SOFTWARE

WEGA

SYSTEMHANDBUCH

EAW *electronic*

P8000

Version 1.1 (2007-10-26)

WEGA - Software

Systemhandbuch

Diese Dokumentation wurde von einem Kollektiv des Kombinates

VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"

erarbeitet.

Herausgeber:
Kombinat
VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Hoffmannstrasse 15-26
Berlin
1193

Verantwortlicher Bearbeiter: P. Hoge

WAE/03-0200-03

Ausgabe: 1/89

Die vorliegende Dokumentation unterliegt nicht dem Aenderungsdienst.

Spezielle Hinweise zum aktuellen Stand der Softwarepakete befinden sich in README-Dateien auf den entsprechenden Vertriebsdisketten.

WEGA-Systemhandbuch

Inhaltsverzeichnis

1. Einleitung

WEGA ist das Hard-Disk-Betriebssystem fuer das Programmier- und Entwicklungssystem P8000. Es ist ein Multi-User-Betriebssystem und kompatibel zum international standardisierten Betriebssystem UNIX System III. Unter dem Betriebssystem WEGA koennen gleichzeitig mehrere Benutzer ueber maximal 8 Terminals am P8000 arbeiten. Jeder dieser Benutzer kann dabei mehrere Programme parallel ausfuehren lassen und seine Dateien vor unberechtigten Zugriffen schuetzen.

Fuer die Systempflege dieses komplexen Softwaresystems ist ein Systemverwalter (Administrator) erforderlich. Der Systemverwalter hat die unterschiedlichsten Aufgaben wahrzunehmen, wie:

- Starten und Abfahren des Systems
- Einrichten und Loeschen von Benutzern
- Pflege der Dateisysteme
- Sicherung der Dateien auf einem Backup-Medium
- Installation und Konfiguration des Betriebssystems
- Installation von Programmpaketen usw.

Durch diesen Dokumentationsband soll er in seiner Arbeit unterstuetzt werden.

Der Systemverwalter muss sich im Betriebssystem eine Sonderstellung verschaffen. Er muss Superuser werden, indem er sich mit dem Benutzernamen "wega" bei einem unter Steuerung des Betriebssystems WEGA bereits laufenden P8000 anmeldet (WEGA login: wega). Nach der Eingabe des Passwortes hat er uneingeschraenkten Zugriff auf alle Dateien und Programme. Das Passwort lautet bei Auslieferung des Systems "root". Es sollte nach der erstmaligen Meldung von WEGA mit dem Kommando passwd(1) geaendert werden. Ist der Systemverwalter bereits als normaler Benutzer im System angemeldet, kann er durch Eingabe des Kommandos su(1) die Zugriffsrechte eines Superusers erlangen.

Die Notation von su(1) entspricht der Beschreibung im WEGA-Programmierhandbuch. Der in Klammern eingeschlossene Ausdruck verweist auf die entsprechende Sektion. Die Sektion M enthaelt die Beschreibung aller Kommandos, die nur fuer den Superuser verfuegbar sind. Diese Sektion bildet den zweiten Teil dieses Dokumentationsbandes.

Beachten Sie bitte !

Einige Superuser-Kommandos sind C-Shell-Prozeduren. Damit diese C-Shell-Prozeduren fehlerfrei arbeiten, sollten die Umgebungsparameter (environment) bei einem Superuser nicht veraendert werden. Das betrifft vor allem die C-Shell-Option "noclobber", die nicht gesetzt werden darf. Programme, wie adduser(1), funktionieren mit dieser Option nicht.

2. Start- und Haltprozeduren

2.1. Voraussetzungen zum WEGA-Start

Voraussetzung zum Start des Betriebssystems WEGA ist eine formatierte und mit dem Betriebssystem WEGA und allen erforderlichen Dateien installierte Systemplatte (Hard-Disk-Laufwerk im Winchester-Beisteller). Das P8000 wird standardmaessig mit einer vollstaendig installierten Systemplatte ausgeliefert. Abschnitt 3 befasst sich mit der Neuinstallation einer Systemplatte.

2.2. Start des Betriebssystems WEGA

Die einzelnen Komponenten des P8000 sind in folgender Reihenfolge einzuschalten:

- P8000-Systemterminal
- P8000-Grundgeraet
- P8000-Winchester-Beisteller
- Drucker
- eventuell weitere Terminals

(Vorhandene Drucker sollten stets eingeschaltet werden. Das Betriebssystem WEGA beginnt sofort nach dem Start mit der Ausfuehrung von eventuell noch vorhandenen Druckauftraegen. Die Terminals koennen bei einem sich in Betrieb befindlichen P8000 beliebig ein- oder ausgeschaltet werden. Sie werden vom Betriebssystem WEGA dementsprechend behandelt.)

Nach dem Einschalten des P8000 bzw. nach Betaetigung der Reset-Taste am P8000-Grundgeraet ist der 8-Bit-Mikrorechnerteil des P8000 aktiv. Nach jedem Einschalten des P8000 wird ein Hardware-Eigentest fuer den 8-Bit-Mikrorechnerteil ausgefuehrt. Das P8000 meldet sich mit dem U880-Softwaremonitor. Die WEGA-Startdiskette wird nun in das Floppy-Disk-Laufwerk 0 eingelegt und die Return-Taste am Systemterminal betaetigt. Der 16-Bit-Mikrorechnerteil wird aktiviert. Er meldet sich mit dem U8000-Softwaremonitor und fordert zum Start von WEGA auf.

Durch Betaetigung der NMI-Taste am P8000-Grundgeraet wird der Start von WEGA in den Multi-User-Mode eingeleitet. Dieser Vorgang wird auch als automatischer WEGA-Start bezeichnet. Er beginnt mit dem Hardware-Eigentest fuer den 16-Bit-Mikrorechnerteil. Der U8000-Softwaremonitor liest nun das primaere Bootstrap-Programm vom physischen Block 0 der Systemplatte ein und startet es. Anschliessend werden das sekundaere Bootstrap-Programm (vom ersten Dateisystem der Systemplatte) und der WEGA-Kern (vom root-Dateisystem) geladen und gestartet. Vom Kern wird das Programm INIT (init(M)) aufgerufen. Das Programm INIT veranlasst die Ausfuehrung der Shell-Prozedur /etc/rc und diese die Ausfuehrung der C-Shell-Prozedur /etc/rc_csh.

Neben anderen Aktivitaeten ruft rc_csh das Programm fsck(M) auf. Dieses prueft die Konsistenz der Dateisysteme der Hard-Disk. Treten dabei Fehler auf, die nicht automatisch beseitigt werden koennen, faehrt das System in den Single-User-Mode. Der Systemverwalter sollte nun die Fehler untersuchen und beseitigen (siehe Abschnitt 4.6.).

Nachdem /etc/rc_csh verschiedene Systemueberpruefungen abgearbeitet hat, fordert es mit datem(M) zur Eingabe des aktuellen Datums und der Uhrzeit auf. Anschliessend werden die Dateisysteme mit mfs(M) bzw. mount(M) eingebunden und Hintergrundprozesse, wie dqueuer(M) und cron(M), aktiviert. Ein erfolgreicher Systemstart wird durch die Ausschrift "WEGA login:" auf dem Systemterminal und allen eingeschalteten Terminals angezeigt.

Zusammengefasst ergibt sich fuer den Start von WEGA in den Multi-User-Mode folgender Ablauf:

1. Einschalten des P8000 (Systemterminal, P8000-Grundgeraet, Winchester-Beisteller, Drucker) bzw.
   Betaetigen der Reset-Taste am P8000-Grundgeraet.
   Der U880-Softwaremonitor meldet sich mit dem Prompt-Zeichen ">".

2. Einlegen der WEGA-Startdiskette in das Floppy-Disk-Laufwerk 0.

3. Return-Taste des Systemterminals betaetigen.

4. Betaetigen der NMI-Taste am P8000-Grundgeraet nach der Ausschrift:

   "U8000-Softwaremonitor Version x.y - Press NMI"

5. Eingabe des Datums nach der Ausschrift:

   "Last known date and time: ..."
   "Enter Date (MM/DD/YY or <cr>):"

   MM, DD, YY bedeuten Monat, Tag, Jahr (12/24/87 fuer 24.12.1987).

6. Eingabe der Uhrzeit nach der Ausschrift:

   "Enter Time (HH:MM):"

   HH, MM bedeuten Stunden und Minuten (13:10 fuer 13.10 Uhr).

