Linux Einführungskurs

Burkhard Obergöker
Oktober 2010

Inhalt

1 Bevor es losgeht ...

Das Betriebssystem Linux ist mittlerweile eines der verbreitetsten Betriebssysteme überhaupt geworden, obwohl es nicht die offensichtliche Einfachheit anderer Betriebssysteme bietet, die in der Werbung propagiert wird. Es gibt eine große Fangemeinde, Entwicklergruppen und Sponsoren, so dass es sich wirklich lohnt, dieses System kennen zu lernen.

In diesem Kurs soll das System "Linux" vorgestellt werden und in Eigenarbeit die eigentliche Benutzung erlernt werden, um ein „Gefühl“ für diese Art des Systems zu bekommen. Dazu muss man sich allerdings ein wenig mit den technischen Gegebenheiten vertraut machen, um zu verstehen, wieso manches funktioniert und anderes nicht funktionieren kann. Schließlich muss man beim Auto fahren auch wissen, warum bei angezogener Handbremse der Wagen nicht vorwärts kommt.

Wem das alles aber viel zu trocken ist und auch schon ein laufendes Linux-System vor sich hat, der sollte im Kapitel „Die Schnittstelle zum Benutzer“ weiterlesen.

2 Historie

Zunächst einmal gab es in der ”Steinzeit” der Rechnerentwicklung die großen Systeme, die ganze Lagerhäuser füllten und nicht mehr Leistung brachten als ein heutiger programmierbarer Taschenrechner. Diese Rechner wurden nur von einer Stelle aus bedient, der Konsole, und konnten auch immer nur eine Aufgabe zurzeit bearbeiten, so dass sie auch dann für andere Aufgaben blockiert waren, wenn sie nur auf eine Eingabe warteten.

Als die Leistungsfähigkeit stieg, stellte man diese Rechner mehreren Personen gleichzeitig zur Verfügung, so dass sich der Rechner mit mehreren Aufgaben beschäftigen konnte. Diese Art der Oberfläche, die als "zeichenorientiert" bezeichnet wird, ist immer noch Standard in der Großrechnerwelt, wie z.B. bei IBM x390.

In der Welt der UNIX-Rechner stellte sich aber ein anderes Konzept vor: So wurde es jedem Benutzer ermöglicht, gleichzeitig mehrere Programme auszuführen, zwischen denen er per Tastendruck wechseln konnte. Das ging natürlich nur, wenn es sich um selbstständig arbeitende Prozesse handelte, die parallel zu der aktuellen Arbeit gestartet werden konnten.

Nun kamen aber schon Arbeitsplatzrechner auf den Markt, die eine einfachere Bedienung zuließen weil sie mit Hilfe der Maus und grafischen Oberflächen benutzt werden konnten. So entwickelte sich im Unix-Bereich eine Komponente, die diese Benutzerfreundliche Oberfläche auch zur Verfügung stellte: X-Windows (oder einfach nur "X") Sie wurde als Modul entwickelt und passt sich hervorragend in das bestehende Betriebssystemkonzept ein, da es lediglich eine Option darstellt.

Im PC-Betriebssystemmarkt konnte man eine analoge Entwicklung beobachten, wobei am Anfang das MS-DOS stand, das üblicherweise per Kommandozeile gesteuert wurde. Von den grafischen Oberflächen setzte sich vorrangig nur ein Hersteller durch, der sich als Marktführer in diesem Bereich entwickeln konnte.

Da dieses Windowssystem ständig kompatibel zu seinem Vorgänger sein wollte, blieb diese Kompatibilität wie ein "Klotz am Bein" ständig als hemmender Faktor im System, was dazu führte, dass Windows nie das aus einem Rechner an Leistung herausholen konnte, zu dem die Hardware eigentlich in der Lage war.

Einen anderen Weg beschritt der finnische Student Linus Torwalds, der aus einer kleinen Semesterarbeit "spaßeshalber" einen Prozess-Scheduler baute, der auf den internen Prozessor-Strukturen des 386er von INTEL basierte. Dieser Betriebssystemvorläufer wurde von ihm und anderen konsequent ausgebaut, bis daraus ein vollständiger Betriebssystemkern wurde, der alle nötigen I/O-Operationen sowie Applikationen verwalten konnte. Dieser Kern ist das eigentliche Linux, weil alles "Drumherum" wie die Systemprogramme, die z.B. zur Dateiverwaltung, Textbearbeitung, Benutzerverwaltung etc. benötigt werden, einer anderen Quelle entstammen, dem **GNU-Projekt**. Dieses Projekt entstammt der Open Software Foundation und ist ein rekursives Akronym: GNU= „**G**NU’s **n**ot **U**NIX“

Linux ist somit im Grunde genommen ein Unix-Derivat, mit dem Unterschied, dass sämtliche Teile (des Kernels) nicht dem Copyright einer bestimmten Firma unterliegen. Sie sind in ausführbarer Form als auch in Sourcecode frei verfügbar, weil sie von Grund auf neu programmiert worden sind. Natürlich wurde UNIX dabei als Vorbild benutzt. Das ist auch daran zu erkennen, dass nur sehr wenige Kernelkomponenten in Assembler und nicht in C vorliegen, was dem Unix-Prinzip entspricht und der Portierung auf andere Hardwareplattformen entgegenkommt.

3 Aufbau des UNIX-(und Linux-) Systems

Wie aus der Historie schon erkennbar, besteht Linux - so wie man es als Linux-Distribution kaufen kann - nicht aus einem festen, unteilbaren Block, sondern setzt sich aus einer Vielzahl von Paketen zusammen, die ihrerseits aus klar definierten Modulen bestehen. So ist es nicht verwunderlich, dass viele dieser Module auch in anderen Systemen wieder zu finden sind. So ist beispielsweise der GNU C++ Compiler nicht nur für Linux- sondern auch für OS/2, HP-UX, AIX und auch als Windows-NT-Version verfügbar.

Das Herz des Linux-Systems ist - wie die Bezeichnung schon andeutet - der Kernel, und somit das eigentliche Linux. Dieser Kern ist für die Hardware verantwortlich und stellt die Schnittstellen zwischen den Applikationen und der Hardware dar. Da der Umgang mit dem Kern direkt nur recht umständlich von Statten geht, wird die nächste Schicht durch die Systemprogramme gebildet, die die Zugriffe auf den Kernel als separate Programme darstellen. Getreu dem Grundsatz "small is beautiful" ist für fast jeden Linux/UNIX-Befehl ein eigenes Programm geschrieben worden, um die Entstehung eines übergroßen "Dinosaurier"-Kerns zu verhindern. So wird gewährleistet, dass wirklich nur die Teile in den Hauptspeicher geladen werden, die auch wirklich benötigt werden.

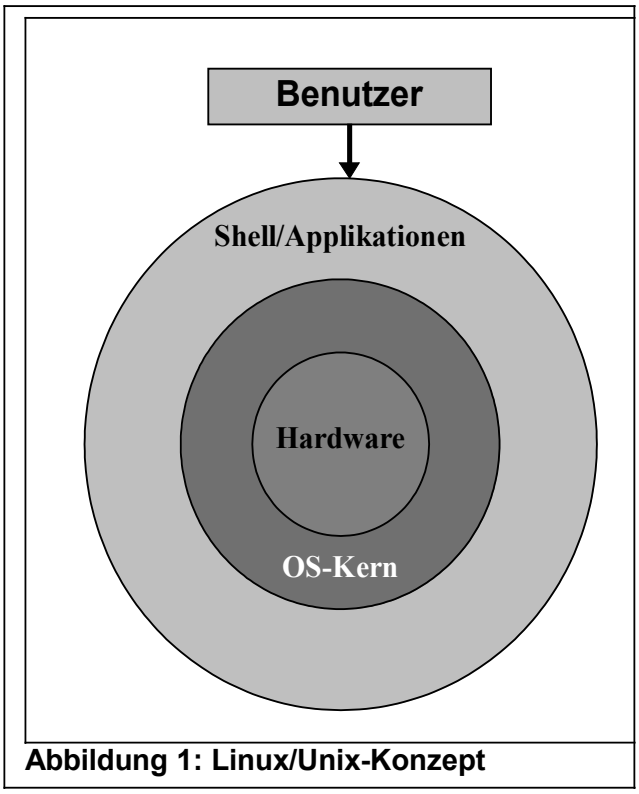

Abbildung 1: Linux/Unix-Konzept

Als oberste Schicht existieren letztendlich die eigentlichen Applikationen, die üblicherweise die einzige Schnittstelle zum System sind, die ein Benutzer zu sehen bekommt. Dabei kann es sich sowohl umso "triviale" Dinge handeln wie einen Dateimanager, wie auch um aufwändige Applikationen von der Textverarbeitung bis hin zum PPS-System.

Diese Modularisierung ermöglicht den unkomplizierten Austausch bestimmter Komponenten wie auch deren Neuentwicklung. So kann die unterste Schicht, die Hardware prinzipiell beliebig geändert werden, solange sie den Mindestanforderungen für ein

solches System entspricht. Tatsächlich existieren Linux-Versionen (Portierungen) für folgende Hardwareplattformen:

- INTEL x86 (386sx bis Pentium III)
- INTEL x86-64 (AMD Opteron)
- INTEL ia64 (Intel Itanium)
- MIPS
- Alpha
- PowerPC (Apple, IBM)
- Motorola 68x00 (Apple Macintosh, Amiga, Atari)
- Acorn
- PA-RISC (Hewlett-Packard)

Diese Liste kann und wird nicht vollständig sein, weil fast täglich neue Komponenten und Portierungen herauskommen.

4 Benutzerverwaltung

4.1 Benutzer

Linux entstammt dem UNIX und hat somit seine Wurzeln in Großrechnersystemen, die grundsätzlich darauf angewiesen sind, ihre Daten zu schützen. Daher muss jeder, der mit einem Linux-System arbeiten möchte, sich zunächst identifizieren. Dazu dient die Kombination aus einem Namen (User) und einem Passwort. Der Name wird vom Superuser festgelegt, während das Passwort von dem betreffenden Benutzer beliebig verändert werden kann. Jeder Benutzer hat ein “home directory”, in dem er üblicherweise uneingeschränkt Dateien anlegen, verändern und entfernen kann, der Rest des Systems bleibt für ihn – wenn überhaupt – nur lesbar.

4.2 Der Superuser (root)

Wie schon angedeutet, existiert im System ein privilegierter Benutzer, der in jedem Unix-System vorhanden ist und dessen Name grundsätzlich “root” (=”Wurzel”) lautet. Dieser *Superuser* hat alle Rechte, die in einem System möglich sind. Selbst wenn eine Datei vor „root“ geschützt sein sollte, kann er sich darüber hinwegsetzen. Unter dem root-Account sollte also nur in Ausnahmefällen gearbeitet werden, da sonst bei Unachtsamkeit die Gefahr besteht, Teile oder das komplette System zu zerstören. Über den root werden sämtliche administrativen Aufgaben erledigt, wie z.B. Benutzer anlegen, Drucker verwalten, Netzwerk konfigurieren etc..

4.3 Gruppen

Weiterhin werden die angelegten Benutzer noch festen Gruppen zugeordnet, deren Namen ebenfalls vom root festgelegt werden. Jeder Benutzer **muss** einer Gruppe angehören, gruppenlose Benutzer gibt es per Definition in Unix-Systemen nicht, es sei denn der root hat fehlerhaft administriert.

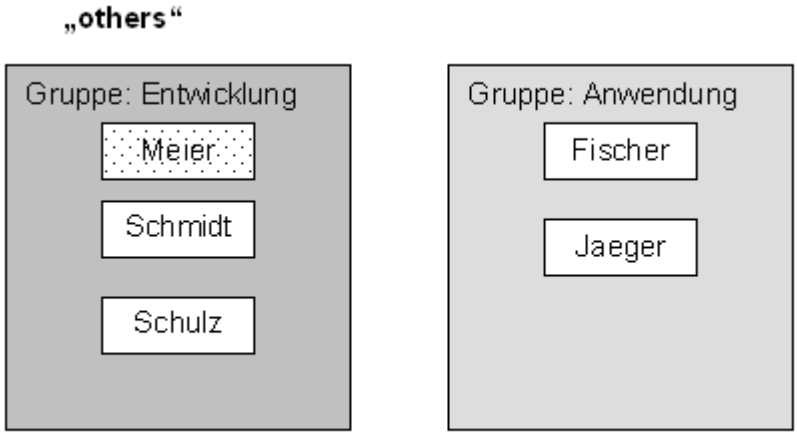

Abbildung 2: Benutzer- und Gruppensystem

Wie in dem obigen Bild angedeutet, teilt sich die Welt für jeden Benutzer in 3 Personenkreise: In „User“, „Group“ und „Others“. Aus der Sicht des Benutzers „Meier“ manifestiert sich diese Struktur folgendermaßen:

- **User:** Alles, was dem Benutzer „Meier“ gehört und unter seiner Berechtigung läuft.

- **Group:** Alles, was zu derselben Gruppe wie er selbst gehört, also in diesem Fall Gruppe „Entwicklung“. Hierzu gehören die Benutzer Schmidt, Schulz und natürlich User Meier selbst und alle Objekte, die zu diesen Benutzern gehören.
- **Others:** Alles, was weder zu Benutzer Meier noch zur Gruppe „Entwicklung“ gehört, also die Benutzer „Fischer“ und „Jaeger“ sowie alle anderen Benutzer, die sonst noch im System existieren. Auch hier manifestiert sich die Zugehörigkeit in den Dateien und Verzeichnissen, die zu den Benutzern oder Gruppen gehören.

5 Dateisysteme

5.1 Die Struktur des Dateisystems

In einem solchen System müssen natürlich auch Daten abgelegt werden, so dass eine Form der Datenspeicherung vonnöten ist. Ein solches System wird als ”Filesystem” bezeichnet, weil es eine Anzahl von Dateien in eine geordnete Struktur bringt.

Unter einer Datei versteht man die kleinste Einheit eines solchen Systems: Eine einheitlichen Sammlung fortlaufender Daten – wie z.B. dieser Text – versehen mit einem eindeutigen Namen, der diese Datei identifiziert. Der Inhalt dieser Dateien wie auch deren Namengebung hat primär keinen Einfluss auf das System.

Diese Struktur ist unter UNIX allg. hierarchisch, d.h. sie ist – ähnlich wie bei MS-DOS – in Verzeichnisse (=Directories) unterteilt, in denen sich wiederum Dateien und Verzeichnisse befinden können.

Das Hauptverzeichnis des Verzeichnis-”*Baumes*” bildet die ”Wurzel”, das ”**root directory**”. Von dieser Stelle aus zweigen alle erreichbaren Unterverzeichnisse ab, wobei ein Verzeichnis wohl mehrere Unterverzeichnisse, aber nur max. 1 Oberverzeichnis besitzen kann. Da das root das Hauptverzeichnis ist, hat dieses auch kein übergeordnetes Verzeichnis.

Soll die Position eines Verzeichnisses innerhalb der Verzeichnisstruktur angegeben werden, so werden einfach alle Verzeichnisse durch ”/” getrennt aneinandergehängt. Die hierarchische Reihenfolge muss dabei eingehalten werden, so dass mit ”/” begonnen werden muss. Das unten abgebildete Beispiel zeigt den Pfad zum Verzeichnis ”/usr/src/linux”. Um eine Datei zu identifizieren wird deren Name einfach an das letzte Verzeichnis – wiederum durch ”/” abgetrennt – angehängt. Der Pfad zur Datei README im bezeichneten Verzeichnis ”/usr/src/linux” würde also ”/usr/src/linux/README” lauten. Achtung: Unter Linux wie auch unter UNIX ist die Groß-/Kleinschreibung signifikant. ”README” ist also eine andere Datei als ”readme”.

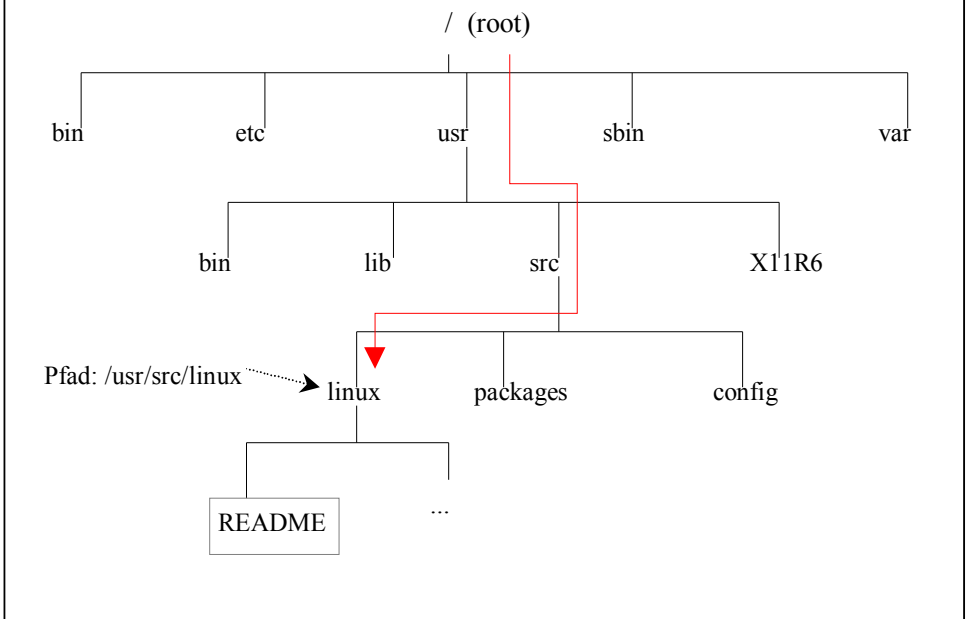

Abbildung 3: Verzeichnisbaum

Um nicht ständig den kompletten Pfad anzugeben, kann auch von dem sog. ”aktuellen Verzeichnis” (= ”working directory”) ausgegangen werden, indem der erste ”/” weg-

gelassen wird. So kann die erwähnte Datei ”README” auch durch ”linux/README” erreicht werden, sofern man sich bereits im Verzeichnis ”/usr/src” befindet.

5.2 Dateitypen

Allgemein gibt es in UNIX/Linux-Systemen 4 verschiedene Arten von Dateien:

1. **Normale Dateien (ordinary files)**

   Diese Dateien enthalten alle Sorten von Daten, die sich in irgendeiner Form verarbeiten lassen. Das können also reine Texte, Datenbankdaten aber auch Programme in maschinenlesbarer Form sein. Der Typ der enthaltenen Daten wird ausschließlich über den Inhalt festgelegt, nicht – wie bei Windows-Systemen – über eine Endung. Tatsächlich wird jedoch oft der Übersicht halber eine Endung angehängt, wie z.B. ”.txt”, ”.html” etc., die aber keinerlei Auswirkungen auf das umgebende System besitzen. Darüber hinaus ist man nicht auf die 3-Stellen-Endung festgelegt. Jeder Dateiname kann bis zu 255 Zeichen lang sein, wobei prinzipiell jedes ASCII-Zeichen erlaubt ist. Es sollten jedoch möglichst nur Buchstaben (ohne Umlaute) und Ziffern verwendet werden, da andere Zeichen zu Problemen führen können.

2. **Verzeichnisse (directories)**

   Verzeichnisse stellen nichts anderes als ”Inhaltsverzeichnisse” dar, die tabellenartig bestimmte Informationen über Dateien darstellen.

3. **Geräte (devices)**

   Jedes Gerät, das vom Rechner aus angesprochen werden kann, wird als eine ”Gerätedatei” dargestellt, um die IO-Befehle möglichst einheitlich zu gestalten. Alle Gerätedateien sollten sich im Verzeichnis ”/dev” befinden. Zu diesen Geräten gehören Tastatur, Bildschirm, CDROM genauso wie z.B. Hauptspeicher, Festplatte, Systemuhr und viele mehr.

4. **FIFO-Dateien (pipes)**

   Diese speziellen Dateien finden Ihre Bedeutung in wenigen Situationen, z.B. beim Datenaustausch zweier Prozesse. Eine FIFO- (First In First Out) Datei wirkt wie eine Warteschlange: Auf der einen Seite wird etwas hineingeschrieben, während auf der anderen Seite etwas herausgenommen wird. „Herausgenommen“ ist bei diesen Dateien wörtlich zu verstehen, weil nicht einfach aus der Datei gelesen, sondern auch die gelesenen Daten automatisch gelöscht werden. Sie werden vornehmlich dann eingesetzt, wenn zwei Programme ”hintereinander” geschaltet werden, also das zweite Programm die Ausgabe des ersten als Eingabe verwendet.

5.3 Technischer Aufbau eines Dateisystems

Ein zusammengehöriges System von Dateien, wird als ”filesystem” bezeichnet, das grob dem Laufwerk unter Windows entspricht (z.B. A:, C:, D: etc.). Besteht ein Linux-System im einfachsten Fall aus einer einzigen Festplatte, so ergibt sich der folgenden Aufbau:

Ab Beginn der Festplatte befindet sich der sog. ”master boot record” (MBR), der dem PC beim Startvorgang (”booten”) die Stelle der nächsten Anweisungen übergibt. Dieser ist sehr klein und darf auch nur an einer bestimmten Stelle existieren.

Ein weiterer besonderer Bereich befindet sich üblicherweise, aber nicht zwingend am Ende der Festplatte: Der Swap-Bereich. Dieser wird während der Laufzeit als virtueller Hauptspeicher genutzt, was im nächsten Kapitel genauer beschrieben wird.

Der Hauptbereich jedoch ist der Datenbereich, das eigentliche File-System. Ein Filesystem wird normalerweise nach seinem ”mount point”, dem Einhängepunkt im Hauptsystem benannt. Das Hauptsystem selbst beginnt mit der Wurzel, der ”root” (”/”) das das einzige Filesystem darstellt, das der Rechner haben **muss**. Ohne Root-Filesystem kann der Linux-Kernel nicht starten.

