des nächsten Zeichens von der angegebenen Datei. 'fgetc' ist eine echte Funktion, 'getc' dagegen ein Makro (deshalb funktionieren Anweisungen wie getc(*f++) nicht!) Bemerkung: getchar() ist identisch mit getc(stdin)
int getw(fp) FILE *fp;	Eingabe des nächsten Wortes (int-Größe) von der angegebenen Datei. Diese Funktion führt keine automatische Ausrichtung auf Wortgrenzen durch.
putc(c, fp) char c; FILE *fp; fputc(c, fp) char c; FILE *fp;	Ausgabe eines Zeichens an die angegebene Datei. 'fputc' ist echte Funktion, 'putc' dagegen ein Makro (deshalb funktionieren Anweisungen wie putc(c, *f++) nicht!) Bemerkung: putchar(c) ist identisch mit putc(c,stdout)
putw(w, fp) FILE *fp;	Ausgabe eines Wortes (int-Größe) an die angegebene Datei. Es erfolgt keine automatische Ausrichtung auf Wortgrenzen!
ungetc(c, fp) char c; FILE *fp	Durch diese Funktion wird ein bereits gelesenes Zeichen wieder in die Eingabedatei zurückgestellt, d.h., beim nächsten Lesezugriff (getc) wird erneut dieses Zeichen gelesen. Wenn ein Zurückstellen nicht möglich ist, wird von 'ungetc' der Wert 'EOF' geliefert.

Zeilenweise Ein-/Ausgabe:

```
char *gets(s)  Einlesen einer Zeile (durch Newline beendet) von der
char *s;       Standardeingabe in die Zeichenkette

char *fgets(s,     fp)    Einlesen von n-1 Zeichen (oder bis zum
char *s;                  Newline) aus der angegebenen Datei in die
FILE *fp;                 Zeichenkette

Beide Funktionen fügen automatisch das Endekennzeichen '\0' hinzu.

puts(s)        Ausgabe der Zeichenkette     und eines abschließenden
char *s;       Newlinezeichens an die Standardausgabe.

fputs(s, fp)   Ausgabe der Zeichenkette 's' (ohne abschließendes
char *s;       Newline) an die angegebene Datei.
FILE *fp;
```

Gepufferte Ein-/Ausgabe:

```
fread(ptr, sizeof(*ptr),     fp)    Lesen von n Feldern der Länge
FILE *fp;                           'sizeof(*ptr)' in einen durch
char *ptr;                          'ptr' adressierten Puffer aus der
                                    angegebenen Datei.

fwrite(ptr, sizeof(*ptr),    fp)    Schreiben von n Feldern der Länge
FILE *fp;                           'sizeof(*ptr)' aus einem durch
char *ptr;                          'ptr' adressierten Puffer in die
                                    angegebene Datei.

Beide  Funktionen liefern als Wert die Anzahl der tatsächlich übertragenen
Felder.
```

Formatierte Ein-/Ausgabe (Datei):

```
fprintf(fp, format, Argumente)     Dies sind die zu 'printf' und
FILE *fp;                          'scanf' analogen Funktionen.
char *format;                      Anstelle der Standardein-/-ausgabe
                                   arbeiten sie mit den angegebenen
fscanf(fp, format, Zeiger ...)     Dateien (erstes Argument).
FILE *fp;
char *format;
```

Positionieren des internen Dateizeigers

```
fseek(fp, offset, type)  Setzen des internen Dateizeigers der
FILE *fp;                angegebenen Datei auf eine neue Position.
long offset;             Diese berechnet sich wie folgt:
                              type:
                                0       Dateibeginn + offset
                                1       akt. Position + offset
                                2       Dateiende + offset
```

```
long ftell(fp)
FILE *fp;
```
Übergibt als Wert (long-Größe) die aktuelle Position des internen Dateizeigers für die angegebene Datei (in Bytes relativ zum Dateianfang).

```
rewind(fp)
FILE *fp;
```
Positionieren des internen Dateizeigers an den Dateianfang (ist identisch mit fseek(fp, 0L, 0)).

