4 Betriebssysteme, Rechnernetze

Das **Betriebssystem** eines Computers ist ein (im allgemeinen sehr umfangreiches) Softwarepaket,das die Arbeit der Anwenderprogramme steuert und überwacht und die Betriebsmittel (Prozessoren, Speicher, Ein- und Ausgabegeräte, Daten, Programme) verwaltet.

Unter einem **Rechnernetz** soll hier ein Netz von **selbständigen Computern** verstanden werden, die untereinander Daten austauschen. Da es heute (bis auf immer seltener werdende "Stand alone"-Dinosaurier, die nicht einmal über ein Modem Kontakt mit der Außenwelt haben) kaum noch Computer gibt, die nicht "mit Kollegen" kommunizieren können, vermischen sich die Aufgaben des Betriebssystems immer mehr mit den Problemen der Netzdienste. Deshalb werden die beiden Themen in einem Kapitel abgehandelt.

4.1 Betriebssysteme

Der in fast allen einschlägigen Lehrbüchern zu diesem Thema zu findende Versuch, die Betriebssysteme z. B. nach der Anzahl gleichzeitig angemeldeter Benutzer ("Single-User"- und "Multi-User"-Systeme), der gleichzeitig zu bearbeitenden Aufgaben ("Single-Tasking"- und "Multi-Tasking"-Systeme), ihrem Zeitverhalten ("Batch"-, "Dialog"- und "Echtzeit"-Systeme) oder anderen Kriterien einzuteilen, endet immer in der Bemerkung, daß fast jedes der existierenden Systeme in fast jeder Hinsicht eine Mischform darstellt. Selbst MS-DOS, das in seiner einfachsten Form (ohne "Windows") als klassisches Single-User-Single-Tasking-System angesehen werden kann, reiht z. B. Print-Jobs in eine Warteschlange ein, und der Benutzer kann sich einem anderen Problem zuwenden (Multi-Tasking in simpelster Form).

Betriebssysteme sind besonders hardwarenah. Deshalb ist es nicht ohne weiteres möglich, ein Betriebssystem auf einen anderen Computer zu übertragen, und deshalb waren in der Vergangenheit "proprietäre (hersteller-abhängige) Systeme" dominierend (in den achtziger Jahren lief z. B. auf Computern der Firma DEC üblicherweise VMS, auf IBM-Rechnern MVS, auf Siemens-Rechnern BS2000).

Die Zeit der proprietären Systeme scheint abgelaufen zu sein. Betriebssysteme, die nicht für die Hardware eines speziellen Herstellers konzipiert sind ("offene Systeme"), verdrängen die zum Teil ganz hervorragenden proprietären Systeme. Dies geht nicht ohne die in der Informatik üblichen Glaubenskriege ab, die mit Vehemenz im allgemeinen von denen ausgefochten werden, die nur ein System kennengelernt haben, dessen Eleganz (häufig durchaus begründet) preisen und alles andere diffamieren. Nur wer sich ganz stark fühlt, sollte es riskieren, mit einem eingefleischten "Apple-Freak" über MS-DOS zu diskutieren. Außer dem Argument, daß es ungleich mehr DOS-Rechner als "Macs" gibt, hat der "DOSianer" ohnehin nicht viel zu bieten (auch der Hinweis auf das jeweils neueste Windows wird von Apple-Freaks eher belächelt), aber gerade das "Verbreitungs-Argument" ist natürlich nicht wegzudiskutieren.

Im PC-Bereich ist MS-DOS bereits Ende der achtziger Jahre zum Industrie-Standard avanciert. Die gewaltige Palette an existierender effektiver Anwendungssoftware wird die Entwickler von Nachfolgesystemen noch lange Zeit zwingen, die Möglichkeit vorzusehen, daß

DOS-Programme (z. B. in einer speziellen "DOS-Box") lauffähig sind (auch auf Apple-Computern können DOS-Programme seit einiger Zeit ablaufen).

Auf Workstations werden (zumindest im technisch-wissenschaftlichen Bereich) die proprietären Systeme mehr und mehr von UNIX verdrängt. Das Angebot an Anwendungssoftware ist gewaltig, und weil UNIX über viele Jahre im Hochschulbereich das bevorzugte Betriebssystem war (und ist), ist außerordentlich viel Spezial-Software frei verfügbar.

4.2 UNIX

4.2.1 Merkmale des Betriebssystems UNIX

Das Betriebssystem UNIX entstand in den Jahren um 1970 in den Bell Laboratories des AT&T-Konzerns. Es enthielt einige für diese Zeit geradezu revolutionäre Konzepte, was wohl die wesentliche Ursache dafür war, daß es über eine sehr lange Zeit von vielen Hardwareherstellern nicht beachtet wurde. Die Firma IBM ("Not invented here" - "Es mag ja gut sein, aber es ist nicht von uns") hat z. B. erst in der Mitte der achtziger Jahre die Gnade gezeigt, UNIX zur Kenntnis zu nehmen.

Zu den wichtigsten Merkmalen des Betriebssystems UNIX, die sich natürlich inzwischen auch in den meisten anderen Betriebssystemen wiederfinden, gehörten von Anfang an:

- Es ist als Multi-User-Multi-Tasking-System konzipiert, es stehen leistungsfähige Werkzeuge für die Kommunikation der Benutzer untereinander zur Verfügung, die beispielgebend für die Realisierung der modernen Netzdienste waren.
- Bis auf sehr wenige hardwarenahe Assemblerroutinen (im wesentlichen die Gerätetreiber) ist das gesamte Betriebssystem in der höheren Programmiersprache C geschrieben. Kein anderes Betriebssystem ist deshalb mit vergleichbar geringem Aufwand auf ein anderes Computersystem portierbar.
- Die peripheren Geräte werden wie Files angesprochen (sind in sogenannten "special device files" beschrieben). Die Eigenschaften des gesamten Computersystems werden in Konfigurationsfiles beschrieben.
- Zu den Merkmalen, die UNIX heute mit zahlreichen Betriebssystemen teilt, gehört die Verwendung eines Kommando-Interpreters ("Shell"), ein hierarchisches File-System, die Möglichkeiten, Ein- und Ausgabe umzuleiten und durch sogenannte "Pipes" von einem Kommando zum nächsten zu schicken und eine große Zahl außerordentlich leistungsfähiger Dienstprogramme ("Utilities").

Nur der eigentliche Betriebssystem-Kern ("Kernel") kommuniziert mit der Hardware (für Kernel und Hardware ist der Beriff "UNIX-Maschine" gebräuchlich). Die Schnittstelle zwischen dem UNIX-Kernel und den Programmen ist in der "UNIX System V Interface Definition" eindeutig beschrieben. Alle Programme (und dazu gehören neben den Anwendungsprogrammen auch fast alle zum Lieferumfang von UNIX gehörenden Programme, die die UNIX-Befehle realisieren, und sogar der Kommando-Interpreter, die Shell) müssen dieser Schnittstellen-Definition entsprechen und sind damit austauschbar.

Seit dem Ende der siebziger Jahre wird die UNIX-Lizenz von der Firma AT&T zu sehr günstigen Konditionen weitergegeben, und fast alle großen Hardware-Hersteller haben ihre UNIX-Version auf den Markt gebracht. Die UNIX-Produkte haben z. B. die Namen HP-UX (Hewlett Packard), Ultrix (DEC), Sinix (Siemens), AIX (IBM), Sun OS und Solaris (SUN), A/UX (Apple). Und dann gibt es noch Xenix, SCO-UNIX, Cromix, Munix, Venix, ...

Im PC-Bereich ist eine UNIX-Version mit dem Namen **LINUX** (nach Linus B. Torvalds aus Finnland) in den letzten Jahren nicht nur deshalb besonders erfolgreich, weil sie frei kopierbar im Internet zur Verfügung steht. Weil auch der Quellcode verfügbar ist, werden ständig von unterschiedlichen Autoren neue Komponenten hinzugefügt. Die Möglichkeit, auf preiswerte Weise die gewaltige Menge der UNIX-Anwendungsprogramme auch auf dem PC nutzen zu können, und die Qualität des LINUX-Systems selbst gestatten die Prognose, daß dieses System sich in den nächsten Jahren stark verbreiten wird.

Das Auseinanderdriften der verschiedenen UNIX-Versionen hat seit Mitte der achtziger Jahre zu verschiedenen Standardisierungsbemühungen geführt. Als nationaler US-Standard wurde 1988 POSIX ("Portable Operation System") vorgestellt. Im gleichen Jahr wurde (u. a. von den Firmen HP, DEC, IBM, Siemens) die OSF ("Open Software Foundation") gegründet, die den inzwischen weit verbreiteten UNIX-Standard OSF/1 und die sehr schöne graphische Benutzeroberfläche OSF/Motif herausgebracht hat.

4.2.2 Prozesse

Prozesse realisieren die Ausführung von Programmen. Für das weitere Verständnis ist die bereits angedeutete Tatsache wichtig, daß sich alle Aussagen sowohl auf Prozesse beziehen können, die Anwender-Programme realisieren (oder von diesen erzeugt wurden), als auch auf Prozesse, die von UNIX-Kommandos oder bereits beim Booten des Systems erzeugt wurden.