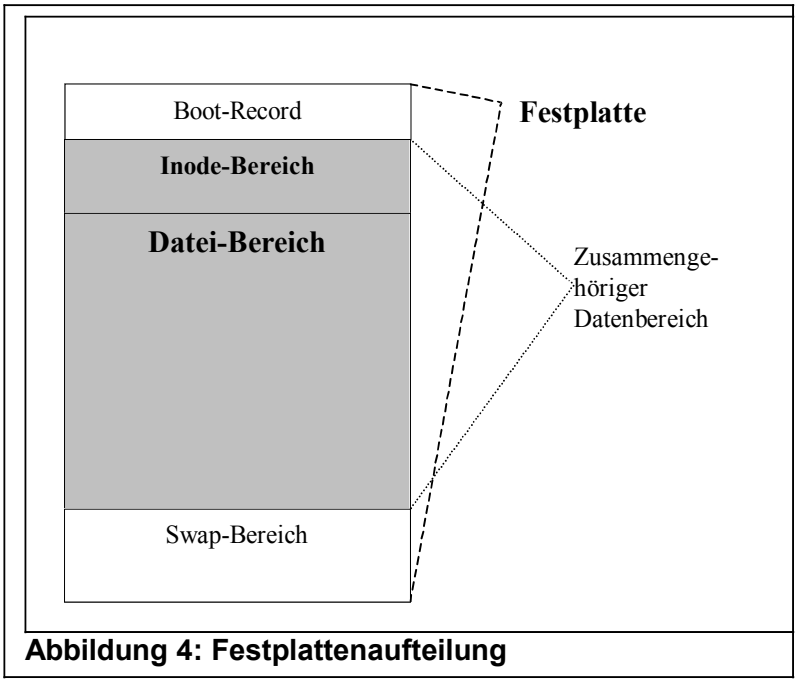

Abbildung 4: Festplattenaufteilung

In einem solchen Dateisystem müssen natürlich nicht nur die Daten selbst, sondern auch die Informationen über deren Lage auf der Festplatte gespeichert werden. Dieser Bereich, der als eine Art Index verstanden werden kann, wird als Inode-Bereich bezeichnet. Dieser wird benötigt, weil die ”normalen” Dateien in 2 Komponenten gespeichert werden. Der erste Teil bildet die Inode, die die physikalische Lage der Datei auf der Festplatte angibt. Der Dateiname ist als Eintrag in dem jeweiligen Verzeichnis zu finden, wobei jeder Eintrag lediglich aus dem Namen und der Inode-Nummer besteht. Alle weiteren Daten, wie z.B. Datum, Größe, Zugriffsberechtigungen etc. sind in der Inode verankert.

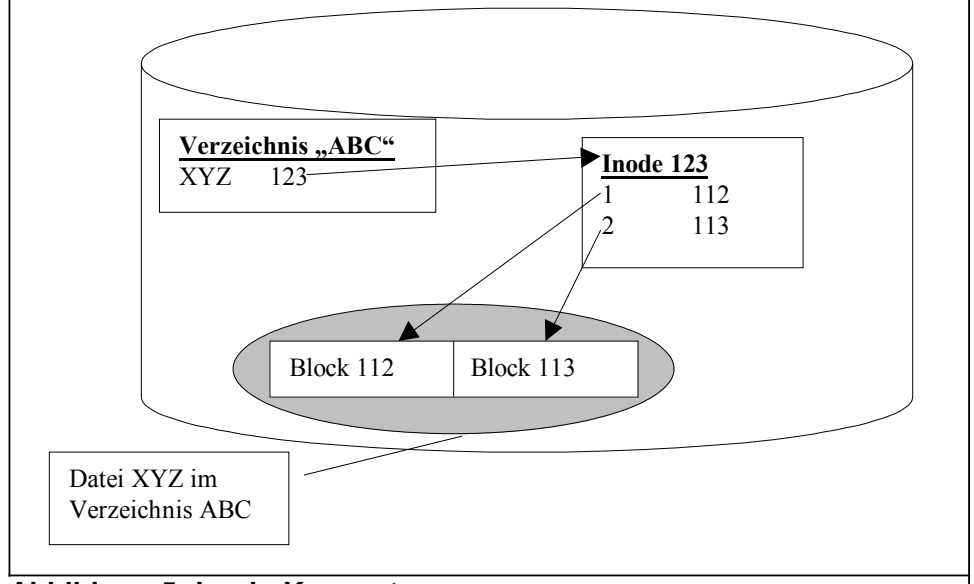

Abbildung 5: Inode-Konzept

Wie sieht nun eine solche Inode aus? Zunächst muss man sich vergegenwärtigen, wie eine Festplatte aufgebaut ist, und wie die Daten auf ihr angelegt werden. Eine Festplatte ist in 3 Dimensionen unterteilt, die als Zylinder, (Schreib-) Kopf und Sektor angegeben werden. Die kleinste Einheit, die auf diese Weise angegeben werden kann wird als Block bezeichnet. Üblicherweise handelt es sich dabei um eine Größe von 512 Byte, also 0,5 KByte. Die meisten Betriebssysteme fassen aber mehrere physikalische Blöcke zu einem logischen Block zusammen, um einen Kompromiss zwischen optimaler Ausnutzung des Speicherplatzes und der Anzahl der vorzuhaltenden Verwaltungsblöcke zu finden. Vernünftigerweise sind das 1, 2 oder 4 KByte pro Block.

Jede Datei hat somit genau eine Inode, die die Datei im Verwaltungssystem repräsentiert. In dem Directory, in dem diese Datei namentlich auftaucht, wird nur noch zusätzlich die Inode-Nummer eingetragen. Alle weiteren Informationen werden aus letzterer entnommen.

Diese Art der Dateispeicherung macht es möglich, dieselbe Datei unter verschiedenen Namen und/oder in mehreren unterschiedlichen Verzeichnissen auftauchen zu lassen. Das ist kein Fehler, sondern durchaus gewollt; es gibt dafür sogar ein Kommando mit dem man solche ”links” (genauer: ”hard links”) erzeugt: Der **”ln”**. Wird eine Datei in unterschiedlichen Verzeichnissen benötigt, braucht sie nicht kopiert zu werden, sondern kann zu den anderen Stelle ”gelinkt” werden. Das führt dazu, dass Änderungen in dieser Datei sofort an allen Stellen sichtbar werden. Wird eine solche Datei an einer der vorhandenen Links gelöscht, so bleibt sie solange bestehen, wie an anderen Stellen noch Links (also Namen) vorhanden sind. Erst wenn der letzte Link gelöscht wird, entfernt Linux auch die Daten der Datei physikalisch aus dem System.

Ein Dateisystem besteht normalerweise jedoch nicht aus einem einzigen Medium, sondern aus mehreren, die miteinander verknüpft sind. Im Gegensatz zu DOS und Windows kennt UNIX aber keine Laufwerksbuchstaben wie A: oder C:, sondern verschachtelt die einzelnen Medien als separate Verzeichisbäume hierarchisch ineinander. Den Vorgang des Zusammenfügens wird im englischen Sprachgebrauch als ”mount” (“to mount” = montieren) bezeichnet, was dem Zusammenbauen einzelner Teile zu einem Großen recht nahe kommt. Wird z.B. eine Diskette, die nur das Verzeichnis ”daten” enthält, mit dem mount-Befehl

```
mount -t msdos /dev/fd0 /mnt
```

in das Verzeichnis ”/mnt” gehängt, so ist das Verzeichnis ”daten” der Diskette ab diesem Zeitpunkt mit dem Pfad ”/mnt/daten” zu erreichen. Genauso wird mit weiteren Festplatten, CDROM-Laufwerken, sogar mit Netz-Resourcen verfahren. Man bekommt also den Eindruck, dass man immer mit einem zusammengehörigen System arbeitet. Der große Vorteil von diesem Verfahren ist dabei, dass man z.B. stark gefüllte Laufwerke einfach durch ”anhängen” weiterer Medien an der entsprechenden Stelle erweitern kann. Mit ein wenig Geschick kann man sogar die alten Pfade beibehalten. So zeigt sich dieses Konzept als wesentlich übersichtlicher und einheitlicher als die Variante der ”Laufwerksbuchstaben”, hat allerdings den Nachteil, dass die Grenzen der jeweiligen Medien nicht immer auf einen Blick sichtbar sind.

5.4 Rechtevergabe

Wie bereits angesprochen, muss sich jeder, der mit einem Linux-System arbeiten will, mit einem Namen, einem “User” identifizieren. Dieser User ist verknüpft mit einer Reihe von Rechten, die sich in den Zugriffsmöglichkeiten der Dateien niederschlägt. Prinzipiell gibt es 3 verschiedene Zugriffsebenen in einem Linux/Unix-System:

- **Lesen (r):** Ansehen des Dateiinhaltes ohne ihn zu verändern, damit ist auch das Recht zum kopieren der Datei enthalten
- **Schreiben (w):** Verändern des Inhaltes, also auch das Löschen oder Überschreiben der Datei.
- **Ausführen (x):** Sofern es sich um ein Ausführbares Programm handelt, kann dieses mit diesem Recht gestartet werden.

Jede Datei gehört wiederum genau einem *User* und einer *group*; sie kann nicht wie ein Benutzer zu mehreren Gruppen gehören. Da sich, wie im vorigen Kapitel beschrieben, die Linux-Welt in 3 Bereiche unterteilt, gibt es für jede Datei 3 Einträge: User, Group und Others. Die Rechte einer Datei kann man mit dem Befehl “**ls –l”** auflisten:

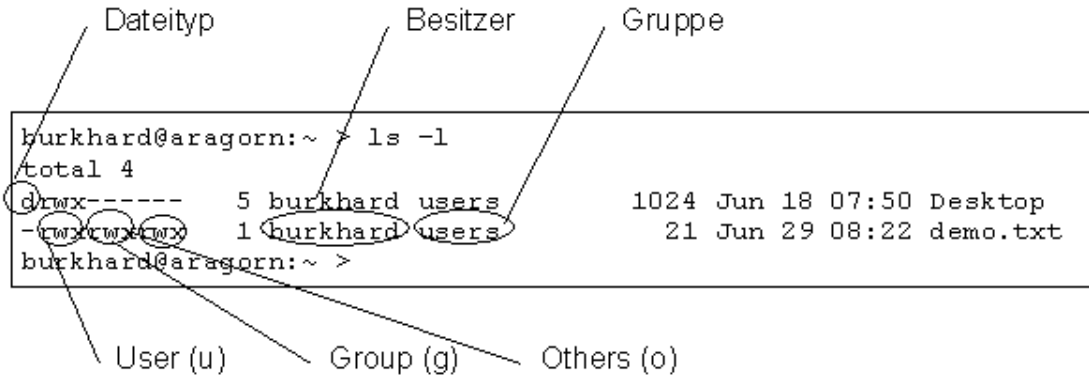

Abbildung 6: Rechtevergabe bei Dateien

Sie erscheinen dann in der linken Spalte der Auflistung und bestehen aus einer 10-stelligen Zeichenkette. Das erste Zeichen steht für den Typ der Datei. Die folgenden teilen sich in 3er-Gruppen auf: Die erste Gruppe legt die Rechte des Dateibesitzers (User, u) fest, die nächste Gruppe die der (Benutzer-) Gruppe (Group, g) und als Letztes werden die Rechte aller übrigen Benutzer angezeigt (Others, o).

Diese Rechte können natürlich auch verändert werden, allerdings nur vom root oder vom Besitzer der Datei. Das Werkzeug für diesen Zweck ist der Befehl “chmod” (change mode), dessen Parameter analog zu der obigen Beschreibung angegeben werden müssen:

```
chmod <Bereich><+|-|=><Recht> <Dateiname>
```

Der “Bereich” stellt dabei die Benutzergruppen aus Sucht des Besitzers dar, also “u”,”g” und “o” oder eine beliebige Kombination von diesen. “Recht” wird als beliebige Kombination der verfügbaren Rechte gesehen, also “r”,”w” und “x”. mit den Zuweisungszeichen können Rechte hinzugefügt (+), entfernt (-) oder definiert (=) werden. Um also die Datei “demo.txt” so zu sichern, dass Allen außer dem Besitzer nur Lesezugriff gewährt wird, so muss der Befehl so lauten:

```
chmod go=r demo.txt
```

6 Das Prozesssystem

Nun besteht ein Betriebssystem nicht allein aus den Dateien, sondern auch aus den laufenden Programmen, die mit den erwähnten Daten arbeiten. Ein laufendes Programm, das in den Hauptspeicher geladen wurde und die Zeit des Prozessors beansprucht, wird in diesem Zusammenhang als ”Prozess” bezeichnet. Die Gesamtheit dieser Prozesse wird als Prozesssystem bezeichnet, und stellt somit das eigentliche ”lebendige” System dar, das auf externe Einflüsse reagieren kann.

Ein laufendes Linux-System besteht grundsätzlich aus einer Vielzahl aktiver Prozesse, die teilweise aufeinander angewiesen sind, da die meisten Prozesse Dienstleistungen für andere Prozesse zur Verfügung stellen. Dieses lässt sich gut an dem Beispiel des Logins, also des Anmeldevorganges verdeutlichen.

> Angenommen, Herr Schulz möchte sich an seinem Rechner einloggen, um seine Arbeit zu beginnen. Das Auf dem Bildschirm dargestellte "login:" ist bereits eine Datenausgabe eines Prozesses, der "getty" genannt wird. Herr Schulz benutzt nun die Tastatur, um seinen Benutzernamen einzugeben. Im Rechner signalisiert die Hardware, dass an dem Gerät "Tastatur" Daten eingegeben werden. Der Kernelprozess nimmt diese Daten entgegen, und sieht in einer Liste nach, ob ein bestimmter Prozess an das entsprechende Gerät gekoppelt ist. Er findet den Prozess "getty" und übermittelt ihm die Daten in einer Form, die dieser verstehen kann.
>
> Der "getty" ist nun dieser Aufgabe nicht gewachsen und startet seinerseits einen Prozess, der "login" genannt wird. Dieser nimmt die Stelle des getty ein und teilt seinerseits dem Kernel mit, dass die Zeichenkette "password:" auf der Standardausgabe ausgegeben werden soll. Der Kernel stellt fest, dass damit der Bildschirm - genauer die Grafikkarte - gemeint ist und kopiert diese Daten an die entsprechende Stelle.
>
> Nachdem nach dem gleichen Spiel das Passwort eingegeben worden ist, nimmt der Prozess "login" Zugriff auf die Passwortdatei. Dieses wiederum endet in einem Zugriff auf die Hardware - der Festplatte - und kann deshalb nicht von ihm selbst ausgeführt werden. Daher wird wieder dem Kernel das entsprechende Kommando gegeben, der seinerseits die geforderten Daten aus der Datei herausholt und sie dem "login" überreicht. Passt das eingegebene Passwort zu dem Benutzernamen, so startet der "login" das Programm, das er aus dem Inhalt der erhaltenen Daten ermittelt. (meistens handelt es sich dabei um eine sog. Shell, dem Kommandozeileninterpreter.) Dieses Programm ersetzt nun den login, so wie dieser vorher den getty ersetzt hat und arbeitet für ihn weiter.

6.1 Struktur der Prozesse

Aus dem obigen Beispiel ist ersichtlich, dass in einem Unix-System eine streng hierarchische Prozessstruktur herrscht. An oberster Stelle steht der Kernelprozess, der sämtliche Hardware-Zugriffe durchführt und ständig laufen muss. Das Stoppen des Kernels ist somit dem Stop des gesamten Systems gleichzusetzen.

Das nächstniedrigere Prozess ist der "init", der als "Vater aller Prozesse" bezeichnet wird. Tatsächlich ist es so, dass grundsätzlich jeder Prozess einen "Eltern"-Prozess (eigentlich: parent process) besitzt, weil unter Unix Prozesse ausschließlich durch Duplizierung und Ersetzung von laufenden Prozessen entstehen. Wird z.B. von der laufenden Kommandozeile, der Shell, die durch den Prozess ”bash” dargestellt wird, ein Editor ”vi” gestartet, so wird zunächst eine Kopie des Prozesses ”bash” im Hauptspeicher des Rechners angelegt. Im zweiten Schritt wird dann der Programmcode der

”bash” durch den des ”vi” überschrieben und ausgeführt. So wird eine relativ einfache und stabile Form der Prozessstruktur erzeugt.

Der Init ist es auch, der beim Hochfahren des Systems steuert, welcher Prozess in welcher Stufe aufgerufen wird. Er kann auch nur ein einziges Mal im System erscheinen, weil jeder folgende Aufruf nur eine Statusänderung des bereits laufenden Prozesses zur Folge hat. Jeder weitere Prozess wird unmittelbar oder mittelbar von dem Init erzeugt, wobei die Höhe der Verschachtelung nur von den Systemressourcen vorgegeben wird.

Durch diese Spezialisierung der Programme ist ein Unix-System grundsätzlich so konzipiert, dass möglichst jedes Programm mehrfach verwendet werden kann und somit die Ausgabe des einen zur Eingabe des nächsten Programmes werden kann.

Da Unix ein "Multitaskingsystem" ist, werden alle Prozesse (quasi) gleichzeitig ausgeführt, was natürlich mit einem einzigen Prozessor technisch nicht möglich ist. Als Kompromiss werden die laufenden Programme im Millisekundenbereich nacheinander bedient, so dass der Benutzer den Eindruck gewinnen kann, dass wirklich alle Programme gleichzeitig laufen. Lediglich die Geschwindigkeit nimmt bei zunehmender Prozessanzahl ab. Stehen dem System allerdings mehrere Prozessoren zur Verfügung, so wird die Last auf alle mehr oder weniger gleich verteilt. Man spricht dann vom "Multiprocessing".

Diese Gleichzeitigkeit der Prozesse hat immense Vorteile, wenn zwei Prozesse aufeinander angewiesen sind. Wenn z.B. der Inhalt einer Textdatei ausgegeben werden soll, so kann das mit dem Programm ”cat” geschehen, das nach klassischer ”EVA”-(Eingabe, Verarbeitung, Ausgabe) Struktur den Inhalt dieser Datei auf den Bildschirm ausgibt

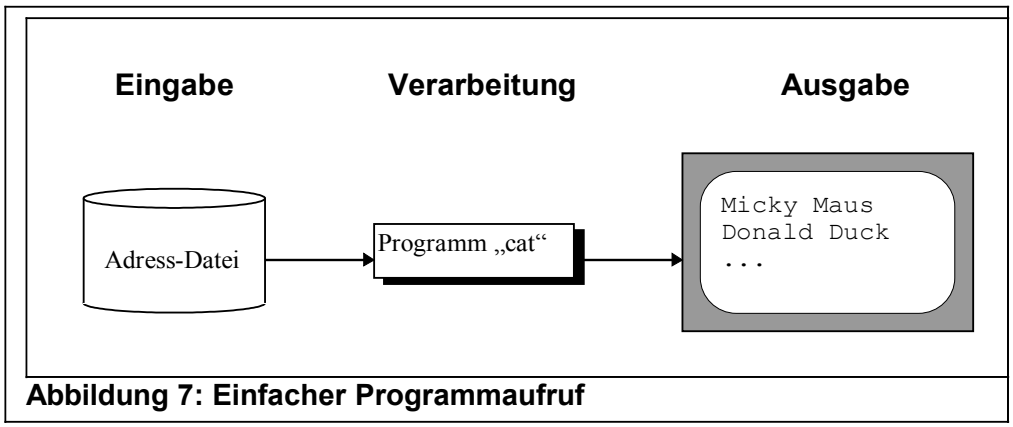

Abbildung 7: Einfacher Programmaufruf

Soll jetzt aber die Ausgabe noch sortiert werden, so müsste erst noch ein Zwischenspeicher angelegt werden. Da die Prozesse aber (quasi) gleichzeitig laufen, kann das Programm ”sort“ einfach zwischengeschaltet werden, so dass keine weitere Speicherung der Daten erforderlich wird.

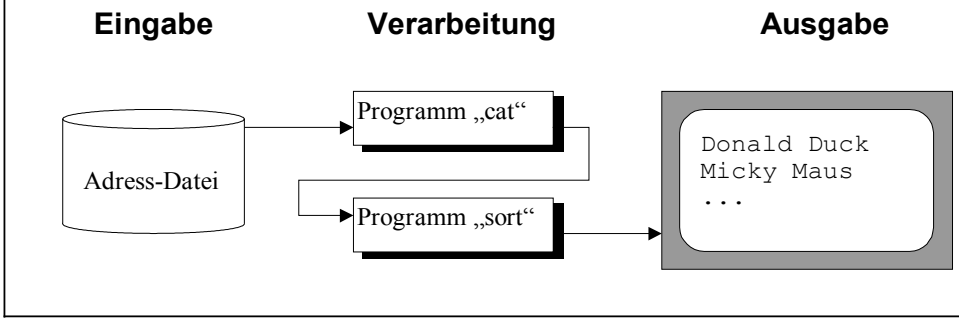

Abbildung 8: Kopplung zweier Prozesse durch eine "Pipe"

Aus Benutzersicht können diese Prozesse grob in 2 Kategorien unterteilt werden:

•**Vordergrundprozesse** benutzen Tastatur und Bildschirm, um mit dem Benutzer zu kommunizieren. Hierzu gehören alle Applikationen, die mit Hilfe von Bildschirmmasken bzw. Fenstern interaktiv bedient werden, wie z.B. Textverarbeitungen, Datenbankapplikationen, Administrationstools etc.

•**Hintergrundprozesse** laufen ohne manuellen Eingriff, so dass sie - einmal gestartet - laufen, bis sie ihre Aufgabe beendet haben oder aber von "außen" gestoppt werden. Hierzu gehören sämtliche Serverprozesse wie Datenbanken, Druckspooler, Netzwerksoftware, aber auch die Betriebssystem-Shell, sofern sie mit einem Skript gestartet wird.

Beide Formen von Prozessen bilden zusammen das gesamte System, das sich mit all seinen Eigenschaften dem Benutzer präsentiert. Die Übergänge zwischen Vorder- und Hintergrundprozessen können fließend sein, weil viele Programme wahlweise als der eine oder der andere Typ gestartet werden können.

Einem Überblick über die gestarteten Programme bzw. Prozesse erhält man mit dem Befehl ”ps” (process status), der jeden Prozess mit einer Nummer versehen auflistet. Diese Nummern werden von dem Betriebssystem vergeben und sind jeweils eindeutig. Das bedeutet, dass keine zwei Prozesse dieselbe Prozessnummer bekommen, wobei allerdings die Nummer eines beendeten Prozesses wiederverwendet werden kann.