Ungepufferte Dateiein-/-ausgabe

Alle Ein-/Ausgaben (außer Standardfehlerausgabe 'stderr') werden vom System aus Gründen der Effektivität gepuffert ausgeführt. Will der Anwender die Pufferinhalte selber überwachen, so ist dies über die Funktion 'setbuf' möglich.

```
setbuf(fp, buf)
FILE *fp;
char *buf;
```
Setzt das Zeichenfeld 'buf' als Ein-/Ausgabepuffer für nachfolgende Operationen. Die Konstante 'BUFSIZ' enthält die notwendige Größe

```
char buf[BUFSIZ];
```

Pufferbereiche können auch über die Funktion malloc(3) bereitgestellt werden.
Wird für 'buf' der Zeigerwert 'NULL' angegeben, so erfolgt die Ein-/Ausgabe völlig ungepuffert.

```
fflush(fp)
FILE *fp;
```
Veranlaßt die Ausgabe aller zwischengepufferten Daten an die angegebene Datei. Die Datei bleibt eröffnet.

Schließen von Dateien

```
fclose(fp)
FILE *fp;
```
Bewirkt das Schließen der angegebenen Datei. Bei gepufferter Arbeitsweise werden die Pufferinhalte geleert und freigegeben. Zum Programmende werden alle eröffneten Dateien automatisch geschlossen.

7.5.3. Mathematische Standardfunktionen

Die mathematischen Standardfunktionen sind in der Bibliothek 'libm.a' zusammengefaßt. Der Anwender muß durch die folgende Preprocessoranweisung

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

die für die Arbeit mit mathematischen Funktionen notwendigen Vereinbarungen in sein Programm integrieren. Beim Übersetzen ist auf die Option '-lm' zu achten.
Da sämtliche Gleitkommaoperationen mit doppelter Genauigkeit ausgeführt werden, liefern die meisten mathematischen Funktionen einen Wert vom Typ 'double'.

mathematische Funktionen:

```
abs(i)                  Absolutwert einer integer-Größe
                        Funktionswert ist vom Typ 'int'

double exp(x)           Exponentialfunktion zur Basis e (2.71828...)
double

double log(x)           Natürlicher Logarithmus zur Basis e
double x;

double log10(x)         Natürlicher Logarithmus zur Basis 10
double x;

double pow(x, y)        x hoch y
double x, y;            (anstelle des aus anderen Sprachen her bekannten
                        Exponentialoperators)

double sqrt(x)          Quadratwurzel
double x;

double floor(x)         Liefert die größte ganze Zahl, die nicht größer
double x;               als der Wert von    ' ist.

double ceil(x)          Liefert die kleinste ganze Zahl, die nicht
double x;               kleiner als der Wert von     ist.

double fabs(x)          Absolutwert einer Gleitkommazahl
double x;               Funktionswert ist ebenfalls eine Gleitkommazahl
                            Typ 'double'

double hypot(x, y)      Liefert als Ergebnis den euklidischen Abstand
double x, y;                        sqrt(x*x+y*y)

double j0(x)            Diese Funktionen berechnen die Besselfunktionen
double x;               der ersten und der zweiten Art für reelle
                        Argumente.
double j1(x)                 jn(): erster Art n-ter Ordnung
double x;                    yn(): zweiter Art n-ter Ordnung

double jn(n, x)
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;

rand()                  Erzeugen einer Pseudozufallszahl (int-Größe)
                        (Periode: 28329)
```

```
srand(seed)          Neusetzen des Startpunktes für den
int seed;            Pseudozufallszahlengenerator

double sin(x)        Trigonometrische Funktionen
double x;            (Argumente sind im Bogenmaß anzugeben)

double cos(x)
double x;

double tan(x)
double x;
                     Umkehrfunktionen
double asin(x)       Arcussinus      (-Pi/2     Pi/2)
double x;

double acos(x)       Arcuscosinus    (0     Pi)
double x;

double atan(x)       Arcustangens    (-Pi/2     Pi/2)
double x;

double atan2(x, y)   atan(x/y)
double x, y;

double sinh(x)       Hyperbolische Funktionen
double x;            0.5(exp(x)-exp(-x))

double cosh(x)       0.5(exp(x)+exp(-x))
double

double tanh(x)       (exp(x)-exp(-x))/(exp(x)+exp(-x))
double x;
```

7.5.4. Auswahl nützlicher Funktionen

Die Standardbibliotheken bieten noch eine Reihe weiterer nützlicher Funktionen, die das Programmieren in C erleichtern. So stehen z.B. Funktionen für folgende Aufgabengebiete zur Verfügung:

- Klassifizierung von Zeichen
  Zeichen- und Zahlkonvertierungen
- Zeichenkettenmanipulation
- Speicherverwaltung
- Status- und Fehlerabfragen
- Informationen über Systemumgebung

u.a.m.

In diesem Abschnitt kann nur eine kleine Auswahl, C-Programmen häufig zu findenden Funktionen gegeben werden.