   Beispiel:

   Last known date and time: Thu Dec 24 13:15:45 1987
   Enter Date (MM/DD/YY or <cr>): 12/31/87
   Enter Time (HH:MM): 23:59
   Thu Dec 31 23:59:00 MEZ 1987

7. Der erfolgreiche Start von WEGA wird durch folgende Ausschriften angezeigt:

   "Going multi-user!"
   "WEGA login:"

2.3. Manueller WEGA-Start

Beim manuellen Start wird das Betriebssystem WEGA mit Hilfe der Bootstrap-Programme in den Single-User-Mode gefahren. Im Single-User-Mode ist nur das Systemterminal aktiv. Fuer bestimmte Aufgaben zur Pflege und Wartung des Systems ist dieser Modus erforderlich. Der Benutzer des Systemterminals ist automatisch Superuser. Der Benutzer sollte daher der Systemverwalter sein.

Die Bootstrap-Programme und der WEGA-Kern werden ueber Kommandoeingaben gestartet. Das System kann bei Bedarf auch im sekundaeren Bootstrap-Programm (Stand-alone-Mode) verbleiben. Hier sind Programme zur Installation der Dateisysteme von WEGA verfuegbar. Ueber eine Kommandoeingabe kann das Betriebssystem WEGA aber auch vom Single-User- in den Multi-User-Mode wechseln.

Das primaere und sekundaere Bootstrap-Programm, der WEGA-Kern sowie alle Stand-alone-Programme koennen von der Hard-Disk, von einer Floppy-Disk im UDOS-Format (WEGA-Startdiskette), von einer Floppy-Disk im WEGA-Dateisystemformat oder von einem mit dem P8000 gekoppelten lokalen Rechner eingelesen werden. (Der lokale Rechner, z.B. ein Personalcomputer PC 1715 mit dem 8-Bit-Betriebssystem UDOS, wird dann anstelle des Systemterminals angeschlossen. Auf dem lokalen Rechner muss das Remote-Programmpaket installiert sein.)

Zusammengefasst ergibt sich fuer den Start von WEGA in den Single-User-Mode folgender Ablauf:

1. Einschalten des P8000 (Systemterminal, P8000-Grundgeraet, Winchester-Beisteller, Drucker) bzw.
   Betaetigen der Reset-Taste am P8000-Grundgeraet.
   Der U880-Softwaremonitor meldet sich mit dem Prompt-Zeichen ">".

2. Einlegen der WEGA-Startdiskette in das Floppy-Disk-Laufwerk 0.

3. Return-Taste des Systemterminals betaetigen.

4. Start des primaeren Bootstrap-Programms nach der Ausschrift

   "U8000-Softwaremonitor Version x.y - Press NMI"

   mit einer der folgenden Kommandoeingabe:

   O D - Laden von Systemplatte (Block 0)
   O U - Laden von UDOS-Diskette (WEGA-Startdiskette)
   O F - Laden von WEGA-Diskette (Block 0)
   O R - Laden vom lokalen Rechner

   Das primaere Bootstrap-Programm meldet sich mit dem Prompt-Zeichen ">".

5. Start des sekundaeres Bootstrap-Programms

   Das sekundaere Bootstrap-Programm wird vom Speichermedium des primaeren Bootstrap-Programms geladen, indem der Dateiname eingegeben wird. Der Name "boot" ist vereinbart, wenn nur das Zeichen "<cr>" eingegeben wird. Soll das Bootstrap-Programm von der Systemplatte geladen werden, muss sich die Datei im ersten Dateisystem der Platte (siehe Abschnitt 4.1.) befinden. Ein vollstaendiger Pfadname (pathname) kann als Dateiname angegeben werden.

   Beispiel einer Eingabe:

   >boot

   Das sekundaere Bootstrap-Programm meldet sich mit der Ausschrift

   "boot"

   und dem Prompt-Zeichen ":".

6. Start des WEGA-Kerns mit einer der folgenden Kommandoeingaben:

   md(0,16000)wega - Laden von Systemplatte
   ud(0,0)wega - Laden von UDOS-Diskette (WEGA-Startdiskette)
   fd(0,0)wega - Laden von WEGA-Diskette (WEGA-Dateisystemformat)
   rm(0,0)wega - Laden vom lokalen Rechner

   Bei Eingabe von "<cr>" ist "md(0,16000)wega" voreingestellt, analog zum automatischen Start. WEGA meldet sich im Single-User-Mode mit dem Prompt-Zeichen "#".

7. Wechsel in den Multi-User-Mode mit:

   INIT 2

   Das System verlaesst den Single-User-Mode und geht in den Multi-User-Mode. Es wird der gleiche Zustand wie nach einem automatischen Systemstart erreicht.

2.4. Start von WEGA ohne Startdiskette

Der 16-Bit-Mikrorechnerteil des P8000 wird mit der Eingabe eines Kommandos vom U880-Softwaremonitor aktiviert und meldet sich wie oben beschrieben. Analog Abschnitt 2.2. oder 2.3. kann nun ein automatischer oder manueller WEGA-Start erfolgen. Unter WEGA koennen nun aber nur die seriellen Kanaele des 8-Bit-Mikrorechnerteils benutzt werden. Alle anderen Komponenten des 8-Bit-Mikrorechnerteils (z.B. die Floppy-Disk-Laufwerke) sind nicht verfuegbar.

Der 16-Bit-Mikrorechnerteil wird wie folgt aktiviert:

1. Einschalten des P8000 (Systemterminal, P8000-Grundgeraet, Winchester-Beisteller, Drucker) bzw.
   Betaetigen der Reset-Taste am P8000-Grundgeraet.
   Der U880-Softwaremonitor meldet sich mit dem Prompt-Zeichen ">".

2. Eingabe des Kommandos: X<cr>

2.5. Abfahren des Betriebssystems WEGA

Das Betriebssystem WEGA wird vom Superuser mit dem Kommando down(M) oder halt(M) automatisch abgefahren. Das System geht dann in den Single-User-Mode und sichert die Daten der Dateisysteme.

Der Abfahrvorgang kann auch manuell ausgefuehrt werden:

1. Information aller angemeldeten Benutzer mit wall(M)

2. Eingabe von:

   INIT 1 oder
   kill -1 1

   Diese Kommandos beenden alle Prozesse im Multi-User-Mode. Das System wechselt in den Single-User-Mode.

3. Eingabe von: sync;sync

4. Reset-Taste am P8000-Grundgeraet betaetigen

5. Ausschalten des P8000, falls gewuenscht.
   Der P8000-Winchester-Beisteller muss vor dem P8000-Grundgeraet ausgeschaltet werden.

Beachten Sie bitte folgende Punkte !

1. Reset-Taste und Netzschalter duerfen im Normalfall erst nach dem ordnungsgemaessen Abfahren des Systems betaetigt werden. Ein falsches Abschalten kann zu Datenverlusten und inkonsistenten Dateisystemen fuehren. Unter unguenstigen Bedingungen kann eine Neuinstallation der Hard-Disk erforderlich werden.

2. Vor dem Wiedereinschalten des Winchester-Beistellers ist eine Pause von mindestens 6 min einzuhalten.

3. Soll der Winchester-Beisteller transportiert werden, ist zuvor das Stand-alone-Programm sa.shipdisk (siehe Abschnitt 2.6.6.) auszufuehren.

2.6. Stand-alone-Mode

Das sekundaere Bootstrap-Programm hat den Charakter eines eigenen kleinen Betriebssystems. Im Stand-alone-Mode laufen verschiedene Programme zur Initialisierung und Installation der Hard-Disk als Systemplatte. Ferner koennen Diagnose-Programme abgearbeitet und der WEGA-Kern geladen werden. Es koennen segmentierte oder nichtsegmentierte Programme ausgefuehrt werden.

Kommandos besitzen im Stand-alone-Mode folgende Syntax:

xx(n,m)name

xx: E/A-Geraet

- md - Hard-Disk
- fd - Floppy-Disk (WEGA-Format)
- ud - Floppy-Disk (UDOS-Format)
- rm - Lokaler Rechner ueber Remote-Programm

n: physische Laufwerknummer
(bei ud und rm 0 eingeben)

m: Blocknummer, ab der das Dateisystem physisch beginnt
(bei ud und rm 0 eingeben)

name: Dateiname
Der Dateiname ist fuer die E/A-Geraete-Treiber md und fd optional. Die Angabe eines Dateinames setzt eine Dateisystemstruktur auf dem Speichermedium voraus. Dateien koennen im Stand-alone-Mode nur zum Lesen eroeffnet werden. Es kann ein vollstaendiger Pfadname angegeben werden. Fehlt der Dateiname, wird das gesamte E/A-Geraet fuer einen Blockzugriff zum Lesen oder Schreiben eroeffnet.