- Ein UNIX-Prozeß belegt im Arbeitsspeicher ein **Code-Segment**, das mindestens einen Teil der ausführbaren Anweisungen des Programmes enthält, ein **Benutzerdaten-Segment** und ein **Systemdaten-Segment**.
- Jedem Prozeß ist als **Prozeß-Identifikator** (**PID**) eine eindeutige Nummer zugeordnet. Fast jeder Prozess ist Abkömmling (**Child Process**) eines anderen Prozesses (**Parent Process**) und kennt den Prozeß-Identifikator seines Eltern-Prozesses (**PPID**).

Beim Booten eines UNIX-Systems wird automatisch ein Prozeß mit **PID = 0** erzeugt ("swapper"), der gewissermaßen zum "Urahn" aller weiteren Prozesse wird. Der Prozeß "swapper" erzeugt sofort einige für den Betrieb wichtige Prozesse ("Dämonen", siehe folgenden Abschnitt), unter anderem den Prozeß "init" (auch ein "Dämon"), der wiederum für jedes angeschlossene Terminal einen "getty"-Prozeß erzeugt.

Die "getty"-Prozesse nehmen an den Terminals die Benutzer-Anmeldungen entgegen (schreiben unter anderem die "login"-Aufforderung). Wenn eine Benutzeranmeldung vom "getty"-Prozeß geprüft und für korrekt befunden wurde, ersetzt er sich durch den Kommando-Interpreter, die "Login-Shell" ("getty"-Prozesse "zeugen kein Kind", sondern ersetzen sich durch einen Prozeß mit der gleichen PID-Nummer). Die Shell (vgl. Abschnitt 4.2.6) wird nun zum "Parent Process" aller Prozesse, die der Benutzer durch Eingeben von UNIX-Kommandos oder Aufruf von Anwendungs-Programmen startet.

- Die Shell kann durchaus auch wieder eine Shell als Kind-Prozeß starten (in vielen UNIX-Systemen stehen unterschiedliche Shell-Programme zur Verfügung, in der Regel bevorzugt ein Benutzer aber ein bestimmtes Shell-Programm und wird als Kind-Prozeß also das gleiche Programm noch einmal starten).
- Wenn ein "Parent Process" wartet, bis ein von ihm gestarteter "Child Process" seine Arbeit beendet hat und danach seine eigene Arbeit fortsetzt, spricht man von "synchron ablaufenden Prozessen". Wenn der "Parent Process" zur Fortsetzung seiner Arbeit die Ergebnisse des "Child Processes" nicht benötigt, kann der "Child Process" auch asynchron ablaufen (er wird in den "Hintergrund" geschickt, was sehr einfach durch Anhängen des Zeichens & an die Kommandozeile realisiert werden kann).
- Ein Prozeß kann auf natürliche Weise beendet werden (z. B. beendet eine Shell ihre Arbeit nach Eingeben des Kommandos "logout"), ansonsten kann ein laufender Prozeß nur durch ein **Signal** beendet werden (mit dem UNIX-Kommando "kill" kann z. B. die "Aufforderung zum sofortigen Selbstmord" an einen Prozeß geschickt werden).
- Wenn ein Prozeß (auf natürliche oder unnatürliche Weise) stirbt, sterben automatisch auch alle noch lebenden Nachkommen. Das bedeutet, daß beim Beenden einer Sitzung (Abmelden mit "logout") alle eventuell noch laufenden Hintergrundprozesse sterben, weil die Shell als ihr "Parent Process" stirbt. Wenn dies verhindert werden soll (man möchte z. B. ein Programm mit sehr langer Rechenzeit über Nacht weiterarbeiten lassen), muß der "Child Process" mit "nohup" (no hang up) gestartet werden.
- Es ist möglich, einzelnen Prozessen unterschiedliche Prioritäten zu geben, die darüber entscheiden, wieviel Rechenzeit ihnen bei der Konkurrenz mit allen parallel ablaufenden Prozessen zugeteilt wird.

4.2.3 Dämonen

Dämonen sind Prozesse, die nicht an einen Benutzer oder ein Terminal gebunden sind. Sie werden im allgemeinen bereits beim Starten des Systems (vom "swapper") erzeugt und erledigen Kontroll- und Verwaltungsaufgaben im Hintergrund.

Der Dämon **cron** schaut in regelmäßigen Abständen in speziellen Files nach, ob dort Eintragungen vorhanden sind, die eine bestimmte Aktion zum gerade aktuellen Zeitpunkt auslösen sollen. Der System-Manager trägt z. B. in die von diesem Dämon inspizierten Files die Kommandos zum Aufräumen im File-System ein, während für den Benutzer mit dem "at"-Kommando die Möglichkeit besteht, beim Dämon "cron" das Starten eines Programms zu einem bestimmten Zeitpunkt in Auftrag zu geben.

Die Druckerwarteschlangen des Systems werden vom Dämon **lpsched** ("Line Printer Scheduler") verwaltet.

Der Dämon **inetd** ("Internet Damon") ist permanent damit beschäftigt, den Anschluß zum Internet auf ankommende Signale zu überwachen, um gegebenenfalls sofort einen "Unterdämon" mit der Erledigung des angeforderten Netzdienstes zu beauftragen.

4.2.4 Das UNIX-File-System

Das hierarchische UNIX-File-System hat gewisse Ähnlichkeiten mit dem File-System von MS-DOS. Zunächst fällt nur der Unterschied ins Auge, daß der "Slash" / dort verwendet wird, wo bei MS-DOS der "Backslash" \ steht. Es gibt ein "Wurzel-Verzeichnis" ("root"), repräsentiert durch / und in diesem Files und Unterverzeichnisse, die wiederum Files und Unterverzeichnisse enthalten dürfen.

Filenamen dürfen im Regelfall aus maximal 14 Zeichen bestehen, viele UNIX-Systeme erlauben wesentlich mehr Zeichen. Im Zeitalter des Datenaustauschs zwischen unterschiedlichen Systemen ist man gut beraten, sich an die (besonders restriktive) DOS-Konvention zu halten (maximal 8 Zeichen, Punkt, maximal dreistellige "Extension") und nur Kleinbuchstaben (UNIX unterscheidet im Gegensatz zu MS-DOS zwischen Groß- und Kleinbuchstaben), Ziffern und den Unterstrich zu verwenden.

Namenserweiterungen ("Extensions") sind unter UNIX möglich, aber nicht so gebräuchlich wie unter MS-DOS, weil ausführbare ("executable") Files auch nicht besonders gekennzeichnet werden müssen (wie unter MS-DOS, wo dafür nur .EXE, .COM und .BAT erlaubt sind).

Es gibt jedoch auch wesentliche Unterschiede zwischen den File-Systemen der Betriebssysteme UNIX und MS-DOS:

> Unter UNIX gibt es stets nur **eine File-Hierarchie** (das Arbeiten mit Laufwerken wie unter MS-DOS ist unter UNIX nicht vorgesehen).
>
> Wenn auf ein weiteres File-System (kann sich z. B. auf einem Diskettenlaufwerk befinden) zugegriffen werden soll, dann muß es in die File-Hierarchie an einem beliebigen Punkt "eingehängt" werden. Dieser Vorgang heißt "mounten".

Mit dem UNIX-Kommando "mount" wird das Wurzel-Verzeichnis eines weiteren File-Systems auf ein beliebiges Verzeichnis (sollte leer sein) der File-Hierarchie abgebildet. Da hierbei Probleme mit den Zugriffsrechten auftreten können, kann dieses Kommando nur für Benutzer mit speziellen Rechten angewendet werden.

> Files werden in UNIX durch einen Satz von Zugriffsrechten vor unberechtigtem Zugriff geschützt.

Der Erzeuger eines Files ist der **owner** (kann seine "Besitzrechte" übrigens mit dem Kommando "chown" - "Change Owner" - abtreten), für ihn, die Mitglieder seiner Benutzer-Gruppe (**group**) und den "Rest der Welt" (**other**) werden gesonderte Zugriffsrechte verwaltet, die durch eine dreistellige Zahl symbolisiert werden. In der Reihenfolge owner-group-other werden durch die drei Ziffern die Zugriffsrechte **read** (Wertigkeit 4, File darf gelesen werden), **write** (Wertigkeit 2, File darf geändert werden) und **execute** (Wertigkeit 1, File darf zur Ausführung gestartet werden) festgelegt, wobei die Wertigkeiten der Zugriffsrechte

addiert werden ("7 darf alles, 0 darf nichts"). Mit dem Kommando "chmod" können die Zugriffsrechte modifiziert werden, mit

chmod 751 filename

werden z. B. dem **owner** alle Rechte, der Gruppe die Rechte **read** und **execute** und dem "Rest der Welt" das Recht **execute** zugebilligt. Das UNIX-Kommando "ls" (ist vergleichbar mit dem DOS-Befehl "DIR") zeigt die Zugriffsrechte an, wenn zusätzlich die Option −l angegeben wird ("Optionen" zu den UNIX-Kommandos werden auch als "Schalter" oder "Flags" bezeichnet und durch das Zeichen − eingeleitet, vergleichbar mit den Schaltern in MS-DOS, die dort mit dem Zeichen / eingeleitet werden). Das Kommanod "ls −l" würde die "Zugriffsrechte 751" z. B. in der Form

−rwxr−x−−x

anzeigen. Das erste Zeichen ist für Files immer −, für Directories **d**, es folgen die Rechte für **owner** (hier **r**ead,**w**rite,e**x**ecute), **group** (hier **r**ead und e**x**ecute) und **other**.