Dies ist einmal die Klassifizierung und die Konvertierung Zeichen. Hierzu ist wiederum eine Preprocessoranweisung notwendig.

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

Dadurch werden die entsprechenden Makrodefinitionen ins Programm integriert. Die nachfolgend aufgeführten Aufrufe sind alle als Makros realisiert. Sie liefern im wahren Fall einen Wert Null (d.h., das zu testende Zeichen gehört der Zeichenklasse an) und ungleich Null sonst.
Das Makro 'isascii' ist für alle Integerwerte definiert. Der Rest der Makros ist nur für Werte definiert, bei denen 'isascii' wahr ist (d.h. für den ASCII-Zeichensatz) und für den einzelnen Wert 'EOF'.

Zeichenklassifizierung	Test
isascii(c)	ASCII-Zeichen (Kode kleiner 0200)
isalpha(c)	Buchstabe
isupper(c)	Großbuchstabe
islower(c)	Kleinbuchstabe
isdigit(c)	Ziffer
isxdigit(c)	Hexadezimalziffer
isalnum(c)	Alphanumerisches Zeichen (Buchstabe oder Ziffer)
isspace(c)	Leer-, Tabulator-, CR-, Newline- oder Seitenvorschubzeichen (FF)
ispunct(c)	Sonderzeichen (weder Steuerzeichen noch alphanumerisches Zeichen)
isprint(c)	druckbares Zeichen (Kode 040 (Leerzeichen) bis 0176 (Tilde))
iscntrl(c)	Steuerzeichen (Kode kleiner als 040 oder gleich 0177 (Delete))
Zeichenkonvertierungen	
toascii(c)	Konvertierung eines Integerwertes in ASCII-Zeichen ((c)&0177)
toupper(c)	Klein- zu Großbuchstaben ((c)-'a'+'A')
tolower(c)	Groß- zu Kleinbuchstaben ((c)-'A'+'a')

In Anwenderprogrammen kann es oft notwendig sein, die Systemzeit auszuwerten. Um entsprechende Zeitfunktionen aus der Standardbibliothek aufrufen zu können, ist die Preprocessoranweisung

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

notwendig. In dieser Datei erfolgt die Vereinbarung speziellen Struktur 'tm', die zur Zeitauswertung dient.

```
struct tm{
     int tm_sec;    /* Sekunden     (0 - 59)  */
     int tm_min;    /* Minuten      (0 - 59)  */
     int tm_hour;   /* Stunden      (0 - 23)  */
     int tm_mday;   /* Tag          (1 - 31)  */
     int tm_mon;    /* Monat        (0 - 11)  */
     int tm_year;   /* Jahr-1900    (0 - 99)  */
     int tm_wday;   /* Wochentag (Sonntag=0) */
     int tm_yday;   /* Tag des Jahres (0 - 365)  */
     int tm_isdst;  /* Sommerzeit bei ungleich 0 */
};
```

Folgende Zeitfunktionen stehen zur Verfügung:

```
char *ctime(clock)
long *clock;
```

Die Funktion 'ctime' konvertiert eine durch 'clock' adressierte Zeit (Anzahl der Sekunden seit dem 1.1.1970 0h; long-Wert), wie sie z.B. durch den Systemaufruf time(2) zurückgegeben wird, in eine ASCII-Zeichenfolge. Der Funktionswert ist ein Zeiger auf eine Zeichenkette aus 26 Zeichen der folgenden Form: Wed Nov 04 08:55:33 1985\n\0

```
struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;
```

Die Funktionen 'localtime' und 'gmtime' liefern einen Zeiger auf eine Struktur 'tm', die die aktuelle Systemzeit in der aufgeteilten Form enthält. Sie korrigiert die Systemzeit für die aktuelle Zeitzone und eine mögliche Sommerzeit.
Die Funktion 'gmtime' liefert die Greenwich-Zeit.

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

Die Funktion 'asctime' konvertiert eine in der aufgeteilten Form (Struktur 'tm') vorliegende Zeit in eine ASCII-Zeichenfolge aus 26 Zeichen. Sie liefert als Wert einen Zeiger auf eine Zeichenkette, die die Zeit in der oben beschriebenen Form enthält.

```
char *timezone(min, sz)
```

Die Funktion 'timezone' liefert den Namen der Zeitzone (Abkürzung aus drei Buchstaben), die zum ersten Argument ('min': Minuten westwärts von Greenwich) gehört. Wenn das zweite Argument ('sz') ungleich Null ist, so wird die Sommerzeit benutzt. Falls keine Abkürzung für die Zeitzone in der internen Zeitzonentabelle existiert, erfolgt die Ausgabe der Differenz zu GMT.