6.2 Signale

Da es für die Hintergrundprozesse schwierig ist, mit dem Rest des Systems zu kommunizieren, wurden die Signale eingeführt. Ein Signal ist nichts anderes als ein numerischer Wert zwischen 0 und 15, der vom Kernel an einen laufenden Prozess gesendet wird. Der angesprochene Prozess entscheidet dann selbst, auf welche Weise er auf diesen Wert reagiert. Um in die Verwendung dieser Prozesse eine möglichst hohe Einheitlichkeit zu bekommen, sind den Signalen bestimmte Bedeutungen zugeschrieben worden. So steht z.B. Das Signal 1 für "hangup", was seine historische Bedeutung in der Verbindung über ein Modem hat und einem laufenden Prozess verdeutlicht, dass sein Kommunikationspartner "aufgehängt", also die Verbindung gekappt hat. In den meisten Fällen sollte sich der Prozess nach Sicherung der aktuellen Daten beenden. Alternativ kann er in einer "Warteposition" verharren, um bei einer erneuten Verbindung die Arbeit wiederaufzunehmen. Das Signal 9 stellt allerdings eine Ausnahme dar: Dieses Signal wird nicht vom angesprochenen Prozess, sondern vom Kernel verarbeitet, der dann dafür sorgt, dass der Prozess **sofort** beendet wird. Eventuell nicht gesicherte Daten gehen also dabei verloren, so dass dieser Befehl nur in Extremfällen benutzt werden sollte.

Ein solches Signal kann entweder mit ”Programmtechnischen Mitteln” versendet werden, oder aber mit einem Programm mit dem viel sagenden Namen ”kill”. Dieses erwartet als Parameter die Nummer des Prozesses (z.B. mit ”ps” ermittelt), dem das Signal gegeben werden soll.

Wurde z.B. versehendlich ein Hintergrundprozess gestartet (z.B. ein Compiler), der sehr viel Ressourcen benötigt und sofort gestoppt werden soll, so muss zunächst die Prozessnummer (PID) mittels

```
$ ps ax| grep meinprozess
```

ermittelt werden. (in diesem Beispiel sei der Name des aufgerufenen Programmes „meinprozess“, während die zugehörige PID „234“ lautet. Nun kann dieser Prozess mit dem Befehl

```
$ kill 234
```

beendet werden. Sollte das jedoch fehlschlagen, kann das Ende erzwungen werden, indem der ”kill” mittels Parameter angewiesen wird, das Signal 9 zu senden:

```
$ kill -9 234
```

Das sollte aber nur die letzte Möglichkeit bleiben, da bei diesem Signal dem ”gekillten” Programm keine Chance gelassen wird, ungesicherte Daten zu speichern.

7 Start und Stop des Linux-Systems

Der Start des Linux-Systems ist relativ einfach. Man drückt auf einen Knopf und wartet, bis man sich einloggen kann. Eine so oberflächliche Beschreibung führt natürlich zu dem Trugschluss, dass sich ein solches System wie eine Kaffeemaschine ein- und ausschalten lässt. Wird dazu das beschriebene Dateien- und Prozesssystem betrachtet, so wird deutlich, dass ein solches Verhalten zu irreparablen Fehlern führen kann.

7.1 Herauffahren (”bootstrap”)

Wird der Rechner eingeschaltet, so wird nach dem Selbsttest der Hardware der erste verfügbare Datenblock der Festplatte geladen, in dem sich dann auch ein ”loader” befinden sollte. Sonst bleibt das System an diesem Punkt stehen. Dieser Loader enthält seinerseits die Informationen über die weitere Vorgehensweise. Er sorgt dafür, dass im weiteren Verlauf der entsprechende Betriebssystemkern in den Hauptspeicher geladen wird und startet ihn auch anschließend.

7.1.1 Der Lader

Diese Aufgabe wird unter Linux von dem ”LILO”, dem ”Linux Loader”, erfüllt. Er ist auch gleichzeitig ein Boot-Manager, der es ermöglicht, mehre Betriebssysteme parallel auf einem Rechner zum Start anzubieten. LILO kann also nicht nur Linux, sondern auch DOS, OS/2, Windows 95/98 oder auch – mit einigen Umständen – Windows NT starten. Die Benutzerführung ist dabei zwar recht dürftig, aber ausreichend.

Erst nachdem der Kern geladen wurde, ist Linux selbst aktiv. Der Kern überprüft zunächst die eigene Systemumgebung, d.h. die verfügbaren und angeschlossenen Geräte. Das wichtigste ist dabei das sog. Root-Filesystem, das an der Stelle ”/” in das System eingebaut ”gemounted” wird. In dieses Filesystem werden mittelbar oder unmittelbar alle weiteren Filesysteme eingehängt. Ist Root vorhanden, wird als Nächstes die Datei /etc/inittab gelesen und interpretiert. Anhand eines vorgegeben ”Boot-Levels” startet das System im weiteren Verlauf mehr oder weniger Prozesse, so dass je nach Umfang des laufenden Systems ein bestimmtes Niveau erreicht wird. Dieser Bootlevel wird entweder interaktiv zu Beginn des Bootvorganges festgelegt oder aber der Datei */etc/inittab* durch den Parameter ”initdefault” entnommen.

7.1.2 Die Start-Skripten

Bis zu diesem Punkt sind so ziemlich alle Unix-Systeme gleich. Die älteren arbeiten nun mit einer zentralen Startup-Datei weiter, der ”/etc/rc”. Einige Linux-Distributionen und alle anderen ”SYSTEM V”-konformen Systeme verzweigen in das Verzeichnis ”/sbin/rc.d ” (bzw. “/etc/ init.d” bei älteren SuSE-Distributionen) und führen die dort vorhandenen Skripten aus, die für den angestrebten Boot-Level benötigt werden.

Üblicherweise startet jeder nachfolgende Level mehr Dienste als sein Vorgänger. Was jedoch in welchen Levels gestartet wird und was nicht, bleibt Interpretationssache der jeweiligen “Designer” des Systems. Die einzigen Level, die bei allen Systemen dieselbe Bedeutung besitzen sind **Level 0, Level S** und **Level 6**. Die folgende Tabelle stellt die Bedeutung der Level am Beispiel der Distributionen von **Red Hat** sowie **S.u.S.E. ab 7.1** dar.

Level	Betriebssystemzustand
0	Das System ist heruntergefahren, es laufen keinerlei Prozesse
S oder 1	Es werden nur die allernotwendigsten Prozesse gestartet, z.B. um tief greifende Veränderungen im System vorzunehmen. Die Bedienung

Level	Betriebssystemzustand
	ist nur von der Konsole aus erlaubt. (**S**ingle User Mode)
2	Das (Basis-)System ist zwar vollständig gestartet**,** jedoch **ohne Netzwerkunterstützung**.
3	Das System ist **mit Netzwerkunterstützung** gestartet. Dieser Level ist sinnvoll für Serversysteme, die keine grafische Oberfläche benötigen
4	- frei -
5	Es wird zusätzlich zu dem vollständigen System der (korrekt konfigurierte) X-Displaymanager und somit die grafische Benutzeroberfläche gestartet. **Vorsicht!** Ist der X-Server nicht korrekt konfiguriert, gibt es an dieser Stelle Probleme. Im schlimmsten Fall kann das System nicht mehr bedient werden!. Dieser Level ist für Desktop und CAD-Systeme geeignet.
6	Das System wird neu gestartet (Reboot). Dieser Level wird auch durch die Tastenkombination [Ctrl]-[Alt]-[Del] initiiert.

Jedes Skript ist dabei für eine bestimmte Software bzw. einen bestimmten Dienst zuständig.

7.2 Herunterfahren (”shut down”)

Wenn das System gestoppt werden soll, müssen zunächst alle laufenden Prozesse gestoppt werden, die sich im System befinden. Denn viele Prozesse enthalten für das System ”lebenswichtige” Daten, die bei einem unkontrolliertem Stromausfall verloren gehen. Da sich das System den momentan nicht benötigten Hauptspeicher als ”Cache” (Zwischenpuffer) verwendet, können viele Daten, die zwar logisch schon auf die Festplatte geschrieben wurden, nicht tatsächlich dort landen. Daher darf ein solches System nicht einfach ausgeschaltet, sondern muss ”heruntergefahren“ werden. Mit diesem ”shut down” des Systems werden alle vorhandenen Prozesse mittels mehrerer vorgegebenen Skripten einzeln beendet. Sind anschließend noch Prozesse aktiv, so werden diese zum Abbruch gezwungen (”kill –9 ...”), dann erst kann das System abgeschaltet werden. Größere Rechner wie Abteilungsrechner und Mainframes werden bei dieser Aktion auch von der Software abgeschaltet, aber auch jüngere PCs und vor allem Notebooks sind zu diesem ”Power down” auch befähigt.

Initiiert wird ein Shutdown ausschließlich vom Superuser, dem „root“, unter dem das Kommando **„shutdown“** gestartet werden muss. Erst wenn das System meldet, dass es angehalten worden ist, darf es ausgeschaltet werden, anderenfalls können nicht vorhersagbare Datenverluste auftreten.

8 Die Schnittstelle zum Benutzer

In den vorangegangenen Kapiteln war wiederholt die Rede von Prozessen, die gestartet werden sollten, nur wie dieses geschieht, blieb bisher unbetrachtet.

8.1 An- und Abmelden am System

Das gestartete Linux zeigt sich mit dem spartanischen Hinweis, indem der Schriftzug “login:” auf dem Bildschirm erscheint. An dieser Stelle kann ein gültiger Benutzername eingegeben und mit der [RETURN]-Taste abgeschlossen werden. Anschließend erscheint “password:” woraufhin das zugehörige Passwort eingegeben werden muss, das nicht auf dem Bildschirm erscheint. Es werden dabei keinerlei Platzhalter wie “*” angezeigt. Bei einem Tippfehler, erscheint natürlich kein Hinweis darauf, welches der beiden Wörter falsch ist, da sonst ein “Einbruch” in das System erheblich erleichtert würde. Sind beide Wörter korrekt, startet eine “Session”, in der entweder weitere Befehle eingeben oder ein voreingestelltes Programm benutzt werden kann. Prinzipiell können beliebig viele Programme sogar ineinander verschachtelt gestartet starten werden, solange nicht das zuerst – durch das Login – gestartete Programm gestoppt wird.

Das Abmelden erfolgt daher wesentlich einfacher als das Anmelden. Sobald das zuerst gestartete Programm beendet wird, beendet sich auch die Session, und es erscheint wieder der Anmeldebildschirm. Das es sich bei diesem Programm meistens um eine „Shell“ handelt, erfolgt die Beendigung der Session mit der Tastenkombination „[Ctrl]-D“.

8.2 Die Shell

Nun, es wurde bereits bei der Beschreibung des Prozesssystems kurz angesprochen, dass nach dem ”Login”, dem Anmelden eines Benutzers, eine sog. „Session“, und damit eine ”Shell” gestartet wird, um die Befehle des Benutzers entgegennehmen zu können. Diese Shell ist natürlich auch nur ein Programm wie alle anderen auch, nur dass sie von dem Benutzer eine Eingabe erwartet, bevor sie irgend etwas bewirkt. Natürlich ist das nicht *ganz* korrekt, denn schließlich initialisiert sie ja auch ihre eigene Umgebung, wenn sie startet. Aus Benutzersicht ist diese Shell ein recht unbequemes Programm, da für alle Aktionen der entsprechende Befehl eingetippt werden muss. Sie arbeitet also weder mit Menü- noch mit Mausunterstützung, geschweige denn mit einer grafischen Oberfläche, sondern erscheint durch ein ”Prompt”, das in den meisten Shells (Bourne-Shell, Korn-Shell, Bourne-again-Shell) als ”$” dargestellt wird. Unter Linux wird meistens die Bourne-again-Shell (bash) eingesetzt, deren Prompt DOS-like den aktuellen Pfad angibt. Der Eingangsbildschirm sieht also üblicherweise so aus:

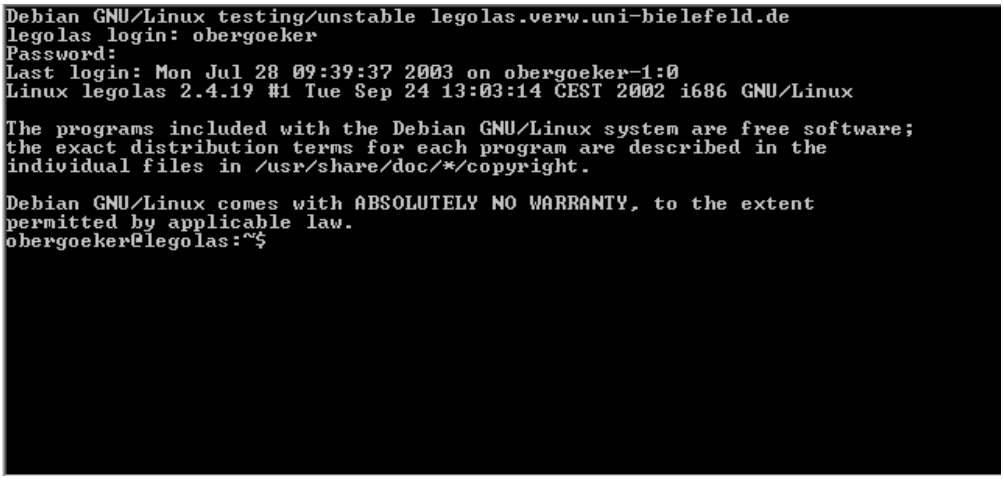

Abbildung 9: Anmeldung im Textmodus

Wenn Sie das Prompt „$“ sehen, benutzen Sie vermutlich die älteste aller Shells, die ”Bourne-Shell” (benannt nach ihrem Entwickler). Nun gibt es aber eine Vielzahl jüngerer Shells, die alle mehr oder weniger zueinander kompatibel sind. So gibt es z.B. außer der Bourne-Shell (sh) die C-Shell (csh), die Korn-Shell (ksh) und die unter Linux am häufigsten verwendete ”Bourne-again-Shell” (bash). Letztere zeichnet sich dadurch aus, dass sie eine etwas bessere Prompt-Darstellung (es wird immer der aktuelle Pfad angezeigt) und eine gute History-Funktion aus (Durch Cursortasten können Befehle zurückgeholt und editiert werden). Außerdem ist sie **fast** 100% zu der Bourne-Shell kompatibel und bietet darüber hinaus eine gute Erweiterung des Befehlsumfanges. DOS-gewöhnte Leute mögen an dieser Stelle aufatmen, denn die Bedienung ist der DOS-Kommandozeile nicht unähnlich, nur die Befehle lauten anders.

Die Shell läuft solange, bis der (interne) Befehl **„exit“** eingegeben oder die Tastenkombination [Ctrl]-[D] gedrückt wird. Wenn die Shell beendet wird, wird auch damit die Session beendet, und das System begibt sich wieder in den Zustand vor dem Start der Shell. Das System selbst bleibt aber in dem aktiven Zustand. Soll weitergearbeitet werden, muss der Anmeldevorgang wiederholt werden.

8.3 Befehle

Das Prompt zeigt an, dass jetzt Befehle abgesetzt werden können, indem sie einfach eingetippt und mit der Return-Taste abgeschlossen werden. Soll beispielsweise der Inhalt des aktuellen Verzeichnisses angezeigt werden, so kann dieses mit dem Befehl ”ls” gelöst werden. Wie bereits erklärt, wird dazu einfach die Zeichenfolge ”ls” eingetippt und mit der Return-Taste abgeschlossen. Das Ergebnis sieht dann ungefähr so aus:

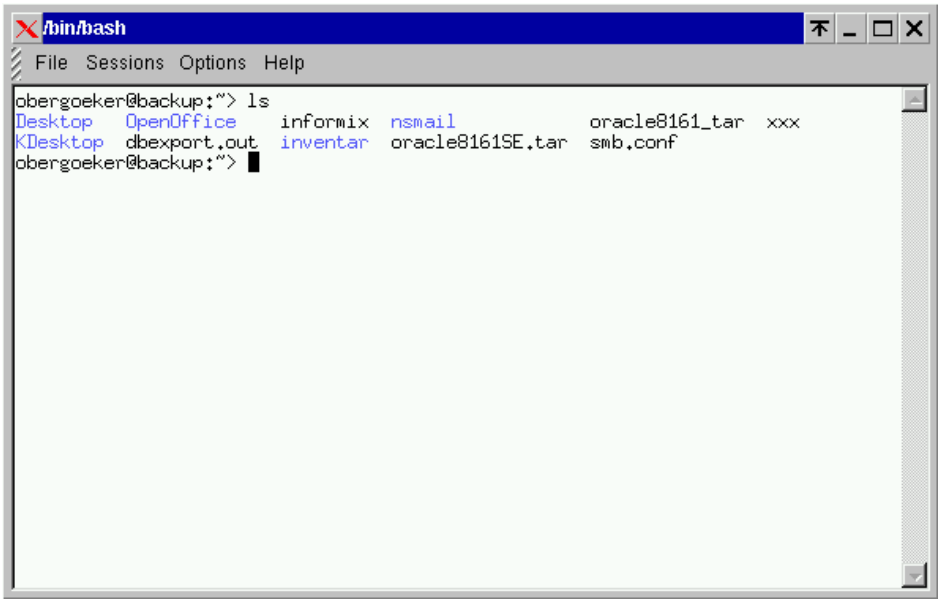

Abbildung 10: Typische Ausgabe des "ls"

Der Befehl ls startet sofort, schreibt sein Ergebnis auf den Bildschirm und gibt anschließend die Kontrolle an die aufrufende Shell zurück, was an dem nachfolgend wieder erscheinenden Prompt erkennbar ist.

Befehle und Programme können in den meisten Fällen synonym betrachtet werden, da die meisten Befehle unter UNIX/Linux als eigene Programme implementiert sind, und nicht als Funktion innerhalb der Shell. Unter DOS sind z.B. viele Befehle direkt in das Äquivalent der UNIX-Shell, dem ”COMMAND.COM”, eingebaut, um die Zeit zu sparen, die für das Laden externer Programme benötigt wird. Unter UNIX ist das kritische Element eher der Hauptspeicher als die Festplattengeschwindigkeit, so dass nach dem Prinzip ”small is beautiful” nur das in den Hauptspeicher geladen wird, das auch wirklich benötigt wird. Die ”eingebauten” Befehle werden als ”interne Befehle”, die separaten Programme als ”externe Befehle” bezeichnet, die alle in den Verzeichnisses /bin und /usr/bin zu finden sind.

Zu den internen Befehlen gehören:

- **cd** Wechseln des Arbeitsverzeichnisses
- **set** setzen von Umgebungsvariablen

Zu den externen Befehlen zählen:

- **ls** Anzeige eines Verzeichnisses
- **cp** Dateien kopieren

Natürlich können auch andere, z.B. selbst geschriebene Programme auf diese Art und Weise gestartet werden. Außerdem kann die Shell selbst auch für kleinere Programme verwendet werden, indem eine Folge von Befehlen in eine Textdatei geschrieben wird. Solche Programme bleiben aber – im Gegensatz zu den Compilersprachen – als Text erhalten und heißen daher ”shell skripts”. Diese Skripten können genauso wie andere Programme direkt als Befehl aufgerufen werden.

8.4 Eingabe und Ausgabe

Eigentlich sollte es nach alter EDV-Terminologie *Eingabe-Verarbeitung-Ausgabe* oder kurz: Das *EVA-Prinzip* heißen, was die prinzipielle Arbeitsweise von Programmen beschreibt. Obwohl dieses Prinzip im Zeitalter der grafischen Oberflächen nicht ganz

dem Stand der Technik entspricht, hat es dennoch seine Berechtigung und ist auch an vielen Stellen wieder zu finden.

Jedes Programm erwartet eine Eingabe und erzeugt eine Ausgabe. Im Zeitalter der Riesenrechner bestand die Eingabe aus einem mehr oder weniger hohen Stapel Lochkarten und die Ausgabe aus einem ebenso hohen Stapel Endlospapier. In unserem Fall ist – am Beispiel der Shell – die Eingabe die Tastatur und die Ausgabe der Bildschirm. Nun kann es aber möglich sein, dass die Ausgabe nicht auf dem Bildschirm erscheint, sondern in einer Datei gespeichert werden soll. Die Umständliche Methode wäre, den Inhalt des Bildschirmes abzutippen und mit einem Editor in einer Datei abzuspeichern.

Da nun jedes Programm sowohl für die Ein- als auch für die Ausgabe sog. ”Kanäle” benutzt, können diese im wahrsten Sinne des Wortes umgeleitet werden. Für diese Zwecke hängt man an den gewünschten Befehl einfach das Zeichen für Eingabe ”<” oder für Ausgabe ”>” gefolgt von einer Datei, die die benötigten Daten enthält bzw. anschließend speichern soll.

> **Beispiel**: Der Inhalt des aktuellen Verzeichnisses soll nicht auf dem Bildschirm abgebildet werden, sondern in der Datei ”inhalt”. Dazu muss nach dem ”ls” das Zeichen ”>” folgen und der Dateiname ”inhalt”:
>
> ```
> $ ls >inhalt
> ```
>
> Soll nun der Inhalt der Datei ”inhalt” zuerst sortiert und dann erst ausgegeben werden, so folgt nach dem Programmaufruf ”sort” das Zeichen ”<” gefolgt von dem Dateinamen ”inhalt”:
>
> ```
> $ sort <inhalt
> ```

Es geht allerdings noch einen Schritt weiter. Da es ja recht umständlich ist, wie im vorigen Beispiel die Daten erst in einer Datei zu speichern, nur um sie anschließend an das nächste weiterzureichen, können diese Programme auch direkt gekoppelt werden. Zu diesem Zweck werden die Programme direkt hintereinander geschrieben, nur getrennt von dem ”Pipe”-Zeichen, dem senkrechten Strich ”|”. Die Daten ”fließen” dann quasi von dem ersten Programm zu dem zweiten. Das obige Beispiel könnte dann in verkürzter Weise so aussehen:

```
$ ls | sort
```

Auf diese Weise können beliebig viele Befehle verkettet werden, um mit mehreren kleinen Einzelschritten das Ziel zu erreichen.

8.5 Dämonen

Keine Angst, jetzt soll kein Abstecher in den ”Exorzisten” folgen, sondern nur eine bestimmte Sorte von Programmen beschrieben werden. Die bisher beschriebenen Befehle und Programme benötigen alle die Tastatur und den Bildschirm, um ihre Arbeit zu erledigen. Das betrifft sowohl den ”ls”, der ja seine Inhaltsangabe auf dem Bildschirm ausgibt, wie auch die Shell selbst, die ihre Befehle von der Tastatur erwartet. Nun gibt es aber eine ganze Reihe von Programmen, die losgelöst von jeglicher Benutzerkommunikation im Hintergrund des Systems laufen, und nur dann aktiv werden, wenn sie benötigt werden (oder wenn sie “meinen”, benötigt zu werden). Diese Eigenschaft der (scheinbar) unberechenbaren Aktivität wurde von den Entwicklern des Unix-Systems mit Dämonen (engl.: daemons) verglichen und so wurde dieser Begriff geprägt. Einige dieser Dämonen werden ständig benötigt wie z.B.:

- lpd der Druckprozess (**l**ine **p**rinter **d**eamon)
- inetd der Internet-Serverprozess (**i**nter**net** **d**aemon)

Diese Prozesse sind zwar selten direkt in Aktion zu sehen, doch werden ihre Aufgaben besonders dann erkennenbar, wenn sie einmal ausfallen.