4.2.5 Einige wichtige UNIX-Kommandos

Die nachfolgend aufgelisteten UNIX-Kommandos sind eine kleine Auswahl. Die in der rechten Spalte angegebenen DOS-Befehle sind vielfach nur bedingt vergleichbar. Durch die Verwendung von Optionen können die Kommandos zum Teil in ihren Auswirkungen sehr stark modifiziert werden. Genaueres erfährt man über das "man"-Kommando, mit dem man sich das Manual auflisten lassen kann, z. B. zeigt

man ls

alles, was man über den "ls"-Befehl zu wissen wünscht.

Auf nicht unerhebliche Unterschiede des besonders häufig zu verwendenden "cp"-Kommandos (Kopieren von Files) zum "COPY" von DOS soll ausdrücklich hingewiesen werden:

- Wie unter DOS müssen "Quelle" und "Ziel" angegeben werden, das "Ziel" darf allerdings nicht wie unter DOS weggelassen werden (DOS kopiert bei nicht angegebenem "Ziel" in das aktuelle Directory).
- Das "Ziel" darf (wie unter DOS) ein Directory sein. Nur in diesem Fall dürfen mehrere Files als "Quelle" angegeben werden (weil das "Ziel" unter UNIX immer anzugeben ist, kann gegebenenfalls der Punkt **.** das aktuelle Directory zum "Ziel" erklären). Wenn mehrere Files in ein Directory kopiert werden, ändern sie ihre Namen nicht (**COPY *.FOR *.SAV** wie unter DOS ist mit dem UNIX-"cp"-Kommando nicht möglich). Für das gleichzeitige Ansprechen mehrerer Files stehen neben den "Wildcards" (auch "Metazeichen" oder "Joker" genannt) ***** und **?**, die auch DOS kennt (allerdings nicht ganz sauber beherrscht), weitere recht effektive Varianten zur Verfügung, z. B. würde **[a-d]*.{c,exe}** alle Files ansprechen, deren Namen mit den Buchstaben a, b, c oder d beginnt und die eine Extension .c oder .exe haben.
- Im Gegensatz zu DOS dürfen unter UNIX "Quelle" und "Ziel" Directories sein. In diesem Fall wird ein gesamter Unterbaum kopiert (ähnlich dem XCOPY von DOS).

UNIX	Bedeutung	DOS
ls	Directory-Inhalt anzeigen	**DIR**
mkdir	Directory erzeugen	**MD**
rmdir	Directory löschen	**RD**
cd	In Directory wechseln	**CD**
pwd	Aktuelles Directory anzeigen ("Wo bin ich?", unter DOS üblicherweise durch den DOS-Prompt angezeigt)	
cp	File(s) kopieren	**COPY, XCOPY**
mv	File(s) verschieben (und gegebenenfalls umbenennen)	**REN** (umbenennen)
rm	File(s) löschen	**DEL**
cat	Textfile-Inhalt anzeigen	**TYPE**
grep	File(s) nach Zeichenfolge druchsuchen	**FIND**
head	Anfang eines Textfiles anzeigen	
tail	Ende eines Textfiles anzeigen	
more	Textfile bildschirmweise "durchblättern"	**MORE**
lp	Textfile über Druckerspooler ausdrucken	**PRINT**
chmod	Zugriffsrechte (Directories und Files) vergeben	
chown	Eigentümer (Directories und Files) wechseln	
whereis	File(s) suchen	**DIR /S**
who	"Wer ist zur Zeit eingeloggt?"	
whoami	"Wer ist an diesem Terminal eingeloggt?"	
passwd	Passwort ändern	
ps	Anzeigen aller laufenden Prozesse	
date	Datum und Uhrzeit anzeigen	**DATE, TIME**
man	Hilfe (Manual-Text)	**HELP**
mail	UNIX-Mail-System	
write	Kommunikation mit eingeloggten Benutzern	
kill	Signal an Prozeß senden	

4.2.6 Shells, Environment, Shellscripts

- Eine **Shell** ist ein Programm, das nach dem Einloggen die Kommandos entgegennimmt, auswertet, spezielle Kommandos sofort selbst ausführt oder aber einen Prozeß startet (Child Process), der das Kommando ausführt. Die UNIX-Shells sind vergleichbar mit dem DOS-Kommando-Interpreter COMMAND.COM.
- Die Shell stellt für eine Sitzung ein **Environment** (Umgebung) in Form von Parametern bereit (z. B. einen Parameter **PATH**, der mit dem gleichnamigen Paramter aus MS-DOS vergleichbar ist). Die Parameter können während der Sitzung verändert werden.
- Ein **Shellscript** enthält eine Folge vom Kommandos (vergleichbar mit den Batch-Prozeduren in DOS) und kann einer Shell übergeben werden, die dann die Abarbeitung aller Kommandos des Shellscripts organisiert. Dabei können die Shells weit mehr als eine Aufeinanderfolge von Kommandos interpretieren. Die in Shell-Scripts möglichen Konstruktionen ähneln einer höheren Programmiersprache: Variablen, arithmetische Ausdrücke, Schleifen, Bedingungen und sogar die Definition und der Aufruf von Funktionen (auch rekursiv) sind möglich. Shellscripts können andere Shellscripts aufrufen (und sogar rekursiv sich selbst). Da sie interpretativ abgearbeitet werden, ist eine Compilierung nicht erforderlich.

Am Ende des Login-Prozesses wird eine Shell gestartet, die zum "Parent Process" aller in der Sitzung zu startenden Prozesse wird. Die meisten UNIX-Systeme werden mit mehreren unterschiedlichen Shell-Programmen ausgeliefert (neben den "Klassikern" **Bourne-Shell**, **Korn-Shell** und **C-Shell** sind inzwischen viele Weiterentwicklungen verfügbar, die erweiterten Komfort wie z. B. eine graphische Oberfläche bieten). Welche Shell nach dem Einloggen gestartet wird, ist für jeden Benutzer in dem Systemfile **/etc/passwd** verzeichnet. Man kann sich beim System-Manager also gegebenenfalls seine "Lieblings-Shell" wünschen, kann natürlich jederzeit ohnehin jede beliebige verfügbare Shell starten.

Nach dem Start führt die Shell automatisch zwei Shellscripts aus: Für jeden Benutzer wird das Shellscript **/etc/profile** abgearbeitet, das unter anderem die Environment-Variablen auf bestimmte Werte setzt. Anschließend wird (wenn vorhanden) ein benutzer-eigenes Shellscript ausgeführt, das im Home-Directory des Benutzers gesucht wird und z. B. für die Korn-Shell den Namen **.profile** und für die C-Shell den Namen **.login** haben muß (Files, deren Namen mit einem Punkt beginnen, werden normalerweise vom "ls"-Kommando nicht mit aufgelistet). Von diesem Shell-Script kann sich der Benutzer also sein ganz persönliches Environment setzen, sich ganz persönlich begrüßen und beliebige weitere Kommandos ausführen lassen. Anschließend wartet die Shell auf die Eingabe von Kommandos.

Die Syntax eines UNIX-Kommandos hat die Form:

command −options argument1 argument2 ...

Die Optionen modifizieren die Wirkung des Kommandos, z. B. listet

ls −a

auch die mit einem Punkt beginnenden "Hidden Files" auf. Mehrere Optionen können (ohne Leerzeichen hinter einem Minuszeichen eingegeben werden z. B. "ls −al"), bei einigen

Kommandos müssen spezielle Optionen (z. B. die "Library-Optionen" bei den Compiler-Aufrufen) jeweils ihr eigenes Minuszeichen haben (im Zweifelsfall siehe: **man**).

Die Argumente können u. a. Filenamen oder Strings sein, z. B. wird mit

grep Dankert rooms phone

in den beiden Files "rooms" und "phone" nach Zeilen mit der Zeichenkette "Dankert" gesucht.

4.2.7 Stream-Editor sed, Bildschirm-Editor vi

Schon zu den ersten UNIX-Versionen gehörten ausgesprochen leistungsfähige Werkzeuge zur Textbearbeitung. Von den zahlreichen Editoren sollen hier nur die beiden wichtigsten kurz vorgestellt werden.

Vom **Stream-Editor sed** (trotz dieses Namens ein sehr nützliches Werkzeug) wird ein Textfile nach Regeln bearbeitet, die entweder in der Kommandozeile spezifiziert oder von einem anderen File gelesen werden. Er ist nicht für die interaktive Arbeit vorgesehen. Das Ergebnis der Bearbeitung wird auf den Bildschirm ("Standard-Ausgabe") ausgegeben, es sei denn, die Ausgabe wird umgeleitet. So wird z. B. durch das Kommando

sed 's/DDR/Bundesrepublik Deutschland/g' oldfile > newfile

im Textfile **oldfile** die angegebene Operation ausgeführt, das Ergebnis steht im File **newfile** (das Zeichen > bewirkt die "Umleitung" der Ausgabe). Die in diesem Beispiel angegebene Operation ist die besonders häufig benutzte "Substitutions-Operation". Das **s** am Anfang steht für **substitution**, das **g** am Ende für **global**: Überall, wo im Text die Zeichenkette 'DDR' vorkommt, wird sie durch 'Bundesrepublik Deutschland' ersetzt.