7.6 Beispiele zur Nutzung von Systemaufrufen und Bibliotheksfunktionen

Zur Veranschaulichung der Anwendung von Systemaufrufen und Bibliotheksfunktionen nun einige Beispiele.

Die ersten beiden Programme 'bsp1a.c' und 'bsp1b.' realisieren zwei einfache Filterprogramme. Im ersten Beispiel werden die von der Standardeingabe kommenden Zeichen auf Kleinbuchstaben getestet ('islower'). Wenn dies der Fall ist, so erfolgt vor der Ausgabe eine Konvertierung in Großbuchstaben ('toupper'). Solch ein Filterprogramm kann z.B. notwendig sein, wenn ein angeschlossener Drucker nur Großbuchstaben ausgeben kann.

```
/*
 * BEISPIEL 1a
 * Filterprogramm: Umwandlung Klein- in Grossbuchstaben
 */

#include <stdio.h>
#include <ctype.h>

main()
{
    int c;
    while ((c=getchar()) != EOF)
        islower(c) ? putchar(toupper(c))   putchar(c);
}
```

Bild 7.5. Programm 'bsp1a.c'

Das Programm 'bsp1b.c' streicht aus einer druckbaren Datei alle Backspacezeichen einschließlich des darauffolgenden Zeichens. Solche Konstruktionen kommen in Textdateien häufig zur Hervorhebung von Namen (Mehrfach- oder Fettdruck) vor. Falls ein angeschlossener Drucker aber kein Backspace (Rückschritt) ausführen kann, so ist die Anwendung dieses Filters vor dem Ausdruck empfehlenswert.

```
/*
 * BEISPIEL 1b
 * Filterprogramm: Eliminieren von Backspace plus naechste Zeichen
 */

#include <stdio.h>
#define BS 8

main()
{
    int c;
    while ((c=getchar()) != EOF)
        if (c==BS) getchar(); else putchar(c);
}
```

Bild 7.6. Programm 'bsp1b.c'

Da diese beiden Programme mit der Standardein-/-ausgabe arbeiten ('getchar' und 'putchar'), können sie unter UNIX, wie im Abschn. 6.2. (Ein-/Ausgabezuweisung) beschrieben, universell eingesetzt werden.

Das im Bild 7.7 dargestellte Sortierprogramm illustriert die Anwendung der Zufallszahlenfunktion und des rekursiven Funktionsaufrufs. Die Funktion 'sortiere' wurde den Benchmarkprogrammen aus [30] entnommen.

```
/*
 * BEISPIEL 2
 * Sortieralgorithmus: Fuellen eines Integerfeldes mit Zufallszahlen,
 *                     Sortieren des Feldes ('quicksort') und
 *                     Ausgabe des Feldes ueber Standardausgabe
 */

#define ANZ 70

int feld[ANZ];

main()
{
    int i, j;
    printf("\nFeld aus Integerwerten mit Zufallszahlen fuellen");
    for (i=0; i<=ANZ; i++)
        feld[i]=abs(rand()%9000);       /* pos. Zahl, max. 9000 */
    printf("\nFeld gefuellt, Sortierung beginnt\7");
    sortiere(0,ANZ);
    printf("\7 - Sortierung beendet");
    printf("\nAusgabe des sortierten Feldes:");
    for (j=0; j<ANZ;) {
        putchar('\n');
        for (i=0; i<10; i++) {  /* 10 Spalten */
            printf("%7d"  feld[j++]);
        }
    }
    putchar('\n');
}

sortiere(unten, oben)   /* Quicksort-Programm */
int unten, oben;
{
    int i, j, mitte, tmp;
    i=unten; j=oben;
    mitte=feld[(unten+oben)/2];
    do {
        while ((i<oben) && (feld[i] < mitte)) ++i;
        while ((j>unten) && (feld[j] > mitte)) --j;
        if (i<=j) {
            tmp=feld[i];        /* vertauschen */
            feld[i]=feld[j];
            feld[j]=tmp;
            i++; j--;
        }
    } while (i<=j);
    if (unten<j) sortiere(unten, j);    /* rekursive Aufrufe */
    if (i<oben) sortiere(i, oben);
}
```

Bild 7.7. Programmbeispiel 'bsp2.c'

Das Rahmenprogramm gibt vor dem Beginn und nach Abschluß des Sortiervorgangs das Glockenzeichen ('\7') ans Terminal aus. Danach erfolgt noch die

