OpenVMS
Einführung*

Raven Information Technologies
Bernd Ulmann

19. Juli 1999

Inhaltsverzeichnis

1 Geschichtlicher Hintergrund

Die Abkürzung VMS steht für *V*irtual *M*emory *S*ystem – Hauptgrund für die Entwicklung von VMS war die Überwindung der Einschränkungen hinsichtlich innerhalb eines Programms verwenndbaren Speichers. Gegen Ende der siebziger Jahre wurde klar, daß ein zukunftsicheres Betriebssystem auf einem virtuellen Speicherkonzept aufsetzen muß. Grundlegend hierbei ist die Aufteilung des Speichers in sogenannte *Seiten* oder *Pages*. Der einem Programm zugeteilte virtuelle Speicher besteht

aus einer Reihe solcher Seiten, die sich jedoch nicht alle zu einem Zeitpunkt im physikalischen Hauptspeicher der Maschine befinden müssen. Vielmehr besteht die Möglichkeit, Speicherseiten, die längere Zeit nicht benötigt wurden, auf einen langsameren Sekundärspeicher auszulagern, um so freien Hauptspeicher für weitere Anfragen zur Verfügung stellen zu können. Auf diese Art und Weise verfügen virtuelle Memory-Systeme über wesentlich mehr Speicher, als dem physikalischen Speicher der aktuellen Maschine entspricht.

In den frühen neunziger Jahren wurde `VMS` in `OpenVMS` umbenannt, um die offene Systemstruktur des Betriebssystemes auch sprachlich auszuweisen. Trotz aller Erweiterungen und Umstrukturierungen, die `VMS` in den mittlerweile über 20 Jahren seiner Geschichte erfahren hat, konnten grundlegende Teile, wie beispielsweise die Benutzerschnittstelle, konsistent gehalten werden, so daß der Umstieg von einer älteren auf eine neuere `VMS`-Version in den meisten Fällen ohne Schwierigkeiten – auch auf Applikationsseite – vollzogen werden kann.

Direkte Vorläufer von `OpenVMS` sind die `DIGITAL`-Betriebssysteme `RSX-11M(+)` beziehungsweise `RT-11` – beide wurden für die Maschinen der `PDP-11`-Baureihe entwickelt, die ihrerseits direkter Vorläufer der `VAX`-Architektur ist.

2 An- und Abmelden

Wie bei (fast) allen Multiuser-Betriebssystemen ist es auch bei `OpenVMS` unerläßlich, sich vor der Arbeit mit dem System anzumelden. Jedem eingetragenen User eines solchen Systems sind hierbei ein *Username* und ein *Paßwort* zugeordnet[1].

Nachdem eine Verbindung zum gewünschten Zielrechner aufgebaut wurde – denkbar sind hierbei beispielsweise

- eine direkte serielle Verbindung zum betreffenden `OpenVMS`-System,
- Zugang über einen sogenannten *Terminalserver*,
- Zugang unter Verwendung eines Netzwerkes, wobei in den meisten Fällen die Protokolle
  - `TCP/IP` beziehungsweise
  - `DECnet`

  zum Einsatz kommen

– fordert das System durch Ausgabe von

```
Username:
```

zur Eingabe des Benutzernamens auf. Im Anschluß hieran wird das Paßwort durch

```
Password:
```

angefordert. Hierbei gilt standardmäßig die Einstellung, daß das Paßwort bis zu dreimal angefordert wird, sofern es nicht korrekt eingegeben wurde. Nach Überschreiten dieses Limits wird die Verbindung automatisch getrennt – treten solche potentiellen Einbruchsversuche mehrfach auf, wird der betreffende Zugangskanal für längere Zeit gesperrt[2].

[1] In speziellen Fällen ist auch die Verwendung von *zwei* Paßwörtern möglich, um eine zusätzliche Sicherheitsstufe zu gewährleisten.

[2] Ein solcher Einbruchsversuch wird als *intrusion* bezeichnet.

Nach erfolgter Anmeldung beim System wartet OpenVMS auf Kommandos, was durch das sogenannte *Prompt* angezeigt wird. Wann immer ein solches Prompt erscheint, ist das System bereit für die Eingabe des nächsten Befehls. Das Standardprompt des OpenVMS-Kommandointerpreters, DCL (siehe Abschnitt 3), ist das Dollarsymbol $.

Außer dem genannten Interpreter verfügen die meisten kommandogesteuerten Subsysteme unter OpenVMS über ein eigenes Prompt-Symbol, anhand dessen man jederzeit feststellen kann, in welchem Programm man sich augenblicklich befindet. So meldet sich das MAIL-System beispielsweise mit

```
MAIL>
```

Ist die Arbeit an dem System abgeschlossen, muß sich der betreffende Benutzer abmelden (*ausloggen*), um hiermit

- zum einen die Terminalverbindung wieder freizugeben und
- zum anderen sicherzustellen, daß niemand die Möglichkeit besitzt, an Stelle eines anderen Users, dessen Verbindung zum OpenVMS-System nicht durch ein Ausloggen beendet wurde, Aktionen auf dem Rechner im Namen eines anderen durchzuführen, zu denen er nicht berechtigt ist.

Das Ausloggen geschieht mit Hilfe des Kommandos

```
$ LOGOUT
```

– hierzu existiert die Variante

```
$ LOGOUT/FULL
```

, die nach Sitzungsende detaillierte Informationen über die Menge in Anspruch genommener Ressourcen ausgibt.

3 DCL

3.1 Grundlagen

Die Abkürzung DCL steht für *D*igital *C*ommand *L*anguage, die Skriptsprache der OpenVMS-Systeme. Der DCL-Kommandointerpreter unterscheidet sich in seiner Funktion und Bedienung zum Teil grundlegend von anderen bekannten Systemen, wie beispielsweise den verschiedenen UNIX-Shells, wie der C- oder der Korn-Shell.

Zentraler Gedanke von DCL ist die Konsistenz des Gesamtsystems, so wurde darauf geachtet, daß Steuerargumente (sogenannte *Qualifizierer*) verschiedener Kommandos dieselbe Gestalt besitzen, wenn durch sie gleiche oder ähnliche Aktionen ausgelöst werden. So verfügen zum Beispiel fast alle DCL-Kommandos über einen Qualifizierer /ALL, der die Ausgabe ausführlicherer Informationen veranlaßt. Der Fall, daß ein solcher Qualifizierer bei einem Kommando /ALL und bei einem anderen Kommando – bei gleicher Funktion – anders heißt, tritt nicht auf[3].

Eine weitere Eigenschaft von DCL ist die Tatsache, daß alle Kommandos in dieser Sprache an die Englische Sprache angelehnt wurden, so daß man sich mit Hilfe des HELP-Systems (siehe Abschnitt 3.2) und kurzem Raten in fast allen Fällen selbst helfen kann – auch, wenn nicht bekannt ist, wie das gesuchte Kommando heißt. Muß

[3]Ein Feature, das man bei UNIX-Shells vergeblich sucht.

man beispielsweise unter UNIX sehr viel wissen, um die Übertragungsgeschwindigkeit einer seriellen Schnittstelle einzustellen (die Idee, daß sich das zugehörige Kommando hinter stty – für set teletype – verbirgt, ist sicherlich nicht intuitiv). Auf der anderen Seite ist das Vorgehen unter DCL sehr direkt: Zunächst die Überlegung, was letztlich getan werden soll – in diesem Falle soll eine Eigenschaft verändert, gesetzt werden, d.h. das gesuchte Kommando ist sicherlich ein SET-Kommando. Was soll modifiziert werden? Ein Terminal. Also führt SET TERMINAL vermutlich in die richtige Richtung. An dieser Stelle (oder auch schon vorher) könnte nun das HELP-System eingesetzt werden, um das gesuchte Kommando gänzlich zusammenzustellen. Da man die Geschwindigkeit einer seriellen Schnittstelle einstellen möchte, ergibt sich das gesuchte DCL-Kommando letztlich zu

```
$ SET TERMINAL/SPEED=9600/PERMANENT
```

Der Qualifizierer /PERMANENT ist zugegebenermaßen nicht naheliegend und bedarf wohl des Hilfesystems. Seine Verwendung liegt in den speziellen Eigenschaften serieller Schnittstellen unter OpenVMS begründet.

Letztlich muß bemerkt werden, daß DCL-Kommandos streng hierarchisch aufgebaut sind. An erster Stelle steht in jedem Fall die Art der auszuführenden Aktion, beispielsweise SET, SHOW, PRINT, usf. Anschließend finden sich Argumente für das gewünschte Kommando, beispielsweise Dateinamen (siehe Abschnitt 4) für ein PRINT-Kommando, sowie Qualifizierer, die eine weitere Sepzifizierung der geforderten Aktion gestatten. So zeigt beispielsweise das Kommando

```
$ SHOW QUEUE SYS$PRINT
```

allgemeine Informationen über die zentrale Druckerwarteschlange SYS$PRINT (siehe Abschnitt 9) an, während

```
$ SHOW QUEUE SYS$PRINT/FULL
```

ausführlichere Informationen über diese Queue anzeigt.

Generell gilt, daß DCL-Kommandos soweit abgekürzt werden können, solange sie eindeutig bleiben. Der obige Befehl läst sich auch als

```
$ SH QUE SYS$PRINT/FU
```

schreiben. Im weiteren Verlauf dieser Kurzeinführung wird auf diese Möglichkeit zugunsten der besseren Merkbarkeit voll ausgeschriebener Kommandos jedoch verzichtet.

3.2 Das Hilfesystem

OpenVMS verfügt über ein äußerst leistungsfähiges Hilfesystem, das durch Eingabe des Kommandos

```
$ HELP
```

gestartet werden kann.

Analog zu den Kommandos des DCL-Interpreters ist auch das Hilfesystem hierarchisch aufgebaut. Auf oberster Ebene finden sich alle Kommandos, zu denen Hilfe vorhanden ist. In dieser Liste finden sich beispielsweise auch die Befehle SET und SHOW. Werden nun weitere Informationen beispielsweise zu SHOW gesucht, kann die folgende Suche durch Eingabe von

```
HELP> SHOW
```

auf diesen einen Befehl eingeschränkt werden. In der nun folgenden Liste finden sich alle möglichen Argumente des gewünschten Kommandos. Hier existiert im genannten Beispiel unter anderem das Argument `USERS`. Durch Eingabe von

```
SHOW Subtopic? USERS
```

wird die Suche auf diesen Teilaspekt des `SHOW`-Kommandos eingegrenzt. `SHOW USERS` seinerseits verfügt nun über eine ganze Reihe möglicher Argumente, wie zum Beispiel `/INTERACTIVE`. Nach Eingabe von

```
SHOW USERS Subtopic? /INTERACTIVE
```

wird nun die Dokumentation zu diesem Qualifizierer ausgegeben. Das `DCL`-Kommando

```
$ SHOW USERS/INTERACTIVE
```

gibt eine Liste aller im Moment interactiv auf dem betreffenden System eingeloggten Benutzer aus[4].

Kann ein `DCL`-Kommando nicht in einer einzigen Zeile eingegeben werden, da es zuviele Parameter benötigt, kann eine unfertige Zeile mit einem Minuszeichen beendet werden, wodurch der `DCL`-Kommandointerpreter dazu veranlaßt wird, die folgende Zeile als Fortsetzung dieses Kommandos aufzufassen (für ein Beispiel siehe Abschnitt 8.3).

3.3 Systemmeldungen

Meldungen, die von `OpenVMS` generiert werden, können stets dem sie erzeugenden Subsystem (z.B. der `DCL`-Kommandointerpreter, das `MAIL`-Subsystem, etc.) zugeordnet werden. Darüberhinaus besitzt jede Meldung einen zugeordneten Schweregrad, anhand dessen sie schnell klassifiziert werden kann. Zuletzt beinhaltet eine solche Meldung noch den eigentlichen Meldungstyp und -text, wie folgendes Beispiel zeigt, in dem ein unbekanntes Kommando, `ASDF`, an `DCL` zur Ausführung übergeben wird:

```
$ ASDF
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 /ASDF/
$
```

Die eigentliche Fehlermeldung beginnt mit `%DCL` – dieser Abschnitt zeigt das erzeugende Subsystem an, in diesem Fall der `DCL`-Kommandointerpreter. Gefolgt wird diese Angabe vom Schweregrad der Meldung, in diesem Fall `-W-` – es handelt sich um eine Warning. Der letzte Teil der Meldungspräambel ist `IVVERB` – die Abkürzung für `invalid verb`. Anschließend findet sich noch eine Klartextbeschreibung des Fehlers, um seine Ursache eingrenzen zu können. In der folgenden Zeile findet sich das fehlerverursachende Statement, `ASDF`.

Allgemein kennt `OpenVMS` die folgenden Schweregrade bei Meldungen:

`-I-`: Dieses Kürzel kennzeichnet eine Meldung vom Typ *informational* – sie stellt keine Fehlermeldung im eigentlichen Sinne dar sondern informiert lediglich über ausgeführte Aktionen, etc.

[4] Neben interaktiven Benutzern existieren auch Subprozesse und Batch-Jobs.