Mit der Option **−f** wird erreicht, daß die Editierkommandos von einem File gelesen werden:

sed −f edcomds oldfile

bearbeitet **oldfile** mit Kommandos, die vom File **edcomds** gelesen werden (da keine "Umleitung" angegeben ist, wird das Ergebnis auf dem Bildschirm erscheinen). Auf diese Weise können beliebig komplizierte Editiervorschriften gespeichert und immer wieder auf andere Files angewendet werden.

Der klassische **Bildschirm-Editor** unter UNIX heißt **vi**. Er wird möglicherweise mehr und mehr durch den moderneren Editor **emacs** verdrängt. Beide Editoren sind außerordentlich mächtig und damit natürlich etwas gewöhnungsbedürftig.

Wenn der **vi** z. B. mit

vi myfile

gestartet wird, darf **myfile** bereits existieren und wird in diesem Fall sofort in den Puffer des Editors geladen (geändert wird nur im Puffer, so daß jederzeit ein "Ausstieg ohne Folgen" für das Original möglich ist). Anderenfalls wird ein leerer Puffer bereitgestellt und beim Abspeichern wird **myfile** erzeugt.

Nach dem Start befindet sich der **vi** im **Kommandomodus**: Tastatureingaben werden nicht als Text, sondern als Kommandos interpretiert (dabei wird zwischen Groß- und Kleinbuchstaben unterschieden).

- Der Cursor kann mit den Cursor- und anderen speziellen Bewegungstasten bewegt werden, u. a. sind folgende weitere **Bewegungsmöglichkeiten** nützlich:

H	springt in die erste auf dem Bildschirm befindliche Zeile,
L	springt in die letzte auf dem Bildschirm befindliche Zeile,
M	springt in die Bildschirmmitte.
87G	springt in die Textzeile 87,
G	springt in die letzte Zeile des Textes.
/muster	sucht vorwärts nach dem String **muster**,
?muster	sucht rückwärts nach dem String **muster**,
/	sucht vorwärts nach dem zuletzt eingegebenen Suchstring,
?	sucht rückwärts nach dem zuletzt eingegebenen Suchstring.

  Der Suchstring darf auch ein "**Regular Expression**" sein, in dem z. B. der Punkt **.** für ein beliebiges Zeichen steht, der Ausdruck **[0-9]** steht für eine beliebige Ziffer.

/e.t	sucht nach Zeichenfolgen **eat**, **ebt**, ... , **e0t**, **e1t** ... usw.
/Da[w-z]k	sucht nach den Zeichenfolgen **Dawk**, **Daxk**, **Dayk**, **Dazk**,
/ist\.	sucht dagegen nach der Zeichenfolge **ist.** (der Backslash \ hebt die Sonderstellung des Metazeichens **.** auf).

- **Löschen und Wiedereinfügen** von Text erledigen u. a. folgende Kommandos:

x	löscht das Zeichen an der Cursor-Position,
X	löscht das Zeichen links neben der Cursor-Position,
J	verbindet die aktuelle Zeile mit der nächsten (Löschen des Zeilenwechsels, Cursor darf sich irgendwo in der aktuellen Zeile befinden),
dw	löscht bis zum Beginn des nächsten Wortes,
d$	löscht bis zum Ende der Zeile,
d^	löscht vom Beginn der Zeile bis zur aktuellen Cursor-Position,
dd	löscht eine komplette Zeile.

  Diesen Kommandos können Wiederhol-Zahlen vorangestellt werden, z. B.:

6x	löscht 6 Zeichen,
3dw	löscht 3 Worte,
18dd	löscht 18 Zeilen.

  Die gelöschten Zeilen befinden sich nach dem Löschen in einem Speicher und können an verschiedenen Stellen wieder eingefügt werden:

P	fügt die gelöschten Zeilen oberhalb der Cursorzeile ein,
p	fügt die gelöschten Zeilen unterhalb der Cursorzeile ein.

- Die Kommandos für die **String-Substitution** müssen wie alle mit dem Doppelpunkt : eingeleiteten Kommandos mit <Return> abgeschlossen werden:

:3,17 s/oldstr/newstr/g	ersetzt in den Zeilen 3 ... 17 alle Strings **oldstr** durch **newstr** (ohne das **g** am Ende wird in jeder Zeile nur der erste **oldstr** ersetzt),

:20,$ s/TM//g entfernt (ersatzlos) alle Strings **TM** ab Zeile 20 aus dem Text (**$** steht hier für die letzte Zeile).

Für den zu ersetzenden String (im ersten Beispiel **oldstr**) darf auch ein "**Regular Expression**" stehen, z.B.:

:% s/^/gcc −c /g fügt am Anfang einer jeden Zeile (^ ist das Metazeichen für den Zeilenanfang) den String "**gcc −c** " ein (**%** in der Bereichsangabe bedeutet "Gesamter Text"),

:% s/[A-Z]x/&y/g hängt an alle Zeichenfolgen, die aus einem beliebigen Großbuchstaben und dem kleinen **x** bestehen, noch ein **y** an, aus **Ax** wird **Axy**, aus **Bx** wird **Bxy** usw. (das "Ampersand" **&** im ersetzenden String steht für die gesamte Zeichenfolge, die vom "Regular Expression" des zu ersetzenden Strings gefunden wurde).

Wenn ein Metazeichen in seiner "ordinären Bedeutung" verwendet werden soll, kann seine Metazeichen-Eigenschaft durch vorangestelltes \ abgeschaltet werden, z. B.:

:% s/\./;/g ersetzt jeden Punkt **.** im gesamten Text durch ein Semikolon **;** (durch das vorangestellte Zeichen \ wird das Metazeichen **.** wieder zu einem gewöhnlichen Punkt) .

- **Änderung rückgängig machen:**

  u ("undo") macht die letzte vorgenommene Änderung rückgängig,
  U macht alle Änderungen an der zuletzt bearbeiteten Zeile rückgängig.

- **Änderungen speichern, Arbeit beenden:**

  :q beendet die Arbeit des **vi**, wenn keine Änderungen vorgenommen wurden,
  :q! beendet die Arbeit ohne Speicherung der vorgenommenen Änderungen,
  :w sichert alle vorgenommenen Änderungen (Überspeichern des alten Files) ohne Verlassen des Editors,
  :wq ("write and quit") sichert alle Änderungen, Arbeit des Editors wird beendet.

Für die eigentliche Texteingabe muß aus dem Kommandomodus in den **Schreibmodus** gewechselt werden, z. B. durch:

- **i** ("insert") fügt den Text jeweils vor der Cursorposition ein,
  o öffnet eine neue Zeile unterhalb der aktuellen Zeile.

Aus dem Schreibmodus gelangt man wieder in den Kommandomodus mit der **Esc**-Taste. Wenn man also aus dem Schreibmodus heraus die Arbeit mit dem Abspeichern des Textes beenden will, ist

<Esc> :wq <Return>

einzugeben.

Die vorgestellten Kommandos sind nur ein ganz kleiner Auszug aus der Fülle der Möglichkeiten, die der **vi** bietet, auch die gegebenen Beispiele demonstrieren zum Teil (z. B. beim Substituions-Kommando) nur wenige Varianten zur Verwendung der Kommandos.

4.2.8 Compiler, Linker, ar und make

Neben dem in jedem UNIX-System vorhandenen C-Compiler existieren Compiler für alle gängigen höheren Programmiersprachen (System-Manager fragen oder einfach probieren). Mit **cc** (C-Compiler), **fc** oder **f77** (FORTRAN-Compiler) bzw. **pc** (Pascal-Compiler) erreicht man auf vielen UNIX-Systemen **Compiler-Treiber**, die neben dem Compiler auch den Linker aktivieren (der Linker kann natürlich auch gesondert aufgerufen werden, meist mit **ld**).

Wer z. B. mit dem MS-FORTRAN-Compiler unter DOS vertraut ist, findet sich sehr schnell zurecht. Wie das **fl**-Programm von MS-FORTRAN erwarten auch die UNIX-Compiler-Treiber den kompletten Programmnamen (einschließlich Extension) als Parameter (üblich unter UNIX sind übrigens **.c** für C-, **.f** für FORTRAN- und **.p** für Pascal-Quell-Programme) und können nur durch den Schalter **c** daran gehindert werden, gleich auch noch den Linker zu aktivieren. MS-FORTRAN-Programmierer, die die Schalter (wie unter DOS üblich) mit / einleiten (z. B. **/c** oder **/FPi**), wundern sich über Fehlerausschriften wie **"Unknown flag −fpi"**, da in DOS-Befehlen Groß- und Kleinschreibung sonst keine Rolle spielt und das Zeichen − von ihnen gar nicht verwendet wurde. Ganz offensichtlich kamen die Enwickler von MS-FORTRAN aus dem UNIX-Lager.

Die von den Compilern erzeugten Objectmoduln haben die Extension **.o** und sind auf höchst angenehme Art untereinander kompatibel, auch wenn sie ihren Ursprung in unterschiedlichen Hochsprachen haben (das Problem der Parametervermittlung, das vielen Programmierern schon innerhalb einer Sprache Schwierigkeiten bereitet, ist natürlich nicht zu unterschätzen).