Ausgabe des sortierten Feldes. Dabei ist der Ausgabebereich für jede Zahl sieben Zeichen lang (Format %7d). Das Bild 7.8 zeigt eine vom Programm 'bsp2.c' erzeugte Ausgabe.

```
Feld aus Integerwerten mit Zufallszahlen fuellen
Feld gefuellt, Sortierung beginnt - Sortierung beendet
Ausgabe des sortierten Feldes:
     35     84     96    670    751    920   1067   1113   1311   1318
   1394   1422   1444   1457   1505   1882   1945   2367   2495   2653
   2749   2983   3054   3060   3183   3524   3754   3767   3894   4051
   4086   4102   4143   4145   4306   4327   4353   4403   4406   4628
   4978   5010   5085   5225   5479   5627   5758   6165   6188   6561
   6629   6803   6834   6851   6967   7089   7137   7212   7248   7419
   7562   7566   7736   7757   7838   7962   8031   8259   8515   8543
```

Bild 7.8. Beispiel einer vom Programm 'bsp2.c' erzeugten Ausgabe

Das folgende Beispielprogramm ermittelt die aktuelle Systemzeit und gibt sie als verschiedene Textzeilen aus. Dabei wird die Anwendung der Bibliothekszeitfunktion 'localtime', der Ausgabefunktion 'printf' und die Arbeit mit der Struktur 'tm' verdeutlicht.

```
/*
 * BEISPIEL 3
 * Zeitprogramm: Ermittelt Systemzeit und gibt sie
 *               als verschiedene Textzeilen aus
 */

#include <time.h>

long jetzt;
struct tm *zeit;
struct tm *localtime();

char *mn[] = {
        "Januar", "Februar", "Maerz", "April", "Mai", "Juni", "Juli",
        "August", "September", "Oktober", "November", "Dezember" };

char *wt[] = {
        "Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag",
        "Freitag", "Sonnabend" };

main()
{
    time(&jetzt);       /* Systemaufruf */
    printf("\nAnzahl der Sekunden seit dem 1.1.70 0h: %ld", jetzt);

    zeit=localtime(&jetzt);  /* Konvertierung in Struktur vom Typ tm */

    printf("\naktuelles Datum: %s, der %d.", wt[zeit->tm_wday], zeit->tm_mday);
    printf(" %s 19%02d", mn[zeit->tm_mon], zeit->tm_year);

    printf("\nes ist der %d. Tag des Jahres", zeit->tm_yday);

    printf("\naktuelle ");
    if (zeit->tm_isdst) printf("Sommerzeit"); else printf("Zeit");

    printf(": %02d:%02d:%02d\n", zeit->tm_hour, zeit->tm_min, zeit->tm_sec);
}
```

Bild 7.9. Programm 'bsp3.c'

Das Bild 7.10 zeigt eine vom Programm 'bsp3.c' erzeugte Ausgabe.

```
Anzahl der Sekunden seit dem 1.1.70 0h: 502748025
aktuelles Datum: Freitag, der 6. Dezember 1985
es ist der 339. Tag des Jahres
aktuelle Zeit: 12:13:45
```

Bild 7.10. Beispiel für eine Ausgabe des Programms 'bsp3.c'

7.7. Compileraufruf und Optionen

In diesem Abschnitt wird die Handhabung des cc-Kommandos unter UNIX beschrieben, d.h. die Erzeugung von abarbeitungsfähigen Programmen und die Anwendung des C-Prüfprogramms 'lint'.

Vorher aber ein Hinweis auf das Programm 'cb'. Nachdem der Anwender seinen C-Quelltext mit Hilfe eines Editorprogramms als Datei erstellt hat, kann er durch das Programm 'cb' (C-beautifier) seinen Quelltext in eine Form bringen, die die Struktur des Programms besser widerspiegelt. Dabei werden Leerzeichen und entsprechende Einrückungen eingefügt.

Aufruf von 'cb':

```
cb < BSP.C > bsp.c
```

Das Programm 'cb' liest von der Standardeingabe den Quelltext (im Beispiel: Datei 'BSP.C') und gibt den formatierten Text über die Standardausgabe (hier: Datei 'bsp.c') aus. Je nach Rechnerversion und UNIX-Implementation kann auch eine unterschiedliche Handhabung möglich sein. Das gleiche gilt für die nachfolgenden Beschreibungen der Optionen von 'cc' und 'lint'. Im Einzelfall sind dann die Handbücher des Rechners zu konsultieren (z.B. über das Kommando 'man').