`-W-`: Eine solche *Warning* ist etwas schwerwiegender als eine Meldung vom Typ `-I-`, stellt aber auch noch keinen Fehler dar, der die Ausführung eines Kommandos oder Programmes unmöglich macht. Dennoch sollte solchen Meldungen nachgegangen werden.

`-E-`: Dieser Meldungstyp kennzeichnet einen *Error* – das Kommando oder Programm konnte nicht korrekt ausgeführt werden.

`-F-`: Eine Steigerung des einfachen Errors ist die Meldung vom Typ *Fatal* – es handelt sich hierbei um einen schwerwiegenden Fehler, der auf jeden Fall untersucht werden sollte!

`-S-`: Mit diesem Kürzel werden allgemeine Systemmeldungen gekennzeichnet.

3.3.1 Beispiele

```
$ SET TERM/SPEED=9600/NOPERMANENT
%DCL-W-NOTNEG, qualifier or keyword not negatable - remove "NO" or omit
 \NOPERMANENT\
$
```

In diesem Beispiel wurde der Versuch unternommen, die Übertragungsgeschwindigkeit einer seriellen Schnittstelle zu modifizieren, ohne darauf zu achten, daß dies nur unter Angabe des Qalifizierers `PERMANENT` möglich ist. Entsprechend generiert der Kommandointerpreter eine Warning, in der darauf hingewiesen wird, daß der eingesetzte Qualifizierer `NOPERMANENT` nicht negierbar ist. Als Abhilfe wird vorgeschlagen, auf das `NO` zu verzichten, oder den gesamten Qualifizierer `NOPERMANENT` wegzulassen.

```
$ MAIL

MAIL> SELECT WASTEBASKET
%MAIL-E-NOTEXIST, folder WASTEBASKET does not exist

MAIL>
```

Hier wurde versucht, innerhalb des `MAIL`-Utilities den Ordner `WASTEBASKET` mit Hilfe des `SELECT`-Kommandos zu öffnen, was fehlschlug, da dieser Ordner nicht existierte.

3.4 Symbole

Wie viele andere Kommandointerpreter auch, bietet `DCL` die Möglichkeit, Variablen zu definieren und zu verwenden. Solche Mechanismen sind nützlich, um beispielsweise eigene Kommandos zu definieren, o.ä.

3.4.1 Definieren von Symbolen

Das folgende Beispiel zeigt, wie ein Kommando `LS` definiert werden kann, bei dessen Aufruf automatisch der Befehl `DIR/SIZ=ALL` zur Ausführung kommt:

```
$ LS :== DIR/SIZ=ALL
```

Durch Verwendung der Zuweisung `:==` wird eine Variable `LS` definiert, deren Inhalt gleich `DIR/SIZ=ALL` ist. Die Verwendung der doppelten Gleichheitszeichen hat zur Folge, daß diese Variable in der globalen Symboltabelle abgelegt wird – ein einfaches Gleichheitszeichen würde die Variable nur in der lokalen Symboltabelle ablegen.

Sobald `DCL` auf ein Symbol an Stelle eines Kommandos stößt, wird es ausgewertet und sein Inhalt als Kommando ausgeführt, so daß die Ausführung des Kommandos `LS` nach obiger Definition in ein `DIR/SIZ=ALL` umgesetzt würde.

Der Doppelpunkt vor den zuweisenden Gleichheitszeichen bedeutet, daß auf der rechten Seite des betreffenden Ausdruckes eine Zeichenkette, ein *String* steht, der vollständig in das Symbol übernommen wird. Zu beachten ist hierbei lediglich, daß diese Zeichenkette mit einem Buchstaben, einem Underscore oder einem Dollarzeichen beginnen muß. Ein Einschließen des String in doppelte Anführungszeichen ist nicht notwendig. Darüberhinaus wird die definierende Zeichenkette vor der eigentlichen Zuweisung in Großbuchstaben konvertiert. Auch mehrfache oder überflüssige Leerzeichen werden in diesem Schritt automatisch entfernt.

Sind diese Seiteneffekte der genannten Zuweisung aus bestimmten Gründen nicht erwünscht, kann der Doppelpunkt weggelassen werden, so daß obiges Beispiel folgende Gestalt hätte:

```
LS == "DIR/SIZ=ALL"
```

Allgemein kann ein Symbol nur erkannt werden, wenn es genau so geschrieben wird, wie in seiner Definition. Da dies den Möglichkeiten von `DCL` im Hinblick auf die Möglichkeiten zur Abkürzung von Kommandos nicht gerecht wird, existiert ein Mechanismus, der es gestattet, Symbole zu definieren, die auch in abgekürzter Form erkannt und aufgelöst werden können.

Soll beispielsweise ein Symbol `WERISTDA` definiert werden, mit dessen Hilfe sich eine Liste aller momentan interaktiv eingeloggten Benutzer anzeigen läßt, könnte dies wie folgt geschehen:

```
$ WERISTDA :== SHOW USERS/INTERACTIVE/FULL
```

Diese Definition greift jedoch nur, wenn das so definierte Kommando stets als `WERISTDA` aufgerufen wird. Soll es bis aus mindestens `WER` abgekürzt werden können, so daß `WER`, `WERI`, `WERIS`, usf. das genannte Kommando ausführen, ist die Definition wie folgt abzuändern:

```
$ WER*ISTDA :== SHOW USERS/INTERACTIVE/FULL
```

Der Stern in obiger Definition bezeichnet die Stelle, bis zu der das Symbol mindestens ausgeschrieben werden muß, um erfolgreich erkannt zu werden.

3.4.2 Anzeigen von Symbolen

Notwendig ist auch ein Mechanismus, mit dessen Hilfe bereits definierte Symbole mit ihren zugeordneten Werten angezeigt werden können. Das betreffende Kommando hat die allgemeine Gestalt

```
$ SHOW SYMBOL LS
```

, falls der Wert des oben deklarierten Symbols `LS` überprüft werden soll. Seine Ausgabe ist in diesem Fall:

```
LS == "DIR/SIZ=ALL"
```

Soll eine Liste aller definierten Symbole erzeugt werden, kann dies durch

```
$ SHOW SYMBOL/GLOBAL/ALL
```

geschehen. Durch Angabe des Qualifizierers `/GLOBAL` wird die globale Symboltabelle als zu durchsuchend spezifiziert, während `/ALL` zur Folge hat, daß *alle* dort eingetragenen Symbole ausgegeben werden.

3.4.3 Foreign-Commands

Ein großer Unterschied zwischen `DCL` und anderen Kommandointerpretern, wie beispielsweise den meisten `UNIX`-Shells, besteht in dem Mechanismus, mit dessen Hilfe Parameter an Programme übergeben werden können, die über die Kommandozeile aufgerufen werden. Während unter `UNIX` nach dem Programmnamen einfach eine Liste der gewünschten Parameter angegeben werden kann, ist dies unter `DCL` nicht direkt möglich, da Programme in den meisten Fällen (mit Ausnahme von Kommandoprozeduren, siehe Abschnitt 7, und installierten Files, siehe `openVMS`-Dokumentation zum `INSTALL`-Utility) nur mit Hilfe des Kommandos `RUN` gestartet werden können.

Soll beispielsweise ein Programm namens `PRIM.EXE`, das alle Primzahlen bis zu einer bestimmten, anzugebenden Zahl berechnet und ausgibt, gestartet werden, führt der Ansatz `RUN PRIM 100` *nicht* dazu, daß das Programm `PRIM.EXE` mit dem Übergabeparameter 100 ausgeführt wird.

In solchen Fällen muß eine Sonderform der oben besprochenen Symbole eingesetzt werden, um das `RUN`-Kommando zu vermeiden. Solchermaßen definierte Kommandos werden auch als *foreign commands* bezeichnet. Sie unterscheiden sich von normalen Symbolen durch ein vorangestelltes Dollarzeichen, das letztlich dem `RUN`-Statement des eigentlichen Programmaufrufes entspricht, so daß für die Ausführung des hypothetischen Programmes `PRIM.EXE` folgende Definition notwendig wäre:

```
$ PRIM :== $PRIM
```

Wird nun `PRIM 100` eingegeben, startet `DCL` automatisch das Programm `PRIM.EXE` und übergibt diesem den Wert 100, der nun durch das Programm ausgewertet werden kann.

3.5 Lexical-Functions

`DCL` bietet – analog zu höheren Programmiersprachen – Funktionen, mit deren Hilfe Angaben über eine Reihe von Systemparametern gewonnen oder modifiziert werden können. Diese Funktionen werden unter `OpenVMS` gewöhnlich als *Lexical Functions* bezeichnet und besitzen eine Syntax, die stark an mathematische Funktionen angelehnt ist.

Grundlegende Eigenschaft einer Funktion ist, daß sie einen Wert zurückliefert. Weiterhin können den meisten Funktionen Werte übergeben werden, auf denen sie operieren – solche Übergabeparameter werden stets in runden Klammern eingeschlossen notiert. Zu beachten ist hierbei, daß auch bei Fehlen solcher Parameter die Klammern (leer) angegeben werden müssen!

Ein Beispiel für eine Funktion, die keine Parameter benötigt und einen Wert zurückliefert, ist `F$TIME` – Rückgabewert sind das aktuelle Systemdatum und die Systemzeit. Der Rückgabewert einer solchen Funktion kann entweder in ein Symbol geschrieben oder direkt mit Hilfe eines `WRITE`-Statements ausgegeben werden:

```
$ ZEIT = F$TIME ()
$ SHOW SYMBOL ZEIT
  ZEIT = "12-JUN-1999 23:18:54.84"
$ WRITE SYS$OUTPUT F$TIME ()
12-JUN-1999 23:18:59.76
```

Im ersten Beispiel wurde der Rückgabewert der Funktion `F$TIME` (man beachte die Konstruktion `()`, wodurch die Funktion als solche überhaupt kenntlich gemacht wird) in ein Symbol `ZEIT` geschrieben, dessen Wert anschließend mit `SHOW SYMBOL` angezeigt wird. Die zweite Variante umgeht diesen Zwischenschritt, indem der Rückgabewert von `F$TIME` direkt mit `WRITE SYS$OUTPUT` auf die Standardausgabe des Systems geschrieben wird.

Mehr Informationen zu den verfügbaren Lexical-Functions findet sich unter dem Stichwort `LEXICALS` im `OpenVMS` Subsystem `HELP`.

4 Das Filesystem

4.1 Grundlagen

Das `OpenVMS`-Filesystem weist einige Besonderheiten gegenüber einer Reihe anderer Filesysteme auf, die im folgenden dargestellt werden sollen. Zum einen unterliegen Filenamen hinsichtlich ihres Formates sehr viel strengeren Einschränkungen, als beispielsweise unter den meisten `UNIX`-Derivaten oder auch `Windows-NT`.

Generell gilt: Ein Filename setzt sich aus drei Teilen zusammen:

1. Dem bezeichnenden Namen der Datei,
2. der Extension, sowie einer
3. Versionsnummer.

Ein typischer Dateiname unter `OpenVMS` wäre also

```
LOGIN.COM;1
```

– die Datei heißt `LOGIN`, hat die Extension `COM`, es handelt sich also um eine `DCL`-Kommandoprozedur und liegt in der Version 2 vor. Die Versionsnummer ist ein Feature, das kaum ein anderes Filesystem bietet – sie wird bei jeder Änderung einer Datei um 1 inkrementiert, wobei im allgemeinen die alte Version der jeweiligen Datei, d.h. ihr Stand *vor* Änderung, erhalten bleibt.

Wird beispielsweise diese Datei `LOGIN.COM;2` editiert und modifiziert (siehe Abschnitt 5) – beispielsweise durch ein Kommando der Form

```
$ EDIT LOGIN.COM
```

–, so wird nach Beenden des Editors mit Abspeichern eine neue Datei mit um eins erhöhter Versionsnummer angelegt. (Um einen Wildwuchs an alten Versionen häufig modifizierter Dateien zu vermeiden, können mit Hilfe des Kommandos `PURGE` alte Versionen gezielt gelöscht werden – siehe `OpenVMS`-Dokumentation.)

4.2 Fileprotections

Unter einem Multiuser-Betriebssystem wie OpenVMS müssen Mechanismen vorhanden sein, um Dateien vor unberechtigtem Zugriff anderer zu schützen. Unter OpenVMS existieren vier Grundrechte, die auf Dateien angewandt werden können und im folgenden aufgelistet sind:

R: *Read*-Recht – hierdurch wird das Lesen der betreffenden Datei erlaubt.

W: Das *Write*-Recht gestattet es, schreibend auf eine Datei zuzugreifen.

E: Um eine Datei, d.h. ein Programm, ausführen zu können, muß das *Execute*-Recht vergeben sein.

D: Gelöscht werden darf eine Datei nur, falls das ihr zugeordnete *Delete*-Attribut gesetzt ist.

Die Benutzer eines OpenVMS-Systems sind grob in vier Kategorien eingeteilt, wobei pro Datei für jede dieser vier Gruppen eine beliebige Kombination der obigen vier Grundrechte eingetragen werden kann. Diese Gruppen sind:

SYSTEM: In dieser Gruppe befinden sich der Systemverwalter, sowie entsprechende Systemprozesse.