Nun können aber auch manuell Programme gestartet werden, die losgelöst von dem eigenen Terminal laufen sollen. Das ist besonders dann recht nützlich, wenn nicht darauf gewartet werden kann oder soll, bis sich das betreffende Programm beendet hat. Soll also ein Programm “prog1” so gestartet werden, dass dem Benutzer unmittelbar nach dem Absetzen des Befehls (und unabhängig von der Laufzeit von prog01) die Kommandozeile wieder zur Verfügung steht, so muss das Zeichen “&” an die Befehlszeile angefügt werden:

```
burkhard@legolas:~/programs > prog01 &
[1] 2702
burkhard@legolas:~/programs >
```

Diese Ergänzung veranlasst die Shell, das Programm zu starten und nicht auf dessen Ende zu warten. Als Information erscheinen in der folgenden Zeile zwei Zahlen: Die erste Zahl ist eine fortlaufenden Nummer der Hintergrundprozesse, die in dieser Shell gestartet wurden. Die Zweite ist eine Systemweit eindeutige Nummer des gestarteten Prozesses, die PID. Statt nun zu warten, bis das Hintergrundprogramm von sich aus terminiert, kann der Abbruch durch den Befehl “kill”, wie er im Abschnitt “Signale” beschrieben wird erzwungen werden.

9 Editoren

Es nützt natürlich alles Wissen über Textdateien nichts, wenn das über die notwendigen Mitteln, mit denen diese erstellt bzw. verändert werden können, fehlt. Zu diesem Zweck gibt es ”Editoren”, die ähnlich wie Textverarbeitungsprogramme arbeiten, jedoch Texte in ”Reinkultur” erzeugen. Textverarbeitungssysteme speichern Informationen über den eigentlichen Text als besondere Zeichen oder Zeichenfolgen zwischen den einzelnen Buchstaben, wie z.B. Fettschrift, Blocksatz, Schriftart etc.. Wenn Programme Konfigurationen aus solchen Dateien lesen würden, wären diese Informationen nicht nur Überflüssig, sie würden auch den Inhalt dieser Datei verfälschen. Auch Programme, die mit einem Compiler übersetzt werden sollen, dürfen keinerlei Daten außer den eigentlichen Texten enthalten.

Für diesen Zweck wurden Editoren entwickelt, mit denen man mehr oder weniger komfortabel solche Texte schreiben kann. Üblicherweise gilt die Regel: Je komfortabler ein Editor ist, umso geringer ist eine Flexibilität.

Unter Linux finden sich eine ganze Reihe von Editoren, z.B. den vi, emacs und pico sowie im grafischen Bereich axe, xvi.und xedit.

9.1 Der “vi”

Das gebräuchlichste Werkzeug im Unix-Bereich ist der Editor “vi” (**vi**sual), der eigentlich nur eine Erscheinungsform des “ex” (**ex**tended editor) ist. Der vi ist nicht nur in allen UNIX-Derivaten vorhanden, er kommt auch mit jedem Terminal aus, d.h. er ist von überall aus zu bedienen. Diese Eigenschaft des “kleinsten gemeinsamen Nenners” hat ihn zu seiner Beliebtheit gebracht. Allerdings ist die Bedienung des vi – gelinde ausgedrückt – gewöhnungsbedürftig. Wer also an die üblichen Textverarbeitungssysteme gewöhnt ist, wird mit dem vi um Anfangsschwierigkeiten nicht herumkommen. Sind diese anfänglichen Schwierigkeiten aber erst einmal überwunden, steht ein mächtiges Werkzeug zur Verfügung, mit dem auch in Ausnahmefällen das System konfigurieren werden kann, z.B. wenn es notwendig wird, das Linux-System mittels des Rettungssystem zu starten.

Die Arbeitsweise des vi gliedert sich in 3 Stati: den **Kommandomodus**, den **Eingabemodus** und den **ex-Modus**. Letzterer stellt eine Art “Kommandozeile” dar, in der vollständig ausgeschriebene Kommandos eingegeben werden müssen.

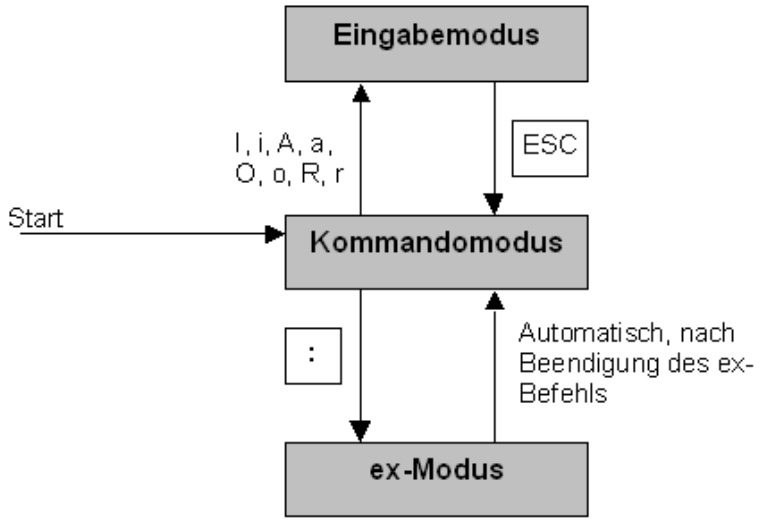

Abbildung 11: Arbeitsweise des "vi"

Im Gegensatz zu den meisten anderen Editoren, kann im vi nicht unmittelbar nach dem Start “drauflos getippt” werden. Vielmehr befindet er sich in einem ***Kommandomodus***, in dem jede Taste einen bestimmten Befehl bedeutet. So kann z.B. mit der Taste <i> in den Einfügemodus gewechselt werden, wobei das “i” natürlich nicht erscheint. Erst die nächste Taste wird als Zeichen in den Editor aufgenommen, sowie jede weitere Taste, bis die <ESC>-Taste gedrückt wird. In diesem Kommandomodus stehen zumeist Befehle zur Verfügung, die sich direkt auf den geladenen Text beziehen, wie z.B. Cursorposition verändern, Zeilen Löschen/Kopieren etc.

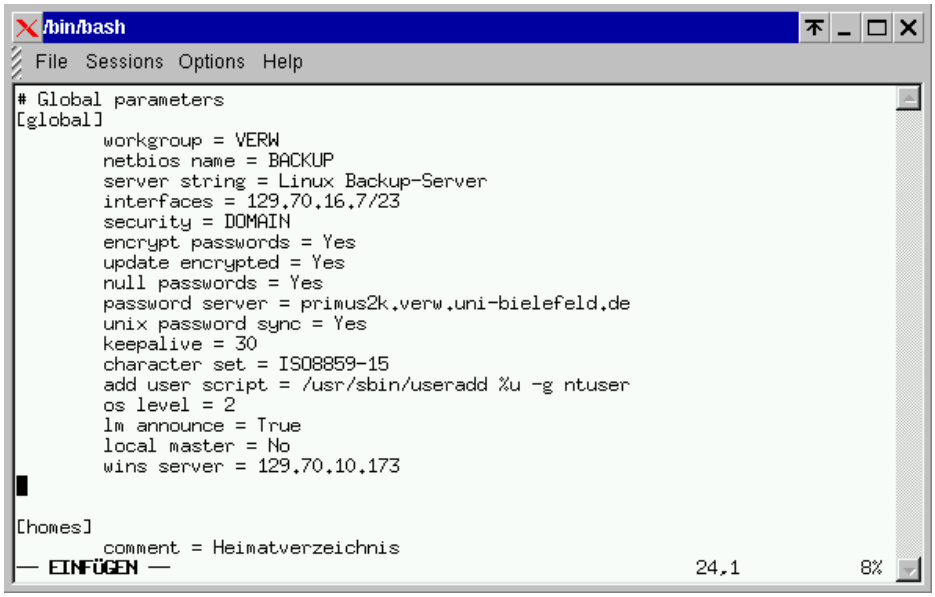

Abbildung 12: Erscheinungsbild des "vi"

Im ex-Modus stehen komplexere Befehle zur Verfügung, die über einzutippende Kommandos erreicht werden. Im Allgemeinen handelt es sich dabei um Befehle, die komplexerer Natur sind, wie z.B. Textbereiche löschen, externe Dateien einfügen, suchen und ersetzen etc..

In den meisten Fällen kann der Eingabemodus optisch nicht vom Kommandomodus unterschieden werden. Wird jedoch der vi-Clone “vim” (der in den meisten Linux-Sys-

temen als vi eingerichtet ist) benutzt, so erscheint in der letzten Bildschirm- bzw. Fensterzeile ein "- INSERT –", wobei im Original-vi dasselbe erreicht werden kann, indem der Editor mit dem Befehl ":set smd" in den Anzeigemodus geschaltet wird. Der ex-Modus macht sich durch ein ":" in der letzten Zeile bemerkbar, und auch dadurch, dass alle Eingaben ausschließlich in der letzten Zeile sichtbar werden.

9.1.1 vi-Kommandos (im Kommandomodus)

a	hinter aktueller Cursorposition einfügen
A	am Ende der Zeile anfügen
c<objekt>	ersetze Objekt (Zeichen) durch die Eingabe
C	ersetze Rest der Zeile durch die Eingabe
d<objekt>	lösche Objekt (Zeichen)
D	lösche den Rest der Zeile
fx	sucht das Zeichen x in der Zeile vorwärts
Fx	sucht das Zeichen x in der Zeile rückwärts
;	wiederhole den letzten Suchbefehl
,	wiederhole den letzten Suchbefehl rückwärts
<CRTL> G	aktuellen Dateinamen anzeigen
i	einfügen hinter dem aktuellen Zeiger
I	einfügen am Anfang der Zeile
o	füge Text nach der aktuellen Zeile ein
O	füge Text vor der aktuellen Zeile an
p	füge zuletzt gelöschten (aber im Puffer noch vorhandenen) Text hinter der aktuellen Zeile an
P	füge zuletzt gelöschten Text vor der aktuellen Zeile an
:	gehe in ex-Modus über
<CRTL> R	gib Bildschirm korrigiert aus

9.1.2 Einige Kommandos im "ex"-Modus

Hierzu muss vi erst in den ex-Modus geschaltet sein :

:w <datei>	Änderungen in gerade bearbeitete Datei/neue Datei zurückschreiben
:e <datei>	angegebene Datei soll editiert werden
:r <datei>	angegebene Datei soll eingelesen werden
:x	verlassen des Editors mit Rückschreiben
:q	verlassen des Editors ohne Rückschreiben
:<anfang>,<ende>s/<zeichenkette1>/<zeichenkette2>	Im Bereich zwischen den Zeilen <anfang> und <ende> nach <zeichenkette1> suchen und bei Erfolg mit <zeichenkette2> ersetzen.

9.2 Der "emacs"

Der Emacs ist ein weiterer Editor, der sehr weit verbreitet ist. Für PC-Benutzer ist seine Benutzung etwas gewohnter als die des vi, doch sind alle Kommandos immer noch mit keinem anderen zu vergleichen. Erfreulich ist die Tatsache, dass jederzeit eine Hilfe angezeigt werden kann, die sämtliche Tastenbelegungen erklärt. Die ungeschlagene Stärke des Emacs ist allerdings, dass er gut programmierbar ist und deshalb auf die Schlüsselwörter von bestimmten Programmiersprachen reagieren kann. Prinzipiell kann er jederzeit durch selbst geschriebene Makros erweitert werden

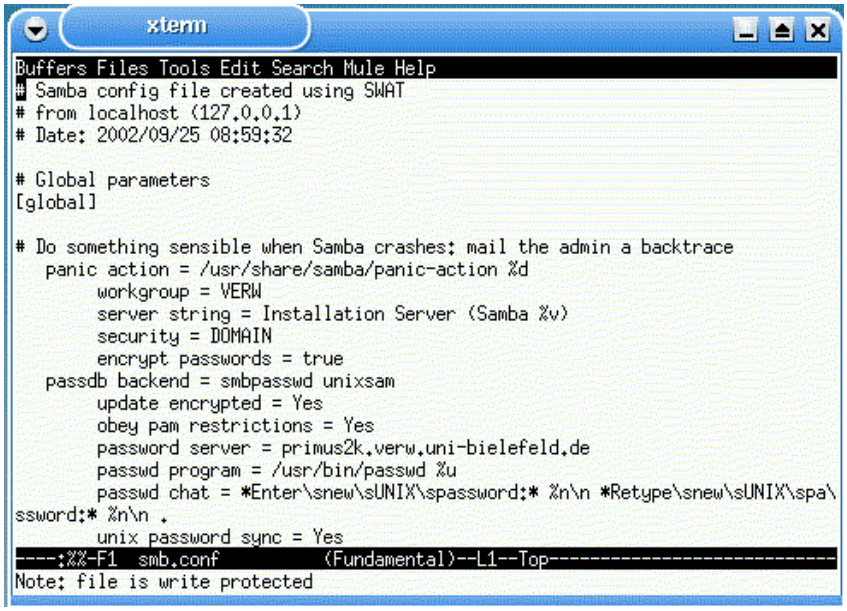

Abbildung 13: Erscheinungsbild des "emacs"

9.2.1 Benutzung des Emacs

Der Emacs ist mit einem Menü versehen, das mit der F10-Taste erreicht werden kann. Die Übliche Form der Steuerung sind aber Tastenkombinationen, die in Zusammanhang mit der Ctrl- (Strg-) Taste oder der ESC-Taste Stehen. Wird in der Dokumentation des Emacs eine Tastenkombination mit vorangestelltem „C“ angegeben, bedeutet das, dass die Ctrl-Taste zusammen mit der nachfolgenden gedrückt werden muss. Der Ausdruck „C-h“ bedeutet also, dass die Ctrl-Taste gedrückt **und gehalten** werden muss, während die Taste „H“ gedrückt wird. Ein Vorangestelltes „M“ bezeichnet die Meta-Taste, mit der üblicherweise „ESC“ gemeint ist. „M-f“ bedeutet also, das die zuerst die ESC-Taste gedrückt **und losgelassen** werden muss und anschließend „F“ betätigt wird.

9.2.2 Einige emacs-Kommandos

C-h	Hilfe aufrufen
C-h t	Tutorial aufrufen
C-k	Alle Zeichen rechts von Cursor löschen (cut)
C-x u	Änderungen aufheben
C-x C-c	Emacs verlassen
C-x C-f	Datei in den Emacs laden
C-x C-s	Datei unter aktuellem Namen speichern
C-x C-w	Datei unter anderem Namen speichern
C-y	gelöschen Text an aktueller Stelle einfügen (paste)

9.3 Der “pico” bzw. "nano"

Der pico oder auch nano ist ebenfalls recht gut zu bedienen, außerdem werden die wichtigsten Befehle in einer Statuszeile angezeigt, so dass immer der Überblick über die vorhandenen Möglichkeiten besteht. Das “^”-Zeichen steht dabei sinnbildlich für die [Ctrl] bzw. [Strg]-Taste (es wäre schließlich langweilig, wenn alle Editoren mit denselben Vokabeln dokumentiert würden ...).

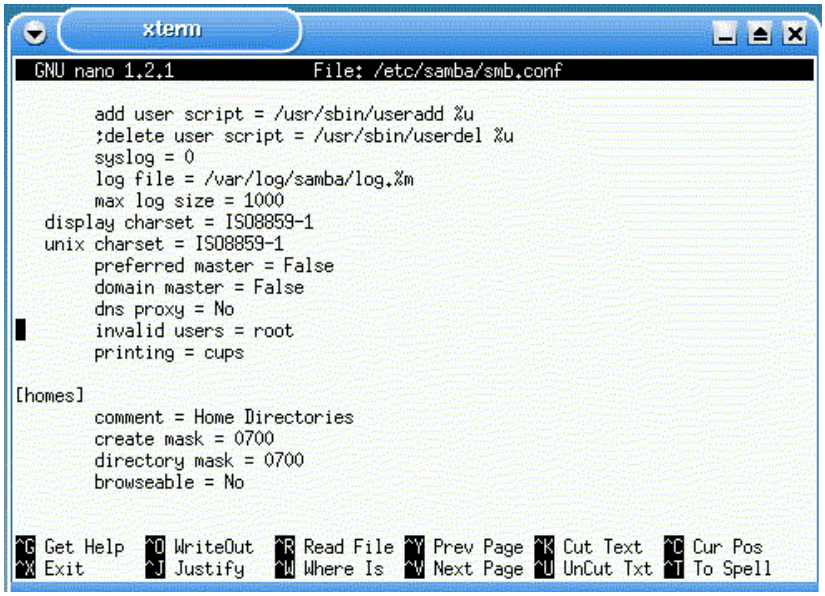

Abbildung 14: Erscheinungsbild des "nano"

Global gesehen entsprechen alle Tasten, die zusammen mit der Control-Taste ([Ctrl] oder [Strg]) gedrückt werden, einem Befehl. Mit den Coursor-Tasten, sowie den PageUp/PageDown-Tasten kann der Cursor durch die edierte Datei bewegt werden.

10 Shell-Programmierung

10.1 Was ist ein SHELL-Skript?

Ein Shell-Skript (shell script) ist im Prinzip nichts anderes, als eine Datei, die in gewohnter Weise Linux-Befehle enthält. Die SHELL ist in der Lage diese Art Dateien so zu verarbeiten, als seien die enthaltenen Befehle von der Tastatur eingegeben worden. Diese Dateien müssen ASCII-Dateien sein, also keine compilierten Programme. Die Shell arbeitet die enthaltenen Befehle dann zeilenweise ab, bis das Ende der Datei erreicht ist. Weil diese Befehle oft einfach nur aus einem "Stapel" von Befehlen bestehen, spricht man an dieser Stelle auch von "Stapel-Dateien" oder im englischen Sprachgebrauch von "batch files"

10.2 Erstellung von SHELL-Skripts

Ein einfaches Beispiel wäre hier die Umsetzung des Befehles "ls -al" in eine Stapel-Datei mit dem Namen "ll". Dazu muss zunächst der Editor aufgerufen werden:

```
$ vi ll_
```

Nun wird in diese Datei der genannte Befehl eingetragen und anschließend der Editor mit speichern verlassen:

```
ls -al
~
~
~
~
~
~
~
:x_
```

Die Datei sieht nun folgendermaßen aus:

```
-rw-r--r--   1 burkhard sysprog      973 Mai 09 1991 ll
```

Hier ist ersichtlich, dass das execute-Bit nicht gesetzt ist, da es sich hierbei eigentlich um eine "normale" Text-Datei handelt. Daher muss die Datei "ll" erst mit dem chmod-Befehl ausführbar gemacht werden:

```
$ chmod u+x ll_
```

Nun sieht die Datei "ll" folgendermaßen aus:

```
-rwxr--r--   1 burkhard sysprog      973 Mai 09 1991 ll
```

Um die Shell nun zu veranlassen, diese Datei auszuführen, genügt es, einfach den kompletten Dateinamen einzugeben:

```
$ ll_
```

Nun verhält sich die Shell so, als sei der Befehl "ls -al" direkt eingegeben worden.

Nun besteht aber noch ein Schönheitsfehler an dieser Datei: Soll nun der Inhalt eines bestimmten Verzeichnisses ausgegeben werden, muss es möglich sein, Parameter mit anzugeben. Da diese nie gleich bleiben, ist es unklug, diese direkt in die Datei hineinzuschreiben. Zu diesen Zweck stellt die Shell die Symbole $1, $2, $3 etc. zur Verfügung. Das Symbol $1 wird bei der Bearbeitung durch den ersten angegebenen Parameter ersetzt, $2 durch den zweiten und so fort. Nicht Benutzte Parameter werden durch den Leer-String ersetzt, so dass z.B. bei nur einem angegebenen Parameter das Symbol $2 ignoriert wird.

Für die erstellte Datei ll kann also ein Parameter verfügbar gemacht werden, wenn in der Datei der Befehl folgendermaßen umgeändert wird:

```
ls -al $1
```

Wird nun z.B. der Befehl

```
$ ll /etc
```

eingegeben, wird auch wirklich der Inhalt des Verzeichnisses /etc ausgegeben.

10.3 Kontrollstrukturen in Shell-Skripts

10.3.1 Die if-Abfrage

Nun kann es vorkommen, dass einige Befehle nicht unbedingt ausgeführt werden dürfen, sondern von bestimmten Bedingungen abhängig sind. Um solch Bedingungen zu testen, existiert in dem UNIX-Befehlspool ein Programm mit dem Namen "test", das - verbunden mit bestimmten Optionen - Bedingungen prüfen kann. die Option f verbunden mit einem Namen, veranlasst test, zu überprüfen, ob eine Datei (file) mit einem solchen Namen existiert. Mit der Option d sucht test nach einem entsprechenden Verzeichnis (directory). So kann das obige Beispiel erweitert werden:

```
if test -d $1
then
     ls -al $1
else
     echo Verzeichnis $1 nicht gefunden!
fi
~
~
:x_
```

Hier ist noch eine Besonderheit hinzugekommen: Das "if". in der normalen Shell-Umgebung kann dieser interne Befehl nicht ausgeführt werden, sondern nur in einem Skript. In der if-Zeile wird die Bedingung geprüft. Ist sie wahr, werden die Befehle hinter dem Wort "then" abgearbeitet und anschließend hinter "fi" weitergearbeitet (sofern dort noch Befehle angeführt sind). Ist die Bedingung falsch, werden die Befehle hinter "else" abgearbeitet.

Das Angegebene Shell-Skript arbeitet also folgendermaßen:

- Zuerst wird geprüft, ob ein Verzeichnis existiert, das den Namen des 1. Parameters besitzt.
- Ist dieses vorhanden, wird mittels ls -al dessen Inhalt ausgegeben.
- Ist es nicht vorhanden, wird die Meldung "Verzeichnis <NAME> nicht gefunden!" ausgegeben, wobei <NAME> für den angegebenen Verzeichnisnamen steht.

Das "else" ist übrigens optional; es kann auch weggelassen werden, wenn es nicht benötigt wird. In der if- Zeile wird übrigen nach dem sog. "return code" abgefragt, der nur dann 0 beträgt, wenn alles fehlerlos gelaufen ist. Im obigen Fall bedeutet dieses, dass test bei einem vorhandenen Verzeichnis 0 zurückgibt, während ein return code Ä 0 bedeutet, dass kein Verzeichnis gefunden wurde.