Beispiele:

```
md(0,16000)sa.mkfs
md(0,0)sys/conf/wega
ud(0,0)sa.format
```

2.6.1. sa.format - Formatieren der Hard-Disk

Das Stand-alone-Programm sa.format dient zum Formatieren der Hard-Disk. Die Platte wird dabei physisch initialisiert, um Schreib- und Lesezugriffe zu ermoeglichen. Die logische Initialisierung der Platte bzw. der Dateisysteme der Platte erfolgt mit dem Programm sa.mkfs (siehe Abschnitt 2.6.3.) oder mit mkfs(M). Nach dem Formatieren wird die Platte zur Kontrolle gelesen. Es koennen "harte" Fehler auftreten, d.h. es koennen Stellen auf der Platte existieren, welche nicht sicher zurueckgelesen werden koennen. Spuren, auf denen solche Fehler auftreten, werden vom Programm sa.format durch mehrfaches Ruecklesen erkannt. Defekte Spuren, die auf dem Gehaeuse der Platte eingetragen sind, muessen manuell in die BTT eingetragen werden.

Die Adressen der defekten Spuren werden in eine Tabelle der defekten Spuren (Bad Track Table, BTT) im Arbeitsspeicher des WDC (Winchester-Disk-Controler) eingetragen und koennen am Ende einer Formatierung auf das Laufwerk geschrieben werden. Nach jedem Reset des WDC bzw. durch ein spezielles Kommando an den WDC wird die BTT von den angeschlossenen Laufwerken in den WDC-RAM eingelesen. Bei normalen Schreib- und Lesezugriffen werden die defekten Spuren ausgeblendet.

Das Programm bietet die Moeglichkeit, beim Formatieren eine schon vorhandene BTT weiter zu benutzen. Es kann ein Bereich einer Platte neu formatiert werden. Eventuell neu hinzukommende defekte Spuren werden in die schon vorhandene BTT eingetragen.

Durch defekte Spuren verringert sich die Kapazitaet der Platte. Bei der Aufteilung der physischen in logische Platten (siehe Abschnitt 4.1.1) wurde dies beruecksichtigt. Je nach Kapazitaet der Platte sind die letzten 200 bis 400 Bloecke bzw. ca. 10 bis 20 Spuren freigehalten. Sollten noch mehr Spuren defekt sein, ist das letzte Dateisystem der Platte zu verkleinern (siehe Abschnitt 5). Es sollte aber auch der Austausch der Platte in Erwaegung gezogen werden. Das Programm sa.format gestattet die Formatierung verschiedener Winchester-Plattentypen.

Nach Aufruf des Programms werden die Firmwareversions-Bezeichnung des WDC und laufwerksabhaengige Parameter ausgegeben:

```
"*** Format Hard-Disk V. X.Y ***"
"  WDC-Firmware-Version: xxxxxxx"
"  Drives: x   Cylinders: xxx   Heads: x"
"  Sectors: xx   Bytes/Sector: xxx"
"  Blocks/Drive: xxxxx"
```

Das Programm fragt nun die Parameter zum Formatieren ab:

" Drive:"
Eingabe der Laufwerknummer (0 oder 1) " Read BTT from HD in WDC-RAM ? "
Es wird abgefragt, ob die Defektspurtabelle des Laufwerks in den WDC-RAM eingelesen werden soll. Bei Eingabe von 'y' oder Bei logischen Fehlern in der Tabelle wird sie ebenfalls geloescht.
Der Inhalt der BTT wird auf dem Systemterminal ausgegeben.

" Manual Input of Bad Track ?"
Nach Eingabe von 'y' oder 'Y' werden die Adressen der defekten Spuren (Zylinder und Kopf) der Tabelle auf dem Laufwerksgehaeuse abgefragt. Die Eingabe von <cr> beendet die Eingabe.

" Format Begin: Cylinder "
Eingabe der Start-Zylindernummer. Der erste Zylinder besitzt die Nummer 0. Als Antwort ist auch 'a' oder 'A' moeglich. In diesem Fall wird die gesamte Platte formatiert. Es erfolgen keine weiteren Abfragen zum zu formatierenden Bereich.

" Head "
Eingabe der Start-Kopfnummer. Die Zaehlung beginnt ab 0.

" Format End: Cylinder "
Eingabe der End-Zylindernummer. Als Antwort ist auch 'e' oder 'E' moeglich. In diesem Fall werden die End-Zylindernummer und die End-Kopfnummer auf die hoechstmoeglichen Werte gesetzt. Es erfolgen keine weiteren Abfragen zum zu formatierenden Bereich. (Letzter Zylinder = Anzahl der Zylinder - 1).

" Head "
Eingabe der Nummer des letztes Kopfes (Anzahl Koepfe -1).

" Ready for format from Cyl x Hd x to Cyl x Hd x :"
Durch Eingabe von 'y' oder 'Y' wird die Formatierung gestartet. Jede andere Eingabe beendet das Programm.

Waehrend des Formatierens wird die aktuelle Spurnummer angezeigt. Aufgetretene defekte Spuren werden angezeigt:

" Format Cyl x Hd x"
" New Entry in BTT: Cyl x Hd x"

Nach Beendigung des Formatierens wird der gesamte Inhalt der BTT auf dem Systemterminal entweder durch

" No Entries in Bad Track Table (BTT)"

(Tabelle enthaelt keine Defektspureintraege) oder durch

" Entries in Bad Track Table:"
" Cyl xxx Hd x"
" ... ..."

angezeigt.
Dann wird nachgefragt, ob die erstellte BTT auf das aktuelle Laufwerk geschrieben werden soll:

" Rewrite BTT from WDC-RAM to HD ? "
Mit der Eingabe von 'y' oder 'Y' erfolgt das Rueckschreiben der BTT. Erst jetzt ist das Laufwerk fuer normale Zugriffe vorbereitet.

Beispiel:

```
*** Format Hard-Disk V. 1.0 ***
  Drive: 0
  Read BTT from HD in WDC-RAM ? n
  No Entries in Bad Track Table (BTT)
  Manual Input of Bad Track ? y
  Bad Track: Cylinder 36
             Head 3
  Bad Track: Cylinder <cr>
  Format Begin: Cylinder a
  Ready for format from Cyl 0 Hd 0 to Cyl 614 Hd 3 ? y
  Entries in BTT
  Cyl xxx Hd x
  Rewrite BTT from WDC-RAM to HD ? y
```

Formatieren einzelner Bereiche

Die Eingabe der Parameter kann auch zum Formatieren einzelner defekter Spuren oder Bereiche der Platte benutzt werden. Das Programm sa.verify ermittelt defekte Spuren der Platte. Konnten die Fehler durch das Formatieren beseitigt werden, muessen nur die Dateisysteme, die auf den defekten Spuren liegen, gegebenenfalls neu installiert werden. Traten beim Formatieren jedoch "harte Fehler" auf, sind das aktuelle und alle folgenden Dateisysteme neu zu installieren, da sich durch Eintraege in die BTT die Berechnung der Blocknummern aendert.

Beispiel:

Es ist ein Sektor auf Zylinder 216, Kopf 1 defekt. Nur diese eine Spur wird neu formatiert:

```
*** Format Hard-Disk V. 1.0 ***
  Drive: 0
  Read BTT from HD in WDC-RAM ? y
  No Entries in Bad Track Table (BTT)
  Manual Input of Bad Track ? n
  Format Begin: Cylinder 216
                Head 1
  Format End:   Cylinder 216
                Head 1
  Ready for format from Cyl 216 Hd 1 to Cyl 216 Hd 1 ? y
  No Entries in BTT
  Rewrite BTT from WDC-RAM to HD ? y
```

2.6.2. sa.verify - Ueberpruefen der Hard-Disk

Das Programm sa.verify ueberprueft die Lesbarkeit der Sektoren eines Hard-Disk-Laufwerks. Eine inhaltliche Pruefung der Datenfelder der Sektoren findet nicht statt. Nach dem Start gibt das Programm einige wesentliche Firmwareparameter des WDC aus:

```
"**** Verify V. X.Y ****"
"  WDC-Firmware-Version: xxxxxxx"
"  Drives: x   Cylinders: xxx   Heads: x"
"  Sectors: xx   Bytes/Sector: xxx"
"  Blocks/Drive: xxxxx"
```

Danach wird die Laufwerknummer abgefragt:

" Drive:"
Eingabe der Laufwerknummer (0 oder 1).