OWNER: Die Gruppe OWNER enthält nur den eigentlichen Besitzer des jeweiligen Files.

GROUP: In dieser Gruppe befinden sich alle Benutzer, die sich in derselben Gruppe wie auch der Besitzer der Datei befinden.

WORLD: WORLD repräsentiert *alle* Benutzer.

4.2.1 Anzeigen von Fileprotections

Um die Schutzrechte der Datei LOGIN.COM anzuzeigen, kann folgendes Kommando zur Anwendung kommen:

```
$ DIR/PROTECTION LOGIN.COM

Directory DISK$USER:[KURS_USER]

LOGIN.COM;2          (RWED,RWED,RE,)
```

Diese Datei darf also von der Gruppe SYSTEM gelesen, geschrieben, ausgeführt und gelöscht werden. Für ihren Besitzer gelten dieselben Rechte, während Mitglieder der Gruppe GROUP nur lesenden und ausführenden Zugriff hierauf haben. Alle anderen Benutzer des Systems, d.h. alle Mitglieder der Gruppe WORLD besitzten keinerlei Rechte auf diese Datei.

4.2.2 Setzen von Fileprotections

Sollen diese Zugriffsrechte einer Datei modifiziert werden, kommt das Kommando SET FILE/PROTECTION=... zum Einsatz. In folgendem Beispiel soll allen Mitgliedern der Gruppe WORLD lesender und ausführender Zugriff auf die Datei LOGIN.COM gewährt werden:

```
$ SET FILE/PROTECTION=W:RE LOGIN.COM
```

Um diese eben vergebenen Rechte wieder zu entfernen, kann

```
$ SET FILE/PROTECTION=W LOGIN.COM
```

ausgeführt werden.

Sollen für mehr als eine Gruppe zu einem Zeitpunkt Rechteänderungen durchgeführt werden, können alle erforderlichen Angaben in Form einer Auflistung angegeben werden. So gewährt beispielsweise

```
$ SET FILE/PROTECTION=(W:R,G:RE) TEST.DAT
```

allen Mitgliedern der Gruppe WORLD lesenden und allen Mitgliedern der Gruppe GROUP lesenden und ausführenden Zugriff auf die Datei TEST.DAT.

4.2.3 ACLs

Es gibt eine Reihe von Fällen, in denen der oben beschriebene Mechanismus zum Schutz von Dateien vor unberechtigtem Zugriff nicht ausreicht – beispielsweise ist es mit den dort beschriebenen Mitteln nicht möglich, eine Datei allen Benutzern einer Gruppe mit Leserechten zugänglich zu machen und hierbei einen bestimmten Teilnehmer dieser Gruppe hiervon auszuschließen.

Zur Behandlung solcher Fälle steht ein zusätzlicher Mechanismus in Form sogenannter *Access Control Lists* zur Verügung. Hiermit kann einer beliebigen Datei eine Liste weiterer Attribute zugeordnet werden, welche die Schutzrechte RWE und D einzelnen Usern oder einzelnen Identifiern zuordnen.

Eventuell für eine Datei eingetragene ACLs können mit DIR/ACL angezeigt werden:

```
$ DIR/ACL LOGIN.COM

Directory DKA100:[KURS_USER]

LOGIN.COM;2
         (IDENTIFIER=DARF_ALLES,ACCESS=READ+WRITE+EXSECUTE+DELETE+CONTROL)
```

In obigem Beispiel wurde der Datei LOGIN.COM eine ACL mit einem Eintrag zugeordnet, die allen Benutzern, die den Identifier (siehe hierzu die OpenVMS-Dokumentation zu AUTHORIZE) DARF_ALLES besitzen, vollständigen Zugriff auf diese Datei sichert. Diese Rechte gelten auch dann, wenn ein solcher Benutzer sich ansonsten beispielsweise in der Gruppe WORLD befindet, für die zum Beispiel nur RE-Rechte eingetragen sind. In diesem Fall erweitert der ACL-Eintrag also die Zugriffsrechte einiger Benutzer auf die betreffende Datei.

Für weitere Informationen, das Eintragen und Verwalten von ACLs betreffend, sei auf die entsprechende OpenVMS-Dokumentation verwiesen, da dieser Punkt den Rahmen der vorliegenden Einführung bei weitem sprengen würde.

4.2.4 Verzeichnisse und Pfade

Das OpenVMS-Filesystem unterscheidet sich in der Art, in der Platten und Verzeichnisse angesprochen werden, stark von anderen Systemen, wie beispielsweise UNIX, MVS, etc. Grundlegendes Element des OpenVMS-Filesystems sind die eigentlichen Festplatten – diese besitzen Namen, wie beispielsweise DKA100, etc. Generell gilt: *Der Name einer Platte innerhalb eines Pfadnamens wird durch einen Doppelpunkt abgeschlossen!*

Nach Angabe der Platte, auf der sich eine bestimmte Datei befindet, folgt die Beschreibung des Verzeichnispfades, unter dem sie zu finden ist. Verzeichnispfade

unter OpenVMS sind stets in eckige Klammern einzuschließen; so bezeichnet beispielsweise [000000] das oberste Verzeichnis einer Platte. Das Kommando

```
$ DIR DKA100:[000000]
```

listet folglich alle Dateien auf, die sich auf der obersten Verzeichnisebene der Festplatte DKA100 befinden – vorausgesetzt, der ausführende User verfügt über die notwendigen Rechte, lesend auf dieses Verzeichnis zugreifen zu dürfen.

Verzeichnisse, sogenannte *Directories*, werden unter OpenVMS stets als Dateien mit der Endung .DIR dargestellt. Zeigt also obiges DIR-Kommando unter anderem eine Datei KURS_USER.DIR;1 an, so handelt es sich hierbei um ein Verzeichnis, dessen Inhalt beispielsweise mit

```
$ DIR DKA100:[000000.KURS_USER]
```

angezeigt werden kann.

An dieser Stelle tritt nun eine besondere Notation in Erscheinung: Werden Verzeichnisse innerhalb von Verzeichnissen angesprochen, so sind diese voneinander durch einen Punkt zu trennen. Werden Verzeichnisse unter dem Wurzelverzeichnis [000000] angesprochen, kann dies auch unter Fortlassung des Wurzelverzeichnisses geschehen,

```
$ DIR DKA100:[KURS_USER]
```

ist also als analog zu obigem Beispiel anzusehen.

4.2.5 Das Default-Directory

Nach dem Einloggen in ein OpenVMS-System befindet man sich in seinem sogenannten *Login-Verzeichnis* – der Ort dieses Verzeichnisses wird bei User-Registration durch den Systemverwalter festgelegt. In diesem Moment stellt das Login-Verzeichnis gleichzeitig das *Default-Verzeichnis* dar. Alle Kommandos, die sich auf Dateien oder relative Pfadangaben beziehen, gehen vom aktuellen Defaultverzeichnis aus – das Kommando

```
$ DIR
```

listet also alle Dateien auf, die sich im momentanen Default-Verzeichnis befinden.

4.2.6 Absolute und relative Pfadangaben

Wie bei einer Reihe anderer Betriebssystem können Pfadangaben entweder absolut eingegeben werden, wie beispielsweise in

```
$ DIR DKA100:[KURS_USER.TEST]
```

oder relativ, wie in

```
$ DIR [.TEST]
```

In diesem Beispiel wird davon ausgegangen, daß das momentane Default-Verzeichnis gleich dem Login-Verzeichnis ist und den Wert `DKA100:[KURS_USER]` besitzt. Das erste `DIR`-Statement listet also mit Hilfe einer absoluten Pfadangabe, d.h. einer Pfadangabe, die unabhängig vom aktuellen Defaultverzeichnis ist, alle Dateien auf, die sich unter `DKA100:[KURS_USER.TEST]` befinden, während die zweite Variante alle Dateien auflistet, die sich in dem Verzeichnis `[.TEST]` unterhalb des aktuellen Defaultverzeichnisses befinden. Da das Defaultverzeichnis als `DKA100:[KURS_USER]` vorausgesetzt wurde, haben in diesem Falle beide Statements denselben Effekt.

Allgemein läßt sich feststellen, daß relative Pfadangaben stets mit einem Punkt nach der öffnenden eckigen Klammer beginnen, da der so spezifizierte Pfad in Abhängigkeit des momentanen Default-Verzeichnisses betrachtet wird.

4.2.7 Anzeigen und Setzen des Default-Verzeichnisses

Das aktuelle Default-Verzeichnis kann mit Hilfe des Kommandos `SHOW DEFAULT` angezeigt werden:

```
$ SHOW DEFAULT
  DKA100:[KURS_USER]
```

Gesetzt wird das Default-Verzeichnis entsprechend mit Hilfe des Kommandos `SET DEFAULT`, wobei hier zwischen relativen und absoluten Pfadangaben zu unterscheiden ist. Soll das Default-Verzeichnis auf das Verzeichnis `DKA100:[KURS_USER.TEST]` gesetzt werden, kann dies, falls das aktuelle Default-Verzeichnis `DKA100:[KURS_USER]` ist, mit einer relativen Pfadangabe, wie beispielsweise in

```
$ SET DEFAULT [.TEST]
```

geschehen, oder – unabhängig vom Wert des momentan gültigen Default-Verzeichnisses – mit

```
$ SET DEFAULT DKA100:[KURS_USER.TEST]
```

4.2.8 Rekursives Suchen in Unterverzeichnissen

Ein wichtiger Mechanismus ist das rekursive Durchsuchen aller Verzeichnisse unterhalb einer bestimmten Position (im allgemeinen dem aktuellen Verzeichnis). Während dies unter `UNIX` mit Hilfe des `find`-Befehls vollzogen wird (Vorgänger hierzu war das Kommando `ws`, *w*alk *s*ubtree, unter `MULTICS`), existiert unter `OpenVMS` eine Möglichkeit, einen solchen Mechanismus in Form einer besonderen Pfadangabe anzustoßen.

Diese Pfadangabe hat stets die Gestalt `[...]`, wobei die drei Punkte kennzeichnen, daß alle Verzeichnisse unterhalb des aktuellen Verzeichnisses angesprochen werden sollen. So können beispielsweise alle Dateien unter dem momentanen Arbeitsverzeichnis durch

```
$ DIR [...]
```

angezeigt werden. Dieser Mechanismus läßt sich auch mit einer absoluten Pfadangabe kombinieren – beispielsweise listet

```
$ DIR DKA100:[000000...]
```

alle Dateien auf, die sich unter dem obersten Verzeichnis der Platte `DKA100` befinden.

4.2.9 Logische Namen

In ähnlicher Art, in der Symbole dazu verwendet werden können, nahezu beliebige Zeichenketten in Variablen abzulegen, können sogenannte *logische Namen* oder *Logicals* dazu eingesetzt werden, Pfadnamen in Variablen abzulegen.

Logicals werden mit Hilfe des Kommandos `DEFINE` definiert und können mit `SHOW LOGICAL` angezeigt werden. Unter `OpenVMS` existiert eine Reihe vordefinierter Logicals, die auf bestimmte, wichtige Teile des Dateisystems zeigen. So besitzt jeder User ein Logical namens `SYS$LOGIN`, das auf das Verzeichnis zeigt, in dem er sich direkt nach dem Einloggen befindet (das sogenannte *Homedirectory*):

```
$ SHOW LOGICAL SYS$LOGIN
   "SYS$LOGIN" = "DKA100:[KURS_USER]" (LNM$JOB_81C07480)
```

In obigem Beispiel zeigt also das Logical `SYS$LOGIN` auf das Verzeichnis `[KURS_USER]` auf der Platte `DKA100` – anstelle einer konkreten Plattenangabe könnte an dieser Stelle wieder ein Logical stehen[5]. Auf einem anders konfigurierten System könnte obiges Kommando die Ausgabe

```
   "SYS$LOGIN" = "DISK$USER:[KURS_USER]" (LNM$JOB_81C07480)
```

nach sich ziehen, wobei `DISK$USER` seinerseits auf das konkrete Device `DKA100` zeigt:

```
$ SHOW LOGICAL DISK$USER
   "DISK$USER" = "DKA100:" (LNM$SYSTEM_TABLE)
```

Existierten bereits für einfache Symbole zwei Tabellen, eine lokale sowie eine globale, in denen die jeweiligen Symbole abgelegt werden können, verfügt das Logicalkonzept von `OpenVMS` über eine sehr viel größere Anzahl verschiedener Tabellen, in denen Logicals gehalten werden. Die jeweilige Tabelle, in der ein Logical gefunden wurde, wird beispielsweise als `(LNM$SYSTEM_TABLE)` angegeben. Hierdurch wird angezeigt, daß sich das betreffende Logical in der Systemtabelle befindet und hiermit für alle Benutzer der Anlage verwendbar ist.

Ein Beispiel für ein Logical, das nur für einen bestimmten Benutzer sichtbar ist, ist `SYS$LOGIN`. Wie oben zu sehen ist, liegt dieses Logical in einer prozeßlokalen Tabelle mit Namen `LNM$JOB_81C07480)` – diese Tabelle gehört dem Prozeß mit der Prozeß-ID (kurz `PID`) `81C07480`.