Da jeder Prozess nach seiner Beendigung einen solchen Wert zurückgibt, kann auch jeder beliebige Programmname in die Bedingungszeile der if-Abfrage eingesetzt werden.

10.3.2 Die while-Schleife

Die while-Schleife arbeitet ähnlich der if-Abfrage:

```
while <Bedingung>
do
     <Befehle>
done
```

Solange die Bedingung hinter dem Wort while wahr ist (den Wert 0 zurückliefert), werden die Befehle zwischen do und done bearbeitet und anschließend wieder die Bedingung überprüft. Ist sie falsch, wird hinter done weitergearbeitet.

10.3.3 Die until-Schleife

Die until-Schleife arbeitet ähnlich der while-Schleife, wobei allerdings die Befehle im Schleifenrumpf (also zwischen do und done) solange abgearbeitet werden, bis die angegebene Bedingung wahr wird.

```
until <Bedingung>
do
     <Befehle>
done
```

Beispiel:

```
until who|grep $1
do
     sleep 100
done
echo "$1 ist jetzt eingeloggt!"
```

Wird dieses Skript als Hintergrundprozess gestartet, wird eine Meldung ausgegeben, wenn der als Argument angegebene Benutzer sich im System eingeloggt hat. Zwischen zwei Überprüfungen wird der Prozess für 100 Sekunden "schlafen gelegt", um nicht unnötig Prozessorzeit zu verbrauchen.

10.3.4 Variablen

Variablen werden oft benötigt, um Schleifen zu steuern. Sie stellen einen Wert dar, der während ihrer Existenz verändert werden kann.

Im einfachsten Fall bekommt eine Variable ihren Wert durch eine einfache Zuweisung:

```
Name='Meier'
```

Nach dieser Operation befindet sich in der Variablen der Wert 'Meier' als Zeichenkette. Natürlich kann dieser Wert auch wieder abgefragt oder ausgegeben werden. Der Befehl

```
echo $Name
```

gibt den Inhalt der Variablen Name auf dem Terminal aus. Wichtig: Dass hier der Inhalt einer Variablen und nicht die Zeichenkette 'Name' gemeint ist, wird durch das $ bekannt gegeben.

Mit dem Programm test können auch Variablen überprüft werden; so würde im obigen Fall die Abfrage

```
test $Name = 'Meier'
```

wahr sein und test somit den return code 0 zurückgeben.

Eine weitere Möglichkeit, eine Variable mit einem Wert zu füllen ist die interaktive Eingabe über den Benutzer:

```
echo "Bitte den Namen eingeben:"
read Name
```

An der Stelle, an der read gestartet wird, wartet der Rechner auf eine Eingabe über die Tastatur und weist der Variablen "Name" den wert zu, den der Benutzer an dieser Stelle eingegeben hat.

Soll die Ausgabe eines Programmes zum Inhalt einer Variablen werden, kann dieses durch das umgekehrte Hochkomma (`) erzwungen werden:

```
Heimverzeichnis=`pwd`
```

Die Variable Heimverzeichnis enthält nun den Namen des aktuellen Verzeichnisses.

10.3.5 Die for-Schleife

Die for-Schleife arbeitet ähnlich der while-Schleife, nur dass hier keine Bedingung zur Steuerung benötigt wird, sondern eine Variable und eine Liste von Werten, mit denen die Variable der Reihe nach gefüllt wird und je ein Schleifendurchlauf getätigt wird:

```
for <Variable> in <Werteliste>
do
     <Befehle>
done
```

Beispiel:

```
echo "Bitte geben Sie die Nachricht ein:"
cat >nachricht

for Name in meier schulz schmidt
do
     write $Name <nachricht>
done
```

In diesem Beispiel wird zunächst durch den cat-Befehl eine Nachricht von dem Benutzer in die Datei "nachricht" geschrieben. Anschließend wird der Befehl "write" dreimal gestartet: Zuerst wird die eingegebene Nachricht an den Benutzer meier gesendet, danach an schulz und zuletzt an schmidt.

11 Drucken

11.1 Der BSD-lpd

Wenn es darum geht, eine Ausgabe auf den Drucker zu erzeugen, wird deutlich, wie stark das Multiuser-Konzept bestimmte Abläufe beeinflusst. Während unter DOS noch von jedem Programm direkt auf die Druckerschnittstelle geschrieben wurde (Anschluss “LPT1”), ist das unter Linux nicht mehr gewünscht, da ja prinzipiell mehrere Benutzer gleichzeitig drucken und sich deshalb blockieren könnten. Um dieses Problem zu beseitigen, benutzt Linux den **Druckerspooler**, der auch aus dem Unix-System stammt.

Es handelt sich dabei um 2 Teile eines Dienstes: der Erste ist ein Dämon, der ständig mitläuft und auf Aufträge wartet, die ihm von dem zweiten Teil, dem Client, mitgeteilt werden. Der Client erzeugt aus den Druckdaten eine Datei, die er in ein vorgegebenes Verzeichnis stellt. Der Dämon nimmt sich die erste dort vorhandene Datei und kopiert die enthaltenen Daten auf den Drucker. Nach Beendigung dieses Auftrags löscht er diese Datei und verfährt auf dieselbe Weise mit der Nächsten. Auf diese Weise können (fast) beliebig viele Benutzer gleichzeitig drucken, ohne dass die Gefahr von “Datensalat” entsteht.

Darüber hinaus ist dieses Drucksystem netzwerkfähig, d.h. der Client kann ohne weitere Softwarekomponenten einen Druck-Dämon auf einem anderen Rechner beauftragen, natürlich vorausgesetzt , dass sich beide im Netzwerk befinden und entsprechend konfiguriert sind.

Der Druckerserver (oder Dämon) heißt “lpd” (**l**ine **p**rinter **d**aemon) und hat keine (direkten) Berührungspunkte mit dem Benutzer. Er wird beim Boot-Vorgang gestartet und erst bei einem Shutdown wieder gestoppt.

Die Client-Komponente heißt “lpr” und wird auch so aufgerufen. Als Parameter werden üblicherweise nur die auszudruckenden Dateien mitgegeben und ggf. den Drucker, sofern mehrere zur Auswahl stehen. Soll also beispielsweise Die Datei “demo.txt” ausgedruckt werden, so kann das durch den Befehl erreicht werden:

```
lpr demo.txt
```

Der Inhalt der Datei demo.txt wird somit auf dem Standard-Drucker ausgegeben. Existieren mehrere Drucker im System, so muss man darauf achten, ob der gewünschte Drucker zugleich der Standarddrucker ist. Ist das nicht der Fall, muss der Name des Druckers hinter der Option “-P” angegeben werden. Der Befehl

```
lpr –Pfarblaser demo.txt
```

sorgt beispielsweise dafür, dass der Ausdruck auf dem Drucker “farblaser” erfolgt.

Normalerweise hat der Druckerspooler ausschließlich die Aufgabe, die Daten – so, wie sie ankommen – auf den Drucker hinüberzutransportieren. Wurde im Linux-System ein Drucker mit Konfigurationswerkzeugen (z.B. mit YaST unter SuSE) eingerichtet, so wird automatisch ein intelligenter **Druckerfilter** der **”apsfilter”** installiert. Es handelt sich dabei um ein Programm, dass die Rohdaten, wie sie dem lpd bereitgestellt werden, überarbeitet werden können. Dabei hat sich die Druckersprache “PostScript” etabliert, da es zu dieser einen Standard gibt und somit eine einheitliche Schnittstelle geschaffen werden kann. Jedes Programm, das den Drucker benutzen möchte, braucht sich also nicht mehr darum zu kümmern, welcher Druckertyp angeschlossen ist, sondern erzeugt einfach PostScript und verlässt sich darauf, dass der apsfilter die Daten für den entsprechenden Drucker richtig aufbereitet.

Die Druckernamen, die im System existieren, können durchaus mehrere auf denselben physischen Drucker verweisen, so dass man verschiedene Fähigkeiten hinter unterschiedlichen Druckernamen verstecken kann. Wird beispielsweise ein Drucker unter SuSE-Linux mit dem Tool YaST erstellt, so werden 3 logische Namen für diesen bereitgestellt:

raw Wird auf diesen Drucker ausgedruckt, verändert apsfilter die Daten nicht. Das macht Sinn, wenn z.B. ein HP Laser Jet angeschlossen ist und das druckende Programm bereits diesen Drucker unterstützt. Auf diese Weise kann die aufwändige Umwandlung in und von PostScript und somit Zeit und Rechnerkapazität erspart werden.

ascii Wenn reiner Text ausgedruckt werden soll (meist Readme's und How-To's), sorgt dieser logische Drucker für die Aufbereitung der Daten. Um Platz auf dem Papier zu sparen, wird normalerweise die Schrift verkleinert, und der Inhalt zweier DIN-A4-Seiten auf einem Blatt ausgedruckt. Dieses Feature ist aber abstellbar

lp2 Dieser Drucker ist der voreingestellte (=Standard-) Drucker. Der apsfilter versucht, den Datenstrom zu erkennen und ihn entsprechend zu verändern oder ihn einfach weiterzugeben.

Da die meisten Systeme mit Druckfunktionen PostScript erzeugen, ist es üblicherweise richtig, den Drucker „lp2“ zu verwenden. Das, was unter Windos der „Standarddrucker“ ist, wird unter Linux in der Umbebungsvariablen „PRINTER“ abgelegt, und lautet meistens „lp“.

11.2 CUPS

Das "Common Unix Printing system" verbreitet sich Zunehmens innerhalb der UNIX-Welt und hat den alten lpr so gut wie verdrängt. Grund hierfür ist nicht nur der größere Leistungsumfang, sondern auch die komfortable Konfigurations-Schnittstelle. Zu den wichtigsten Vorteilen zählen:

- Unterstützung mehrerer Protokolle (LPD, HTTP, IPP, JetDirect)
- Zugangsbeschränkung auf IP-Ebene
- Benutzerauthentifizierung
- Accounting (Abrechnung)

CUPS ist dabei voll kompatibel zum alten BSD-lpd und kann auch über die "alten" Tools (lpr,lpq,lprm ...) bedient und verwaltet werden. Somit kann der lpd **problemlos** durch dieses System ersetzt werden.

Fast jede Linux-Distribution enthält eigene Konfigurations-Tools, um Drucker unter CUPS einzurichten, als unabhängige Schnittstelle bietet sich aber das Web-Interface an, das über jeden HTML-Browser bedient werden kann. Da CUPS standardmäßig auf den TCP-Port 631 "hört", lautet die entsprechende URL

```
http://localhost:631
```

Unter dieser Adresse sollte dann dieses Bild erscheinen:

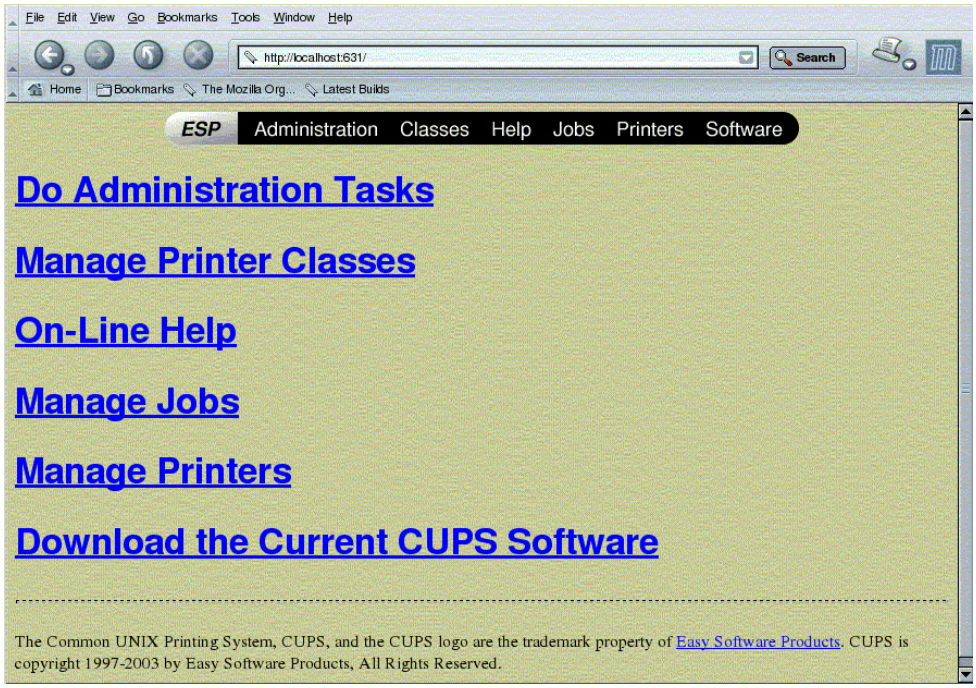

Abbildung 15: CUPS-Konfiguration

Wird hier der Punkt „Manage Printers“ ausgewählt, kann interaktiv ein neuer Drucker angelegt werden, der anschließend Systemweit und Netzwerkweit verfügbar ist. Je nach Vorkonfiguration des CUPS werden bereits im lokalen Netz verfügbare (CUPS-) Drucker automatisch erkannt und eingebunden. Eine manuelle Nachkonfiguration entfällt in diesem Fall – noch nicht einmal Druckertreiber werden dann benötigt.

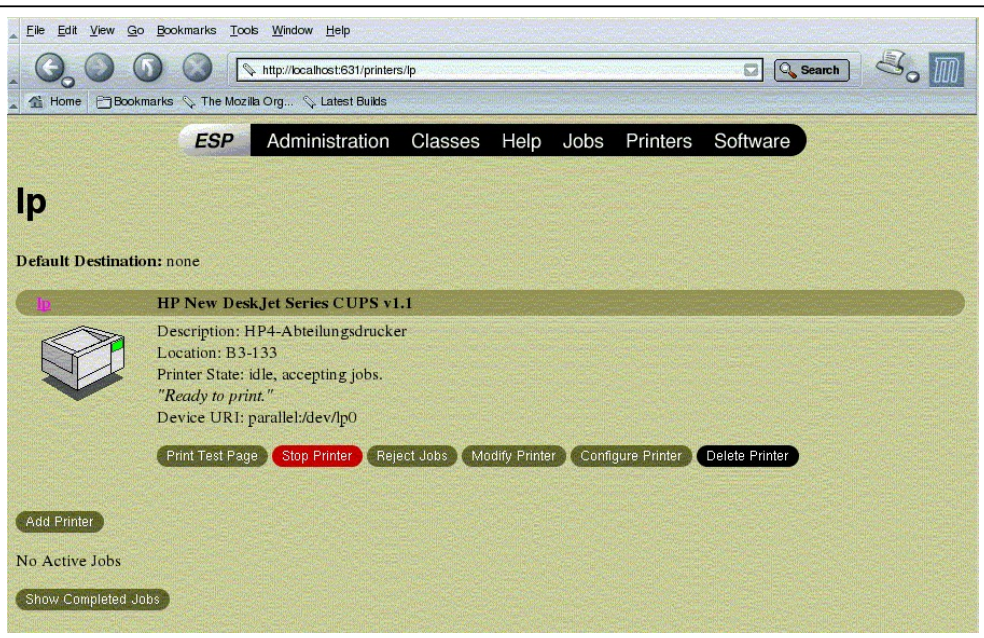

Abbildung 16: Drucker in CUPS

Unter „Printers“ können bereits konfigurierte und bereitgestellt Drucker kontrolliert und nachbearbeitet werden. Besonders die Papiergröße ist von Bedeutung, werden doch die meisten Drucker zunächst nur mit „US Letter“ eingerichtet. Dann sollte die Papiergröße („pagesize“) auf „A4“ geändert werden.

Abbildung 17: Konfiguration ein DirectJet Druckers

Besondere Netzwerk-Drucker können durch Angabe eines URI-Pfades angeschlossen werden. HP-Drucker verwenden beispielsweise häufig die JetDirect- Schnittstelle, die über den Port 9100/tcp angesprochen wird. Dann kann der Drucker wie im obigen Bild über diese Adresse verbunden werden:

```
Socket://<drucker>:9100
```

Der Name des Druckers (<drucker>) ist dann der DNS-Name oder die IP-Adresse des Druckers, so wie er im Netzwerk erscheint.

Wie alle (guten) UNIX-Programme lässt sich auch CUPS durchgängig über reine ASCII-Textdateien konfigurieren, die sich üblicherweise in dem Verzeichnis **`/etc/cups`** finden lassen.

12 Datensicherung mit „tar“

Die Datensicherung ist unter Sewrver-Systemen ein zentraler Punkt, der auch unter Linux mit entsprechender Sorgfalt unterstützt wurde. Für diese Aufgabe existieren – gemäß der UNIX-Philosophie – eine Menge kleiner programme, die zusammengesetzt werden können, um ein betimmtes Ziel zu erreichen. In diesem Fall handelt es sich um den „**t**ape **ar**chiever“, dem „**tar**“. Der Aufruf dieses Programms lautet folgendermaßen:

```
tar -<Optionen> <Verzeichnis>
```

Der „tar“ ist sowohl für die Erstellung als auch für die Extraktion und Überprüfung, was über die anzugebenen Optionen gesteuert wird:

- -c (**c**reate) Erzeugen eines neuen Archivs.
- -x (e**x**tract) Entpacken eines Archivs.
- -t (**t**able) Inhaltsangabe des Archivs ohne zu entpacken.
- -z (**z**ip) Komprimieren des Archivs mit „gzip“ (nur GNU-tar).
- -j (**??**) Komprimieren des Archivs mit „bzip2“ (nur GNU-tar).
- -v (**v**erbose) Auflistung der archivierten oder entpackten Dateien.
- -f (**f**ile) Archivdatei oder –gerät, das erzeugt oder entpackt werden soll.

Die Option „-f“ , die die Archivdatei bestimmt, sollte **immer** angegeben werden, weil sonst nicht unbedingt voraussagbar ist, wohin das Archiv gespeichert wird. Wie der name schon vermuten lässt, ist tar eigentlich auf Magnetbänder spezialisiert, die sequenziell geschrieben und gelesen werden. Weil unter Linux ohnehin alle Geräte Dateien sind, kann statt dem Standard-Magnetband (/dev/rmt0) auch eine beliebige Datei benutzt werden, für die lediglich genügend Platz auf dem betroffenen Filesystem frei sein muss.

Beispiel 1:

Es soll das Verzeichnis „/home/burkhard“ in die Datei „/tmp/burkhard.tar“ archiviert werden. Die nahe liegende Lösung sähe so aus:

```
tar –cvf /tmp/burkhard.tar /home/burkhard
```

Somit würden alle Dateien und Unterverzeichnisse von /home/burkhard archiviert werden, jedoch würde der absolute Pfad „/home/burkhard“ dazu führen, dass die Dateien beim entpacken zwingenderweise an dieselbe Stelle geschrieben werden. Die flexiblere Methode ist die Angebe eines relativen Pfades, wobei vorher in das Verzeichnis navigiert wird, das das zu archivierende enthält, in unserem Fall also „/home“, wonach der komplette Befehl so aussieht:

```
cd /home
tar –cvf /tmp/burkhard.tar ./burkhard
```

Auf jeden Fall sollte **immer** die letzte Methode verwendet werden, weil sonst die Gefahr besteht, dass bei der Restauration vorhandene Dateien überschrieben werden.

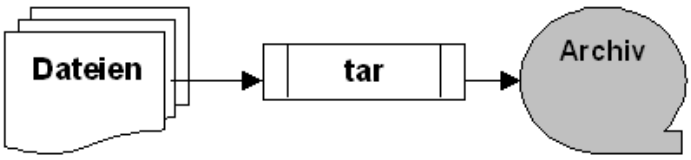

Abbildung 18: Datensicherung mit "tar"

Beispiel 2:

Die erstellte Archivdatei soll in das Verzeichnis /data/archiv entpackt werden. Die Lösung ist nur mit einem Archiv möglich, das nur relative Pfade enthält, wie es in dem 2. Teil des ersten Beispiels der Fall ist. Zunächst muss in das Verzeichnis navigiert werden, in dem die Daten angelegt werden sollen. Anschließend wird „tar“ mit der Option „-x“ statt „-c“ aufgerufen:

```
cd /data/archiv
tar –xvf /tmp/burkhard.tar
```

Der Parameter „Verzeichnis“ kann beim extrahieren fehlen, es sei denn, es sollen nur bestimmte Dateien entpackt werden. In diesem Fall müssen die gewünschten Dateien mit dem Namen angegeben werden, unter dem sie im Archiv abgelegt wurden.

In diesem Beispiel werden jedoch alle archivierten Verzeichnisse mit dem relativem Pfad „./burkhard/...“ gespeichert, so dass das Verzeichnis „burkhard“ im aktuellen Verzeichnis angelegt wird; d.h. das Verzeichnis lautet nun „/data/archiv/burkhard“.

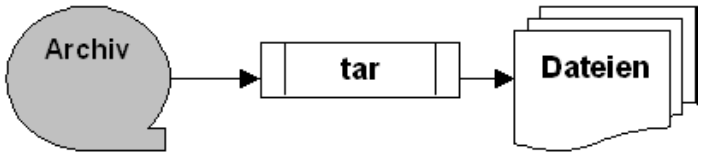

Abbildung 19: Datenrestauration mit "tar"

Wichtig ist an dieser Stelle zu wissen, dass mit „tar“ immer **nur normale Dateien** und Verzeichnisse archiviert werden können. Der „tar“ ist **nicht** in der Lage, Gerätedateien zu archivieren, weshalb er auch nicht für eine vollständige Systemsicherung geeignet ist. Für diesen Zweck ist das Werkzeig „cpio“ das geeignetere, zumal es wesentlich mehr Möglichkeiten bietet. Leider ist es recht umständlich zu verwenden und sollte auch nur nach ausgiebigem Studium der Literatur („man cpio“, s. folgendes Kapitel) verwendet werden.

Der GNU-tar enthält im Unterschied zu den tar-Versionen anderer UNIX-Derivate die Option „-z“, mit der die „on the fly“-Komprimierung bzw. Dekomprimierung mittels „gzip“ initiiert wird. Der gzip ist dabei nicht fest im tar enthalten, sondern muss als externes Modul im aktuellen Pfad vorhanden und ausführbar sein.