Das Programm sa.verify liest die Defektspurtabelle des Laufwerks ein. Lesefehler fuehren zum Abbruch des Programms. Sind Eintraege vorhanden, wird die Tabelle auf dem Systemterminal dargestellt:

" x Entries in BTT of Drive y"
" Cyl xxx Head x"
" Cyl xxx Head x"

Das Programm fragt nun die Parameter fuer den Verify-Lauf ab:

" Begin: Cylinder "
Eingabe der Start-Zylindernummer.

" End: Cylinder "
Eingabe der End-Zylindernummer.

Anschliessend wird das Ruecklesen gestartet. Die aktuelle Zylindernummer wird ausgegeben:

" Cylinder xxxx"

In der Defektspurtabelle eingetragene Spuren werden beim Ruecklesen uebergangen. Bei Auftreten eines Fehlers werden der Fehlercode sowie die Zylinder- und Kopfnummer des fehlerhaften Sektors ausgegeben. Der Benutzer wird gefragt, ob das Ruecklesen fortgesetzt werden soll:

"Error x --- Dr x Cy xxx Hd x continue ?"
Eingabe von 'y' bewirkt das Fortsetzen des Rueckleseng.

Sind alle Zylinder gelesen, wird das Programm mit der Ausgabe folgender Nachricht beendet:

"Verify complete"

2.6.3. sa.mkfs - Initialisieren von Dateisystemen

Das Stand-alone-Programm sa.mkfs initialisiert ein leeres Dateisystem. Auf dem Speichermedium wird eine Dateisystemstruktur (siehe Abschnitt 7.3.3.) angelegt. Dazu werden der Superblock, die Inode-Liste (i-list) mit den Inodes, die Freiblockliste und das root-Directory erstellt. Die Anzahl der Inodes wird entsprechend der vorgegebenen Dateisystemgroesse ueber einen festen Teiler berechnet. Der Sektorversatz und die Anzahl der Sektoren pro Zylinder sind im Programm auf 1 (kein Versatz) und 72 eingestellt.

Das Programm sa.mkfs erwartet folgende Eingaben:

"file system size:"

Es wird die Groesse des Dateisystems in Bloecken (je 512 Byte) eingegeben.

"file system:"

Es ist der Name des Dateisystems, in der Form xx(n,m) (siehe Abschnitt 2.6.) einzugeben.

Beispiel:

```
file system size: 7000
file system: md(0,16000)
```

2.6.4. sa.install - Installieren von Dateisystemen

Das Programm sa.install dient zum Installieren des root- oder /usr-Dateisystems der Systemplatte. Dazu werden die WEGA-Vertriebsdisketten des entsprechenden Dateisystems benutzt. Zur Steuerung der Installation ist auf jeder Diskette die Datei CONTENTS vorhanden. Sie wird als erste Datei jeder Diskette eingelesen. Das Programm sa.install gestattet die vollstaendige Installation aller Dateien der Disketten, laesst aber auch eine dialogorientierte Auswahl und ein schnelles Nachladen von Dateien zu. Eine eventuelle Auswahl sollte jedoch nur bei genauer Systemkenntnis erfolgen.

Folgende Eingaben sind notwendig:

"Enter Date (MM/DD/YY) : "

Eingabe des Installationsdatums (Monat/Tag/Jahr).

"input file system : "

Es wird der Name des E/A-Geraetes bzw. des Dateisystems in der Form xx(n,m) erwartet, von dem die Vertriebsdisketten eingelesen werden sollen (fd(0,0) oder ud(0,0)).
Nach der Definition des Eingabegeraetes, wird die Datei CONTENTS von diesem Geraet eingelesen.

"output file system : "

Es wird der Name des zu installierenden Dateisystems in der Form xx(n,m) erwartet.

Beispiel:

```
Enter Date (MM/DD/YY) : 4/3/87
input file system : fd(0,0)
output file system : md(0,16000)
```

Das Eintragen der Dateien in ein Dateisystem kuendigt sa.install auf dem Systemterminal an:

"..... (n/y/a/A/q/Q) ? :"

Der Systemverwalter kann eine eventuelle Auswahl treffen:

A Alle Dateien, die in der Datei CONTENTS enthalten sind, werden installiert.

a Alle Dateien, die in der Datei CONTENTS dem aktuellen Directory zugeordnet sind, werden installiert. Das aktuelle Directory wird angezeigt.

y Die angezeigte Datei wird installiert.

n Die angezeigte Datei wird nicht installiert.

q Alle folgenden Dateien, die in der Datei CONTENTS dem aktuellen Directory zugeordnet sind, werden nicht installiert.

Q Alle folgenden Dateien, die in der Datei CONTENTS enthalten sind, werden nicht installiert.

Nach der Installation der Dateien einer Diskette fordert sa.install mit

"next input disk ? (y/n) : "

zum Einlegen der naechsten WEGA-Vertriebsdiskette auf. Nach Eingabe von "y" (yes) wird die Datei CONTENTS der neuen Diskette eingelesen. Vor Eingabe von "y" muss die Diskette bereits gewechselt sein. Nach der letzten Diskette des aktuellen Dateisystems ist "n" (no) einzugeben. Das Programm sa.install ist beendet.

2.6.5. sa.cat - Ausgabe einer Datei

Das Stand-alone-Programm sa.cat gibt eine druckbare Datei auf dem Systemterminal aus. Das Programm sa.cat erwartet die folgende Eingabe:

"File:"

Es ist der Name des Datei, in der Form xx(n,m)name (siehe Abschnitt 2.6.) einzugeben.

Beispiel:

File: md(0,16000)/etc/inittab

2.6.6. sa.shipdisk - Transportvorbereitung Hard-Disk

Die Schreib-/Lesekoepfe der Winchester-Laufwerke muessen zum Transportieren in eine Parkposition gefahren werden. Das Stand-alone-Programm sa.shipdisk realisiert diesen Vorgang. Das Programm erwartet keine Eingaben. Nach der Beendigung des Programms ist sofort die Reset-Taste am P8000-Grundgeraet zu betaetigen.

Beispiel:

ud(0,0)sa.shipdisk
Reset-Taste am P8000-Grundgeraet betaetigen.

2.6.7. sa.timer - Stellen des Uhrenmoduls

Mit dem Stand-alone-Programm sa.timer kann der Uhrenmodul im P8000-Compact neu gestellt werden. Ist der Uhrenmodul nicht vorhanden, wird die Fehlermeldung

"Timer not available"

ausgegeben. Das Programm endet. Ist der Uhrenmodul installiert, werden Datum und Zeit angezeigt und das Programm fordert zur Neueingabe auf. Bei Eingabe von Return bleibt der alte Wert erhalten.

"Date is: MM/DD/YY"
"Time is: HH:MM:SS"
"Enter new Date (MM/DD/YY) : "
(Eingabe neues Datum oder Return)
"Enter new Time in GMT (HH:MM) : "
(Eingabe neue Uhrzeit in GMT oder Return)
"Enter Return to start time : "
(Eingabe von Return startet die Uhr)
"Date is: MM/DD/YY"
"Time is: HH:MM:SS"
(Anzeige neues Datum und Zeit)

2.6.8. sa.diags - Hardware-Diagnose

Mit dem Stand-alone-Programm sa.diags wird dem Anwender die Moeglichkeit gegeben, individuelle Hardware-Tests fuer den 16-Bit-Mikrorechnerteil beliebig oft abzuarbeiten. Empfohlen wird die Verwendung dieses Programms besonders fuer das Auffinden von Fehlern, die nur zeitweilig auftreten. In solch einem Fall sollte man das Diagnoseprogramm, bzw. den entsprechenden Einzeltest, laengere Zeit laufen lassen.

Fuer den Aufruf von sa.diags wird zunaechst das

primaere Bootstrap-Programm (!!!)

von der WEGA-Startdiskette geladen, das sich mit dem Prompt-Zeichen ">" meldet. Danach wird durch die Eingabe von "sa.diags <cr>" das Diagnoseprogramm von der WEGA-Startdiskette geladen und gestartet. Es meldet sich mit der Ausschrift "Stand alone Hardware-Eigentest".