5 Der Editor `EDIT`

Die beiden Standardeditoren unter `OpenVMS` sind `EDIT` und `EVE`, wobei in den folgenden Abschnitten nur `EDIT` näher behandelt werden wird – zu `EVE` sei auf die `OpenVMS`-Dokumentation verwiesen.

5.1 Grundlagen

`EDIT` ist ein äußerst leistungsfähiger, bildschirmorientierter Editor, der mit Hilfe des Kommandos `EDIT` gestartet werden kann. Optional kann diesem Kommando der Name der zu bearbeitenden Datei mitgegeben werden – existiert die genannte Datei noch nicht, wird sie im Lauf der sich anschließenden Editorsitzung erzeugt.

Soll beispielsweise die Datei `WELCOME.TXT` editiert werden, besitzt der Editor-Aufruf folgende Gestalt:

[5] Konkrete Angaben von Devicenamen sind nach Möglichkeit aus Gründen leichterer Wartbarkeit stets durch logische Namen zu ersetzen!

```
$ EDIT WELCOME.TXT
```

Wie alle Editoren, trennt auch EDIT zwischen Kommandos und einzugebendem Text. Da es unpraktisch ist, ständig zwischen Eingabe- und Kommandomodus hin- und herschalten zu müssen, liegen eine Reihe häufig benötigter Funktionen auf speziellen Tasten, dem sogenannten *Keypad* (siehe Abschnitt 5.2). Über das Keypad kann auch zwischen Eingabe- und Kommandomodus gewechselt werden.

5.2 Das Keypad

Das Keypad besteht aus dem Ziffernblock auf der rechten Seite der Tastatur. Im Laufe der Jahre haben sich unter OpenVMS zwei verschiedene Belegungsarten dieses Tastaturbereiches entwickelt, von denen im folgenden stets die EDT-Variante verwendet wird. Um sicherzustellen, daß nach dem Start von EDT die EDT-Keypadbelegung geladen wird, muß in

```
SYS$MANAGER:SYLOGICALS.COM
```

mit Hilfe von

```
DEFINE /SYSTEM/TRANSLATION_ATTRIBUTES=TERMINAL EVE$KEYPAD EDT
```

ein entsprechendes Logical angelegt werden. Die Belegung des EDT-Keypads ist wie folgt:

```
                GOLD key functions are shown in reverse.
                   ------------------------    -------------------------------
  To get help on   | HELP  |      Do       |  |       |       |       |       |
  commands, type   |KeyDefs|               |  |       |       |       |       |
  a command name   |_______|_______________|  |_______|_______|_______|_______|
  or ? and press    ------------------------    -------------------------------
  RETURN.          | Find  |Ins Her|Remove |  | Gold  | HELP  |FndNxt | Del L |
                   |Wil Fin|Restore|Sto Tex|  |  key  |KeyDefs| Find  |Res Lin|
  To list all key  |_______|_______|_______|  |_______|_______|_______|_______|
  definitions,     |Select |Pre Scr|Nex Scr|  |MovByPa| Sect  |Append | Del W |
  type Keys and    | Reset |Pre Win|Nex Win|  |  Do   | Fill  |EDT Rep|Res Wor|
  press RETURN,    |_______|_______|_______|  |_______|_______|_______|_______|
  or press GOLD-           |Move up|          |Forward|Reverse|Remove | Del C |
  HELP.                    |  Top  |          |Bottom |  Top  |Ins Her|Res Cha|
                    _______|_______|_______   |_______|_______|_______|_______|
  To show a key    |Mov Lef|Mov Dow|Mov Rig|  | Word  |  EOL  | Char  |       |
  definition, use  |StaOfLi|Bottom |EndOfLi|  |ChngCas|Del EOL|SpecIns|Return |
  the SHOW KEY     |_______|_______|_______|  |_______|_______|_______| Subs  |
  command.                                   |   EDT Line    |Select |       |
                    Use the DO key to enter  |   Open Line   | Reset |       |
                       advanced commands     |_______________|_______|_______|
Buffer: HELP
Press the key that you want help on (RETURN to exit help):
```

5.2.1 Spezielle Tastenbezeichnungen

Einige Tasten (vier) des Keypads besitzen besondere Namen, die auch in den folgenden Abschnitten und Beispielen verwendet werden. Hierbei handelt es sich um die vier nebeneinanderliegenden Tasten der obersten Zeile des Ziffernblocks:

PF1: Diese Taste wird auch als GOLD-Taste bezeichnet, da auf einigen VT52-Terminals die entsprechende Taste gelb gefärbt war. Die GOLD-Taste dient quasi als Shift-Taste für die anderen Tasten des Keypads. Wird zuerst diese Taste und daran anschließend eine weitere Taste des Keypads betätigt, wird die Funktion, die in obigem Diagramm in der jeweils unteren Zeile steht, ausgeführt. Wird eine Keypad-Taste ohne vorangehendes Betätigen der GOLD-Taste gedrückt, wird die in der jeweils oberen Zeile beschriebene Funktion ausgeführt.

PF2: Durch Drücken dieser Taste wird das Hilfe-System des Editors aufgerufen – hiermit wurde das obige Keypad-Diagramm erzeugt.

PF3: Alle Suchfunktionen können durch diese Taste gestartet werden.

PF4: Mit PF4 können ganze Zeilen von der aktuellen Cursorposition bis zum Zeilenende gelöscht werden, GOLD PF4 fügt die zuletzt gelöschte Zeile an der momentan Cursorposition wieder ein.

Alle weiteren Tasten des Keypads werden durch Voranstellen des Bezeichners KP gekennzeichnet, so repräsentiert KP. beispielsweise die Punkt-Taste auf dem Keypad.

5.3 Nützliche Keypad-Kommandos

Funktion	Kommandos
Textanfang	GOLD KP5
Textende	GOLD KP4
Anfang der Zeile	KP0
Ende der Zeile	KP2
Wortweise springen	KP1
Text markieren, Anfang	KP.
Text löschen	KP6
Text einfügen	GOLD KP6
Kommandomodus aktivieren	GOLD KP7
Text suchen	GOLD PF3
Weitersuchen	PF3
Arbeitsrichtung aufwärts	KP5
Arbeitsrichtung abwärts	KP4

5.4 Kommandos im Kommandomodus

Einige Kommandos können ausschließlich innerhalb des Kommandomodus in einer EDIT-Sitzung eingegeben werden, da sich für sie keine Abkürzungen in Form einer Keypad-Taste finden. Der folgende Abschnitt listet einige häufig gebrauchte Kommandos auf, für eine tiefergehende Beschreibung sei an dieser Stelle auf das HELP-System des Editors bzw. die entsprechende Dokumentation verwiesen.

EXIT: Dieses Kommando beendet den Editor unter Abspeichern aller im Laufe der Sitzung vorgenommenen Änderungen.

QUIT: Im Unterschied zu EXIT verläßt QUIT den Editor, *ohne* die vorgenommenen Eingaben und Änderungen abzuspeichern. Wurden Änderungen vorgenommen, kann der Editor erst nach Beantworten folgender Sicherheitsabfrage wirklich verlassen werden:

```
Buffer modifications will not be saved, continue quitting [Yes]?
```

Die Defaultanwort auf diese Frage ist `Yes`, d.h. nach Betätigen der Eingabetaste werden alle Änderungen verworfen und der Editor verlassen.

`GET:` Mit Hilfe dieses Kommandos kann eine Datei in den Editor geladen werden. Jede Datei, die in den Editor geladen wurde, bekommt einen Buffer-Namen zugewiesen, über den sie beim Arbeiten mit mehreren geladenen Dateien (Arbeiten mit mehreren Buffern) referenziert werden kann.

`WRITE:` Der aktuelle Buffer-Inhalt wird mit diesem Kommando abgespeichert. Optional kann ein Dateiname mit angegeben werden, falls sie nicht unter ihrem originalen Namen abgelegt werden soll.

`SHOW BUFFER:` Hiermit können alle aktiven Buffer angezeigt werden. Angenommen, der Editor wurde mit

```
$ EDIT TEST.TXT
```

gestartet und im weiteren Verlauf der Editor-Sitzung wurde das Kommando

```
GET LOGIN.COM
```

eingegeben, so erzeugt `SHOW BUFFER` folgende Ausgabe:

```
 Buffer name                    Lines  Attributes

  TEST.TXT                          0
  LOGIN.COM                        26
```

Mit Hilfe der Cursor-Tasten kann nun einer der beiden Buffer ausgewählt und durch Drücken der Eingabetaste zum aktuellen Buffer deklariert werden. Zu beachten ist, daß, falls mehrere Buffer gleichzeitig verwendet werden, das Kommando `EXIT` *alle* modifizierten Bufferinhalte zurückspeichert – nicht nur den aktuell dargestellten!

`SPAWN:` Mit Hilfe des `SPAWN`-Kommandos kann der Editor zwischenzeitlich verlassen werden, wobei ein neuer Subprozeß erzeugt wird. Innerhalb dieses Subprozesses kann ohne Einschränkungen auf `DCL` zugegriffen werden. Nach Ausloggen mit `LOGOUT` wird die zuvor verlassene Editor-Session wieder augenommen.

6 Das MAIL-System

6.1 Grundlagen

Unter `OpenVMS` steht ein leistungsfähiges Mailsystem zur Verfügung, daß es gestattet, Mails unter Verwendung einer Vielzahl unterschiedlicher Protokolle zu versenden und zu empfangen. Standardmäßig unterstützt wird `DECnet`, aber auch Dienste aus dem Bereich der `TCP/IP`-Familie sind verfügbar (vorausgesetzt, daß ein `TCPI/IP`-Protokollstack, wie beispielswiese `UCX` oder `TGV/MULTINET` installiert ist).

6.1.1 Das Folder-Konzept

Nach außen hin stellt das MAIL-Utility eine Verzeichnisstruktur zur Verfügung, mit deren Hilfe Mails beliebig kategorisiert und sortiert werden können. Die einzelnen Verzeichnisse dieser Struktur werden als *Folder* bezeichnet. Ein Beispiel für einen solchen Folder ist das Verzeichnis MAIL, in dem alle bereits gelesenen Nachrichten abgelegt werden.

Nach Aufruf des MAIL-Utilities können die vorhandenen Folder durch

```
MAIL> DIR/FOLDER
```

angezeigt werden. Im Grundzustand, d.h., falls keine eigenen Folder angelegt wurden und keine neuen, ungelesenen Mails vorliegen, aber zumindest eine Mail bereits früher gelesen wurde, wird hier lediglich der bereits erwähnte Ordner MAIL angezeigt.

Sobald eine neue Nachricht eingeht, wird ein neuer Folder, NEWMAIL, erzeugt, in dem sich die neuen, ungelesenen Nachrichten befinden. Der Wechsel zwischen verschiedenen Foldern kann mit Hilfe des Befehls SELECT, gefolgt vom Namen des gewünschten Folders, vollzogen werden.

Befindet man sich beispielsweise in MAIL, während eine neue Nachricht eingeht, kann der Folder NEWMAIL mit

```
MAIL> SELECT NEWMAIL
```

zum aktuellen Standardfolder erklärt werden, so daß nun Zugriff auf die neu eingegangenen Nachrichten besteht. Wird das Programm MAIL aufgerufen, *nachdem* neue Nachrichten empfangen wurden, wird der Standardfolder sofort auf NEWMAIL gesetzt, da neue Nachrichten stets höchste Priorität besitzen.

Welche Nachrichten sich in einem Folder befinden, kann durch das Kommando DIR eruiert werden. Dieses bezieht sich stets auf den aktuellen Folder, nach Ausführung des obigen Kommandos würden folglich alle Nachrichten, die sich in NEWMAIL befinden, angezeigt.

6.2 Transportmechanismen und Versenden von Mails

Wie bereits erwähnt, steht für den Versand von Nachrichten eine Reihe verschiedener Protokolle zur Verfügung, unter denen das heutzutage sicher meistgenutzte SMTP darstellt.

Eine Nachricht kann mit Hilfe des Kommandos SEND abgeschickt werden – im Anschluß hieran erfragt das System die Adresse des Ziels, sowie in einem weiteren Schritt den gewünschten Titel der zu versendenden Nachricht. Die Auswahl des benötigten Protokolls geschieht explizit bei der Angabe der Zieladresse.

- Soll beispielsweise eine Nachricht an einen User, dessen Account sich auf der selben Maschine befindet, versandt werden, reicht als Adressangabe dessen Benutzername.
- Befindet sich der Account des Empfänger auf einer Maschine, die vom Sendersystem aus direkt per DECnet erreicht werden kann, hat die Adresse die allgemeine Form

  ```
  MASCHINE::USERNAME
  ```

- Soll eine Mail über das SMTP-Protokoll versandt werden, muß über die Angabe der eigentlichen Zieladresse der Form

  ```
  username@maschine.domain.suffix
  ```

  hinaus das genannte Protokoll spezifiziert werden, so daß eine vollständige Adressangabe zum Versand einer Internet-Mail die Form

  ```
  SMTP%"username@maschine.domain.suffix"
  ```

  besitzt.