Das vom Linker (bzw. Compilertreiber, wenn nicht **−c** angegeben wurde) erzeugte ausführbare Programm hat den Namen **a.out**, der natürlich geändert werden sollte (mit **mv**), besser ist es, gleich beim Compiler- bzw. Linker-Aufruf (mit der Option **−o**) einen speziellen Namen vorzuschreiben (man beachte den Unterschied zum Link-Programm von MS-DOS, bei dem ohne spezielle Namensangabe das ausführbare Programm den Namen vom ersten angegebenen Objectmodul übernimmt, versehen mit der Extension **.EXE**). Das ausführbare Programm wird mit voller Namensangabe (einschließlich eventuell vorhandener Extension) gestartet (im Gegensatz zu DOS, wo das ausführbare Programm eine Extension haben muß, die dann allerdings beim Aufruf weggelassen werden darf).

Beispiele:

cc hp.c **<Return>**
a.out **<Return>**
... übersetzt das C-Programm **hp.c**, aktiviert den Linker (es entsteht **a.out**) und startet das ausführbare Programm.

cc −o hp hp.c **<Return>**
hp **<Return>**
... übersetzt das C-Programm **hp.c**, aktiviert den Linker (es entsteht **hp**) und startet das ausführbare Programm.

Das Erzeugen und Verwenden von Objectmodul-Libraries ist mit **ar** (steht für "**archives**") genauso einfach und effektiv (und damit empehlenswert) wie unter MS-DOS mit dem Library-Manager LIB. Leider gibt es kleine Unterschiede bei verschiedenen UNIX-Derivaten hinsichtlich der Namensgebung, meist gut beraten ist man mit der Beachtung folgender

- **Empfehlung** (gegebenenfalls siehe "**man ar**"): Der Name einer persönliche Library beginnt mit **lib** und hat die Extension **.a**, z. B.: **libpriv.a**.

Beispiel: **cc −c up1.c**
... übersetzt **up1.c** (es entsteht **up1.o**).
ar −r libpriv.a up1.o
... bringt Objectmodul **up1.o** in die Library **libpriv.a** ein, die erzeugt wird, wenn sie noch nicht existiert. Die Option **−r** sorgt dafür, daß in einer bereits existierenden Library ein darin vorhandenes **up1.o** ersetzt wird.
cc −o hp hp.c −L . −lpriv
... übersetzt das Hauptprogramm **hp.c**. Der Linker durchsucht (veranlaßt durch die Option **−lpriv**) die Library **libpriv.a**, die er allerdings nur findet, wenn mit der Option **−L** das Directory angegeben wird, in dem sich die Library befindet (hier wird durch den Punkt **.** das "Current Directory" spezifiziert). Es entsteht (Option **−o hp**) ein ausführbares Programm **hp**.

Wie unter DOS werden den Libraries nur die benötigten Moduln entnommen, beim Arbeiten mit mehreren Libraries ist jedoch die Reihenfolge der Angabe der Libraries wichtig. Jede Library muß dem Linker mit einer eigenen Option **−l** bekanntgemacht werden, die Libraries werden in der angegebenen Reihenfolge durchsucht, wobei eine Referenz eines Moduls auf eine vorher durchsuchte (weiter links im Kommando stehende) Library ungelöst bleibt.

Für die Verwaltung umfangreicher Projekte ist **make** ein sehr nützliches Werkzeug. Der Programmierer beschreibt in einem **Makefile** die Abhängigkeiten der einzelnen Programmteile voneinander und die Kommandos, mit denen die Programmteile erzeugt werden, wenn eine Änderung ausgeführt wurde. Das Kommando **make** ist damit in der Lage zu entscheiden, welche Operationen ausgeführt werden müssen, um das Projekt "up to date" zu halten (bei einem Quellprogramm, das ein jüngeres Datum als der zugehörige Objectmodul hat, wird von **make** die Übersetzung veranlaßt), unnötige Operationen werden vermieden, Beispiel:

♦ In einem "Mini"-Projekt wird das ausführbare Programm **mpexe** durch Compilieren des Quellprogramms **mp.c** und Linken mit den Objectmoduln **up1.o** und **up2.o** erzeugt, die ihrerseits aus den Quellprogrammen **up1.c** und **up2.c** erzeugt werden. In einem projekt-eigenen Directory befinden sich die Quellprogramme **mp.c**, **up1.c**, **up2.c** und ein

Makefile:

```
mpexe:      mp.c  up1.o  up2.o
        cc −o mpexe  mp.c  up1.o  up2.o
```

Die erste Zeile kennzeichnet die Abhängigkeit des ausführbaren Programms vom zugehörigen Quellprogramm und den beiden einzubindenden Objectmoduln. Die zweite (durch Einrückung mit der **TAB**-Taste - nicht durch Leerzeichen - als solche gekennzeichnete) Zeile enthält das Kommando, mit dem das ausführbare Programm erzeugt wird.

Beim Starten von **make** würden zunächst die beiden Objectmoduln mit

cc −c up1.c
cc −c up2.c

erzeugt werden (weil keine speziellen Kommandos für das Erzeugen von **up1.o** und **up2.o** im **Makefile** existieren, werden von **make** diese sinnvollen Kommandos verwendet). Anschließend wird die Kommandozeile zum Erzeugen von **mpexe** ausgeführt (wie im **Makefile** angegeben).

Bei einem erneuten Start von **make** würde nur die Mitteilung erscheinen:

make: 'mpexe' is up to date

Wenn jedoch z. B. **up1.c** verändert wird (es genügt **touch up1.c**, das Kommando **touch** ändert nur Uhrzeit und Datum auf die aktuellen Werte), dann würden bei einem anschließenden Aufruf von **make** sowohl **up1.o** als auch (wegen der Abhängigkeit) **mpexe** neu erzeugt werden (eine Erneuerung von **up2.o** wird als nicht erforderlich erkannt).

Natürlich könnten auch für **up1.o** und **up2.o** Abhängigkeiten (z. B. von anderen Programmen oder von Include-Files) und spezielle Kommandozeilen für ihre Erzeugung definiert werden.

4.2.9 GNU

Das Projekt **GNU** (die Abkürzung steht für "**GNU is not UNIX**", was eigentlich eine "rekursive Nicht-Definition" ist) wurde 1985 von der Free Software Foundation begründet und verfolgt das Ziel, Software ohne finanzielle oder juristische Einschränkungen zur Verfügung zu stellen. Da die Software im Quellcode verfügbar ist, existieren von den zum Teil ganz hervorragenden Programmen vielfach bereits "Nicht-UNIX-Versionen". Auf eine kleine Auswahl besonders leistungsfähiger Programme soll hier hingewiesen werden:

- **gcc** ist ein ANSI-C-Compiler (auch für MS-DOS verfügbar).
- **g++** ist ein C++-Compiler.
- **f2c** setzt FORTRAN-Quellprogramme in C-Quellprogramme um. Auch wenn die erzeugten C-Programme sich nicht ganz leicht lesen lassen, so funktioniert die Umsetzung doch ausgesprochen zuverlässig, wenn der Quellcode der FORTRAN-77-Norm entspricht.
- **ghostscript** und **ghostview** (auch als MS-DOS- und MS-Windows-Versionen verfügbar) gestatten das Ansehen von PostScript-Files auf dem Bildschirm und das Konvertieren für die Ausgabe auf Nicht-PostScript-Druckern.
- **gzip** ist ein leistungsfähiger File-Komprimierer.
- **emacs** ist ein mächtiger Editor, der auf ASCII-Terminals und X-Windows-Umgebungen läuft.

Da die GNU-Programme wie das UNIX-Derivat LINUX frei kopierbar im INTERNET zur Verfügung stehen, ist gerade die Kombination von LINUX mit den sehr leistungsfähigen GNU-Tools für den privaten Gebrauch und die Benutzung an Hochschulen besonders interessant (auch eine spezielle UNIX-Shell, die sogenannte "Bourn-again-Shell" **bash** findet sich im GNU-Projekt).

Zum Quelltext der GNU-Programme gehört vielfach gleich ein **Makefile**, mit dem auf bequeme Art die ausführbaren Programme erzeugt werden können.

4.3 Rechnernetze

Es werden Netze von selbständigen Computern betrachtet (im Gegensatz zu Terminalnetzen, bei denen mehr oder weniger "dumme" Terminals von einem zentralen Computer bedient werden). Der Benutzer arbeitet in einem solchen Netz auf einem Computer, der mit anderen Computern kommunizieren und von dort Dienste abfordern kann.

4.3.1 Netztopologien, Hardware

Die Verbindungen in Netzen werden vornehmlich durch Kupferleitungen, Glasfaserkabel und Funkstrecken realisiert. Aus topologischer Sicht sind sternförmige Netze, Bus- und Ringnetze dominierend (und Mischformen aus diesen Standardtopologien). Nach einer strengen Vorschrift muß geregelt sein, welche Stationen wann senden dürfen und wie eine Station die für sie bestimmten Nachrichten aus dem Nachrichtenstrom herausfiltert.

Bei einem **Sternnetz** sind mehrere externe Rechner mit einem Zentralrechner verbunden, über den der gesamte Datenaustausch abgewickelt wird (und der alleinverantwortlich für die korrekte Adressierung der Nachrichten ist und der natürlich immer weiß, von welchem Absender eine Nachricht bei ihm aufläuft).