Der Um im erten Beispiel ein komprimiertes Archiv zu erhalten, müsste der benötigte Befehl folgendermaßen lauten:

```
tar –cvzf /tmp/burkhard.tar.gz ./burkhard
```

Die Erweiterung des Archivs mit er Endung „.gz“ ist nicht zwingend erforderlich, doch hilft sie, auf den ersten Blick die Art des Archivs und damit die korrekten Extraktions-werkzeuge zu finden.

13 Hilfe

Im Gegensatz zu anderen Betriebsystemen ist unter Linux fast **jeder** verfügbare Befehl dokumentiert. Es handelt sich dabei um ein umfangreiches Referenzwerk, die "**man pages**" (=Manual Pages), die auch in gedruckter Form erhältlich sind, doch ist es sehr angenehm, "mal eben" online nachlesen zu können, wie der benötige Befehl denn benutzt werden muss. Zum Ansehen dieser Referenzen wird ein kleines Programm benötigt, das den treffenden Namen "**man**" (=Manual) trägt. Als Parameter wird dann noch der gesuchte Befehl eingegeben, so dass mit dem Kommando

```
man ls
```

alles zum Befehl "ls" erhalten werden kann. Es muss also schon bekannt sein, zu *welchem* Befehl die Hilfe gebraucht wird, denn es gibt keine Suchfunktionen nach Problemen oder Stichwörtern. Ein Blick in das Verzeichnis /bin oder /usr/bin kann schon die richtige Schreibweise zeigen, da unter Linux fast jedes ausführbare Programm einem Befehl entspricht.

Der "man" durchsucht übrigens der Reihe nach 8 Bände nach dem entsprechenden Kommando, wobei in jedem Band eine bestimmte Gruppe von Befehlen bzw. Funktionen behandelt wird:

Band 1: User Commands (Benutzerkommandos)

Band 2: System Calls (C-Funktionen aus dem Kernel)

Band 3: Subroutines (C-Funktionen aus Libraries)

Band 4: Devices (Gerätedateien)

Band 5: File Formats (Konfigurationsdateien)

Band 6: Games (Einfache Textbasierte Spiele)

Band 7: Miscellaneous (Sonstiges)

Band 8: System Administration (Administrations-Kommandos)

Da es in einigen Bänden gleiche Kommandos gibt, kann der "man" dazu gezwungen werden, nur in einem bestimmten Band zu suchen, indem die Nummer des Bandes als erster und der gesuchte Befehl als zweiter Parameter angegeben wird. Soll beispielsweise etwas über das Programm "chmod" in Erfahrung gebracht werden, reicht der Befehl

```
man chmod
```

aus, während die gleichnamige C-Funktion nur mit

```
man 2 chmod
```

angezeigt wird.

Zum Glück kann aber auch über man selbst eine Hilfeseite aufgerufen und damit die Struktur der Bände angezeigt werden, indem dieser Befehl ausgeführt wird:

```
man man
```

14 Grafische Oberflächen

14.1 X-Server und Windowmanager

Mit den beschriebenen Möglichkeiten des Betriebssystems können hervorragend mit geringen Mitteln gute Serversysteme aufgebaut werden, aber für zeitgemäße Desktop-Anwendungen reichen textbasierte Konsolen einfach nicht aus. In der Unix-Welt etablierte sich daher in den 80er Jahren ein System, das als X-Window bezeichnet wird, oder einfach nur “X”. Dieses System fügt sich harmonisch in die vorhandene Text-Welt ein, so dass alles, das im Text-Modus läuft, ebenso unter X läuft.

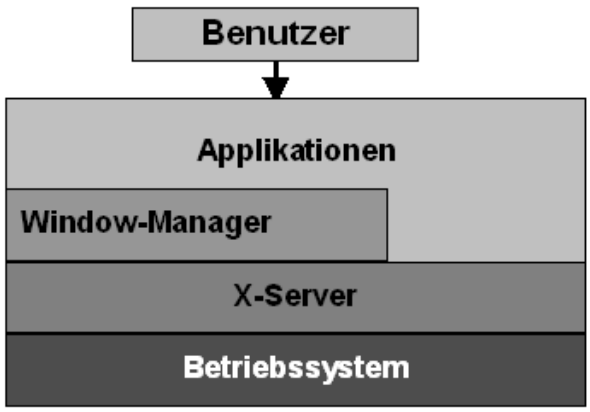

Abbildung 20: Aufbau des X-Window-Systems

X ist – ganz im Sinne von UNIX/Linux – ein streng strukturiertes System, das sich schichtweise um das eigentliche Betriebssystem herum legt. Die Basis bildet der sog. X-Server, ein Programm, das auf der Maschine gestartet wird und die Kommunikation mit der Grafikkarte (und der Tastatur/Maus) aufnimmt. Der X-Server ist somit das einzige hardwareabhängige Modul und kann nur einfache Grafikbefehle ausführen wie z.B. Rechteck/Kreis zeichnen, Text ausgeben, Objekt mit Farbe füllen etc. Komplexere Strukturen wie z.B. das Zeichnen einer Menüleiste oder das Bewegen eines Fensters muss in Form komplexerer Befehle von einer übergeordneten Schicht mitgeteilt werden. Es gibt also für jede Hardware einen bestimmten X-Server, so dass es z.B. einen für Standard-VGA, einen für S3-Grafikkarten, einen für Mach64-Chipsätze etc. Da der X-Server der unter UNIX “X” heißen muss, wird ein *symbolic link* von /usr/bin/X11 nach X gesetzt. Ist der passende X-Server vorhanden und richtig konfiguriert, kann mit dem Befehl “xinit” den reinen X-Server starten, um einen Eindruck zu bekommen, was dieser eigentlich darstellt.

Die nächste Schicht bildet der “Windowmanager”. Er übersetzt Direktiven wie “zeichne Fenster mit Menüleiste” in für den X-Server verständliche Grafikbefehle um. Der Windowmanager stellt das “Look an Feel” der Oberfläche für den Benutzer bereit, übernimmt also das Design der Oberfläche, wie sie sich dem Benutzer bietet. Der Windowmanager bestimmt, wie der Hintergrund aussieht, wie die Knöpfe am Fensterrand aussehen, welche Menüs bereitstehen etc. Da auch in der Linux-Welt unterschiedliche Vorlieben existieren, sind schon die unterschiedlichsten Windowmanager programmiert worden. Da gibt es z.B. den fvwm mit seinem eigenen Stil, olvwm, der die SUN-Oberfläche nachbildet, der bowman, der wie die NextStep-Oberfläche aussieht, den amiwm, der den Amiga nachbildet und auch den fvwm95, der den meisten bekannt vorkommen wird. Achtung: Auch wenn Linux auf einmal wie ein anderes Sys-

tem aussehen wird, bedeutet das noch lange nicht, das die Linux-fremde Software hier läuft. Es handelt sich lediglich um eine optische Simulation.

Den größten Zulauf hat allerdings momentan der **KDE**, der nicht irgendetwas anderes emuliert, sondern seinen eigenen Stil festlegt. Er ist besonders einfach zu konfigurieren und bietet zudem noch eine große Menge weiterer Tools und Applikationen, die aus dem Projekt des KDE hervorgegangen ist. Der KDE ist sehr komfortabel, benötigt aber sehr viel Speicherplatz, so dass er auf sehr kleinen Systemen nicht eingesetzt werden sollte.

Das X-Window-System ist nicht essenzieller Bestandteil des Linux-Systems, sondern ist ein optionales Paket, dass den Linux-Rechner zum grafischen Desktop-System erweitert. Wird der Rechner ausschließlich als Serversystem eingesetzt, so sollte X nicht installiert werden, da es viel Festplatten- und RAM-Speicher benötigt. Ein Textbasierter Linux-Rechner kann mit 64 MB durchaus gut arbeiten. Wird jedoch X gestartet und noch entsprechende Applikationen (Office-Pakete, Grafikbearbeitung etc.), so wird mindestens 128 MB benötigt. Wird als Window-Manager dann noch KDE verwendet, sollte das System schon mindestens 256 MB RAM-Speicher bereit halten. Durch die Unix-Architektur des Speichers kann natürlich durch Erhöhung des Swap-Speichers zu mehr virtuellem Speicher gelangt werden, doch ist ab einer bestimmten Grenze das System dermaßen stark mit der Ein- und Auslagerung von Speicherseiten beschäftigt, dass es sehr langsam wird und deshalb ein effektives Arbeiten nicht mehr möglich ist.

Da X grundsätzlich so konzipiert wurde, dass die grafische Oberfläche auf einem anderen Rechner liegt als die ausgeführten Applikationen, benötigt X grundsätzlich die Netzwerkunterstützung. Auch wenn der PC nicht einmal eine Netzwerkkarte besitzt, wird trotzdem ein Software-Netzwerk aufgebaut, das dieselben Eigenschaften wie ein „echtes“ Netzwerk aufweist. X ist somit eine echtes Client-Server-System.

14.2 Installation von X

Mit der X-Installation ist hier nicht die Detailabstimmung, sondern nur die Einrichtung der Grafikkarten- und Monitor-Unterstützung gemeint. Bei einem Vergleich mit einem Windows-System würde das der Installation des Grafikkartentreibers gleichkommen. im Großen und Ganzen betrifft Installation 2 Aufgaben:

a) Die Auswahl des Servers:

b) Die Konfiguration der Grafikmodi

Beide sind recht Komplex, so dass schon recht früh Werkzeuge für diese Aufgaben gebaut wurden. So gibt es unter SuSE den SaX bzw. SaX2, unter Red Hat den Xconfigurator, während von XFree86 selbst die Programme xf86config und XF86Setupfür die 3er und den xf86cfg mitgeliefert werden. Der xf86config zeichnet sich dadurch aus, dass er völlig auf eine grafische Oberfläche verzichtet, und daher das Mittel der Wahl ist, wenn die anderen versagen. Da sich die Entwicklergruppe um den XFree86 seit einiger Zeit überworfen hat wird von vielen Distributoren mittlerweile das X.Org-System favorisiert. So liefert SuSE seit der Version 9.0 und Debian seit Sarge gar nicht mehr das XFree86 aus. Für den Benutzer ändert sich seitens der Bedienung und Konfiguration mit den bekannten Werkzeugen nichts, nur die administrative Seite muss andere Werkzeuge benutzen.

14.3 X starten

Von der Kommandozeile aus kann – immer vorausgesetzt, dass X bereits installiert ist – X zusammen mit dem Windowmanager durch den Befehl “startx” aufgerufen werden. Viele Distributionen (SuSE, Ubuntu, Knoppix) verwenden ein „Autologin“, wobei

der bei der automatisch eingerichtete Benutzer nach dem Starten des Systems automatisch angemeldet wird. In beiden Fällen wird X mit einem Windowmanager gestartet, so dass das System sofort benutzbar ist.

Sollte eine Situation entstehen, in der die Bedienung der Maus nicht mehr möglich ist, oder aber kein Menü mehr zur Verfügung steht, sorgt die Tastenkombination [Ctrl]-[Alt]-[Backspace] für sofortiges Beenden des X-Servers und damit aller darin laufenden Programme.

14.4 Mit X booten

Nun ist es recht umständlich, erst eine Text-Session zu starten, nur um anschließend den X-Server zu laden. Wenn es sich ohnehin um ein Desktop-System handelt, kann Linux veranlasst werden, sofort beim Startvorgang X zu starten und auch die Anmeldung dem X-Server (genauer: dem Displaymanager) zu überlassen. Dazu muss das System in den Level gebootet werden, in dem auch der “xdm”, der X-Display Manager gestartet wird. In der SuSE-Distribution und Red Hat geschieht das in **Level 5**. In dem folgenden Beispiel sei der Level 3 vorgegeben, der auf 2 Arten automatisch gestartet werden kann.

- Die Distribution bietet ein Konfigurationstool (z.B. den YaST von SuSE oder LinuxConf von Red Hat), mit dem die grafische Oberfläche beim Start eingestellt werden kann.
- Die zentrale Steuerdatei, die /etc/inittab, wird so verändert, dass der Eintrag “initdefault” so aussieht:

```
id:initdefault:5:
```

Mit dem Befehl “init 5” oder beim nächsten Start des Linux-Systems wird dann automatisch der Displaymanager gestartet, über den ein Login ermöglicht wird:

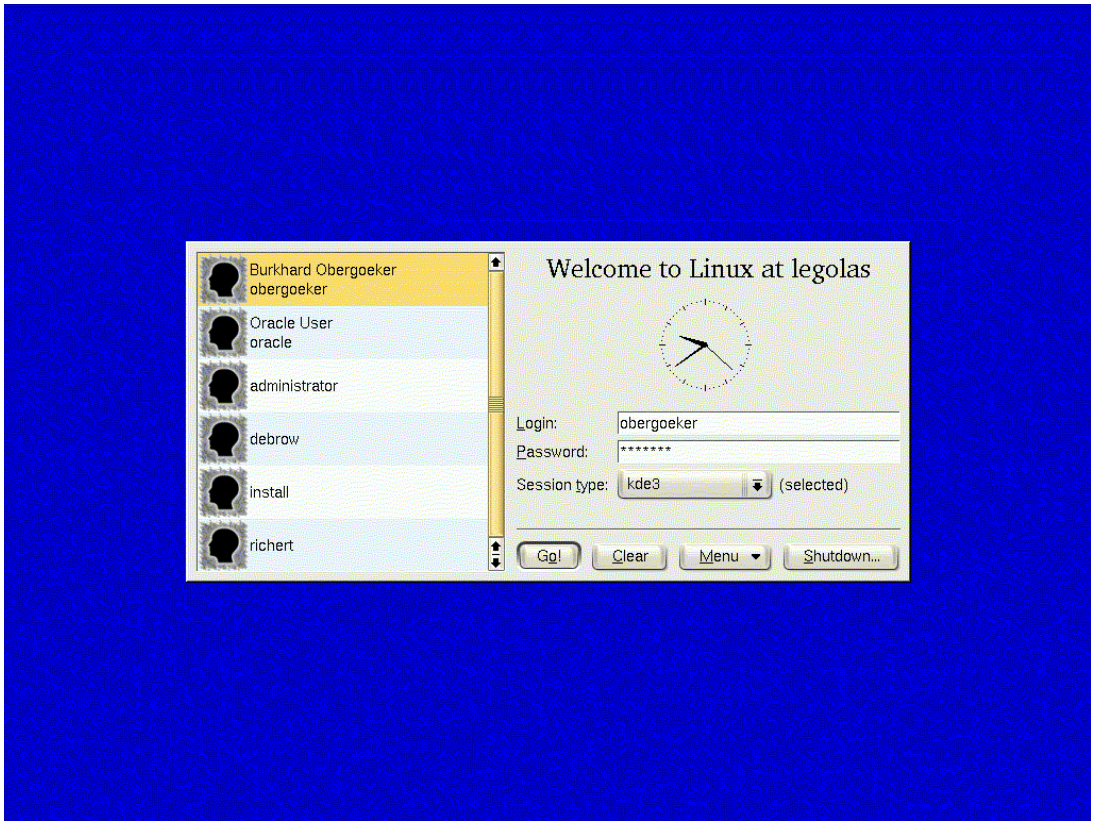

Abbildung 21: Anmeldung im Grafikmodus

Der Anmeldedialog des xdm bzw. kdm erwartet dieselbe Kennung, die auch im login-Vorgang im Textmodus erwartet wird. Anschließend wird eine Session (Sitzung) gestartet, wie sie auch im vorigen Kapitel beschrieben wurde. Technisch gesehen besteht kein Unterschied darin, ob nun die Session von dem xdm (kdm) oder der Kommandozeile gestartet wird. Nur ist die grafische Oberfläche komfortabler, zumal dann, wenn vorrangig X-basierte Applikationen verwendet werden sollen.

15 Benutzung des KDE

Da sich im Linux-Umfeld der KDE besonders etabliert hat, soll auf diesen etwas genauer eingegangen werden.

Zunächst sollte vorausgeschickt werden, dass der KDE durchaus nicht unmittelbar an Linux gekoppelt ist, sondern vielmehr ein völlig eigenständiges Projekt darstellt. Tatsächlich ist der KDE in vielen UNIX-Umgebungen zu finden, wie beispielsweise FreeBSD, Solaris, HP-UX.

KDE wurde entwickelt, um eine intuitive Oberfläche für UNIX-Systeme zur Verfügung zu stellen, damit auch Nicht-UNIX-Experten ohne besonderes Vorwissen ein solches System bedienen können. Dabei wurde das bereits vorhandene und etablierte X-Window-System als Grundlage genommen, um alle technischen Vorteile wie Netzwerkfähigkeit, Modularität, Plattformunabhängigkeit etc. weiterhin nutzen zu können.

Alle bislang verfügbaren Programme mit grafischer Oberfläche können daher unter KDE ohne Anpassungen genutzt werden.

KDE stellt nicht nur eine Bedieneroberfläche, sondern bietet auch Software-Entwicklern eine Umgebung an, mit der sehr komfortabel entwickelt werden kann. Daher gibt es bereits eine große Menge Software, die mit Hilfe dieser Tools entwickelt wurden.

Die bekanntesten sind die Umgebung Kdevelop und das Office-Paket Koffice. Unter www.kde-apps.org stehen viele solcher Produkte bereit.

15.1 Der Desktop

In der Standardeinstellung des KDE4 präsentiert sich die Benutzerschnittstelle ungefähr wie auf dem folgendem Bild.

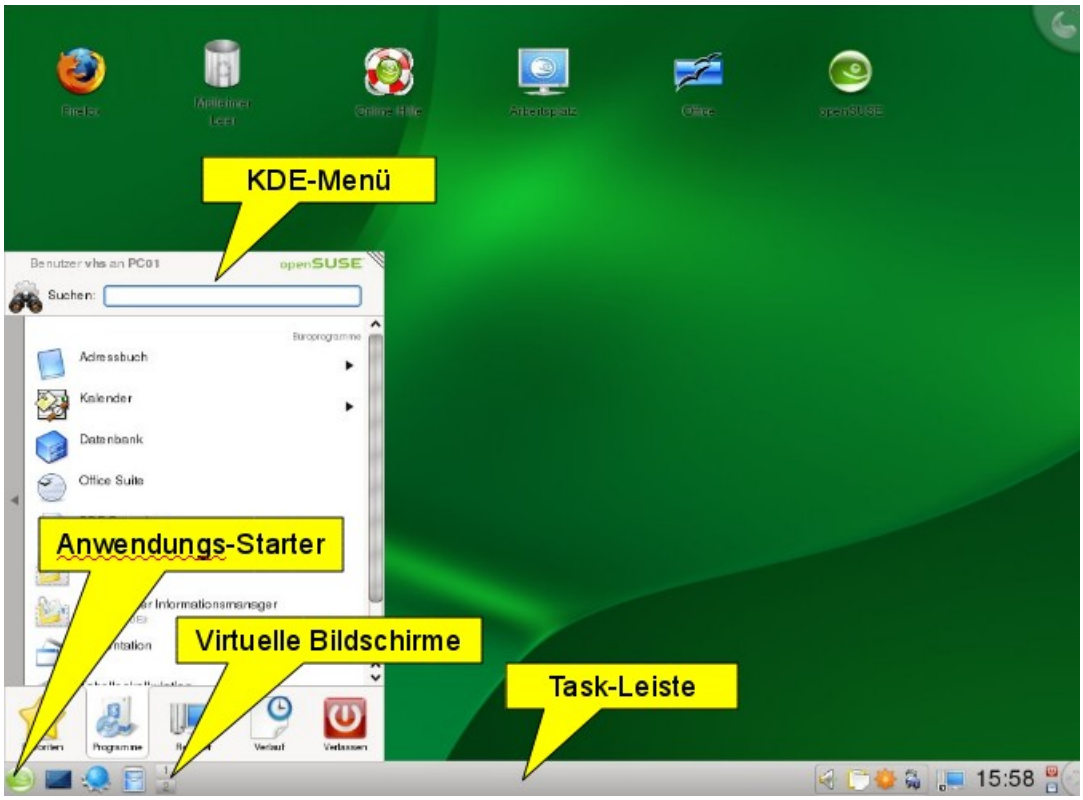

Abbildung 22: KDE4-Session

Der KDE-Desktop besteht aus mehreren Modulen, die sehr frei angeordnet und konfiguriert werden können, aber in der Voreinstellung viel Ähnlichkeit mit dem Desktop von MS-Windows aufweisen. Dadurch soll die Umgewöhnung möglichst vereinfacht werden. Zentraler Punkt ist hier auch der **Anwendungs-Starter**, über den - wie bei MS-Windows - die meisten Programme und Aktionen gestartet werden können. Ein Klick öffnet das Menü. Unter „Verlassen“ befinden sich die Steuerungsfunktionen, wie das “Logout”, das die Sitzung beendet oder das „Shutdown“, das den Rechner herunter fährt.

In der **Kontrollleiste (=Taskleiste)**, in der sich auch der Starter befindet, werden alle laufenden Programme als Eintrag aufgeführt. Ein Klick auf den jeweiligen Eintrag holt die Anwendung in den Vordergrund. Sofern sich diese auf einem andern virtuellen Bildschirm befindet, wird zu diesem gewechselt. Wird der Platz knapp, können mehrere Fenster desselben Programmes zu einem Eintrag gruppiert werden. Erst wenn darauf geklickt wird, werden alle zugehörigen Einträge aufgeführt

Um eine bessere Übersicht zu behalten, können mehrere **virtuelle Bildschirme (Arbeitsflächen)** verwendet werden, die jeweils wie ein einzelner Bildschirm arbeiten. Die Anzahl dieser Arbeitsflächen ist zwischen 1 und 20 wählbar.

Ist die OpenGL-Funktion eingeschaltet wird ein Wechsel zwischen diesen Arbeitsfläche wie ein sich drehender Würfel animiert.

Ein Äquivalent zur zentralen „Systemsteuerung“ ist jedoch nicht in jeder Distribution zu finden, da unterschiedliche Distributionen unterschiedliche Werkzeuge verwenden. Die meisten Funktionen finden sich jedoch in dem Bereich „System“ oder „Rechner“. SuSE-Linux (obiges Bild) positioniert hier den Yast, Debian (und Kubuntu) siedeln hier separate Konfigurationsprogramme an.

Abbildung 23: Geräte-Überwachung

Wird ein wechselbarer Datenträger (CD/DVD, USB-Stick, externe Festplatte etc.) an das System angeschlossen, wird die Geräte-Überwachung aktiv, die diese Datenträger als Liste anbietet. Ein Rechtsklick bietet mögliche Aktionen an, die meistens in der Anzeige im Dolphin bestehen. Diese Liste wird nur einige Sekunden angezeigt, kann aber jederzeit durch einen Klick auf das Geräte-Überwachungs-Symbol in der Kontrollleiste wieder hervor geholt werden.