Die folgenden Beispiele verdeutlichen die angesprochenen Mechanismen:

1. Es ist eine Nachricht an den User KURS1, dessen Account sich auf der lokalen Maschine befindet, zu versenden:

   ```
   $ MAIL
   MAIL> SEND
   Address: KURS1
   Subject: Testmail
   Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
   Testnachricht...
   <CTRL/Z>
   ```

2. Eine Mail an einen Benutzer auf einer entfernten, aber per DECnet erreichbaren Maschine, deren Name HUGIN sei, läßt sich folgendermaßen versenden:

   ```
   $MAIL
   MAIL> SEND
   Address: HUGIN::KURS1
   usf.
   ```

3. Abschließend soll eine Nachricht an die Internetmailadresse ulmann@raven-infotech.de verschickt werden:

   ```
   $ MAIL
   MAIL> SEND
   Address: SMTP%"ulmann@raven-infotech.de"
   Subject: usw.
   ```

Zu beachten ist, daß eine Mail erst abgesandt wird, nachdem das Textende durch Eingabe von <CTRL/Z> kenntlich gemacht wurde. Soll die aktuelle Nachricht nicht versendet werden, kann durch <CTRL/C> ein Abbruch erzwungen werden.

Falls die eingeschränkten Editierfähigkeiten der zeilenweise Eingabe einer Mail, wie in den obigen Beispielen, als nicht ausreichend erachtet werden, kann durch Verwendung des Kommandos

```
SEND/EDIT
```

ein Editor zur Erstellung der Nachricht aufgerufen werden. Auch hier gilt das eben gesagte: Abgeschickt wird die Mail durch <CTRL/Z>, Eingabe von <CTRL/C> bricht die laufende Editorsession ohne Versenden ab.

6.3 Empfangen von Mail

Geht eine neue Nachricht ein, wird automatisch ein neuer Folder Namens NEWMAIL angelegt. Dieser wird beim Aufruf des MAIL-Utilities direkt als Standardfolder deklariert, so daß ohne zusätzliche Kommandos Zugriff auf neue Nachrichten besteht. Das Vorhandensein neuer Nachrichten wird nach Aufruf von MAIL durch eine entsprechende Meldung angezeigt.

Alle Nachrichten innerhalb eines Folders besitzen eine Nummer, mit deren Hilfe sie gezielt angesprochen werden können. Sind beispielsweise drei neue Nachrichten eingegangen, befinden sich in NEWMAIL Nachrichten mit den Nummern 1, 2 und 3. Diese können (nach einem eventuell vorangegangenen DIR-Kommando) direkt unter Angabge ihrer laufenden Nummer gelesen werden.

Wird keine Nummer eingegeben, sondern nur die RETURN-Taste betätigt, wird die erste Nachricht im aktuellen Folder angezeigt, danach die zweite, usf. Alle gelesenen Nachrichten werden nach Verlassen des MAIL-Programmes (siehe Abschnitt 6.7) in den Folder MAIL kopiert – ist der Folder NEWMAIL leer, wird er gelöscht und erst bei Eintreffen neuer Nachrichten wieder erzeugt.

6.4 Extrahieren von Mails

In einigen Fällen ist es nützlich, eine gerade gelesene Mail in Form einer Datei abspeichern zu können, um sie beispielsweise in folgenden Schritten mit anderen Programmen (Editoren, etc.) bearbeiten zu können.

Das Kommando EXTRACT extrahiert den Inhalt der aktuell gelesenen Nachricht in eine Datei, deren Name als Argument des Kommandos angegeben werden kann. Soll beispielsweise die gerade gelesene Mail in eine Datei EINLADUNG.TXT extrahiert werden, kann dies durch

```
EXTRACT EINLADUNG.TXT
```

geschehen.

6.5 Löschen von Nachrichten

Meist ist es nicht sinnvoll, alle jemals eingegangenen Nachrichten aufzubewahren – nicht zuletzt, da die Performance von MAIL mit steigender Anzahl von vorhandenen Nachrichten herabgesetzt wird.

Eine Mail kann gezielt mit Hilfe des Kommandos DELETE gelöscht werden – hierbei ist zu beachten, daß sie in diesem Schritt nicht wirklich gelöscht, sondern in einen besonderen Folder, WASTEBASKET, verschoben wird, aus dem sie, solange sie sich darin befindet, auch wieder in den Folder MAIL (oder andere Ordner) verschoben werden kann. Das Leeren des Folders WASTEBASKET, der hierdurch gleichzeitig selbst gelöscht wird, geschieht beim Verlassen des MAIL-Utilities, siehe Abschnitt 6.7.

6.6 Anlegen von Foldern und Verschieben von Mails

In den vorangegangenen Abschnitten wurden bereits drei Folder angesprochen:

1. Der Folder MAIL – in ihm werden alle bereits gelesenen Nachrichten abgelegt,
2. der Folder NEWMAIL – er wird erzeugt, sobald neue Nachrichten eintreffen und verschwindet, nachdem die letzte in ihm enthaltene Mail gelesen wurde,
3. der Folder WASTEBASKET – in ihn werden alle Nachrichten, die mit Hilfe des Kommandos DELETE gelöscht wurden, kopiert. Er wird – mitsamt seinem Inhalt – gelöscht, wenn das MAIL-Programm beendet wird (siehe Abschnitt 6.7).

Allgemein läßt sich feststellen:

- Ein Folder wird gelöscht, sobald er keine Nachrichten mehr enhält,
- ein Folder wird erzeugt, indem mindestens eine Nachricht in ihn verschoben wird.

Das Verschieben von Nachrichten geschieht mit Hilfe des Befehls `MOVE`, wobei als Argument der Name des Zielfolders anzugeben ist. Soll beispielsweise die gerade gelesene Nachricht in einen Folder `PRIVAT` verschoben werden, der zu diesem Zeitpunkt nicht notwendigerweise existieren muß, hätte das Kommando die Form

```
MAIL> MOVE PRIVAT
```

Existiert bereits ein Folder `PRIVAT`, wird die Nachricht ohne weitere Fragen seitens des Systems dorthin verschoben, während im anderen Falle der Zielfolder nach einer Sicherheitsabfrage neu angelegt wird.

6.7 Verlassen des `MAIL`-Utilities

Zum Verlassen des `MAIL`-Programmes stehen zwei verwandte, aber dennoch unterschiedliche Kommandos bereit: `EXIT` und `QUIT`. Beide beenden das Programm `MAIL`, jedoch leert `EXIT` hierbei den eventuell vorhandenen Folder `WASTEBASKET` und löscht dessen Inhalt (erst hierdurch werden mit Hilfe von `DELETE` gelöschte Nachrichten wirklich gelöscht), während `QUIT` das Programm ohne Aufühen dieser Tätigkeit verläßt – mit `DELETE` gelöschte Mails gehen nach einem `QUIT` folglich nicht verloren.

7 Kommandoprozeduren

Kommandoprozeduren stellen im Grunde eine Ansammlung von `DCL`-Kommandos dar, die in der Reihenfolge ihres Auftretens abgearbeitet werden[6].

Zusätzlich zu den im interaktiven Modus zur Verfügung stehenden `DCL`-Kommandos stehen weitere Befehle bereit, mit deren Hilfe die Abarbeitungsreihenfolge der Kommandos einer solchen Kommandoprozedur beeinflußt werden kann. Dies sind im wesentlichen die Befehle `IF`, `ELSE`, `ENDIF`, `GOTO`, `GOSUB` und `RETURN`.

Kommandoprozeduren werden in der Hauptsache eingesetzt, um komplexe oder zeitaufwendige Abläufe in Form eines sogenannten *Batches* zu automatisieren, aber auch Voreinstellungen einzelner Benutzer können über eine spezielle Kommandoprozedur namens `LOGIN.COM` gesetzt werden.

Kommandoprozeduren sollten stets im Filenamen die Extension `.COM` besitzen, da dies die Voreinstellung des `DCL`-Interpreters ist. Aufgerufen werden Kommandoprozeduren durch das Zeichen `@` – so startet beispielsweise `@COMPILE` die Kommandoprozedur `COMPILE.COM`, die sich im aktuellen Default-Verzeichnis befindet. Die Extension `.COM` wird von `@` automatisch ergänzt, sofern sie nicht beim Aufruf mit angegeben wurde. Endet der Filename einer Kommandoprozedur nicht auf `.COM`, muß der vollständige Dateiname beim Start angegeben werden.

7.1 Beispiele

Das folgende Beispiel stellt eine Kommandoprozedur dar, mit deren Hilfe ein `VAX-C`-Programm `PRIM.C` übersetzt und gelinkt werden kann:

[6] Unter `UNIX` beispielsweise werden solche Prozeduren üblicherweise als *Shell-Scripten* bezeichnet.

```
$! Compiler starten
$ CC PRIM
$! Linker starten
$ LINK PRIM, SYS$LIBRARY:VAXCRTL/LIBRARY
```

Zu beachten ist, daß die einzelnen Kommandozeilen einer DCL-Kommandoprozedur mit einem $ beginnen müssen, Kommentarzeilen beginnen mit den Zeichen $!.

Die eben dargestellte Kommandoprozedur ist in ihrem Verhalten sehr starr, da sie bei jedem Aufruf stets dasselbe C-Programm übersetzt. Nützlich wäre eine Prozedur, die die beiden dargestellten Schritte für ein beliebiges C-Programm ausführen kann. Hierzu ist die Angabe und Verarbeitung eines Parameters notwendig.

Parameter können an DCL-Kommandoprozeduren direkt nach dem Filenamen nach dem @-Kommando übergeben werden. Soll beispielsweise die Prozedur AUSGABE.COM mit dem Parameter TEXT gestartet werden, hätte der entsprechende Aufruf folgende Form:

```
$ @AUSGABE TEXT
```

Innerhalb einer Kommandoprozedur sind eventuelle Aufrufparameter in Form spezieller Variablen (Symbole) sichtbar, die mit P1 bis P8 bezeichnet werden. Eine Prozedur, die ihren ersten Übergabeparameter (und nur diesen) auf dem Bildschirm ausgibt, könnte folgendermaßen geschrieben werden (AUSGABE.COM)

```
$ WRITE SYS$OUTPUT P1
```

P1 enthält den Wert des ersten Übergabeparameters des aktuellen Prozeduraufrufes, der durch das WRITE-Statement auf die Standardausgabe SYS$OUTPUT geschrieben wird. WRITE verwendet automatisch den *Wert* einer Variablen, wie beispielsweise P1 – in anderen Fällen, in denen dies nicht selbständig geschieht, kann eine Auswertung durch Einschließen des entsprechenden Symbols in einfache Anführungszeichen ' erzwungen werden. Da dies jedoch den Rahmen dieser Kurzeinführung überschreitet, sei auch an dieser Stelle auf die entsprechende OpenVMS-Dokumentation verwiesen.

7.2 Eine Login-Prozedur

Eine Kommandoprozedur mit herausragender Bedeutung ist die Datei LOGIN.COM im Default-Verzeichnis SYS$LOGIN. Sie wird automatisch bei jedem Einloggen in das System ausgeführt und ist der Ort, an dem benutzerspezifische Voreinstellungen und Definition ausgeführt werden können. Solcher Voreinstellungen können beispielsweise Terminaleinstellungen sein, aber auch Symbole und Foreign-Commands fallen hierunter.

Ein einfaches LOGIN.COM könnte folgende Gestalt besitzen:

```
$ SET TERMINAL/INSERT
$ PWD :== SHOW DEFAULT
```

Hier wird ein Symbol PWD definiert, das die Ausführung von SHOW DEFAULT veranlaßt – weiterhin wird das Terminal des Benutzers in den Insert-Modus umgestellt.

So einfach diese Prozedur auf den ersten Blick scheint, so große Schwierigkeiten kann ihre Anwendung nach sich ziehen. Für den Fall eines interaktiven Logins geschieht nichts unerwartetes. Dies ändert sich jedoch, sobald der betreffende

User einen Batch-Job (siehe Abschnitt 9) startet. Da auch Batch-Jobs die Login-Prozedur(en) durchlaufen, kommt auch diese Kommandoprozedur zur Ausführung, scheitert jedoch bereits an ihrem ersten Kommando, da ein Batch-Job kein zugeordnetes Terminal besitzt, das auf Insert-Modus umgeschaltet werden könnte.

Es wird folglich eine Unterscheidung zwischen interaktivem und nicht-interaktivem Login notwendig, um solche Probleme auszuschließen. Dies kann durch Verwendung der Lexical-Function `F$MODE` (siehe Abschnitt 3.5) geschehen, wie in folgendem Beispiel ausgeführt:

```
$ IF F$MODE () .EQS. "INTERACTIVE"
$ THEN
$  SET TERMINAL/INSERT
$ ENDIF
$!
$ PWD :== SHOW DEFAULT
```

Das Kommando `SET TERMINAL/INSERT` kommt hier nur dann zur Ausführung, wenn es sich um ein interaktives Login handelt, da nur in diesem Fall der Rückgabewert von `F$MODE` gleich `INTERACTIVE` ist (die Vergleichsoperation `.EQS.` steht für *E*qual *S*tring). Das Symbol `PWD` wird in jedem Fall definiert, steht also auch Batch-Jobs, etc. zur Verfügung.