Der Aufruf eines C-Übersetzungslaufs hat die allgemeine Form:

```
cc Optionen Dateiname
```

Den Dateinamen können wahlweise Optionen vorangestellt sein, mit denen bestimmte Aktionen im Übersetzungsprozeß veranlaßt werden. Optionen sind durch Minuszeichen gekennzeichnet.
Standardmäßig werden die angegebenen Dateien kompiliert (Aufruf C-Compiler), assembliert (Assembleraufruf) und verbunden (Linkeraufruf). Dateinamen, die mit '.c' enden, werden als C-Quellen erkannt und vom C-Compiler übersetzt. Bei Dateien, die mit '.s' enden, wird der Compilerlauf übersprungen. Alle anderen Dateinamen werden als Namen von C-kompatiblen Objektprogrammen (z.B. durch einen früheren cc-Lauf erzeugt) oder von Bibliotheken mit C-kompatiblen Funktionen genommen. Diese werden dann mit den durch Übersetzung entstandenen Objektprogrammen, in der angegebenen Reihenfolge, zu einem ausführbaren Programm verbunden. Bei einem fehlerfreien Ablauf wird das ausführbare Programm als Datei mit dem Namen 'a.out' erzeugt, falls durch Optionen nichts anderes festgelegt wurde.

Bei der Dateinamensgebung der Zwischendateien gibt implizite Festlegungen:

Name.s	Assemblerquelltext
Name.o	Objektkode

Die Zwischendateien werden automatisch gelöscht, falls durch Optionen nichts anderes festgelegt wurde.

Mögliche Optionen für das cc-Kommando:

-O Aufruf des Optimierungslaufs vom C-Compiler

-S Übersetzt die angegebenen C-Quellprogramme und erzeugt Assemblerquellkodedateien, deren Namen auf .s' enden. Die weiteren Schritte (Assemblieren und Verbinden) werden unterdrückt.

Übersetzt die angegebenen Quellen (Compiler- und Assemblerlauf). Die Linkphase wird unterdrückt. Es entstehen Objektkodedateien mit der Endung '.o'.

-P Es wird der Preprocessorlauf für die angegebenen Dateien ausgeführt. Der erzeugte Text (ohne #-Anweisungen) wird in Dateien mit der Endung abgelegt.

-E Analog -P, das Ergebnis wird aber an die Standardausgabe ausgegeben.

-DName
-DName=Wert
Definiert für den Preprocessor einen Namen (wie bei #define innerhalb der Quelldatei). Wenn kein Wert gegeben ist, wird standardmäßig 1 eingesetzt.

-UName Aufheben einer #define-Anweisung für den angegebenen Namen.

-IDirectoryname
Ändern des Suchalgorithmus für include-Dateien. Dateinamen, die nicht mit '/' beginnen, werden zuerst in dem Directory gesucht, in dem sich die Quelldatei befindet, dann in den Directories der I-Optionen und zuletzt in den Standard-verzeichnissen.

-p Veranlaßt den Compiler, einen Kode zu erzeugen, der die Anzahl der Aufrufe einer jeden Funktion zählt. Bei der Ausführung des so erzeugten Programms wird durch automatische Aufrufe der Bibliotheksfunktion 'monitor(3)' ein sog. Laufzeitprofil in der Datei 'mon.out' abgelegt (siehe dazu auch: Kommando prof(1)).

Alle weiteren Optionen werden als Optionen für den Verbinder 'ld' aufgefaßt und an ihn weitergereicht. Vom cc-Kommando werden keine Optionen an den Assembler 'as' übergeben.
Beim Verbinden wird automatisch die Bibliothek '/lib/libc.a' mit angehängt. Als Beispiele für häufig benutzte Linkeroptionen beim cc-Aufruf seien '-o' und '-l' erwähnt.

-o Name Legt den Namen für die ausführbare Datei fest (anstelle 'a.out', z.B. cc -o bsp bsp.c).

-lx Gibt eine weitere Bibliothek mit dem Namen /lib/libx.a für den Verbinder zur Auflösung von externen Referenzen an. Diese Option ist z.B. für die mathematischen Standardprogramme notwendig. (/lib/libm.a ->Option: -lm, für Pascal-Bibliothek: /lib/libpas.a ->Option: -lpas)

Eine vollständige Beschreibung der Linker-Optionen ist im Handbuch unter ld(1) zu finden oder über das man-Kommando (man ld) einsehbar.