Das Diagnoseprogramm ist als ein Baum von Menues aufgebaut. Menuebestandteile sind entweder Untermenuenamen oder Funktionen, die ausgefuehrt werden. Das Programm kommentiert sich selbst. Dem Anwender werden jeweils alle Eingabemoeglichkeiten angezeigt. Durch Eingabe eines Kommandos wird eine Auswahl getroffen und man gelangt in das naechste Menue. Es besteht die Moeglichkeit Einzeltests aufzurufen, fuer diese einen Wiederholungszaehler einzustellen und sie dann abzuarbeiten.

Weiterhin ist es moeglich, die interne Testliste anzuzeigen, in der alle Einzeltests enthalten sind. Diese Liste kann vollstaendig, oder beginnend bei einer als "aktuell" gekennzeichneten Zeile, abgearbeitet werden. Dabei kann fuer jeden der Tests die Anzahl der Wiederholungen individuell festgelegt werden.

Werden bei der Abarbeitung der Tests Hardware-Fehler festgestellt, so erfolgt die Fehlerausgabe ueber das angeschlossene Terminal. Es werden eine Fehlernummer und max. 4 Fehlerparameter angezeigt. Die entsprechenden Erlaeuterungen sind der Fehlerliste fuer den Hardware-Eigentest P8000-Grundgeraet im Band: "Einfuehrung in die Software des Geraetesystems P8000", Anhang B, zu entnehmen.

Das Diagnoseprogramm enthaelt gegenwaertig folgende Tests:

- EPROM-CRC-Test
- SRAM-Speichertest
- UA855-PIO-Peripheriebausteintest
- UA857-CTC-Peripheriebausteintest
- UA856-SIO-Peripheriebausteintest
- DRAM-Speichertest

Weitere Tests sind in Vorbereitung.

Zum Programm sa.diags existiert eine gesonderte Dokumentation im Sammelband "WEGA-Dienstprogramme".

2.7. Single-User-Mode

Im Single-User-Mode (Einbenutzerbetrieb) des Betriebssystems WEGA kann der Systemverwalter verschiedene Funktionen zur Pflege und Wartung des Software-Systems ausfuehren, Konfigurationen und Installationen vornehmen. Diese Aufgaben sind oft nicht mit dem Multi-User-Betrieb vertraeglich. Die Bedienung erfolgt ueber das Systemterminal, alle anderen Terminals sind nicht aktiv. Der Systemverwalter ist im Single-User-Mode automatisch Superuser. Der Start des P8000 in den Single-User-Mode wird im Abschnitt 2.3. und der Wechsel vom Multi-User- in den Single-User-Mode im Abschnitt 2.5. beschrieben. Die mount-Verbindungen der Dateisysteme wird dabei nicht hergestellt bzw. automatisch geloest. Der Wechsel vom Single-User- in den Multi-User-Mode erfolgt mit

```
INIT 2
```

2.8. Einrichten und Loeschen von Benutzern

Unter WEGA existieren die Shell-Programme adduser(M) und rmuser(M) zur Installation von neuen Benutzern im System bzw. zum Entfernen von Benutzern aus dem System. Die Programme duerfen nur ausgefuehrt werden, wenn alle Dateisysteme der Systemplatte mit mount eingebunden sind. Der Aufruf der Programme erfolgt daher am besten im Multi-User-Mode. Beide Programme arbeiten dialogorientiert.

2.8.1. Benutzer einrichten

Das Programm adduser ermittelt aus der Datei /etc/passwd (siehe passwd(5)) eine Benutzer-Identifikation (userid), indem das hoechste vorhandene Benutzer-Id inkrementiert wird. Die Datei /etc/passwd enthaelt Informationen ueber alle Benutzer, wie Benutzer-Id, Passwort (verschluesselt), Home-Directory (Directory, in das nach dem Login gewechselt wird), Login-Shell (im allgemeinen /bin/csh, da diese mehr Moeglichkeiten als die Standard-Shell /bin/sh bietet) und Gruppen-Id. Die meisten Benutzer sind der Gruppe "other" zugeordnet. Gruppen sind in der Datei /etc/group (siehe group(5)) vereinbart. Ein Benutzer kann Mitglied mehrerer Gruppen sein. Die Gruppenzugehoerigkeit kann geaendert werden (siehe newgrp(1) und setgid(2)). Mit dem Login wird ein Benutzer Mitglied seiner Login-Gruppe. Das Programm adduser fragt nach einem Initialisierungs-Passwort fuer den neuen Benutzer. Nach dem ersten Login sollte es vom Benutzer mit passwd(1) geaendert werden.

Zusaetzlich koennen der Systemverwalter oder der Benutzer in dem Home-Directory des Benutzers eine Shell-Initialisierungsdatei anlegen. Diese besitzt bei Benutzung der C-Shell (/bin/csh) den Namen .cshrc und bei Benutzung der Standard-Shell (/bin/sh) den Namen .profile.

Eine einfache .cshrc koennte so aussehen:

```
#
set path = (. /bin /usr/bin /z/bin)
```

Die erste Zeile (#) teilt der Shell mit, dass es sich um eine C-Shell-Prozedur und nicht um eine Standard-Shell-Prozedur handelt. Die zweite Zeile gibt die Reihenfolge der Directorys an, die nach Kommandos durchsucht werden. Das Beispiel zeigt allerdings die schon in der Shell voreingestellte Reihenfolge. Zur Einbeziehung eines privaten bin-Directorys des Benutzers kann die Zeile geaendert werden in:

```
set path = (. ~/bin /bin /usr/bin /z/bin)
```

2.8.2. Benutzer loeschen

Das Shell-Programm rmuser(M) entfernt Benutzer aus dem System. Alle Gruppeneintraege werden geloescht. Der Systemverwalter kann ausserdem das Passwort des Benutzers aendern, um damit den Zugang zum System zu sperren. Der Eintrag in /etc/passwd bleibt jedoch bestehen. Er wird fuer die Identifikation der noch vorhandenen Dateien des ehemaligen Benutzers benoetigt. Das Programm rmuser sucht diese Dateien im System und legt die Dateinamen in der Datei files.<user> ab. Der Systemverwalter kann ueber die Dateien verfuegen.

2.9. Konfiguration der seriellen Kanaele

Das P8000 besitzt 8 serielle Kanaele, vier auf dem 8-Bit- und vier auf dem 16-Bit-Mikrorechnerteil. Alle Kanaele sind unter WEGA verfuegbar. Bei Auslieferung des P8000 ist WEGA fuer den Anschluss von 7 Terminals (V.24 oder IFSS, 9600 Baud, Terminaltyp: P8000-Terminal) und fuer den Anschluss eines Druckers (V.24 oder IFSS, 9600 Baud, XON/XOFF- oder Hardware-Protokoll ueber DTR) installiert:

Kanal	Name in /dev
tty0	tty0
tty1	console
tty2	tty2
tty3	lp
tty4	tty4
tty5	tty5
tty6	tty6
tty7	tty7

Das Systemterminal wird als "console" bezeichnet. Es entspricht tty1 und kann wie ein normales Terminal benutzt werden. Zusaetzlich erscheinen auf ihm Systemfehlermeldungen. Die seriellen Kanaele lassen sich durch den Superuser

im Single-User-Mode konfigurieren.

2.9.1. Modem anschliessen

Die seriellen Kanaele koennen zum Anschluss eines Terminals (abgeruestete V.24- oder IFSS-Schnittstelle) oder zum Anschluss eines Modems (V.24) installiert werden. Die Hardware des P8000 stellt nur mit Kanal tty4 eine unter WEGA nutzbare voll ausgebaute V.24-Schnittstelle bereit. Tty4 und tty5 sind nicht als IFSS-Schnittstellen nutzbar. Zur Konfiguration wird das Kommando ttyconfig(M) (siehe auch mdmctl(2)) benutzt. Es wird von der Startprozedur /etc/rc_csh aufgerufen. Sie enthaelt die Kommandozeile:

```
ttyconfig -t 0-7
```

Bei Konfiguration von tty4 als Modem aendert sie sich in:

```
ttyconfig -t 0-3, 5-7 -m 4
```

Die Standardeinstellung des Systems ist Terminalmodus fuer alle Kanaele.

Im Modemmodus erfolgt automatisch ein Logout, sobald das DTR-Signal inaktiv wird. Dem Terminalkanal zugeordnete Prozesse werden beendet.