In einem **Busnetz** hängen alle Computer (parallel) an einem Strang. Da alle Nachrichten dadurch zu allen Netzteilnehmern gelangen können, müssen ihnen die Adressen mitgegeben werden. Alle Teilnehmer "lauschen" ständig am Netz, die für sie bestimmten Nachrichten werden aufgenommen (in einen eigenen Speicher kopiert, nicht etwa vom Bus entfernt), die übrigen ignoriert. An jedem Ende des Busses ist ein "Dämpfer" angebracht, der die nicht mehr weiter zu transportierenden Signale "schluckt". Die Frage, welche Station senden darf, ist komplizierter und wird in Busnetzen meist nach dem CSMA/CD-Verfahren geregelt ("carrier sense multiple access with collision detection"): Eine Station darf nur senden, wenn die Leitung frei ist. Dabei kann es zu Kollisionen kommen, wenn zwei Stationen etwa gleichzeitig bei vermeintlich freier Leitung zu senden beginnen. Sie müssen deshalb ständig prüfen ("collision detection"), ob solch ein Fall eingetreten ist und dann sofort beide das Senden einstellen und eine zufallsbestimmte Zeitspanne (damit nicht wieder beide gleichzeitig beginnen) bis zum nächsten Sendeversuch warten. Das **Ethernet**, bei dem die Computer untereinander mit Koaxialkabeln in Bustopologie verbunden sind, ist der gegenwärtig am weitesten verbreitete Standard für lokale Rechnernetze.

In einem **Ringnetz** sind die Verbindungen ringförmig geschlossen (es gibt "weder Anfang noch Ende" des Strangs). Die Nachrichten kreisen in einem vorgegebenen Richtungssinn. Wie beim Busnetz nehmen die einzelnen Stationen nur die an sie adressierten Nachrichten auf. Die Frage, wer senden darf, wird in Ringnetzen meist nach dem **Tokenverfahren** geregelt. Ein bestimmtes Zeichen ("Token"), dessen Bitmuster fest vereinbart und deshalb von allen Stationen erkannt wird, kreist ständig im Ring, und nur die Station, die das Zeichen gerade besitzt, darf senden. Wenn die gesendete Nachricht einmal im Kreis gelaufen ist, wird sie vom Absender selbst wieder aus dem Ring genommen, und das Token wird an die nächste Station weitergereicht. Kollisionen sind damit ausgeschlossen. Wenn in einer zu sendenden Nachricht zufällig das Bitmuster enthalten ist, das dem Token entspricht, so wird dieser Teil der Nachricht nach fest vereinbarten Regeln "verfälscht". Die empfangende Station kennt die

"Verfälschungsregeln" und korrigiert die Nachricht. Das am weitesten verbreitete System, das nach diesem Verfahren arbeitet, ist der **IBM Token Ring**.

Neben den Kabeln und den Netzanschlüssen der Computer ("Netzkarten") gibt es noch weitere Hardware-Bestandteile in Netzen, z. B.:

Ein **Repeater** ist ein Gerät zur Verstärkung der Signale, das sich weder um den Inhalt noch um die Adressen der weiterzuleitenden Signale kümmert. Das Medium, über das die Signale aufgenommen werden, und das Medium, an das sie weitergeleitet werden, können unterschiedlich sein (z. B.: Übergang vom Koaxialkabel zum Glasfaserkabel).

Eine **Bridge** verbindet zwei gleichartige Netzsegmente miteinander und gibt an das jeweils andere Netzsegment nur die Daten weiter, die für einen Empfänger in diesem Segment bestimmt sind, muß also die Adressen auswerten.

Ein **Router** kann zwei gleichartige oder unterschiedliche Netze (z. B. zwei Netze mit unterschiedlicher Topologie) miteinander verbinden und muß damit zumindest teilweise eine Protokoll-Umwandlung vornehmen (siehe folgenden Abschnitt).

4.3.2 Protokolle

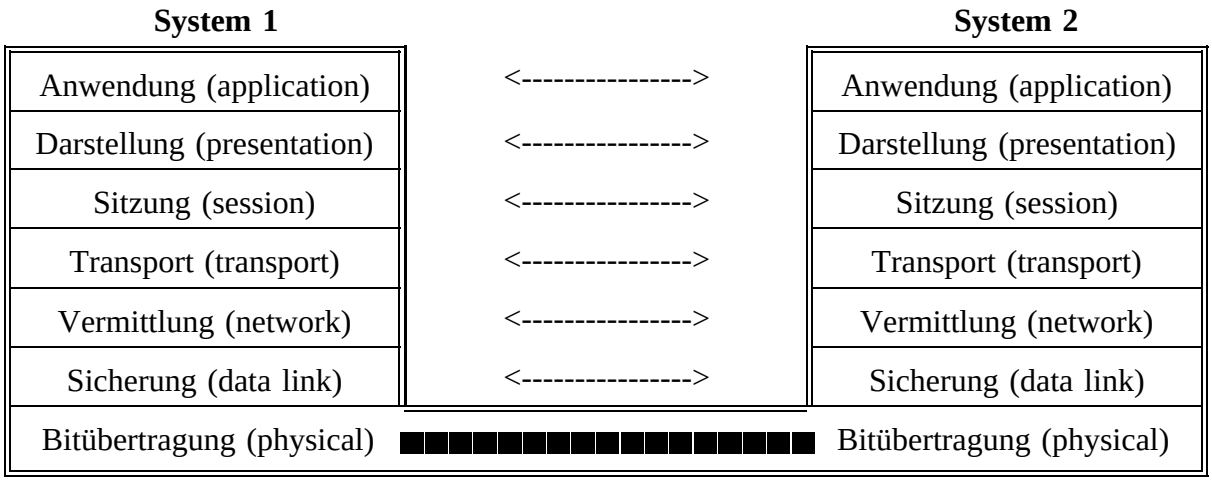
Von **leitungsvermittelter Kommunikation** wird gesprochen, wenn zwischen den beteiligten Computern eine feste Verbindung auch dann vorhanden ist, wenn kein Datenaustausch stattfindet. Bei **Paketvermittlung** werden die zu übertragenden Informationen in Datenpakete verpackt, die mit Adressen (und Informationen, in welcher Reihenfolge die Pakete wieder zusammengefügt werden müssen) versehen und einzeln "auf die Reise geschickt" werden. Dabei können in großen Netzen die Pakete durchaus unterschiedliche Wege gehen. In beiden Fällen sind von den beteiligten Stationen feste Regeln einzuhalten, da sonst eine Verständigung nicht möglich ist.

> Als **Netz-Protokoll** werden alle Vereinbarungen und Normen bezeichnet, die zur Verständigung der Teilnehmer in einem Netz erforderlich sind.

Natürlich war es unvermeidlich, daß verschiedene Hard- und Softwarehersteller und Betreiber von Netzen unterschiedliche Protokolle definierten (die gegenwärtig besonders verbreiteten Protokolle hören auf so schöne Namen wie TCP/IP, Novell-IPX, Decnet-LAT, ISO-OSI, IBM-SNA, Appletalk, ...). Dies führte schon sehr früh zu Standardisierungsbemühungen, die besonders von der **ISO** ("International Organization for Standardization") vorangetrieben wurden. Gegenwärtig sieht es so aus wie mit den meisten Standardisierungsversuchen im Computerbereich: Nicht der von der ISO entwickelte Standard setzt sich durch, das wesentlich ältere und besonders auf UNIX-Anlagen häufig verwendete **TCI/IP**-Protokoll (sprich: "ti-ßi-pi-ei-pi") hat sich als Quasi-Standard etablibiert.

Die ISO-Bemühungen haben aber immerhin zu einem theoretischen Modell für die Kommunikation in offenen Systemen geführt, an dem sich viele Probleme der Kommunikation in Netzen sehr schön darstellen lassen, das

ISO/OSI-Schichtenmodell (OSI steht für "Open Systems Interconnection"):



Das Modell symbolisiert zwei Computer, die miteinander kommunizieren. Jede Schicht eines Computers korrespondiert **logisch** (nicht physikalisch!) mit der entsprechenden Schicht auf der gleichen Stufe des anderen Computers. Eine **physikalische Verbindung** (z. B.: Kabel) gibt es nur auf der untersten Ebene.

Jede Schicht eines Computers erwartet eine bestimmte Art von Diensten von der unter ihr liegenden Schicht und liefert ihrerseits Dienste an die über ihr plazierte Schicht ab:

Über der obersten Schicht sitzt der Benutzer (user), der mit einem Anwendungsprogramm (application) arbeitet (er verfaßt z. B. einen Brief mit einem Mail-Programm). Die Darstellungsschicht darunter kümmert sich um eine geeignete Form der Informationsdarstellung (ASCII- oder anderer Zeichensatz, komprimiert, nicht komprimiert, ...). Die Sitzungsschicht (session) ist mit der Verwaltung beschäftigt (Einloggen, Paßwortabfrage, Synchronisation der Datenübertragung, ...). Die Transportschicht (transport) bereitet die Daten für den Transport vor (Pakete herstellen) und entpackt die ankommenden Pakete. Die Vermittlungsschicht (network) ist für die Adressierung und die Auswahl des Transportweges zuständig. Der eigentliche Transport der Daten wird von der Sicherungsschicht (data link) überwacht (diese fordert bei erkannten Fehlern eine Wiederholung der Übertragung an) und von der untersten Schicht (physical) physikalisch ausgeführt.

Der Nachteil dieses schönen Modells ist, daß das zugehörige ISO-OSI-Protokoll praktisch nur eine untergeordnete Rolle spielt, andere Protokolle sich aber mit diesem Modell nur bedingt abbilden lassen.