C-Prüfprogramm 'lint'

Dieses Prüfprogramm dient zur zusätzlichen Prüfung von C-Programmen. Es ermittelt Fehlerursachen, überflüssige Programmteile und mögliche Probleme bei der Übertragung des Programms auf andere Rechner. Es führt eine strengere Typprüfung als der C-Compiler aus. Es erkennt unerreichbare Befehle, Schleifen, die nicht an ihrem Anfang starten, nicht benutzte, aber vereinbarte automatische Variablen und logische Ausdrücke, deren Wert konstant ist. Weiterhin werden die Funktionsaufrufe überprüft. Dabei werden Funktionen ermittelt, die in einigen Fällen Werte liefern und in anderen nicht. Es erfolgen Meldungen, wenn Funktionen mit unterschiedlicher Anzahl von Argumenten aufgerufen werden und wenn Funktionswerte nicht weiter verwendet werden. Implizit wird angenommen, daß alle Dateien zu einem Programm verbunden werden. Sie werden durch 'lint' auf gegenseitige Kompatibilität überprüft.
Zur Überprüfung der richtigen Anwendung von externen Funktionsaufrufen können 'lint' in der gleichen Art wie dem Linker 'ld' Bibliotheksnamen übergeben werden (z.B. -lm für die mathematische Standardbibliothek). Implizit benutzt 'lint' die Standardbibliothek 'llib-lc', bei der Option -p (Portabilitätsüberprüfung) die Bibliothek 'llib-port'.
Die allgemeine Form des Aufrufs von 'lint' ist:

```
lint Optionen Dateiname
```

Bei der Angabe von Optionen können mehrere Buchstaben mit einemmal angegeben werden, z.B. -pnv. Von 'lint' werden auch die Optionen -D, -U und -I des cc-Kommados akzeptiert.

Mögliche Optionen für das lint-Kommando:

Prüfung, ob long-Variablen int-Variablen zugewiesen werden.

-b Prüfung auf break-Anweisungen, die nicht erreicht werden können. (Die Programme 'lex' und 'yacc' erzeugen viele solcher Konstruktionen.)

Reklamiere Portabilitätsprobleme.

-h Führe eine Reihe heuristischer Tests aus, die helfen, Fehler zu erkennen, Überflüssiges zu reduzieren und den Programmierstil zu verbessern.

Führe keine Überprüfung der Kompatibilität mit der Standardbibliothek durch.

-p Überprüfung der Kompatibilität mit anderen C-Dialekten (z.B. IBM).

Reklamiere keine Funktionen und Variablen, die verwendet werden, aber nicht vereinbart sind (oder umgedreht). Dies ist sinnvoll, wenn 'lint' auf Dateien angewendet wird, die eine Untermenge eines größeren Programms sind.

Unterdrücke die Reklamation von unbenutzten Funktionsargumenten.

Prüfe auf extern vereinbarte Variablen, die nicht benutzt werden.

Alle Funktionen, die nicht zurückkehren (wie z.B. exit(2)), können von 'lint' nicht "verstanden" werden. Entsprechende Fehlermeldungen muß der Anwender in Zusammenhang mit seinem Programm interpretieren.

Eine Reihe von Kommentaren in C-Programmen beeinflußt ebenfalls die Arbeitsweise von 'lint'.

/*NOTREACHED*/ Ab dieser Stelle erfolgt keine Reklamation mehr nicht erreichbaren Anweisungen.

/*VARARGSn*/ Unterdrückt die Überprüfung auf eine variable Anzahl von Argumenten in der nachfolgenden Funktionsvereinbarung. Es werden nur die Datentypen der ersten n Elemente überprüft; fehlendes n wird als Null genommen.

/*ARGSUSED*/ Schaltet die Option -v für die nächste Funktion ein.

/*NOSTRICT*/ Schaltet die strenge Typprüfung im nächsten Ausdruck aus.

/*LINTLIBRARY*/ Wenn dieser Kommentar am Anfang einer Datei steht, so unterdrückt er die Reklamation von nicht benutzten Funktionen in dieser Datei.

Das Prüfprogramm 'lint' unterstützt den erfahrenen Programmierer bei der Entwicklung größerer Programme. Es hilft Fehler zu vermeiden, die sonst beim Übersetzen von C-Programmen nicht erkannt werden, und dient damit der eigenen Sicherheit.