Da UNIX-Systeme keine Laufwerksbuchstaben verwenden, werden alle zusätzlich angehängten Speichermedien in vorgegebene Verzeichnisse eingehängt (=gemountet) und dieser dann mit dem voreingestellten Dateimanager – meistens der Dolphin – geöffnet. Dieser „automount“ ist eine Funktion des KDE und wird in dieser Form nur unter diesem Windowmanager angeboten. So muss beispielsweise ohne jegliche Grafische Oberfläche auf den manuell gestarteten mount-Befehl zurück gegriffen werden.

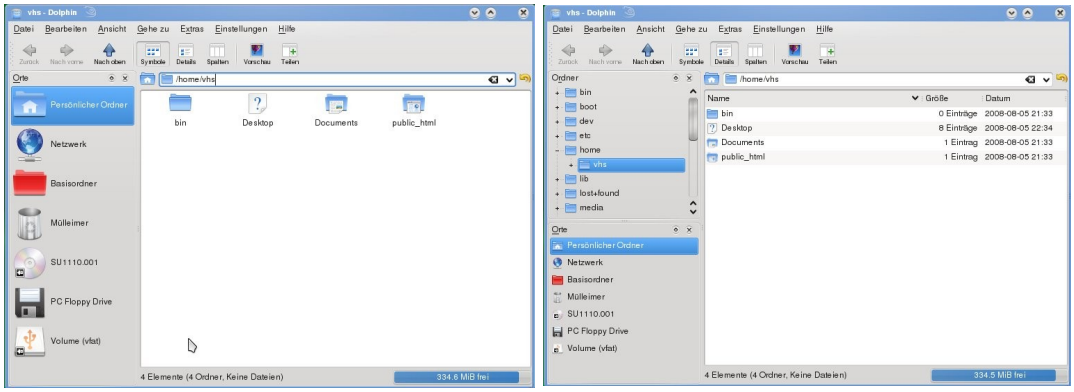

Abbildung 24: Dateimanager Dolphin

Das wichtigste Werkzeug im KDE4 ist der **Dolphin**, der gleich mehrere Aufgaben übernehmen kann.

Er ist für alle Aufgaben zu gebrauchen, die Dateien betreffen, daher kann er mit dem Explorer von Microsoft verglichen werden, er kann jedoch viel mehr!

Er zeigt alle Dateien in dem gerade ausgewählten Verzeichnis an und startet bei einem **einfachen Klick** die zugehörige Anwendung. Es werden jeweils die Programme gestartet, die als Standard-Anwendung zu diesem Dateityp hinterlegt wird. Der Dolphin orientiert sich dabei – wir der Rest des Linux-Systems auch – nicht an dem Dateinamen, sondern an dem Inhalt der Datei. So kann es nicht passieren, dass durch die Vergabe eines unpassenden Namens eine Datei nicht mehr brauchbar ist.

Er kann außerdem über fast jedes beliebige Netzwerk-Protokoll (ssh, smb, nfs, ftp, http etc.) mit anderen Rechnern Kontakt aufnehmen und Dateien übertragen, Wechsellaufwerke können über ihn eingebunden, bearbeitet und auch ausgeworfen werden, selbst Vorschauen von Bildern können dargestellt werden.

Wenn in der linken Seitenleiste die Ordner eingeblendet werden, wird das Dateisystem von Linux im Dolphin deutlich. Jeder Benutzer hat sein eigenes „Home-Directory“, das durch eine Klick auf den ***Ort*** „*Persönlicher Ordner*“ erreicht wird. Tatsächlich ist das der Ordner, der mit dem Login identisch und im Unterverzeichnis **/home** zu finden ist. Nur in diesem Bereich hat ein „normaler“ Benutzer Schreib-Rechte, alles andere bleibt so geschützt. Somit werden ungewollte Veränderungen des Betriebssystems – wie beispielsweise durch Viren und Trojaner - verhindert.

Zusätzlich angeschlossene Datenträger (=Laufwerke) werden zwar einzeln unter „Orte“ sichtbar, im Gesamt-System erscheinen sie aber als Unterverzeichnisse von **/media** eingehängt (=gemountet). Sie können aber auch an jedem beliebigen anderen Verzeichnis eingehängt werden. So entstehen die von Windows gewohnten Laufwerksbuchstaben gar nicht erst. Das gesamte Verzeichnissystem wird dadurch flexibler und übersichtlicher.

16 Kernel-Konfiguration

Wie schon Eingangs erwähnt, ist das eigentliche Linux der sog. „Kernel“, also das zentrale Stück des Betriebssystems, das die Grundfunktionen zur Verfügung stellt. In diesem Kernel werden sämtliche Hardwarekomponenten (außer Grafikkarten) konfiguriert und unterstützt. Im Gegensatz zu anderen Betriebssystemen werden die Hardware-Treiber nicht von externen Anbietern zur Verfügung gestellt, sondern liegen vollständig als Source in jedem Linux-System bereit.

Eine Standard-Installation enthält bereits einen vollständigen Kernel, der alles enthält, was im System benötigt wird. Das diese Kernel jedoch mehr enthalten, als unbedingt notwendig, lohnt es sich, einen „eigenen“ Kernel zu bauen.

Zu diesem Zweck werden außer den Kernel-Sourcen der C-Compiler (gcc / egcs), der „GNU make“ und zusätzliche Utilities (binutils) benötigt. Nach erfolgter Installation ist der gesamte Verzeichnisbaum unter „***/usr/src/linux***“ zu finden, in denen – sauber in Unterverzeichnissen organisiert – der gesamte C-Quellcode sämtlicher Treiber befindet. Ein geübter C-Programmierer kann also durchaus diese Treiber an seine eigene Hardware anpassen, wenn bestimmte Eigenschaften nicht gewünscht sind oder hinzugefügt werden sollen.

16.1 Konfigurationstools

Um aber auch nicht-Programmierern die Möglichkeit des Kernelbaus zu erschließen, werden mehrere Konfigurationstools bereitgestellt, die sich unterschiedlich komfortabel bedienen lassen, damit aber auch unterschiedliche Anforderungen an das umgebende System stellen. Grundsätzlich kann eine Kernelmodifikation nur vom „root“ durchgeführt werden und auch nur aus dem Verzeichnis /usr/src/linux oder wo auch sonst der Kernel-Baum seinen Ursprung hat. Alle Tools werden mit dem Programm „make“ erstellt und auch gleichzeitig aufgerufen.

Das spartanische „config“ benötigt lediglich eine Kommandozeile und fragt der Reihe nach **alle** Einstellungen interaktiv ab. Das dauert üblicherweise sehr lange, weil mittlerweile sehr viele Einstellungen veränderbar sind. Gestartet wird es mit dem Befehl.

```
make config
```

Schöner und auch weniger fehleranfällig ist das grafisch orientierte „xconfig“, das sehr elegant per Maus bedient werden kann. Leider erwartet dieses jedoch nicht nur ein laufendes X-System, sondern auch die Komponenten Tcl/Tk, mit denen die grafischen Objekte erstellt werden. Gestartet wird dieses Tool mit diesem Befehl:

```
make xconfig
```

Das dritte Tool bietet einen guten Kompromiss zwischen den beiden genannten. Es nennt sich „menuconfig“ und ist lediglich auf ein unterstütztes ASCII-Terminal angewiesen, das mit der Console jedem Linux-Rechner zur Verfügung steht. Es benötigt also keine grafische Oberfläche, keine zusätzlichen Module und ist dennoch komfortabel bedienbar. Es wird mit dem Befehl

```
make menuconfig
```

gestartet und besitzt folgendes Erscheinungsbild:

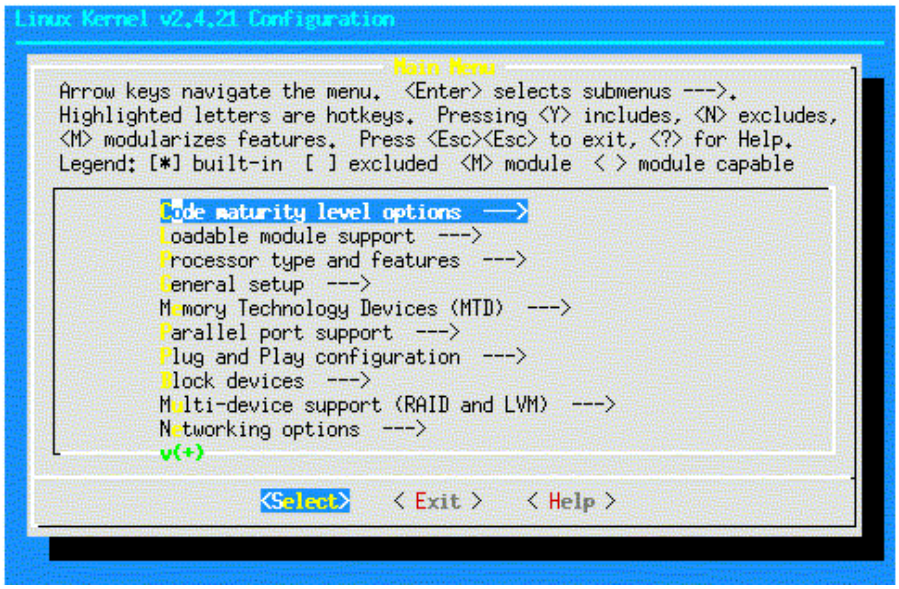

Abbildung 25: Erscheinungsbild des "menuconfig"

Die einzelnen Menüpunkte werden mit den Cursortasten angewählt und mit der Return-Taste bestätigt. In den einzelnen Sektionen können die entsprechenden Treiber entweder fest in den Kernel integriert werden (dargestellt durch ein „*“) oder aber als Modul konfiguriert werden (durch ein „M“ dargestellt).

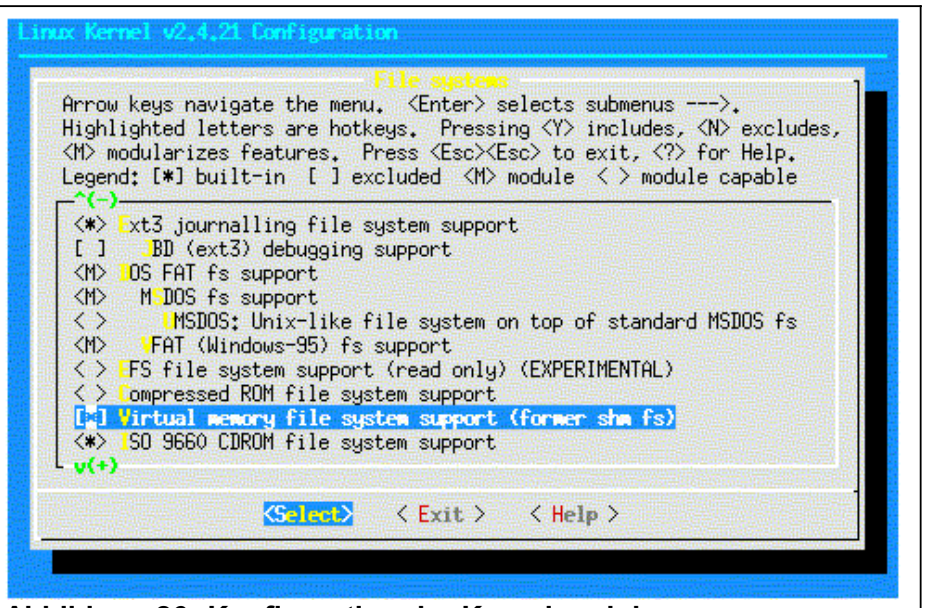

Abbildung 26: Konfiguration der Kernelmodule

Module bieten den Vorteil, dass sie während der Laufzeit des Systems gestoppt, geändert und wieder zum System hinzugefügt werden können, ohne dass ein Neustart notwendig wird. Weil nun aber Module erst dann gestartet werden können, wenn das Root-Filesystem bereits gemountet wurde, dürfen Treiber, die vorher benötigt werden, nur direkt in den Kernel eingebunden werden. Zu diesen gehören beispielsweise IDE-, SCSI-, Filesystem- und CDROM-Treiber.

16.2 Übersetzen des Kernels

Bevor jetzt ein neuer Kernel generiert wird, sollte **auf jeden Fall eine getestete Bootdiskette** bereitliegen, wie sie bei der Installation üblicherweise erzeugt wird. Sollte der

neu generierte Kernel nicht lauffähig sein, kann dann das System immer noch mit der Diskette gestartet werden.

Sind alle Treiber und Module wie gewünscht konfiguriert, kann der Kernel Übersetzt und eingerichtet werden. Für diesen Zweck müssen die folgenden Schritte eingehalten werden:

1. Sofern noch ein Kernel der Version 2.4.x verwendet wird, müssen die Abhängigkeiten im Verzeichnisbaum aktualisiert werden. Seit kernel 2.6.x ist dieses jedoch nicht mehr nötig. Der Befehl hierzu lautet:
   `make dep`
2. Übersetzen des Kernels und dessen Installation in das Verzeichnis „/“ oder „/boot“ (SuSE und Red Hat) sowie LILO-Aufruf. Im Fall von Red Hat und S.u.S.E. muss daruf geachtet werden, dass der neue Kernel auch tatsächlich im Verzeichnis „/boot“ landet. In dem Makefile, das mit den Kernel-Paketen von „kernel.org“ und dessen Mirrors ausgeliefert wird, ist das Root-(„/“)-Verzeichnis vorgegeben. Allerdings kann das durch den Eintrag „*INSTALL_PATH=/boot*“ im erwähnten Makefile („*/usr/src/linux/Makefile*“) korrigiert werden.
   `make bzlilo`
3. Übersetzen der Module
   `make modules`
4. Installation der Module im System unter /lib/modules/<*kernel-version*> (z.B. /usr/lib/2.4.17)
   `make modules_install`

Der Übersetzungsvorgang – auch Compiler-Lauf genannt) benötigt sehr viel Rechenkapazität, weshalb es sich nicht empfiehlt, diesen Vorgang auf einem System zu starten, das von mehreren Anwendern gerade benutzt wird.

Ist der Kernel neu gebunden und installiert worden (make bzlilo), muss der Rechner neu gebootet werden, bei der Neuübersetzung eines Modules reicht der Neustart desselben.

Arbeitet der Kernel nicht wie gewünscht, muss erneut die Konfiguration durchgeführt und die obige Vorgehensweise wiederholt werden, wobei diesesmal nur die Teile neu übersetzt werden, die verändert wurden. Der Compilerlauf ist dadurch wesentlich kürzer als beim ersten Mal.

Funktionieren Kernel und Module zur Zufriedenheit, kann der Speicherplatz wieder freigegeben werden, der von den Zwischenprodukten belegt wird. Der hierfür benötigte Befehl lautet:

```
make clean
```

Die Konfigurationsinformationen bleiben dabei erhalten, so dass eine anschließende Neukonfiguration auf der vorhandenen aufbauen kann. Diese Informationen werden übrigens in der Datei „***.config***“ abgelegt, die bei einer neuen Kernelversion lediglich in das Hauptverzeichnis des neuen Kernel kopiert und mit dem Befehl

```
make oldconfig
```

an die neue Umgebung angepasst werden kann.

17 Freie Applikationen

Die Open-Source-Gemeinde propagiert den Gedanken der Freien Software, die unter der GPL, der **GNU General Public License** vertrieben wird. Grob bedeutet das, dass die Software in binärer Form und im Sourcecode weitergegeben werden darf, wenn keine Bezahlung für die Software verlangt wird. Lediglich Gebühren für Medien und Vervielfätigung sind erlaubt. Darüberhinaus muss gewährleisttet werden, dass die Software grundsätzlich zusammen mit dem Autorenverzeichnis ausgeliefert wird. Die Finanzielle Unterstützung erfolgt ausschließlich durch Dokumentation und Service bzw. Support.

Der Linux-Kern sowie die GNU-Software, die als „drumherum“ mitgeliefert wird, unterliegt der GPL. Natürlich ist darauf zu achten, dass nicht alle von der Distribution mitgelieferten Software zwangsweise frei sein muss. Besonders häufig werden Demo-Versionen kommerzieller Software der Distribution hinzugefügt oder Besondere Absprachen mit den Herstellern getroffen. In diesem Fall darf das Medium, auf dem sich die entsprechende Software befindet, nicht ohne weiteres kopiert und weitergegeben werden.

17.1 Bezugsquellen

Obwohl die meisten Linux-Distributionen bereits eine sehr Umfangreiche Sammlung freier Software mitliefern, lohnt sich immer ein Blick auf einschlägige Web-Sites, auf denen Freeware in Gigabytes zum Download bereitliegen. Die Palette reicht dabei vom Texteditor über HTML-Browsern bis hin zu komplexen Datenbansystemen.

Eine kleine Auswahl von Web-Sites, auf denen das reichhaltige Angebot durchstöbert werden kann:

- http://www.linux.de
- http://www.linuxberg.com
- http://www.freshmeat.net
- http://www.linuxarchives.com/
- ftp://ftp.funet.fi
- ftp://ftp.gwdg.de

17.2 Installation

Um es vorwegzuschicken: Die Installation von Software kann **nur durch den „root“** geschehen. Normale Benutzer können zwar einzelne ausführbare Programme in ihre eigenen Verzeichnisse kopieren, systemweite Installation ist ihnen aber vorenthalten.

Die meisten Pakete liegen in Sourcecode-Form vor, die vor dem Einsatz zunächst mit dem Compiler übersetzt und anschließend installiert werden müssen. Üblicherweise ist jedoch die Installation in der mitgelieferten Dokumentation („README“, <Paket>.lsm, „INSTALL“ etc.) gut beschrieben. Vor allem sollte darauf geachtet werden, welche Pakete von der zu installierenden Software vorausgesetzt werden. Die zentrale Komponente, die von allen Komponenten vorausgesetzt wird, ist der C-Compiler (gcc), um die Sourcen in eine Maschinenlesbare Form zu übersetzen.

Die Pakete, die auf den oben genannten Servern bereitliegen, sind üblicherweise mit der Endung „.tar.gz“ versehen, was bedeutet, dass die voriegende Software aus mehreren Dateien bestegt, die erst mit „tar“ in eine Archivdatei zusammengesetzt und anschließend mit „gzip“ komprimiert wurden (s. Kap. Datensicherung). Um ein solches paket einzurichten, sollte zunächst ein separates Verzeichnis für die Quelldateien an-

gelegt werden. In den meisten Linux-Distributionen befindet sich bereits ein Verzeichnis „/usr/src/“ in dem ein solches Verzeichnis eingerichtet werden kann. Dorthin wird nun das Paket kopiert und entpackt, indem der Befehl

```
tar –xvzf archivdatei
```

eingegeben wird. Anschließend sollte in das neu entstandene Verzeichnis gewechselt und eine Datei „README“, „INSTALL“, „LIESMICH“ oder Ähnliches gesicht und mit „less“ oder einem Editor gelesen werden. Dort befindet sich üblicherweise eine Anleitung zur weiteren Vorgehensweise.

Beispiel

Die Datei „iptraf-1.4.3.tar.gz“ (ein TCP/IP-Sniffer) wurde von einem Server heruntergeladen und in das Verzeichnis „/usr/src/myfiles“ kopiert. Im nächsten Schritt wird mit dem Befehl

```
tar –xvzf iptraf-1.4.3.tar.gz
```

das archivierte Verzeichnis entpackt, woraufhin ein Verzeichnis iptraf-1.4.3 erstellt und alle benötigten Dateien dorthinein extrahiert werden. In diesem Verzeichnis ist dann folgender Inhalt zu finden:

```
gandalf:/usr/src/myfiles/iptraf-1.4.3 # ll
total 67
drwx------   4 root     root         1024 Nov 20 20:45 .
drwxr-xr-x   3 root     root         1024 Nov 20 20:42 ..
-rw-------   1 root     root        17878 Apr 15  1999 CHANGES
-rw-------   1 root     root        18271 Jul  8  1998 COPYING
drwx------   2 root     root         1024 Dec 19  1998 Documentation
-rw-------   1 root     root         3498 Dec 19  1998 INSTALL
-rw-------   1 root     root        11402 Nov  1  1998 README
-rw-------   1 root     root         1961 Dec 24  1998
README.interfaces
-rw-------   1 root     root          939 Dec  9  1998
README.platforms
-rw-------   1 root     root         2147 Dec  3  1998
README.rvnamed
-rw-------   1 root     root         1379 Nov  1  1998 WHATELSE
drwx------   2 root     root         2048 Apr 15  1999 src
gandalf:/usr/src/myfiles/iptraf-1.4.3 #
```

Besonderes Augenmerk sollte dabei auf die Datei „INSTALL“ gelegt werden, die die notwendigen Informationen zur Installation enthält. Aber auch die „README-Dateien sollten nicht ignoriert werden, da sich für viele Probleme Informationen beinhalten, die vielleicht erst im nachinein entstehen können.

In der erwähnten INSTALL-Dateiist dann auch die Information zu finden, dass bereits lauffähige Binary-Versionen mitgeliefert wurden, die einfach nur noch installiert werden müssen. Sollen diese binaries jedoch neu erzeugt werden (Was auf jeden Fall der bessere Weg ist), wartet die Datei mit diesem Inhalt auf:

```
...

RECOMPILATION

Should you wish to recompile the program (perhaps to reduce the size
of the binary by letting it use the shared versions of the ncurses
and panels libraries), you will need these:

        1.  Kernel 2.0.0 or later, with sources decompressed in
            /usr/src/linux.  Earlier versions may still work, but
cannot
            be confirmed.  Kernel 2.0.34 or higher is recommended.
        2.  ncurses 1.9.9e or later.  Earlier versions have
undesirable
```

```
            keystroke and overlapping window refresh behavior.
        3.  gcc 2.7.0 or later.

To compile, just cd to the src directory and type "make" at the
shell
prompt.  You may want to edit the Makefile to tweak some options
before
you compile.  There should be no errors.

...
```

Nachdem also nun mittels „cd src“ ind das gewünschte Verzeichnis gewechselt wurde, kann mit den beiden Befehlen

```
make
make install
```

die Übersetzung und die Installation durchgeführt werden. Allerdings sollte in der Datei „Makefile“ nachgesehen werden, wohin denn nun die Binärdatei installiert wird. Wenn z.B. mit einem SuSE-System gearbeitet wird, ist der Installationspfad „/usr/local/bin“ falsch, es sollte daher **vor der Installation** in der Datei der Eintrag „TARGET“ auf „/usr/bin“ umgeändert werden. Ist dann die Übersetzung und die Installation fehlerfrei abgelaufen, kann das neue Programm – wie in der mitgelieferten Dokumentation beschrieben – genutzt werden.