2.9.2. Terminaltyp aendern

In der Datei /etc/ttytype (siehe ttytype(5)) erfolgt eine Zuordnung zwischen Terminalkanal und dem Typ jedes angeschlossenen Terminals. Der Typ wird durch maximal 7 Zeichen abgekuerzt und muss in der Datei /etc/termcap (siehe termcap(5)) enthalten sein.

In der Datei /etc/termcap sind die Parameter und Funktionen ausgewaehlter Terminaltypen definiert. Der Systemverwalter kann weitere Typen eintragen. Verschiedene Programme, z.B. der Editor vi, benutzen diese Informationen fuer die Bildschirmsteuerung.

Die Eintraege in /etc/ttytype koennten wie folgt aussehen:

```
P8 tty0
P8 console
P8 tty2
P8 tty3
PV tty4
PV tty5
PV tty6
PV tty7
```

P8 ist die Abkuerzung fuer P8000-Terminal in der Betriebsart ADM31-

Steuerzeichenfolgen (abgeruesteter Funktionsumfang).
PV ist die Abkuerzung fuer P8000-Terminal in der Betriebsart VT100-Steuerzeichenfolgen entsprechend ANSI X 3.64 bzw. ISO DP 6429 (abgeruesteter Funktionsumfang).

2.9.3. Terminals deaktivieren

Das Programm /etc/INIT (siehe init(M)) eroeffnet beim Start des Multi-User-Modes alle in der Datei /etc/inittab (siehe inittab(5)) eingetragenen Kanaele fuer das Login. Gelingt das Eroeffnen eines oder mehrerer Kanaele nicht, werden fuer diese Kanaele neue Versuche gestartet. Kanaele, an denen keine Terminals angeschlossen sind, sollten in der Datei /etc/inittab deaktiviert werden.

Die /etc/inittab hat fuer den Multi-User-Mode die Eintraege:

```
2:00:c:/etc/GETTY tty0 2
2:co:c:/etc/GETTY console 2
2:02:c:/etc/GETTY tty2 2
2:03:k:
2:04:c:/etc/GETTY tty4 2
2:05:c:/etc/GETTY tty5 2
2:06:c:/etc/GETTY tty6 2
2:07:c:/etc/GETTY tty7 2
```

Die Beschreibung der Datei inittab ist im Abschnitt 7.2.2. oder in init(M) enthalten.

Ist z.B. Kanal tty2 unbenutzt, wird der entsprechende Eintrag geaendert in:

```
2:02:k:        oder

2.02:k:/etc/GETTY tty2 !
```

Ueber Kanal tty3 darf kein Login erfolgen, da an diesem Kanal der Drucker angeschlossen ist. Gleiches gilt bei Anschluss anderer E/A-Geraete.

2.9.4. Baudrate aendern

Das Kommando /etc/GETTY (siehe getty(M)) stellt unter anderem die Baudrate fuer einen Terminalkanal ein. Zur Aenderung der Baudrate, ist der Datei /etc/inittab der zweite Parameter von GETTY zu aendern. Alle Terminals sind auf 9600 Baud (Parameter = 2) eingestellt.

Wird zum Beispiel fuer tty4 die Baudrate 2400 Baud festgelegt, ist der Eintrag in der Datei /etc/inittab wie folgt zu aendern:

```
2:04:c:/etc/GETTY tty4 6
```

Die Baudraten sind im Abschnitt 7.2.3. und in getty(M) angegeben. Die Baudrate des Systemterminals darf nicht geaendert werden.

Fuer "deal-up" Kanaele (Telefonleitung ueber Vermittlung) kann GETTY einen dritten Parameter erhalten. Dieser legt die Zeit in Sekunden bis zum Loesen der Verbindung fest, falls keine Reaktion auf das Login erfolgte.

2.9.5. Zusaetzliches Terminal anschliessen

Ist kein Drucker im P8000-System erforderlich, kann ein weiteres Terminal angeschlossen werden. Folgende Schritte sind auszufuehren:

1. Loeschen von /dev/lp mit rm /dev/lp

2. Drucker aus Datei /usr/spool/queuer/config streichen (siehe Abschnitt 7.1.3.)

3. chmod 622 /dev/tty3

4. Eintrag in /dev/inittab aendern

5. Terminaltyp in /etc/ttytype eintragen

6. ttyconfig in /etc/rc_csh anpassen

2.9.6. Zusaetzlichen Drucker installieren

Mit dem Druckertreiber im WEGA-Kern koennen zwei Drucker betrieben werden. Der zweite Drucker wird anstelle eines Terminals angeschlossen. Jeder serielle Kanal (ausser console) ist dafuer geeignet. Die Minor-Geraetenummer bezeichnet den entsprechenden seriellen Kanal. Zur Unterscheidung zwischen ersten (lp) und zweiten Drucker (lp2), wird bei lp2 zusaetzlich Bit 7 der Minor-Geraetenummer gesetzt. Darueber hinaus kann an jedem tty-Kanal (ausser console) ein Drucker (vorzugsweise mit Textqualitaet, siehe auch Abschnitt 7.1.) mit XON-/XOFF-Protokoll ueber den tty-Treiber betrieben werden.

Soll lp2 z.B. an Kanal tty7 angeschlossen werden, sind folgende Aenderungen auszufuehren:

1. Deaktivieren des Logins (siehe Abschnitt 2.9.3.) in der Datei /etc/inittab.

2. Anpassen von ttyconfig in /etc/rc_csh.

3. Eintragen von lp2 in das Directory /dev mit:

   /etc/mknod /dev/lp2 c 9 135

4. Zugriffsrechte aendern mit:

   chmod 200 /dev/lp2 /dev/tty7

5. Einbinden des Druckers in die Spooler-Datei /usr/spool/queuer/config (siehe Abschnitt 7.1.3.)

2.9.7. Aendern der Parameter des Druckertreibers

Der Druckertreiber lp(4) realisiert eine Formatierung der Ausgabeseiten. Mit dem Programm setlp(M) kann die Seitengestaltung veraendert werden. Es lassen sich die Anzahl der Zeilen pro Seite, die Anzahl der Zeichen pro Zeile und der linke Rand setzen. Die Standardwerte betragen 66, 128 und 0. Mit dem Programm stty(1) lassen sich die Uebertragungsparameter der V.24- bzw. IFSS-Schnittstelle (Standardwerte: 9600 Baud, 8 bit, keine Paritaet) beeinflussen. Beide Programme sollten bei Bedarf in die Datei rc_csh vor Aufruf von dqueuer eingefuegt werden:

```
/bin/stty 4800 < /dev/lp
/etc/setlp -d /dev/lp -l 66 -w 128 -s 10
/etc/dqueuer -r
```

Dieses Beispiel setzt fuer den Drucker lp die Baudrate auf 4800 Baud, die Anzahl der Zeilen pro Seite auf 66, die Anzahl der Zeichen pro Zeile auf 128 und die Anzahl Leerzeichen als linken Rand auf 10.

2.10. Aenderung der Zeitzone

WEGA ist auf mitteleuropaeische Zeit (MEZ) eingestellt, Sommerzeit ist nicht aktiviert. Die Zeitzone ist ggf. an verschiedenen Stellen zu aendern:

1. Aendern der TZ environ(7)-Variablen
   in den Dateien /etc/rc, /etc/cshprofile, /etc/inittab und /etc/profile.

   TZ hat das Format:

   ```
         xxxnzzz
   ```

   xxx - Abkuerzung der oertlichen Standardzeit
   zzz - Abkuerzung der oertlichen Sommerzeit
   n - Zeitverschiebung in Stunden westlich von GMT

TZ fuer die mitteleuropaeische Zeit lautet "MEZ-1MES", andere Zeitzonen sind z.B. "PST8PDT oder EST5EDT".

2. Aendern der Zeitzone im WEGA-Kern
In der Datei /usr/sys/h/sysparm.h werden die Systemkonstanten DSTFLG und TIMEZONE geaendert. DSTFLAG = 1 stellt die Sommerzeit ein. TIMEZONE enthaelt die Anzahl der Minuten der Zeitverschiebung westlich von GMT, fuer MEZ -1*60. Nach der Aenderung wird ein neuer WEGA-Kern generiert, siehe dazu Abschnitt 5.

3. Installation der WEGA-Systemplatte

Das P8000 wird vom Hersteller mit einer installierten Systemplatte ausgeliefert. Die mitgelieferte Standardsoftware ist in den Dateisystemen root und /usr enthalten. Eine Kopie davon befindet sich auf den root- und /usr-Vertriebsdisketten. Diese Diskette bilden die Basis zur Neuinstallation der Systemplatte.