4.3.3 On-Line-Dienste

In der zweiten Hälfte der sechziger Jahre entstand in den USA das erste Computernetz (ARPANET), das auf zwei revolutionären Prinzipien beruhte: Es gab keine Zentrale, und das Funktionieren bestimmter Verbindungen war nicht die Voraussetzung für das Funktionieren des Netzes. Als es (etwa zwanzigjährig) starb, hinterließ das ARPANET einen gewaltigen

Nachkommen, das **INTERNET**, das heute Millionen von Computern weltweit miteinander verbindet.

Das INTERNET ist gewachsen als Hochschul- und Behördennetz, was Vor- und Nachteil zugleich ist: Das Fehlen von leistungsabhängigen Abrechnungsmechanismen kommt den in Budget-Katagorien denkenden Anwendern in den Hochschulen entgegen, für die kommerzielle Nutzung erwies sich das INTERNET als nur bedingt geeignet. Da es auch keine Stelle gibt, die mit dem, was im Netz angeboten wird, so richtig Geld verdienen kann, fehlten viele Dienste, die für eine kommerzielle Nutzung sinnvoll sind.

In den letzten Jahren etablierten sich zahlreiche kommerzielle Netzbetreiber. CompuServe ist gegenwärtig weltweit der größte On-Line-Dienst, mit America Online und Prodigy folgen dichtauf weitere in den USA beheimatete Firmen, die aber weltweit agieren. In Deutschland ist gegenwärtig BTX der Telekom (mit der "Jedermann"-Version DATEX-J) am weitesten verbreitet. Die Kosten, die für die Benutzung der On-Line-Dienste anfallen, sind auch für Privatpersonen bezahlbar. Es kann prophezeit werden, daß in wenigen Jahren der Zugang zu einem On-Line-Dienst so selbstverständlich ist wie heute das private Telefon.

Physikalisch wickelt sich der Netzbetrieb vornehmlich über das bestehende Leitungsnetz (Telefonnetz) ab. Ein Computer, ein Modem und die Absicht, sich bei einem On-Line-Dienst anzumelden, genügen, schon ist man weltweit dabei.

Einen besonderen Anreiz für die Teilnahme an den kommerziellen On-Line-Diensten ist die Möglichkeit, über diese Dienste in das INTERNET und damit zu einem gewaltigen Angebot an Informationen zu kommen. Für den Hochschulbereich ist das INTERNET ohnehin ein konkurrenzlos großer Fundus an Informationen und zum Teil ganz hervorragenden Anwender-Programmen.

4.4 INTERNET

Das INTERNET arbeitet mit TCP/IP-Protokollen ("Transmission Control Protocol/Internet Protocol", die Definitionen der Protokolle sind im INTERNET frei zugänglich), die sich nur bedingt in das im Abschnitt 4.3.2 vorgestellte ISO/OSI-Schichtenmodell einordnen lassen: Die drei obersten Schichten werden von den Anwendungsprotokollen abgedeckt, die für die einzelnen Dienste (Telnet, FTP, siehe nachfolgende Abschnitte) definiert und von der entsprechenden Software realisiert werden. Nach dem TCP-Protokoll werden die Aufgaben der Transportschicht erledigt, das IP-Protokoll definiert die Arbeit der Vermittlungsschicht.

4.4.1 Adressen

Die am INTERNET angeschlossenen Computer haben eine weltweit eindeutige **IP-Adresse** (32-Bit-Integer, zur besseren Lesbarkeit wird die Zahl mit einigen Punkten durchsetzt, im Rechenzentrum Berliner Tor gibt es z. B. einen Rechner mit der IP-Adresse 141.22.192.4). Diese Zahlen können als Adressen angegeben werden, es ist allerdings wesentlich übersichtlicher, die von den Betreibern frei wählbaren mehrgliedrigen Namen zu verwenden, die in sogenannten "Domains" verwaltet werden.

Die "Top-Level-Domain" (im Network Information Center in Kalifornien) verwaltet das letzte Glied des Namens (und kümmert sich um die weiteren Namensglieder nicht) und hat Deutschland sinnvollerweise **de** zugewiesen. Alle Adressen der INTERNET-Computer in Deutschland enden also auf **de**. Der nationale Netzverwalter (Rechenzentrum der Uni Karlsruhe) verwaltet das vorletzte Glied des Namens, der Fachhochschule Hamburg ist **fh-hamburg** zugewiesen, und die "Domain" **fh-hamburg.de** darf nun wieder ein Namensglied verwalten, so daß schließlich ein Name wie **fbr036.rzbt.fh-hamburg.de** eindeutig einen Computer **fbr036** in der "Domain" **rzbt.fh-hamburg.de** (Rechenzentrum Berliner Tor in der FH Hamburg in Deutschland) beschreibt.

Wenn ein Computer über einen solchen Namen angesprochen wird, kann über die **Name-Server** diese Adresse schließlich in eine IP-Adresse umgesetzt werden.

4.4.2 Telnet

Mit Telnet kann man sich auf einem entfernten Computer ("Remote Host") einloggen. Der eigene Computer wird dabei zum Terminal degradiert, was zur Folge hat, daß eine Reihe seiner Fähigkeiten nicht mehr zugänglich sind, z. B. kann man auf die eigenen Laufwerke nicht mehr zugreifen, so daß man mit Telnet nicht ohne weiteres Files vom eigenen auf den entfernten Computer übertragen kann (und umgekehrt natürlich auch nicht). Im Novell-Netz des Rechenzentrums Berliner Tor kann man sich mit

RUN TELNET

diesen Netzdienst zugänglich machen (vorläufig beschränkt auf den Raum 201). Wer einen Account auf einem UNIX- oder VMS-Rechner des Rechenzentrums besitzt, kann sich dann via Telnet auf einem dieser Rechner einloggen und dann auf diesem Rechner arbeiten. Man kann sich natürlich auch auf irgendeinem Rechner irgendwo in der Welt einloggen, sofern man eine Zugangsberechtigung hat.

Viele Rechner gestatten ein "Guest Login" mit natürlich nur beschränkten Rechten. Zum Beispiel kann man sich auf einem Rechner der "Akademischen Software Kooperation" in Karlsruhe, wo viele interesannte Programme zu finden sind, mit dem "Login Name" **ask** und dem Paßwort **ask** einloggen, wenn man mit

telnet askhp.ask.uni-karlsruhe.de

eine Verbindung zu diesem Rechner hergestellt hat. In vielen UNIX-Systemen leistet der "Remote Login"-Befehl **rlogin** den gleichen Dienst.

Auch unter Windows ist das Arbeiten auf "Remote Hosts" möglich (im Novell-Netz im Rechenzentrum Berliner Tor wird dieser Dienst mit **RUN TELNETW** gestartet).

So richtig interessant wird es, wenn man auf diesem Wege in ein **X-Window-System** eindringt: Ein X-Window-System verlagert (grob gesprochen) einen Teil der (zum Bildaufbau benötigten) Intelligenz vom Computer in das Terminal (ein sogenanntes X-Terminal, das vom Computer mit einer speziellen Sprache angesprochen wird und selbst wie ein kleiner hochspezialisierter Rechner den Bildaufbau erledigt, vergleichbar ist der Vorgang mit dem Ablauf in einem PostScript-Drucker). Wenn man über Telnet mit einem X-Window-System arbeitet, dann muß der eigene Rechner die Funktion des X-Terminals übernehmen (und wird auf diese Weise nicht ausschließlich zum "dummen" Terminal).

4.4.3 File-Transfer (FTP, Anonynous-FTP, KERMIT)

Mit **FTP** ("File Transfer Protocol") können Files von einem Computer zu einem anderen übertragen werden. Natürlich benötigt man auf beiden Computern die erforderlichen Rechte (Leseberechtigung für Files, Schreibberechtigung im Ziel-Directory).

Im Novell-Netz des Rechenzentrums Berliner Tor startet man mit

RUN FTP

das Programm (vorläufig beschränkt auf den Raum 201), das sich dann mit dem Prompt **ftp>** meldet und auf weitere Eingaben wartet.

Der unter FTP verfügbare Befehlssatz beschränkt sich auf die Befehle, die für den File-Transfer und das Manöverieren und Orientieren in File-Systemen erforderlich sind (man kann also gegenüber dem "Remote Login" via Telnet auf dem "Remote Host" nur vergleichswenig machen). Mit **?** bekommt man eine Befehlsübersicht, mit **? get** z. B. Hilfe für den Befehl zum File-Transfer vom entfernten Rechner zum lokalen Rechner.

Die Verbindung zum entfernten Rechner wird mit der **open**-Anweisung hergestellt, danach wird die Zugangsberechtigung geprüft, die man z. B. mit einem Account auf diesem Rechner durch Absolvieren der üblichen Einlogg-Prozedur nachweisen kann.

In vielen Rechenzentren (besonders im Hochschulbereich) stehen für die Netzteilnehmer auf FTP-Servern Informationen und Programme unentgeltlich zur (Kopier-)Verfügung. Der Zugang ist über einen imaginären Benutzernamen (meistens **anonymous** oder **ftp**) möglich, bei der Paßwortabfrage ist es üblich, seine eigene komplette E-Mail-Adresse anzugeben, z. B. landet man mit

ftp> open ftp.uni-paderborn.de

auf dem FTP-Server der Universität Paderborn, beantwortet die Frage nach dem Login-Namen mit **anonymous** und die Paßwortabfrage mit der eigenen kompletten E-Mail-Adresse, die z. B. **dkt@rzbt.fh-hamburg.de** lauten könnte. Danach kann man sich gegebenenfalls Files kopieren (mit **get**, mit **ls** erfährt man, was angeboten wird), mit dem FTP-Kommando **bye** werden die Verbindung und das FTP-Programm beendet.