7.3 SYS$MANAGER:SYLOGIN.COM

Zusätzlich zu dem eben erwähnten, benutzerspezifischen `LOGIN.COM` exisitiert eine systemweite Login-Prozedur, mit deren Hilfe Voreinstellungen vorgenommen werden können, die für alle User gelten sollen. Diese findet sich unter

```
SYS$MANAGER:SYLOGIN.COM
```

und kann jederzeit explizit durch das Logical `SYS$SYLOGIN` referenziert werden.

8 Arbeiten mit Platten und Bändern

Wie bereits in Abschnitt 4.2.4 erwähnt, werden Plattenbezeichnungen stets mit einem Doppelpunkt abgeschlossen. In den vorangegangenen Beispielen wurde bereits eine Platte angesprochen, nämlich das Device `DKA100`. Sowohl Platten als auch Bandlaufwerke besitzen solche Devicebezeichnungen, die aus drei Buchstaben, einer Nummer sowie einem abschließenden Doppelpunkt bestehen. Der führende Buchstabe kennzeichnet den Devicetyp:

- Plattenlaufwerke beginnen stets (von historischen Devices, wie manchen `MASSBUS`-Laufwerken, etc. einmal abgesehen) mit einem `D` wie *Disk*,
- während Magnetbandlaufwerke mit einem `M` für *Magnetic Tape* beginnen.

Der zweite Buchstabe kennzeichnet den Controllertyp, an dem das betreffende Device angeschlossen ist – die wichtigsten hierbei sind `K`, `I`, `U` und `S`:

`K`: Hiermit wird unter anderem der interne `SCSI`-Controller von `OpenVMS`-Workstations bezeichnet.

`I`: `DSSI`-Controller werden hierdurch gekennzeichnet.

`U`: `UNIBUS`- und `QBUS`-Plattencontroller haben diesen Bezeichner gemeinsam.

`S:` Mit `S` wird ein logischer Controller bezeichnet, der ein *Shadow-Set* zur Verfügung stellt (gespiegelte Platten).

Der dritte (und letzte Buchstabe) einer Platten- oder Magnetbandbezeichnung kennzeichnet die Nummer der Controllers in der Maschine: Ein `A` bezeichnet den ersten Controller, ein `B` den zweiten, usf.

Zuletzt wird die Adresse der Platte am jeweiligen Controller durch die sich anschließende Zahl vor dem Doppelpunkt angegeben, wobei manche Controller dreistellige Nummern, andere hingegen nur einstellige verwenden. Sind dreistellige Zahlen angegeben, stellt die höchstwertige Ziffer die Device-Adresse der betreffenden Einheit dar, d.h.

```
DKA100:
```

ist die Platte mit der `ID` 1 am ersten internen `SCSI`-Controller einer Workstation, während

```
MUC3:
```

das Bandlaufwerk mit der `ID` 3 am dritten Controller in einem `UNIBUS`- oder `QBUS`-System repräsentiert.

8.1 Labels und Initialisierung

Sowohl Platten als auch Bänder verfügen über ein sogenanntes *Label* – dies ist ein symbolischer Name, der dem Device, bzw. dem eingelegten Band gegeben werden kann (und meist auch muß), mit dessen Hilfe eine Zuordnung erleichtert werden soll.

Um eine Platte oder ein Band für eine folgende Verwendung vorzubereiten, ist das Device zu *initialisieren*. Hierbei wird zum einen das benötigte Label geschrieben und zum anderen legt `OpenVMS` in dieser Phase benötigte Datenstrukturen an (bei Platten fallen hierunter beispielsweise die Datei `INDEX.SYS` und andere, über die die Zuordnung von Blöcken und Dateien auf der Platte vorgenommen werden).

Das Kommando `INITIALIZE` zum Initialisieren von Platten und Bändern erwartet zwei Parameter (neben einer Vielzahl optionaler Qualifizierer, siehe `HELP INITIALIZE`). Der erste Parameter ist die Bezeichnung des Devices, das initialisiert werden soll, beispielsweise `DKA100:`, der zweite enthält den gewünschten Labelnamen, der dem Device zugeordnet werden soll.

Um beispielsweise die Platte `DKA100` mit dem Label `USERDISK` zu initialisieren, ist folgendes Kommando notwendig:

```
$ INITIALIZE DKA100: USERDISK
```

Im Anschluß hieran kann die Platte für weitere Verwendung gemountet werden (siehe Abschnitt 8.2). Zu beachten ist, daß eine solche Initialisierung *alle* eventuell auf dem Device vorhandenen Daten löscht!

8.2 Mounten und Dismounten von Platten und Bändern

Bevor auf eine Platte oder auf ein Magnetbandlaufwerk zugegriffen werden kann, muß das betreffende Device *gemountet* werden. Hierdurch wird es dem System bekanntgemacht, wobei zwei grundlegende Zugriffsmodi zu unterscheiden sind:

1. Normaler, filestrukturierter Zugriff und

2. blockweiser Zugriff ohne Struktur.

In den meisten Fällen wird der erste Fall vorliegen – hier muß beim Mounten des betreffenden Devices der jeweilige Labelname angegeben werden. Das MOUNT-Kommando wird nur dann durchgeführt, wenn angegebenes Label und wirkliches Label des Devices übereinstimmen. Um beispielsweise die im vorangegangenen Abschnitt initialisierte Platte DKA100 zu mounten, um sie für die Benutzer des Systems freizugeben, kann folgender Befehl verwendet werden:

```
$ MOUNT DKA100: USERDISK
```

Der zweite erwähnte Zugriffsmodus mountet ein Device in der Art, daß es ohne Filestruktur angesprochen werden kann (und muß). Dies ist beispielsweise nützlich, um Platten und Bänder von Fremdsystemen mit Spezialprogramm bearbeiten zu können, oder, um Backup-Operationen durchführen zu können (siehe Abschnitt 8.3).

Soll ein Device als Blockdevice ohne Filestruktur gemountet werden, kann der Qualifizierer /FOREIGN angewandt werden. Hierbei ist *kein* Label des jeweiligen Devices erforderlich! Das Kommando

```
$ MOUNT/FOREIGN DKA100:
```

mountet beispielsweise die Platte DKA100 ohne Filestruktur, so daß beispielsweise ein bereits erstelltes Backupsaveset zurückgespielt werden kann (siehe Abschnitt 8.4).

Um ein Device wieder freizugeben, sei es, weil das System heruntergefahren wird, sei es aus Wartungsgründen, wegen Bandwechsel, oder weswegen auch immer, muß es, falls es zuvor gemountet war, entsprechend *dismountet* werden. Dies hat zur Folge, daß eventuell noch nicht zurückgeschriebene Cache-Inhalte geschrieben werden, daß noch offene Files geschlossen werden, und daß das betreffende Device im Anschluß hieran nicht mehr zur Verfügung steht (bis zum nächsten MOUNT-Kommando).

Die einfachste Form eines DISMOUNT-Kommandos hat folgende Form:

```
$ DISMOUNT DKA100:
```

Hierdurch wird die Platte DKA100 aus dem System entfernt und steht nicht mehr für Ein-/Ausgabeoperationen zur Verfügung. In diesem Zusammenhang ist eine Besonderheit im Umgang mit Bändern und Wechselplatten zu beachten: Nach einem DISMOUNT-Kommando werden Bänder und Wechselplatten ausgeworfen, was nicht immer gewünscht ist – vor allem, wenn kein physikalischer Zugriff auf ein Bandlaufwerk besteht, ist es ein sehr unpraktisches Verhalten, wenn ein Band nach einem Dismount ausgeworfen wird. Um dies gezielt unterdrücken zu können, kann der Qualifizierer /NOUNLOAD verwendet werden. So dismountet beispielsweise das Kommando

```
$ DISMOUNT MKA500:/NOUNLOAD
```

das momentan im Magnetbandlaufwerk MKA500 befindliche Band, wirft es aber nicht aus, so daß es mit einem nachfolgenden MOUNT-Kommando wieder geladen werden kann.

8.3 Sichern von Platten

Der folgende Abschnitt geht nur auf die Möglichkeit des Sicherns einer gesamten Platte in Form eines sogenannten *Image-Savesets* ein. Das hierfür eingesetzte Utility `BACKUP` ist sehr mächtig und in der Lage, auch inkrementelle Backups, physikalische Backups, etc. durchzuführen, was jedoch über den Rahmen dieser Einführung bei weitem hinaus geht. An dieser Stelle sei für eingehendere Informationen, das `BACKUP`-System betreffend, auf die entsprechende `OpenVMS`-Dokumentation verwiesen.

Ein sogenanntes Image-Saveset oder Image-Backup stellt ein vollständiges Backup einer Platte dar, nach dessen Restaurierung die Zielplatte logisch äquivalent zur Quellplatte ist. Im Unterschied zu einem *physical* Backup müssen hierbei jedoch Quell- und Zielplatte nicht vom gleichen Typ sein, sofern die Zielplatte mindestens genügend Blöcke besitzt, um alle Dateien der Quellplatte aufzunehmen.

Dreh- und Angelpunkt aller Backup-Maßnahmen unter `OpenVMS` ist das `BACKUP`-Utility. Es erhält mindestens zwei Pfadangaben als Aufrufparameter:

1. Eine Quell- und
2. eine Zielangabe. Falls es sich, wie in diesem Beispiel, um ein Image-Backup handelt, muß das Zieldevice mit `MOUNT/FOREIGN` gemountet werden. Handelt es sich um ein Band, kann das `BACKUP`-Utility diese Aufgabe übernehmen, bei einer Platte muß diese zuvor von Hand gemountet werden (siehe Abschnitt 8.4).

Dies können ganz allgemein in beiden Fällen ganze Platten, Bänder, oder auch nur Verzeichnisse, oder einzelne Dateien sein, wobei in diesem Abschnitt nur die Möglichkeit behandelt wird, ein Image-Backup von einer Platte anzufertigen. Folgendes Beispiel kopiert den gesamten Inhalt der Platte `DKA100` auf ein Band im Magnetbandlaufwerk `MKA500`, wobei das Band das Label `USRBCK` bekommt:

```
$ BACKUP DKA100:/IMAGE MKA500:USERPLATTE.BCK/SAVE/LOG -
_$ /IGNORE=(LABEL, INTERLOCK)/LABEL=USRBCK
```

Sein Aufbau ist wie folgt:

- Zunächst wird als Datenquelle die Platte `DKA100:` spezifiziert.
- Der direkt folgende Qualifizierer `/IMAGE` gibt an, daß ein Image-Backup erzeugt werden soll.
- Das zu erstellende Backupfile soll auf das Magnetband `MKA500:` unter dem Filenamen `USERPLATTE.BCK` geschrieben werden.
- Der Qualifizierer `/SAVE` gibt an, daß es sich hierbei um ein wirkliches Backup-*Saveset* handeln soll[7].
- Die Angabe von `/LOG` hat zur Folge, daß eine Liste der kopierten Dateien ausgegeben wird, anhand derer sich der Fortschritt der Kommandoausführung verfolgen läßt.
- Der Qualifizierer `/IGNORE` bekommt eine Liste, bestehend aus zwei Parametern, übergeben:

[7]Wäre dieser Qualifizierer nicht angegeben, würde der Versuch unternommen werden, alle Dateien einzeln (unter Berücksichtigung der Verzeichnisstruktur des Quellaufwerkes) auf das Band zu schreiben. Dies würde einerseits daran scheitern, daß ein Magnetband keine Verzeichnisstruktur unterstützt und andererseits daran, daß die Ausnutzung des auf dem Band zur Verfügung stehenden Platzes sehr schlecht wäre, da eine Vielzahl kleiner Blöcke mit entsprechenden Verlusten in den `GAP`s geschrieben würde! Ganz abgesehen davon, daß die mit dem Qualifizierer `/IMAGE` kollidieren würde...

`INTERLOCK`: Diese Angabe hat zur Folge, daß Dateien, die nicht geschlossen sind, sondern noch von einem Programm schreibend bearbeitet werden, nicht zu einem Abbruch des `BACKUP`-Kommandos führen. Solche Dateien werden nicht gesichert, gehen also verloren – im allgemeinen sollte darauf geachtet werden, daß auf dem zu sichernden Device keinerlei Dateien geöffnet sind. Dieser Parameter ist lediglich als Sicherheitsleine zu verstehen, die hoffentlich nicht zur Anwendung kommen wird.

`LABEL`: Durch diese Option wird erreicht, daß das eingelegte Band, unabhängig von seinem aktuellen Label (falls beispielsweise ein altes Band überschrieben werden soll) verwendet wird. Ein vom gewünschten Label abweichendes Bandlabel führt in diesem Fall nicht zum Abbruch von `BACKUP` oder einem Operator-Request.