Die meisten anderen Pakete erzwingen noch eine Reihe vorangehender Konfigurationen, die sich entweder darauf beschränken, eine Datei namens „configure“ auszuführen, oder aber eine manuelle Anpassung verschiedenster Dateien vorziehen. Der prinzipielle Ablauf einer Installation eines solchen Paketes ist jedoch immer recht ähnlich.

17.3 RPM-Pakete

Wird eine „Red Hat“- oder „S.u.S.E.“- Distribution (oder darauf aufbauende) benutzt, werden die meisten Pakete mittels „rpm“, dem „Red Hat Package Manager“ eingerichtet. Die Vorgehensweise ist bei diesem System wesentlich einfacher, allerdings können nur solche Pakete problemlos installiert werden, die auf die jeweilige Distribution zugeschnitten sind. Wird also z.B. ein Red Hat-Paket in einem SuSE-System eingespielt, so werden üblicherweise Nacharbeiten notwendig, wie z.B. das Anpassen der Startskripten und die Umsetzung der Libraries. Die Pakete erscheinen dann in Form einer Datei mit der Endung „.rpm“ und können mit dem Befehl

```
rpm –i <paketdatei>
```

Installiert werden Meistens lässt sich das jedoch schon über ein zentrales Konfigurationstool erledigen, wie der YaST unter Linux oder der frei verfügbare „*xrpm*“. Die Übersicht über die installierten Pakete kann mit dem Befehl

```
rpm –q -a
```

Erreicht werden, die Entfernung von Paketen über

```
rpm –e <paketname>
```

Achtung: Der Paketname muss genauso geschrieben werden, wie er in der RPM-Datenbank abgelegt wurde, sonst wird er nicht erkannt.

17.4 DEB-Pakete

Bei einer auf Debian basierenden Distribution ([K]Ubuntu, Knoppix) wird der Debian-Pakage-Manager verwendet, für den als „Frontend“ der APT (Advanced Packaging Tool) benutzt wird. Die Installation setzt entweder ein Medium (CD/DVD, Festplatte)

oder das Internet voraus. Bei letzterem ist die Installation eines Software-Paketes extrem einfach. Soll beispielsweise „Kmahjongg“ installiert werden, reicht der Befehl:

```
apt-get install kmahjongg
```

Dabei wird nicht nur das Paket selbst, sondern auch alle Pakete installiert, die als Voraussetzung benötigt werden. Die Quellen für die installierbaren Pakete werden in der Datei `/etc/apt/sources.list` festgehalten, so dass sie jederzeit mit einem Texteditor angepasst werden können. Die Informationen zu den verfügbaren Paketen und deren Abhängigkeiten werden lokal abgelegt, so dass sie mit der Zeit veralten und daher vor einer wichtigen Installation mit

```
apt-get update
```

auf den neuesten Stand gebracht werden müssen.

Ein Upgrade auf eine neue Distributions-Version benötigt lediglich die Befehle:

```
apt-get update
apt-get dist-upgrade
```

Natürlich gibt es auch übersichtliche Werkzeuge, mit denen die Pakete visuell ansprechend verwaltet werden können. Für die Text-Konsole bietet such „***aptitude***“, an, unter KDE der „***kpackage***“.

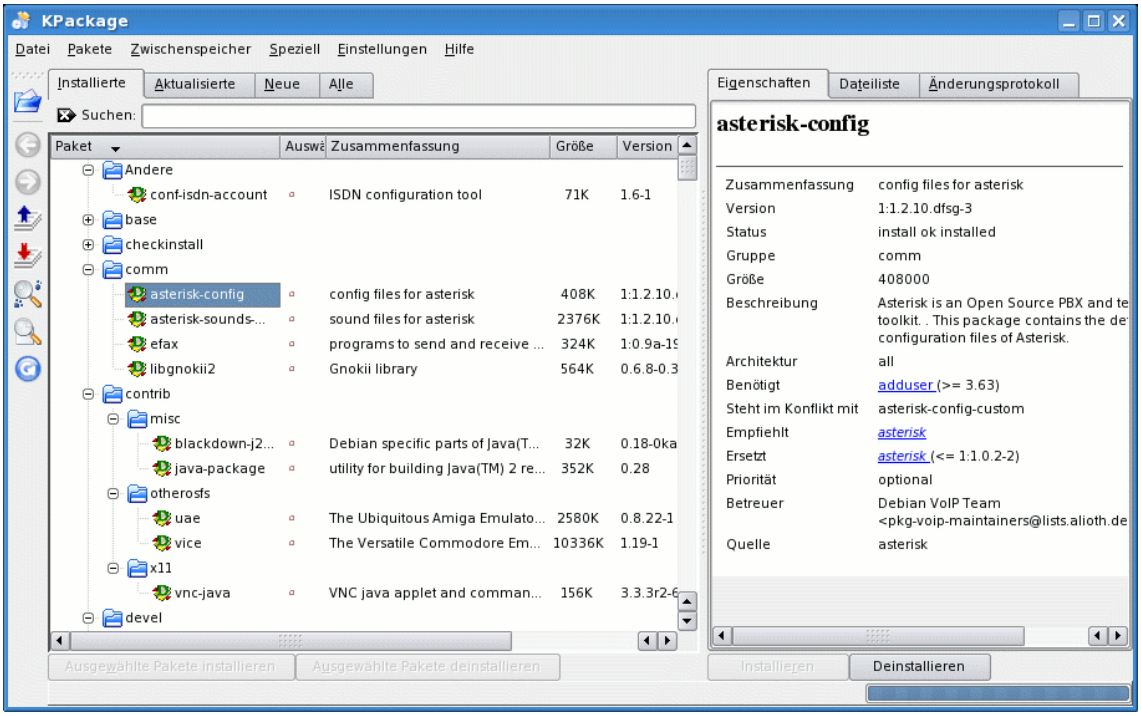

Abbildung 27: Kpackage

18 Kommerzielle Applikationen

Es gibt bereits eine große Menge verfügbarer X-Software für Linux, wobei kommerzielle Software genauso bereitsteht wie Free- oder Shareware. In den meisten Fällen der kommerziellen Applikationen darf eine privat genutzte Version kostenlos eingesetzt werden, während für kommerzielle Zwecke der normale Preis gezahlt werden muss.

18.1 Office-Software

Zweifellos ist die Office-Software – also Textverarbeitung, Tabellenkalkulation, Terminplaner – die populärste Software überhaupt geworden, zumindest im privaten Bereich. Daher haben es auch viele Softwarehäuser nicht versäumt, auf den Linux-Zug aufzuspringen und eine Version ihrer Applikationen für dieses Betriebssystem zu portieren. Zu den bekannten Arten der Desktop-Office-Programme gehören:

- StarOffice 8.0 und die freie Variante OpenOffice 2.0
- Koffice
- WordPerfect 8.0
- Applixware

Eine Applikation unter Linux wird ähnlich bedient wie unter Windows, zumal die meisten SW-Hersteller Ihre Software für beide Betriebssysteme anbieten. Es bleiben jedoch die Unterschiede, die der verwendete Windowmanager mit sich bringt. Der KDE ist da schon sehr Windows- ähnlich, wer es aber noch MS-konformer haben mochte, kann den fvwm95 einsetzen. Natürlich gibt es noch eine Reihe weiterer Systeme aus diesem Bereich, die für Linux geschrieben worden sind.

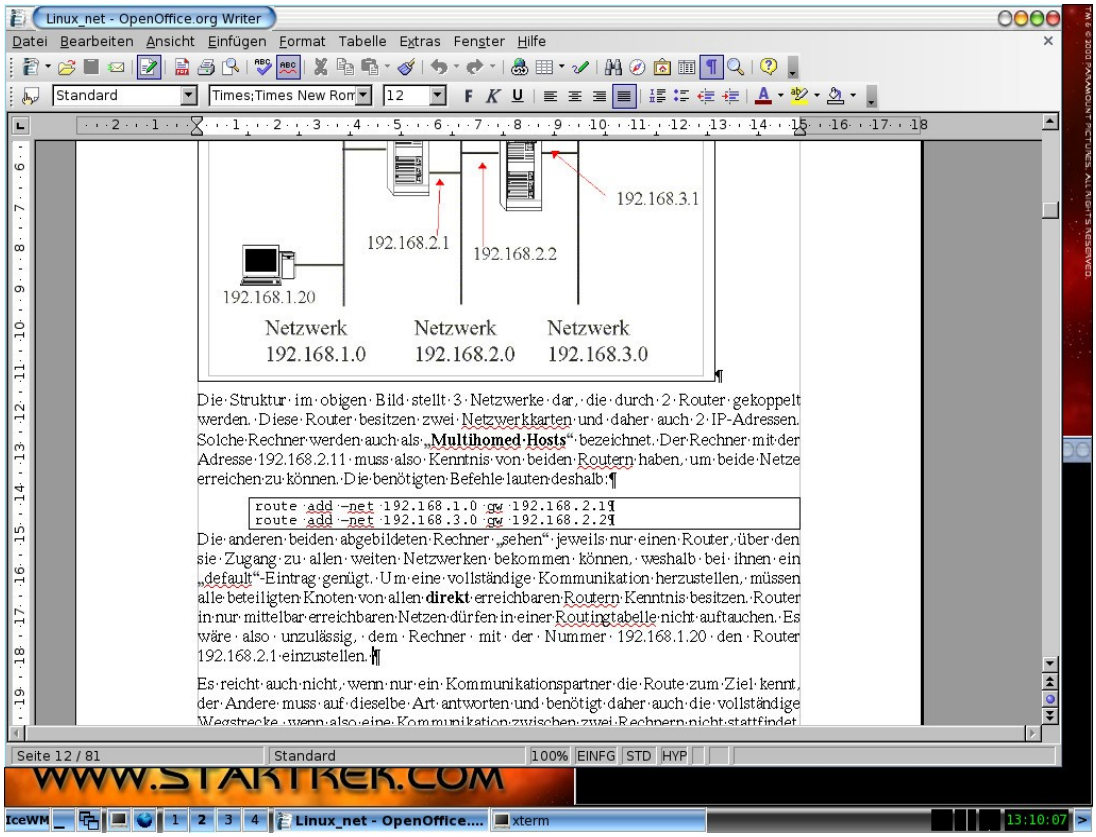

Abbildung 28: OpenOffice 2.0

18.2 Datenbanken

Viele der “großen” Datenbankhersteller haben sich bereit gefunden, auch Linux zu unterstützen. Daher sind große, komplexe Systeme für den Einsatz unter Linux bereit. Diese Datenbanken haben üblicherweise gar nichts mit grafischen Oberflächen zu tun, sondern laufen als Hintergrundprozesse im System. Soll das Linux-System als Datenbankserver eingesetzt werden, ist es lohnenswert, X-Windows gar nicht erst zu starten, da X recht viel Speicher braucht, und für einen Server, der ohnehin kein Arbeitsplatz sein sollte, überflüssig ist.

Natürlich werden unter Linux vorzugsweise Datenbanken eingesetzt, die unter einer OpenSource-Lizenz stehen. Zu den bekanntesten gehören:

- PostgreSQL
- MySQL

Diese Datenabken stehen den kommerziellen Produkten kaum nach. Sogar zum Support gibt es Unternehmen, die sich auf diese Systeme spezielisiert haben Aus dem Kommerziellen Bereich stehen aber ebenfalls alle namhaften Hersteller auf der Linux-Seite:

- Oracle Workgroup Server (Oracle)
- Informix SE/Dynamic Server (IBM/Informix)
- DB/2 (IBM)
- Adabas (SAG)
- Interbase (Borland)

- Visual LNX++ (LNX)

All diese Datenbanken haben den großen Vorteil, dass auf Grund der offen gelegten Linux-Sourcen ihre Umgebung manuell angepasst werden kann, was auf anderen Systemen nicht möglich ist.

18.3 Kaufmännische Software

Mittlerweile hat sich auch im kaufmännischen Bereich herumgesprochen, dass mit Linux ein System bereitsteht, dass kostengünstig und leistungsfähig ist und zudem noch eine eingebaute Multiuser-Fähigkeit besitzt. Die bekanntesten Produkte sind hier:

- **Lx-Office:** Eine freie kaufmännische Software, die vollständig Web-orientiert aufgebaut und daher weitestgehend Plattformunabhängig ist. (http://www.lx-office.org/index.php)
- **Absolute Busy**: Komplettlösung für FiBu, KLR, Lohnbuchhaltung etc.

18.4 Spezielle Pakete

Da Linux ja nun vorrangig aus dem universitären Bereich stammt, sind wissenschaftliche Programme besonders häufig dort vorzufinden. Einige Beispiele:

- **Mathematica**: Mathematikpaket, zur Lösung von Problemen in numerischer, grafischer und symbolischer Form
- **Maple V**: Ein Mathematikprogramm, in dem sich numerische Berechnungen der höheren Mathematik durchführen und visualisieren lassen.
- **ARCAD X11**: Ein 3D-CAAD-Programm für Architektur, Raumplanung und Bauingenieurwesen.

Durch X bietet sich eine komfortable Oberfläche, die den Lernaufwand für die Bedienung des neuen Systems minimiert. Linux kann also als ein sehr gutes Desktop-System eingesetzt werden, das die Stabilität eines Linux-Systems mit dem Bedienkomfort eines Windows-Systems kombiniert, vorausgesetzt, man hat die Hürde bis zur korrekten Installation des X-Servers überwunden.

19 Befehlsübersicht

Es gibt Tausende von Befehlen unter Linux. Um wenigstens die häufigsten griffbereit zu haben, soll diese Übersicht eine Hilfestellung geben. Die Beschreibungen sind keinesfalls vollständig, wenn alle Optionen zu einem Befehl benötigt werden, bietet sich der Aufruf der ManPages an. (s. Kap. 12 Hilfe)

19.1 Kommandos zur Dateiausgabe

Befehl	Aufgabe	Aufruf
`cat`	Ausgabe oder Konkatenation von Dateien	cat <Datei> [<Datei2> ...] Beispiel: cat meldung cat meldung meldung2
`less`	Seitenweise Ausgabe eines Textes. "less" ist ein Filter, der die Standardeingabe als Input und die Standardausgabe als Output benutzt.	less <Datei> [<Datei2> ...] Beispiel: cat brief |less less <brief
`lpr`	Ausgabe von Dateien über den Printer-Spooler (= Druck von Dateien). In der Regel kann "lpr" auch als Filter eingesetzt werden.	lpr <Datei> [<Datei2> ...] Beispiel: lpr drucktext cat drucktext | lpr
`lpq`	Ausgabe der aktuellen Druckerwarteschlange in der Form Auftragsnummer, Benutzername, Speicherplatz, Datum	lpq Beispiel: lpq
`lprm`	Abbruch eines Druck-Auftrages mit der angegebenen Auftragsnummer (s. lpq)	lprm <Auftragsnummer> Beispiel: lprm 12345

19.2 Informationen über Dateien

Befehl	Aufgabe	Aufruf
`df`	Ausgabe der freien Speicherplatzes Datenträgers	df [<Gerätename>] Beispiel: df /dev/hda2
`du`	Anzahl der durch einen Dateibaum belegten Blöcke	du <Verzeichnis> Beispiel: du /usr/spool/mail du .
`find`	Suche nach Dateien mit vorgegebener Charakte-	find <Verzeichnis> <Ausdruck>

Befehl	Aufgabe	Aufruf
	ristik	Beispiel: find /u/maier -name hilfe –print find . –exec chown meier {} \;
`ls`	Inhaltsverzeichnis eines Verzeichnisses	ls [<Optionen>] [<Dateien>|<Verzeichnis>] Beispiel: ls ls -al /u/maier
`pwd`	Zugriffspfad des aktuellen Verzeichnis	pwd Beispiel: pwd

19.3 Dateisystemorientierte Kommandos

Befehl	Aufgabe	Aufruf
`cd`	Positionieren auf ein Unterverzeichnis; "cd" ohne Parameter positioniert auf das HOME-Verzeichnis	cd [<Verzeichnis>] Beispiel: cd cd .. cd /usr/games
`mkdir`	Erzeugen eines neuen Unterverzeichnisses	mkdir <Verzeichnisname> Beispiel: mkdir texte
`rmdir`	Löschen eines Unterverzeichnisses (nur möglich, wenn in dem Unterverzeichnis keine Dateien sind außer ”.” und ”..”)	rmdir <Verzeichnisname> Beispiel: rmdir texte

19.4 Modifikation von Dateien

Befehl	Aufgabe	Aufruf
`chmod`	Ändern der Zugriffsrechte	chmod <Modus> <Dateiname> Beispiel: chmod go-w sicher.dat chmod 640 ganzsicher.dat chmod 400 paranoia.dat
`cp`	Kopieren einer (oder mehrerer) Datei(en)	cp <Datei> <Neue_Datei> Beispiel: cp altertext.txt neuertext.txt

Befehl	Aufgabe	Aufruf
ln	Vergabe eines weiteren Namens für dieselbe Datei; der Inhalt wird im Gegensatz zu "cp" nicht verdoppelt!	ln <Datei> <Neuer_Name> Beispiel: ln readme liesmich ln –s /u/maier/daten /u/schulz/daten
mv	Ändern eines Dateinamens	mv <Alter_Name> <Neuer_Name> Beispiel: mv programm programm2
rm	Löschen einer Datei	rm <Datei> [<Datei2> ...] Beispiel: rm alte.daten rm alte.daten wichtige.daten

19.5 Suchen, Sortieren und Vergleichen

Befehl	Aufgabe	Aufruf
cmp	Vergleicht 2 Dateien	cmp <Datei1> <Datei2> Beispiel: cmp gutes_prog besseres_prog
comm	Sucht in 2 Dateien gemeinsame Zeilen und gibt in drei Spalten aus, welche Zeilen nur in der ersten, welche nur in der zweiten und welche Zeilen in beiden Dateien vorkommen.	comm <Datei1> <Datei2> Beispiel: comm gutes_prog besseres_prog
diff	Ermittelt Unterschiede zweier Dateien und gibt sie als "sed"-Kommandos aus	diff <Datei1> <Datei2> Beispiel: diff gutes_prog besseres_progr
grep	Sucht Textmuster in Dateien	grep [<Optionen>] <Textmuster> <Datei> Beispiel: grep "maier" /etc/passwd
join	Vereinigt 2 Dateien	join <Datei1> <Datei2> Beispiel: join teil1 teil2
sort	Sortiert und mischt Textdateien, kann auch als Filter benutzt werden.	sort <datei> Beispiel: sort adressen_liste cat adressen | sort >sortierte_liste
wc	Zählt Buchstaben, Worte und Zeilen einer Datei; Option -c zählt nach Zei-	wc [-cwl] <datei> Beispiel:

Befehl	Aufgabe	Aufruf
	chen, -w nach Worten und -l nach Zeilen	wc -l adressen
echo	Textausgabe von angegebenen Argumenten, wobei auch wildcards (*,?) benutzt werden können	echo <argumente> Beispiel: echo "Hallo, Welt" echo adr*

19.6 Programmerstellung und Behandlung

Befehl	Aufgabe	Aufruf
ld	Linker, Binder, Normalerweise vom Compiler implizit aufgerufen	
make	Atomatische Neugenerierung eines Programms us mehreren Moduln, benötigt eine Datei mit em Namen "makefile"	make [<datei>] Beispiel: make make bzlilo
Sh	Starten einer Untershell, ggf. um ein Shell kript abzuarbeiten	sh [<datei>] Beispiel: sh sh generiere_programm

19.7 Compiler (Programmiersprachen)

Befehl	Aufgabe	Aufruf
cc	C-Compiler	cc <datei> [-o<dauei>] Beispiel: cc hallo.c cc hallo.c –o hallo
ppc386	Pascal-Compiler	ppc386 [<option>] <datei> Beispiel: ppc386 -So hallo.pas

19.8 Administrative Befehle

Befehl	Aufgabe	Aufruf
Passwd	Eigenes Passwort ändern, fremde Passwörter darf nur root ändern.	passwd [<user>] Beispiel: passwd passwd meier

19.9 Vergleich DOS/Linux-Befehle

DOS-Befehl	Linux-Befehl
attrib	chmod
copy / xcopy	cp
del	rm
dir	ls
echo	echo
edit	vi / emacs / pico ...
fdisk	fdisk
find	find
md	mkdir
mem	ps / top
more	more / less
print	lpr
ren	mv
scandisk / chkdsk	fsck / e2fsck
set	set
sort	sort
type	cat

19.10Die „M“-Tools

Befehl	Bedeutung
mdir <Laufwerk> mdir a:	Verzeichnis des angegebenen DOS-Laufwerks angeben
mcopy <Quelle> <Ziel> mcopy /etc/smb.conf a:	Datei vom Linux- ins DOS-System kopieren oder umgekehrt
mdel <Datei> mdel a:test.txt	Datei im DOS-Laufwerk löschen
mmd <Verzeichnis> mmd ablage	Verzeichnis auf dem DOS-Laufwerk anlegen
mrd <Verzeichnis> mrd ablage	(leeres) Verzeichnis vom DOS-Laufwerk löschen

20 Wichtige Verzeichnisse und Dateien

20.1 Verzeichnisse

Verzeichnis	Bedeutung
/boot	Kernel
/bin /usr/bin	Ausführbare Dateien (Kommandos)
/sbin /usr/sbin	Administrative Kommandos
/etc/init.d /etc/rc.d	Startup-Skripten (S.u.S.E. und Red Hat)
/etc	Konfigurationsdateien
/lib	Programm-Biliotheken (libraries)
/usr/X11R6	Verzeichnisse des X-Window-Systems
/usr/doc	Dokumentation des Systems
/usr/src	Qualldateien (z.B. Linux-Kernel)
/var/log	Log-Dateien

20.2 Dateien

Datei	Bedeutung
/boot/vmlinuz	Aktuell installierter Kernel
/etc/inittab	Startlevel-Kontrolldatei
/etc/profile	Global gültiges Logon-Skript
/etc/passwd	User-Verwaltungsdatei
/etc/shadow	Passwort-Datei zu /etc/passwd
/etc/group	Gruppen-Verwaltungsdatei
/etc/conf.modules	Konfiguration der Kernel-Module
/etc/isapnp.conf	Konfiguration der PnP-Geräte
/etc/rc.config	Zentrale Konfigurationsdatei (**nur** Red Hat + S.u.S.E.)
/var/log/boot.msg	Log-Datei des Kernel-Bootvorganges