Einen noch etwas komfortableren Zugang hat man über MS-Windows (im Novell-Netz im Rechenzentrum Berliner Tor werden Windows und sofort dieser Dienst mit **RUN FTPW** gestartet).

Die Übertragung der Files zwischen unterschiedlichen Systemen erfolgt mit einer gewissen Intelligenz, man sollte darauf achten, daß ein Unterschied zwischen Text-Files und Binär-Files gemacht wird. Im Zweifelsfall ist man mit der Einstellung "Binär" (**binary)** gut beraten, dann allerdings wird z. B. bei Übertragung von DOS nach UNIX (und umgekehrt) bei Text-Files die in beiden Systemen unterschiedliche Darstellung des Zeilenwechsels nicht repariert. Wenn man also sicher ist, nur Text-Files (z. B. Programme im Quellcode) zu übertragen, sollte man den Text-Modus (**ascii**) einstellen. Während sich "binary" übertragene Text-Files i. a. (wenn überhaupt erforderlich) mit erträglichem Aufwand reparieren lassen, sind "ascii" übertragene Binär-Files (z. B. ausführbare Programme) Kandidaten für den Abfalleimer.

Der Transfer von Files von einem Computer zum anderen wurde früher vornehmlich mit dem (inzwischen etwas in die Jahre gekommenen) Programm **kermit** ausgeführt, das für sehr viele Systeme (von Atari über DOS und VMS bis zu UNIX und noch viele, viele andere) verfüg-

bar ist. Da **kermit** auch eine Terminal-Emulation enthält, erledigt es sogar neben den FTP-Aufgaben auch Telnet-Dienste. Es ist frei kopierbar und wird sicher noch einige Zeit seine Fans haben.

4.4.4 Network File System (NFS)

Mit **NFS** kann ein File-System physikalisch auf verschiedenen Computern liegen, stellt sich für den auf einem Computer arbeitenden Benutzer jedoch wie ein einheitliches File-System dar. Ein mit MS-DOS arbeitender Benutzer greift zum Beispiel auf ein Directory eines UNIX-Rechners zu, das sich ihm (und seinen Anwendungsprogrammen) wie ein zusätzliches (DOS-)Laufwerk darstellt.

Das Laufwerk auf dem entfernten Rechner ist nach dem Prinzip des "Einhängens eines File-Systems in ein anderes" ("mounten", vgl. Abschnitt 4.2.4) angebunden. Natürlich kann es dabei zu Problemen mit den Zugriffsrechten kommen (**mount** setzt auch unter UNIX spezielle Berechtigungen voraus).

Im Novell-Netz des Rechenzentrums Berliner Tor startet man mit

RUN NFS

diesen Dienst (vorläufig beschränkt auf den Raum 201). Danach steht dem Benutzer ein zusätzliches Laufwerk E: zur Verfügung, das sich physikalisch auf einem UNIX-Rechner befindet. Um die vollen Zugriffsrechte für dieses Laufwerk zu bekommen, muß sich der Benutzer (Berechtigung dafür vorausgesetzt) "auf diesem Laufwerk einloggen" (mit einem UNIX-Account) und kann dann für die von ihm erzeugten Files alle unter UNIX möglichen Zugriffsrechte vergeben.

NFS kann auch genutzt werden, um auf periphere Geräte (z. B.: Drucker) des entfernten Rechners zuzugreifen. Nach **RUN NFS** im Novell-Netz des Rechenzentrums Berliner Tor werden Hinweise aufgelistet, wie man einen Drucker "mounten" kann (ein doppelseitig druckender DIN-A4-Laser-Drucker für die Ausgabe von PostScript-Files wird bei RUN NFS automatisch "gemounted").

4.4.5 Auskunftsdienste

Die nicht mehr überschaubare Menge an Information, die im Internet "herumliegt", erfordert zusätzliche Netzdienste, um die gewünschten Informationen zu finden. Von den zahlreichen Diensten wird hier nur eine kleine Auswahl vorgestellt.

Archies helfen beim Suchen nach Files, deren Namen man (zumindest ungefähr) kennt. Auf dem eigenen Computer muß ein "Archie-Client" eingerichtet sein, ein Programm, das sich in den "Archie-Servern" des Internet nach den gesuchten Files umsieht. Im Novell-Netz des Rechenzentrums Berliner Tor startet man mit

RUN ARCHIE

einen Archie-Client (vorläufig beschränkt auf den Raum 201) unter MS-Windows, dessen Benutzeroberfläche selbsterklärend ist.

Mit "Archies" kann man herausfinden, wo sich ein File befindet, zur Beschaffung steht der FTP-Dienst zur Verfügung (moderne "Archie"-Versionen erledigen auch diese Aufgabe).

Auf einer höheren Stufe an Intelligenz und Bequemlichkeit arbeiten die **Gopher**. Auch hier läuft lokal ein "Gopher-Client" (solche Programme liegen übrigens im Internet frei kopierbar herum, bei der Beantwortung der Frage, wo man man sie finden kann, helfen zum Beipiel "Archies" und "Gopher"). Die Gopher-Server im Internet wissen viele der gesuchten Informationen selbst nicht, wissen aber, welcher andere Gopher es weiß. Der Benutzer merkt in der Regel kaum etwas davon, wenn der von ihm angezapfte Gopher sich die Information bei einem Kollegen beschafft. Mit einem Gopher-Client kann man sich die gewünschten Files auch gleich beschaffen lassen.

Im Novell-Netz des Rechenzentrums Berliner Tor startet man mit

RUN GOPHER

einen Gopher-Client (vorläufig beschränkt auf den Raum 201), unter MS-Windows.

4.4.6 World Wide Web (WWW)

Unter den Informationssystemen nimmt seit einigen Jahren das **World Wide Web** ganz eindeutig eine Sonderstellung ein. Der Grund dafür war sicher zunächst, daß die nach dem **HTTP**-Standard ("Hypertext Transport Protocol") übertragenen **Hypertexte** auch den hartnäckigsten "Computer-Muffeln" ein Navigieren in den Informationen ermöglichten (in dem Moment, wo ich diesen Satz schreibe, fallen mir allerdings einige "Muffel" ein, die ganz sicher auch das WWW unbeschadet überstehen werden). Mit dem Erfolg kommt der Erfolg: Ein viel benutztes System ist auch für die Anbieter interessanter, das Angebot wächst, und das System wird noch interessanter.

Hypertexte sind Informationen, die besonders gekennzeichnete Stichworte enthalten, die auf weitere Informationen verweisen, die in anderen Files auf anderen Servern (in einem anderen Land) gespeichert sein können (eigentlich ist es zutreffender, von **Hypermedia** zu sprechen, denn die Files enthalten natürlich auch Bilder, Videos und akustische Informationen).

Lokal muß ein Client-Programm gestartet werden, das als **Browser** bezeichnet wird. Gegenwärtig ist dafür ein Programm mit dem Namen **MOSAIC** besonders beliebt. Nach dem Starten eines WWW-Browsers landet man in seiner **Home Page**, dem Startpunkt für alle weiteren Wanderungen durch die Informationswelt. Im allgemeinen bietet eine Home Page zahlreiche (durch Unterstreichen markierte) Stichworte an, die man mit der Maus anklicken kann, worauf man in der nächsten Seite landet, die möglicherweise wieder solche Stichworte enthält usw.

Es gibt noch eine andere (vielfach schnellere) Möglichkeit, an eine gewünschte Information zu gelangen. Alle für das WWW aufbereiteten Informationen sind durch einen eindeutigen **Uniform Resource Locator (URL)** gekennzeichnet (gewissermaßen die Adresse der Information), und nach dem Eingeben eines URL wird sofort versucht, die zugehörigen Informationen zu lesen und die WWW-Seite aufzubauen. Die URL haben einen ähnlichen Aufbau wie die Internet-Adressen, z. B. landet man mit http://www.informatik.tu-muenchen.de in der Home-Page des WWW-Servers der Technischen Universität München (mit vielen unterstrichenen Begriffen, mit denen man sich zu weiteren Informationen durchhangeln kann).

Im Novell-Netz des Rechenzentrums Berliner Tor startet man mit

RUN WWW

MS-Windows und den WWW-Browser NETSCAPE (vorläufig beschränkt auf den Raum 201), und es erscheint die Home-Page der Fachhochschule Hamburg (natürlich mit zahlreichen unterstrichenen Begriffen ...).

Gerade bei der Nutzung des WWW wird immer wieder deutlich, daß es natürlich noch sehr viele Engpässe auf der "Daten-Autobahn" gibt, und die Anzahl der Netz-Teilnehmer wächst schneller als die Übertragungskapazität. Es genügt ein "Flaschenhals" auf der Übertragungsstrecke, um erhebliche Verzögerungen zu versuchen. Wenn man frustriert einige Minuten auf den Aufbau einer angewählten Seite warten muß, sollte man die Zeit nutzen, um darüber nachzudenken, daß es trotz alledem doch faszinierend ist, vor einem Computer in Hamburg zu sitzen, eine WWW-Seite eines Servers in München zu betrachten, um durch einen Mausklick auf ein Stichwort dieser Seite die nächste Seite aus Washington anzufordern.