- Zuletzt wird mit `/LABEL=USRBCK` das gewünschte Label für das eingelegt Band vereinbart. Stimmen aktuelles Bandlabel und hier spezifiziertes Label nicht überein, wird (dank `IGNORE=LABEL`) das aktuelle Label überschrieben.

8.4 Zurückspielen von Backup-Savestes

Das Zurückspielen eines gemäß obigem Beispiel erstellten Savesets von einem Sicherungsband auf Platte gestaltet sich wie folgt:

```
$ MOUNT/FOREIGN DKA100:
$ BACKUP MKA500:USERPLATTE.BCK/SAVE/LOG DKA100:/IMAGE/LOG
$ DISMOUNT DKA100:
```

In diesem Beispiel wird das in Abschnitt 8.3 erstellte Image-Backupfile `USERPLATTE.BCK` von einem Band in Laufwerk `MKA500` auf die Zielplatte `DKA100` zurückgespielt.

9 Batch- und Printerqueues

`OpenVMS` stellt ein äußerst leistungsfähiges Batch- und Print-Queue-System zur Verfügung, das in den folgenden Abschnitten kurz dargestellt wird.

9.1 Grundlagen

Der Hauptzweck einer Queue besteht in der unbeaufsichtigen Verarbeitung vorbereiteter Jobs[8]. Dies können zum einen Jobs sein, die in Form von Kommandoprozeduren vorliegen, zum anderen fallen hierunter jedoch auch Druckaufträge. Entsprechend dieser Aufteilung, existieren unter `OpenVMS` zwei grundlegende Typen von Queues:

1. Batch-Queues, in die abzuarbeitende Kommandoprozeduren gestellt werden können, sowie
2. Print-Queues, über die Druckaufträge verarbeitet werden.

9.2 Einrichten von Queues

Grundlegendes Element aller Queues eines `OpenVMS`-Systems ist der sogenannte *Queue-Manager* – dieses Programm kontrolliert die Verarbeitung der in die Queues gestellten Aufträge. Für Queues und den Queuemanager gilt, daß sie *nur einmal* erzeugt werden müssen! Es ist weder notwendig, noch korrekt, Queues, etc. bei jedem

[8] Unter `MULTICS` hießen solche Jobs auch *unattended jobs*.

Systemstart durch entsprechende Einträge in der zentralen Startupprozedur (siehe Abschnitt 11.3) neu zu generieren!

Das einmalige Einrichten eines Queue-Managers geschieht mit Hilfe des Kommandos

```
$ START/QUEUE/MANAGER/NEW
```

Im Anschluß hieran können Queues erzeugt werden:

9.2.1 Batch-Queues

Batch-Queues dienen, wie bereits angedeutet, der automatischen Verarbeitung vorbereiteter Jobs in Form von Kommandoprozeduren. Soll beispielsweise lediglich ein ausführbares Programm als Batch gestartet werden, muß es aus einer Kommandoprozedur heraus aufgerufen werden, die dann ihrerseits im Batch läuft.

Zunächst zur Einrichtung einer Batch-Queue: Im folgenden wird die Standard-Queue `SYS$BATCH` erzeugt, die in jedem Fall auf einem neuen `OpenVMS`-System erzeugt werden sollte, da eine Reihe von beispielsweise Startupprozeduren beim Boot hiervon Gebrauch machen können, was sich unter Umständen sehr günstig auf die Startupzeit auswirkt.

Das Kommando zum Erzeugen einer Batch-Queue namens `SYS$BATCH` besitzt folgende Gestalt:

```
$ INITIALIZE/QUEUE/BATCH/AUTOSTART_ON=MARS:: -
_$ /JOB_LIMIT=10/BASE_PRIORITY=3 SYS$BATCH
```

Hiermit wird eine sogenannte *Autostart* Queue erzeugt, d.h. sie befindet sich nach einem Reboot des Systems in dem Zustand (d.h. gestartet oder gestoppt), in dem sie sich zum Zeitpunkt des Shutdowns befand. Um dieses Feature nutzen zu können, muß in der Startupprozedur `SYS$MANAGER:SYSTARTUP_VMS.COM` die Zeile

```
$ ENABLE AUTOSTART /QUEUES
```

aktiviert werden. Durch obiges `INITIALIZE`-Kommando werden weiterhin eine Reihe von Voreinstellungen für die neue Queue vorgenommen:

- Es handelt sich um eine Batch-Queue (Parameter `/BATCH`),
- die Anzahl maximal in dieser Queue laufender Jobs wurde auf 10 festgelegt, wobei als
- Basispriorität dieser Jobs 3 eingestellt wurde. Dies liegt eine Stufe unter der Priorität normaler interaktiver Prozesse und soll somit verhindern, daß sich eine große Rechenlast in dieser Batch-Queue extrem auf die Antwortzeiten für interaktive Benutzer auswirkt.

Vor ihrer Verwendung muß die generierte Batch-Queue einmal mit

```
$ START/QUEUE SYS$BATCH
```

gestartet werden[9]. Bei Bedarf kann die Queue mit

```
$ STOP/QUE SYS$BATCH
```

[9]Es besteht auch die Möglichkeit, eine Queue direkt bei ihrer Erzeugung zu starten. Hierfür muß lediglich der zusätzliche Qualifizierer `/START` im `INITIALIZE`-Kommando angegeben werden.

auch wieder gestoppt werden. Allgemein können Queues hiermit nur dann angehalten werden, wenn gerade kein Job bearbeitet wird. Sollen eventuell gerade aktuve Jobs abgebrochen und die Queue ohne Warten gestoppt werden, kann der zusätzliche Qualifizierer `/RESET` angegeben werden.

Ein Batch-Job kann nun mit Hilfe des Kommandos `SUBMIT` in eine Batch-Queue gestellt werden. Soll beispielsweise die Kommandoprozedur `COMPILE.COM` als Batch in der Queue `SYS$BATCH` ausgeführt werden, ist hierzu folgender Befehl notwendig:

```
$ SUBMIT/QUEUE=SYS$BATCH COMPILE.COM
```

Die Angabe `/QUEUE=SYS$BATCH` kann in diesem Fall auch fortgelassen werden, da die Queue `SYS$BATCH` die Default-Queue für alle Batch-Jobs darstellt, bei deren Start kein expliziter Queuename angegeben wurde.

Da einem Batch-Job naturgemäß kein Terminal zugeordnet ist (sonst handelte es sich um einen interaktiven Prozeß), werden alle Ausgaben, die im Verlauf seiner Verarbeitung anfallen, in ein Logfile geschrieben, das, falls im `SUBMIT`-Statement keine abweichenden Angaben vorgenommen wurden, denselben Namen, wie die Kommandoprozedur, die im Batch ausgeführt wird, jedoch mit der Extension `.LOG`, besitzt. Dieses Logfile wird im Login-Verzeichnis des Users, unter dessen Account der betreffende Batch läuft, abgelegt. Darüberhinaus gilt die Standardeinstellung, daß nach Ende eines Batch-Jobs dieses Logfile automatisch auf den Systemstandard-Drucker ausgegeben wird. Ist dies nicht erwünscht, was der Regelfall sein dürfte, kann die Erzeugung eines solchen Ausdrucks durch Angabe des Qualifizierers `/NOPRINT` unterdrückt werden.

9.2.2 Print-Queues

Das folgende Beispiel zeigt, wie die Default-Drucker-Queue `SYS$PRINT` eingerichtet werden könnte:

```
$ INITIALIZE/QUEUE/ON=TTA1:/START SYS$PRINT
```

Die solchermaßen erzeugte Drucker-Queue zeigt auf einen Drucker, der an der seriellen Schnittstelle `TTA1` angeschlossen ist – hierbei ist darauf zu achten, daß die Parameter dieser Schnittstelle mit denen des betreffenden Druckers kompatibel sind. Entsprechende Einstellungen sind über das Kommando `SET TERMINAL` vorzunehmen, wobei zu beachten ist, daß diese Einstellungen bei jedem Systemstart erneut vorgenommen werden müssen, da es sich um nicht-permanente Definitionen handelt!

Eine Besonderheit im Zusammenhang mit Druckerqueues ist das sogenannte *Spooling* – dieser Mechanismus dient zur Zwischenspeicherung der zu druckenden Aufträge, so daß mit dem Fortfahren eines Jobs nicht auf die Beendigung eines Druckjobs gewartet werden muß. Notwendig hierzu ist die Angabe einer Platte, auf der die Spool-Datei(en) angelegt werden sollen. Im folgenden Beispiel wird die Systemplatte, die stets unter dem logischen Namen `SYS$SYSDEVICE` angesprochen werden kann, als Spooldevice verwendet (wovon im Regelfall abzuraten ist, da eine durch Spoolfiles gefüllte Systemplatte einen regulären Betrieb eines `OpenVMS`-Systems unmöglich macht!):

```
$ SET DEVICE/SPOOLED=(SYS$PRINT, SYS$SYSDEVICE:)
```

Eine Platte, die als Spooldevice dient, kann übrigens nur dismountet werden (siehe Abschnitt 8.2), wenn keine Devices mehr auf ihr gespooled werden. Vor einem Dismount einer solchen Platte sind also alle betreffenden Devices mit

```
$ SET DEVICE/NOSPOOLED SYS$PRINT
```

auf NOSPOOLED zu setzen.

Druckaufträge werden mit Hilfe des PRINT-Befehls in die jeweils gewünschte Druckerqueue gestellt. Soll beispielsweise die Datei LOGIN.COM auf dem Standarddrucker ausgegeben werden, kann dies durch

```
$ PRINT LOGIN.COM
```

geschehen. Soll eine andere Printer-Queue verwendet werden, kann dies durch Angabe des Qualifizierers /QUEUE=... erzwungen werden.

9.3 Verwalten von Queue-Einträgen

Jeder Job, der in eine Queue, egal, ob Batch- oder Print-Queue, gestellt wird, kann durch eine eindeutige Jobnummer, die ihm durch das jeweilige SUBMIT- beziehungsweise PRINT-Kommando zugeteilt wird, referenziert werden. Das Kommando

```
$ SHOW ENTRY
```

zeigt alle Jobs des betreffenden Users an, die noch zur Bearbeitung anstehen oder gerade verarbeitet werden.

Ein Eintrag in einer Queue kann gezielt unter Zuhilfenahme seiner Job-Nummer gelöscht werden. So entfernt beispielsweise

```
$ DELETE/ENTRY=612
```

den Job mit der Nummer 612 aus seiner Queue.

9.4 Löschen von Queues

Im allgemeinen gilt, daß nur Queues gelöscht werden können, die sich im Zustand *stopped* befinden (siehe Abschnitt 9.2.1). Unter Umständen muß die betreffende Queue also mit Hilfe eines Kommandos der Form

```
$ STOP/QUEUE/RESET ...
```

angehalten werden, bevor sie gelöscht werden kann. Das Löschen einer Queue kann dann durch

```
$ DELETE/QEUEUE ...
```

geschehen.

10 Userverwaltung

Eine der wichtigsten Tätigkeiten eines Systemverwalters besteht im Anlegen und Verwalten von Benutzer-Accounts.

10.1 Grundlagen

Dreh- und Angelpunkt der gesamten Benutzerverwaltung unter OpenVMS ist die Datei SYS$SYSTEM:SYSUAF.DAT – in ihr sind alle Einstellungen der einzelnen Accounts, sowie die Paßwörter enthalten. Bearbeitet werden kann sie mit Hilfe des Programmes AUTHORIZE, dessen Aufruf über

```
$ MCR AUTHORIZE
```

vollzogen wird. Zu beachten ist an dieser Stelle, daß das Arbeitsverzeichnis zuvor *unbedingt* mit

```
$ SET DEFAULT SYS$SYSTEM
```

auf SYS$SYSTEM umgesetzt werden muß, da AUTHORIZE andernfalls versucht, unter dem aktuellen Verzeichnis ein neues UAF.DAT zu erzeugen!

10.2 Privilegien

OpenVMS stellt ein ausgefeiltes Privilegien-Schema zur Verfügung, das weit über die simplen Mechanismen beispielsweise eines UNIX-Systems hinausgeht. Insgesamt existieren fast 40 verschiedene Rechte, die einzeln vergeben oder verweigert werden können. Hierdurch ist eine feine Abstufung der einzelnen Berechtigungen möglich. So kann beispielsweise einem Benutzer das Recht zugestanden werden, Queues zu verwalten – hierfür ist das OPER-Privileg zuständig, während ein anderer Benutzer beispielsweise physikalische IOs auf Devices durchführen darf (PHY_IO).

10.3 UICs

Jedem Benutzer wird bei seiner Einrichtung eine eindeutige Nummer, die sogenannte *UIC*[10] zugeordnet. Eine solche UIC besteht aus zwei Zahlen, von denen die erste die Gruppe, welcher der betreffende User zugehören soll, kennzeichnet und die zweite den User selbst identifiziert.

In diesem Zusammenhang ist zu beachten, daß UICs stets als Dupel von Oktalzahlen dargestellt und eingegeben werden[11]!