Die Installation der Dateisysteme root und /usr erfolgt im sekundaeren Bootstrap-Programm (Stand-alone-Mode). Die root- und /usr-Vertriebsdisketten sind im WEGA-Dateisystemformat beschrieben (fuer aeltere Versionen wurde das UDOS-Format benutzt).
Die Installation von zusaetzlichen Softwarepaketen erfolgt unter dem Betriebssystem WEGA. Die Disketten der Zusatzpakete sind im tar(5)-Format beschrieben. Die Installation der zusaetzlichen Softwarepakete erfolgt mit dem Programm winstall(M). Fuer jedes Zusatzpaket existiert eine Datei README, in der spezielle Informationen zum jeweiligen Softwarepaket enthalten sind. In den folgenden Abschnitten wird die vollstaendige Installation der Standardsoftware beschrieben.

Beachten Sie bitte !

Alle Angaben zu den logischen Dateisystemen beziehen sich auf die Standardaufteilung der Systemplatte, die im Abschnitt 4.1. beschrieben ist. Die Standardaufteilung Ihrer Systemplatte finden Sie in der README-Datei auf der WEGA-Startdiskette.

3.1. Ueberpruefen der Systemplatte

Voraussetzung zum Einrichten der Dateisysteme ist eine fehlerfreie Systemplatte. Alle benutzten Sektoren der Hard-Disk muessen sich physisch lesen und beschreiben lassen. Der Inhalt ist dabei nicht von Bedeutung. Mit dem Programm sa.verify (Beschreibung Abschnitt 2.6.2) kann ein Hard-Disk-Laufwerk auf defekte Spuren ueberprueft werden. Defekte Spuren werden mit dem Stand-alone-Programm sa.format (Beschreibung Abschnitt 2.6.1.) neu formatiert. Spuren, die sich nicht durch Neuformatieren reparieren lassen, werden von sa.format in die "Bad Track Table" des Laufwerks aufgenommen und nicht mehr benutzt.

Neue Hard-Disk-Laufwerke muessen natuerlich vor der ersten Benutzung vollstaendig formatiert werden.

WEGA arbeitet mit Blocknummern zum Adressieren von 512 Byte Bloecken (Sektoren) der Platte. Eine Blocknummer wird wie folgt in die entsprechende Spurnummer (Kopf, Zylinder) umgerechnet:

(Blocknummer : Anzahl der Sektoren : Anzahl der Koepfe) + 1
= Zylindernummer; Rest = Kopfnummer

Zylinder 0 ist fuer die "Bad Track Table" reserviert. In der Formel sind jedoch eventuell defekte Spuren nicht beruecksichtigt. Die Programme sa.format und sa.verify zeigen die Anzahl der Zylinder, Koepfe und Sektoren der Winchesterplatte(n) an.

Aufruf der Programme:

```
ud(0,0)sa.verify
ud(0,0)sa.format
```

3.2. Installation des Dateisystems root

1. Einrichten des leeren Dateisystems mit dem Stand-alone-Programm sa.mkfs (Beschreibung Abschnitt 2.6.3.):

   ```
   ud(0,0)sa.mkfs
   file system size: 7000
   file system: md(0,16000)
   ```

2. Installation aller Dateien des Dateisystems root mit dem Programm sa.install (Beschreibung Abschnitt 2.6.4.):

   ```
   ud(0,0)sa.install
   ```

   Anschliessend 1. Diskette des Dateisystems root in das Floppy-Disk-Laufwerk 0 einlegen, dann Eingabe von:

   ```
   Enter Date (MM/DD/YY) : 5/1/87
   input file system : fd(0,0)
   output file system : md(0,16000)
    .... (n/y/a/A/q/Q) ? : A
   ```

   Nach der letzten Diskette des Dateisystems root wird wieder die WEGA-Startdiskette in das Laufwerk 0 eingelegt.

3.3. Installation des Dateisystems /usr

1. Einrichten des leeren Dateisystems:

   ```
   ud(0,0)sa.mkfs
   file system size: 13000
   file system: md(0,0)
   ```

2. Installation der Dateien:

ud(0,0)sa.install

Anschliessend 1. Diskette des Dateisystems /usr in das Floppy-Disk-Laufwerk 0 einlegen, dann Eingabe von

```
Enter Date (MM/DD/YY) : 5/1/87
input file system : fd(0,0)
output file system : md(0,0)
 .... (n/y/a/A/q/Q) ? : A
```

Nach der letzten Diskette des Dateisystems /usr wird wieder die WEGA-Startdiskette in das Laufwerk 0 eingelegt.

Die weitere Installation der Systemplatte erfolgt unter WEGA im Single-User-Mode.

3.4. Initialisieren der Dateisysteme /tmp und /z

Mit mkfs(M) koennen die leeren Dateisysteme /tmp und /z angelegt werden. Der alte Inhalt geht dabei vollstaendig verloren. Aeusserste Vorsicht ist also geboten, um nicht irrtuemlich ein anderes Dateisystem zu zerstoeren.

Eingabe von:

```
mkfs /dev/rtmp 4000
mkfs /dev/rz xxxx      (abhaengig vom Typ der Platte)
```

Das Dateisystem /z geht bis auf Sonderfaelle bis zum Ende der Platte, siehe README auf der WEGA-Startdiskette.

3.5. Benennen der Dateisysteme

Jedes Dateisystem erhaelt mit labelit(M) einen Namen. Dieser wird im Superblock abgelegt.

```
labelit /dev/rroot root 0
labelit /dev/rusr /usr 0
labelit /dev/rtmp /tmp 0
labelit /dev/rz /z 0
```

3.6. Block 0 schreiben

Das primaere Bootstrap-Programm wird mit dd(1) auf Block 0 des ersten Dateisystems (/usr) der Systemplatte geschrieben:

```
dd if=/pb.image of=/dev/rmd0 conv=sync count=1
```

3.7. Erzeugen der Directorys lost+found

In jedem Dateisystem sollte ein Directory "lost+found" vorhanden sein. Das Programm fsck(M) traegt in dieses Directory alle Dateien ein, fuer die kein Eintrag in einem Directory existiert. Fuer eventuelle Eintraege muessen im Directory mindestens 10 Bloecke freigehalten sein, da zum Zeitpunkt der Ueberpruefung des Dateisystems keine Aenderungen im Dateisystem erfolgen koennen.

Die Dateisysteme muessen fuer das Anlegen der Directorys lost+found mit mount(M) verbunden werden. Dafuer kann die Shell-Prozedur mfs(M) Verwendung finden. Fuer das Anlegen der Directorys lost+found existiert die Shell-Prozedur mlf.

```
mfs
mlf / /usr /tmp /z
umfs
```

4. Verwaltung der WEGA-Dateisysteme

4.1. Systemplatte

4.1.1. Dateisysteme der Systemplatte

Jede physische Hard-Disk kann unter WEGA in bis zu 10 logische Platten aufgeteilt werden. Jede logische Platte enthaelt ein Dateisystem.

Die Systemplatte (Laufwerk 0) ist wie folgt aufgeteilt:

Dateisystem-Geraetename	Dateisystem-Systemname	Start-(Offset) Blocknummer	Laenge in Bloecken
/dev/md0 bzw. /dev/usr	/usr	0	13000
/dev/swap		13000	3000
/dev/md2 bzw. /dev/root	/	16000	7000
/dev/md3 bzw. /dev/tmp	/tmp	23000	4000
/dev/md4 bzw. /dev/z	/z	27000	bis Ende

Die Laenge von /dev/md4 ergibt sich aus der Speicherkapazitaet des jeweiligen Winchesterlaufwerks.

4.1.2. Verbinden von Dateisystemen (mount)

Der Dateisystem-Geraetename ist der Name, ueber den das System auf den entsprechenden Bereich der Platte zugreift. In dem Directory /dev existieren dazu die erforderlichen Geraetedateien (special files). Fuer jedes E/A-Geraet koennen mehrere Namen vereinbart werden. Das kann ueber einen Link mit ln(1) oder durch Anlegen einer eigenen Geraetedatei mit mknod(M) erfolgen, zum Beispiel:

```
mknod /dev/root b 0 2
```

Der Dateisystem-Systemname ist Teil des Pfadnamens. Die Datei /usr/bin/man z.B. befindet sich im Dateisystem /dev/usr. Die Verbindung zwischen Dateisystem und Systemnamen wird mit dem Kommando mount(M) hergestellt, zum Beispiel:

```
mount /dev/usr /usr
```

Es wird dabei das root-Directory des zu verbindenden Dateisystems mit einem Directory des Dateisystems root verbunden. Ueber dieses Directory wird auf die Dateihierarchie des verbundenen Dateisystems zugegriffen.