Literaturverzeichnis

[1] UNIX (TM) Time-Sharing System: UNIX Programmer's Manual Seventh Edition Volume 1, 1A and 2B. Bell Telephone Laboratories Incorporated, Murray Hill, New Jersey 1979

[2] UNIX (TM) User's Manual Release 3.0. Bell Telephone Laboratories Incorporated, Murry Hill, New Jersey 1980

[3] Programmers Manual for UNIX System III Volume 2A/2B. Western Electric Company Incorporated, 1981

[4] UNIX Time-Sharing System: The Bell System Technical Journal (UNIX Sonderheft). July-August 1978, Vol. 57, No. 6, Part 2

[5] Kernighan, B. W.; Ritchie, D. M.: The C Programming Language. Englewood Cliffs, New Jersey: Prentice Hall Inc., 1978

[6] Kernighan, B. W.; Ritchie, D. M.: Programmieren in C mit dem C Reference Manual in deutscher Sprache. München/Wien: Carl Hanser Verlag, 1983

[7] Wettstein, H.: Aufbau und Struktur von Betriebssystemen. München/Wien: Carl Hanser Verlag, 1978

[8] Yates, J. L.: UNIX and the standardization of small computer systems. BYTE, 1983, Vol. 8, No. 10, pp. 160-166

[9] UNOS 68/35. microbit, 1984, No. 3, p. 22

[10] Thomas, R.; Yates, J. L.: UNIX Anwenderhandbuch. München: te-wi Verlag, 1983

[11] Bourne, S. R.: The UNIX System. London: Addison-Wesley Publishing Company, 1983

[12] Stanka, Z.; Lösch, S.: UNIX - Führer durch das System. München: te-wi Verlag, 1984

[13] McGilton, H.; Morgan, R.: Einführung in das UNIX System. Hamburg: McGraw-Hill Book Co., 1984

[14] Banahan, M.; Rutter, A.: UNIX - Lernen, verstehen, anwenden. München/Wien: Carl Hanser Verlag, 1984

[15] Bourne, S. R.: The UNIX Shell. The Bell System Technical Journal Vol. 57, No. 6, Part 2, pp. 1971-1990

/16/ Ritchie, D. M.; Johnson, S. C.; Lesk, M. E.; Kernighan, B. W.: The C Programming Language. The Bell System Technical Journal Vol. 57, No. 6, Part 2, pp. 1991-2019

/17/ Gauthier, R.: Using the UNIX System. Reston Virginia: Reston Publishing Company, Inc., 1981

/18/ Detering, R.: UNIX-Handbuch. Düsseldorf: Sybex-Verlag GmbH, 1984

/19/ Illik, J. A.: Erfolgreich Programmieren mit C. Düsseldorf: Sybex-Verlag GmbH, 1984

/20/ Schirmer, C.: Die Programmiersprache C. Das Lehr- und Arbeitsbuch für die Praxis mit vollständiger Beschreibung der Standardbibliotheksfunktionen. München/Wien: Carl Hanser Verlag, 1985

/21/ Feuer, A. R.: Das C-Puzzlebuch (C-Programmier-Training). München/Wien: Carl Hanser Verlag, 1985

/22/ Clauss, M.; Fischer G.: Sprachbeschreibung C-Sprache unter den Betriebssystemen MUTOS-8000 und MUTOS-1630. VEB Robotron Buchungsmaschinenwerk Karl-Marx-Stadt 1985

/23/ Sprachbeschreibung Programmiersprache C MGS K1600. VEB Robotron-Elektronik Dresden 10/83

/24/ Einert, E.: Die Programmiersprache C. rechentechnik/datenverarbeitung 21(1984)9, S. 10-13

/25/ The C Language. Artikelserie in BYTE, August 1983, p. 43 ff.

/26/ Johnson, S. C.; Kernighan, B. W.: The C Language and Models for System Programming. BYTE, August 1983, pp. 48-60

/27/ Joyce, J.: A C Language Primer. BYTE, August 1983, pp. 64-78

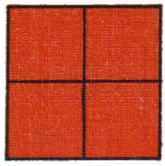
/28/ Pol, B.: Die Sprache C. ELEKTRONIK (1983) H. 2, S. 43-49; H. 3, S. 53-59; H. 4, S. 61-65

/29/ C-Compiler im Vergleich. Computer Persönlich (1985) Heft 5, S. 109 - 115

/30/ Geschwindigkeitstest mit C-Compilern. Computer Persönlich (1985) Heft 6, S. 62 - 64

Sachwörterverzeichnis

Notizen



In übersichtlicher Form werden Kenntnisse über das Betriebssystem UNIX, über die Kommandosprache Shell und über die Programmiersprache C vermittelt. Im Mittelpunkt stehen dabei die Darstellung der Basiskonzepte und der Standardkommandos von UNIX sowie der Sprachelemente von C.
Durch sorgfältig gestaltete Bilder und Tafeln wird sowohl das Einarbeiten erleichtert als auch das Nachschlagen effektiv unterstützt.

ISBN 3-341-00863-2