Als Beispiel sei der Benutzer SYSTEM mit Hilfe des AUTHORIZE-Utilities näher betrachtet:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> SHOW SYSTEM

Username: SYSTEM                          Owner:  SYSTEM MANAGER
Account:  SYSTEM                          UIC:    [1,4] ([SYSTEM])
CLI:      DCL                             Tables: DCLTABLES
Default:  SYS$SYSROOT:[SYSMGR]
LGICMD:   LOGIN
Flags:
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                 Sat Sun
No access restrictions
Expiration:            (none)    Pwdminimum:  8   Login Fails:     0
Pwdlifetime:         30 00:00    Pwdchange:  16-JUL-1999 00:15
```

[10] *U*ser *I*dentification *C*ode

[11] Ein Relikt von RSX-11M, von dem sich OpenVMS letztlich herleitet.

```
Last Login: 17-JUL-1999 23:11 (interactive), 17-JUL-1999 22:36 (non-interactive)
Maxjobs:         0  Fillm:       300  Bytlm:        32768
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:         40  JTquota:       4096
Prclm:          10  DIOlm:         40  WSdef:          256
Prio:            4  ASTlm:         50  WSquo:          512
Queprio:         0  TQElm:         30  WSextent:      2048
CPU:        (none)  Enqlm:        200  Pgflquo:      40960
Authorized Privileges:
  ACNT      ALLSPOOL  ALTPRI    AUDIT     BUGCHK    BYPASS    CMEXEC    CMKRNL
  IMPERSONATDIAGNOSE  DOWNGRADE EXQUOTA   GROUP     GRPNAM    GRPPRV    IMPORT
  LOG_IO    MOUNT     NETMBX    OPER      PFNMAP    PHY_IO    PRMCEB    PRMGBL
  PRMMBX    PSWAPM    READALL   SECURITY  SETPRV    SHARE     SHMEM     SYSGBL
  SYSLCK    SYSNAM    SYSPRV    TMPMBX    UPGRADE   VOLPRO    WORLD
Default Privileges:
  ACNT      ALLSPOOL  ALTPRI    AUDIT     BUGCHK    BYPASS    CMEXEC    CMKRNL
  IMPERSONATDIAGNOSE  DOWNGRADE EXQUOTA   GROUP     GRPNAM    GRPPRV    IMPORT
  LOG_IO    MOUNT     NETMBX    OPER      PFNMAP    PHY_IO    PRMCEB    PRMGBL
  PRMMBX    PSWAPM    READALL   SECURITY  SETPRV    SHARE     SHMEM     SYSGBL
  SYSLCK    SYSNAM    SYSPRV    TMPMBX    UPGRADE   VOLPRO    WORLD
Identifier                      Value           Attributes
  NET$MANAGE                    %X91F50002
UAF>
```

Die `UIC` des Accounts `SYSTEM` ist `[1,4]` – dieses Format wird im Zusammenhang mit `UIC`s stets angewandt. Hierbei ist 1 die Gruppennummer, die Systemgruppe, und 4 die Usernummer. Das Loginverzeichnis des Benutzers ist `SYS$SYSROOT:[SYSMGR]` (das Systemmanager-Verzeichnis), die beim Login auszuführende Kommandoprozedur heißt `LOGIN.COM`. Es bestehen keinerlei Restriktionen hinsichtlich der Tage, an denen sich der User `SYSTEM` einloggen darf, das Paßwort muß mindestens 8 Zeichen aufweisen, es läuft nach 30 Tagen ab, während der Account selbst nicht verfällt und es liegen bislang keine fehlerhaften Loginversuche vor. Darüberhinaus sind die Zeiten des letzten Logins – sowohl interaktiv, als auch nichtinteraktiv – angegeben.

Für Parameter, wie beispielsweise `Maxjobs` gilt im allgemeinen, daß ein Wert von 0 für *unbegrenzt* steht – der User `SYSTEM` unterliegt also keinen Beschränkungen hinsichtlich der Anzahl gleichzeitig laufender Jobs im System.

Im Anschluß hieran wird eine Liste der genehmigten Privilegien[12] sowie der defaultmäßig aktivierten Privilegien[13] ausgegeben. Hierbi gilt, daß nur die Defaultprivilegien wirklich aktiv sind. Es ist durchaus möglich, daß weniger Privilegien als Default eingetragen sind, als als autorisiert. Um ein nur genehmigtes Privileg für einen laufenden Prozeß zum aktiven Privileg zu machen, muß der Befehl

```
$ SET PROCESS/PRIVILEGE=...
```

verwendet werden.

10.4 Einrichten von Benutzern

Im folgenden wird ein Benutzer `GAST` mit Passwort `GAST` angelegt, dessen Loginverzeichnis sich unter `DISK$USER:[GAST]` befindet, wobei die Login-Kommandoprozedur `LOGIN.COM` heißt. Er soll keine Privilegien außer den Standardprivilegien besitzen und das Anfangspaßwort muß beim ersten Login geändert werden. Als `UIC` soll `[1000,1]` verwendet werden:

[12] Authorized Privileges
[13] Default Privileges

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD GAST/UIC=[1000,1]/DEVICE=DISK$USER/DIR=[GAST]-
_UAF> /PASSWORD=GAST/FLAGS=NODISUSER

%UAF-I-PWDLESSMIN, new password is shorter than minimum password length
%UAF-I-ADDMSG, user record successfully added
%UAF-I-RDBADDMSGU, identifier GAST value [001000,000001] added to rights database
UAF> EXIT
```

Der Qualifizierer /FLAGS=NODISUSER löscht das DISUSER-Flag – ist dieses Flag gesetzt, kann sich der betreffende User nicht mehr einloggen. Beim Anlegen eines neuen Benutzers ist DISUSER die Voreinstellung, die explizit überschrieben werden muß!

Nach der so erfolgten Einrichtung des Benutzers GAST muß noch dafür Sorge getragen werden, daß sein Homeverzeichnis existiert und auch ihm gehört:

```
$ CREATE/DIRECTORY DISK$USER:[GAST]
$ SET FILE/OWNER=GAST DISK$USER:[000000]GAST.DIR
```

Nach diesen beiden Schritten exisitert nun ein Verzeichnis [GAST] auf dem Device DISK$USER, das dem Benutzer GAST gehört.

Sobald sich GAST das erste Mal einloggt, wird er gezwungen, sein Paßwort neu zu setzen (dies kann durch Ändern des Expire-Parameters in AUTHORIZE umgangen werden, was jedoch aus Sicherheitsgründen nicht der Regelfall sein sollte!):

```
 Welcome to OpenVMS (TM) VAX Operating System, Version V7.2

Username: gast
Password:
 Welcome to OpenVMS (TM) VAX Operating System, Version V7.2 on node TICHY

Your password has expired; you must set a new password to log in


New password:
Verification:
```

10.5 Ändern von Benutzereinstellungen

Das Ändern von Benutzereinstellungen in SYSUAF.DAT wird ebenfalls mit Hilfe des AUTHORIZE-Utilities vollzogen. Prinzipiell können hierbei alle Qualifizierer und Parameter zum Einsatz kommen, die auch bei der Neuanlage eines Benutzers verwendet werden können. Lediglich das Kommando ADD ist durch den Befehl MOD zu ersetzen.

10.6 Löschen von Benutzern

Das Löschen von Benutzern aus der Userdatenbank SYSUAF.DAT geschieht in AUTHORIZE mit Hilfe des Kommandos REMOVE. Zu beachten ist hierbei, daß lediglich die den Benutzer betreffenden Einträge aus SYSUAF.DAT entfernt werden. Sein Homeverzeichnis, sowie alle zu ihm gehörenden Dateien werden hiervon nicht beeinflußt.

Soll beispielsweise der Benutzer GAST aus obigem Beispiel wieder gelöscht werden, sind folgende Schritte zu vollziehen:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> REMOVE GAST
%UAF-I-REMMSG, record removed from system authorization file
%UAF-I-RDBREMMSGU, identifier GAST value [001000,000001] removed from rights database
UAF> EXIT
$ DELETE DISK$USER:[GAST...]*.*.*
$ DELETE DISK$USER:[000000]GAST.DIR.
```

Hierbei ist das erste DELETE-Statement unter Umständen mehrmals auszuführen, falls Unterverzeichnisse unter DISK$USER:[GAST] existieren, die, falls sie wiederum Unterverzeichnisse enthalten, nicht einem einzigen Durchlauf entfernt werden können.

11 Boot und Shutdown eines OpenVMS-Systems

11.1 Grundlagen

Bevor ein System wie eine VAX oder ALPHA unter OpenVMS genutzt werden kann, muß das Betriebssystem geladen werden – dieser Vorgang wird allgemein als *Boot* bezeichnet (abgesehen von der IBM-Mainframe-Welt, wo der analoge Mechanismus als *IPL*, *I*nitial-*P*rogram-*L*oad benannt ist).

Vor dem Abschalten einer solchen Maschine muß ein *Shutdown* durchgeführt werden, um sicherzustellen, daß keine Daten verloren gehen, die unter Umständen noch nicht auf die Platten geschrieben wurden, daß alle Applikationen ordnungsgemäß beendet wurden, etc.

11.2 Booten des Systems

Die Bootvorgänge von VAX- oder ALPHA-Systemen unterscheiden sich mitunter stark voneinander (gerade, was die Angabe spezieller Parameter an den Bootloader betrifft), so daß für eine eingehendere Beschreibung dieses Vorganges auf die Dokumentation des jeweils vorhandenen Systems verwiesen sei.

Im allgemeinen werden die genannten Systems (unter der Voraussetzung, daß alle konfigurierbaren Parameter, wie das gewünschte Bootdevice, etc., korrekt konfiguriert wurden) durch das Kommando B gebootet.

Im Verlauf des Bootvorganges werden eine Reihe von Kommandoprozeduren und Programmen gestartet und ausgeführt, die dafür sorgen, daß alle Subsysteme gestartet werden, Page- und Swapfiles installiert werden, usf. Die zentrale Startupprozedur ist hierbei SYS$MANAGER:SYSTARTUP_VMS.COM (auf VMS-Systemen *vor* Version 6.0 war diese Prozedur noch unter SYS$MANAGER:SYSTARTUP_V5, etc., aufzufinden!). Für eine kurze Beschreibung des Aufbaus dieser Prozedur, siehe Abschnitt 11.3.

11.3 Die zentrale Startup-Prozedur

Zentraler Bestandteil des Startupvorganges eines OpenVMS-Systems ist die Kommandoprozedur SYS$MANAGER:SYSTARTUP_VMS.COM. An dieser Stelle werden alle Kommandos eingetragen, die bei einem Neustart des vorliegenden Systems automatisch ausgeführt werden sollen. Hierunter fallen beispielsweise Startupprozeduren für Produkte, wie Compiler, Datenbanken, etc., aber auch die Startup-Aufrufe für installierte Netzwerkprodukte und -protokolle, wie DECnet, TCP/IP, usw.

Auch das Mounten allgemein verfügbarer Platten kann und sollte an dieser Stelle vorgenommen werden, wobei die einzelnen MOUNT-Kommandos nur dann zur

Ausführung gelangen sollten, wenn das betreffende Device auch vorhanden ist, was beispielsweise mit Hilfe der lexical function F$GETDVI überprüft werden kann. Soll beispielsweise die Platte DKA100 mit dem Label USERDISK gemountet werden, könnte folgende Kommandozeile verwendet werden:

```
$ IF F$GETDVI ("DKA100:", "EXISTS") THEN MOUNT DKA100: USERDISK
```

Das Kommando MOUNT DKA100: USERDISK gelangt hierbei nur dann zur Ausführung, wenn das gewünschte Device vorhanden ist, d.h. der Aufruf von F$GETDVI den Rückgabewert TRUE liefert.

11.4 Die Shutdown-Prozedur

Analog zur eben angesprochenen Startupprozedur existiert eine Shutdownprozedur unter SYS$MANAGER:SYSHUTDOWN.COM, in der alle Kommandos eingetragen werden können, die bei einem Shutdown des Systems automatisch ausgeführt werden sollen. Hierunter fallen beispielsweise Shutdownprozeduren für Datenbanksysteme, für Netzwerkprotokolle und -produkte, etc.

11.5 Shutdown des Systems

Der eigentliche Shutdown eines Systems sollte vom Systemmanager-Account aus gestartet werden (d.h. als User SYSTEM). Er wird eingeleitet durch den Aufruf

```
$ @SYS$SYSTEM:SHUTDOWN
```

Diese Routine sorgt dafür, daß alle eventuell noch nicht auf Platte geschriebenen Daten, die sich unter Umständen also nur im Cache befinden, zurückgeschrieben werden, daß eventuell Anpassungen von Parametern innerhalb eines OpenVMS-Clusters vorgenommen werden, etc.

Unter normalen Umständen sollte ein OpenVMS-System *niemals* ohne vorheriges Ausführen von @SYS$SYSTEM:SHUTDOWN abgeschaltet werden. Für schwere Fälle existiert das Programm OPCRASH.EXE, das einen gezielten Crash des Systems erzwingt und schneller zum Halt-Zustand führt, als ein geordneter Shutdown. Auch OPCRASH.EXE sollte nur im Notfall eingesetzt werden, da hierbei beispielsweise die lokale Shutdownprozedur nicht ausgeführt wird, etc.