Im Dateisystem root sind die Directorys /usr, /tmp und /z zum Verbinden mit den Dateisystemen /dev/usr, /dev/tmp und /dev/z enthalten.

Nach dem Start von WEGA in den Single-User-Mode besteht noch keine mount-Verbindung. Zum Herstellen oder Loesen der Verbindungen existieren die Shell-Prozeduren /etc/mfs und /etc/umfs.

4.1.3. Benutzen der Dateisysteme

Im Dateisystem /dev/root sind die am meisten benutzten, im Dateisystem /dev/usr die etwas weniger benutzten WEGA-Programme und Dateien enthalten. Im Dateisystem /dev/tmp werden alle temporaeren Dateien (z.B. Zwischendateien vom Compiler) abgelegt. Das Dateisystem /dev/z ist fuer die Benutzer und zur Installation spezieller Programme reserviert. Der swap-Bereich /dev/swap dient zum Auslagern von Prozessen. Er wird ausschliesslich durch den WEGA-Kern benutzt und ist nicht als Dateisystem initialisiert.

4.1.4. Optimierung der Plattenzugriffe

Zur Gewaehrleistung von schnellen Plattenzugriffen ist das am meisten benutzte Dateisystem /dev/root in der Mitte der Systemplatte angeordnet. Die Pfadvariable (path variable) sollte so gesetzt sein, dass zuerst das eigene Directory, dann das Directory /bin und zum Schluss /usr/bin durchsucht werden. Sind Programme in einer /z/bin installiert, sollte sie in die Suchreihenfolge einbezogen werden.

Die Aufteilung der Hard-Disk in mehrere logische E/A-Geraete bzw. Dateisysteme erhoeht die Zuverlaessigkeit des Systems. Zu den aufwendig zu installierenden Dateisysteme /dev/root und /dev/usr sind ueberwiegend nur Lesezugriffe notwendig. Fehler im Dateisystem sind dadurch recht selten und mit fsck(M) leicht zu beheben. Alle wesentlichen Schreibzugriffe erfolgen in den Dateisystemen /dev/tmp und /dev/z. Im Dateisystem /dev/tmp sind temporaere Dateien abgelegt. Dieses Dateisystem kann bei Fehlern jederzeit mit mkfs(M) neu initialisiert werden. Nichtbehebbare logische und behebbare physische Fehler erfordern nur die Neuinstallation des betroffenen Dateisystems. Das Formatprogramm sa.format gestattet dabei, wie schon beschrieben, auch die Formatierung einzelner Bereiche einer Hard-Disk.

4.2. Kopieren von grossen Dateien

Vor dem Kopieren von grossen Dateien sollte mit df(M) die Anzahl der freien Bloecke festgestellt werden. Zehn Prozent der Bloecke eines Dateisystems sollten als Reserve frei bleiben, um den Fehler "Out of space on dev x/y" (Dateisystem voll siehe Abschnitt 6.3.) zu vermeiden. Ein volles Dateisystem kann ernsthafte Probleme verursachen. Besonders in den Dateisystemen root und /usr sollte der Fehler nicht auftreten. Das Dateisystem /usr kann durch viele Druckerausgaben ueber den Spooler voll werden. Es sollte, falls der Fehler haeufig auftritt, vergroessert werden.

Beachten Sie bitte !

Die Datei /usr/adm/wtmp (siehe acct(M)) muss in regelmaessigen Abstaenden bereinigt werden, da sie staendig groesser wird.

4.3. Aendern der Plattenaufteilung

Die Aufteilung der Systemplatte kann vom Systemverwalter geaendert werden. Dabei duerfen jedoch keinesfalls die Dateisysteme /dev/root und /dev/usr verkleinert werden.

1. Generierung eines neuen WEGA-Kerns und gegebenenfalls eines neuen sekundaeren Boot-Programms (siehe Abschnitt 5).

2. Kopieren dieser beiden Programme (boot, wega) auf ein Backup-Medium (Floppy-Disk).

3. Neuinstallation der Systemplatte mit der geaenderten Plattenaufteilung (siehe Abschnitt 3).

Zusaetzliche Dateisysteme werden entsprechend Abschnitt 4.4. installiert.

4.4. Einbinden eines zweiten Hard-Disk-Laufwerks

Der Systemverwalter kann ein zweites Hard-Disk-Laufwerk in WEGA einbinden. Es ist wie folgt vorzugehen:

1. Hardware-Anschluss realisieren

2. Generierung eines neuen WEGA-Kerns (siehe Abschnitt 5).

3. System mit dem neuen Kern in den Single-User-Mode starten.

4. Anlegen der Geraetedateien fuer die neuen Dateisysteme in dem Directory /dev mit dem Kommando mknod(M). Es muessen die Eintraege fuer die Block- und raw-Geraete erzeugt werden.

Die Dateisystem-Geraetenamen fuer Laufwerk 1 sind mit

md10 bis md19

festgelegt (fuer Laufwerk 0: md0 bis md9).

Laufwerk 1 soll z.B. in zwei logische Platten aufgeteilt werden:

```
mknod /dev/md10 b 0 10
mknod /dev/md11 b 0 11
mknod /dev/rmd10 c 0 10
mknod /dev/rmd11 c 0 11
```

5. Zugriffsrechte und Eigner setzen:

   ```
   chmog 640 bin 0 /dev/md10 /dev/md11
   chmog 640 bin 0 /dev/rmd10 /dev/rmd11
   ```

6. Eintragen der neuen Dateisysteme in die Shell-Prozedur /etc/mfs.

7. Eintragen der neuen Dateisysteme in die Shell-Prozedur /etc/rc_csh zur automatischen Ausfuehrung des Kommandos fsck(M). (Fuer Dateisysteme mit mehr als 30000 Bloecken sollte bei fsck die Option -t und ein "scratch" Dateiname angegeben werden.)

8. Eintragen der neuen Dateisysteme in die Check-Liste /etc/checklist des Programms fsck.

9. Anlegen der mount-Directorys:

   ```
   mkdir /z1 /z2
   ```

10. Initialisieren der Dateisysteme mit mkfs(M):

    ```
    mkfs /dev/rmd10 <Laenge in Bloecken>
    mkfs /dev/rmd11 <Laenge in Bloecken>
    ```

11. Benennen der Dateisysteme mit labelit(M):

    ```
    labelit /dev/rmd10 /z1 1
    labelit /dev/rmd11 /z2 1
    ```

12. Mount-Verbindung herstellen:

    ```
    mount /dev/md10 /z1
    mount /dev/md11 /z2
    ```

13. Anlegen der Directorys lost+found:

```
mlf /z1 /z2
```

14. Zugriffsrechte und Eigner der lost+found setzten:

```
chmog 644 bin 0 /z1/lost+found /z2/lost+found
```

15. Wechsel in den Multi-User-Mode mit:

```
sync
INIT 2
```

4.5. Sichern von Dateisystemen

Ein Platten-Dateisystem kann vom Superuser zur Sicherung auf ein Backup-Medium ausgelagert werden. Es wird ein "Dump" des Dateisystems hergestellt. Im P8000 findet dazu die Floppy-Disk Verwendung. Die Kommandos dump(M) und restor(M) sind zum Auslagern bzw. Rueckspeichern der Dateisysteme vorhanden, z.B.:

```
dump 0u /dev/rz
restor r /dev/rz
```

Ein Dump darf nur erfolgen, wenn das Dateisystem nicht mit mount verbunden ist. Aktivitaeten im Dateisystem machen den Dump unbrauchbar.

Beim P8000 sollte jedoch vorrangig darauf orientiert werden, dass jeder Benutzer eigenverantwortlich seine Dateien mit tar(1) auf Disketten sichert. Darueber hinaus koennen Dateien mit den Programmen putud(1) auf UDOS-Disketten bzw. putfile(1) ueber remote auf UDOS- oder SCP-Disketten gesichert werden.

Von den Einsatzbedingungen des P8000 ist abhaengig, wie oft ein Dump auszufuehren ist. In der Regel werden jeden Monat ein Level 0 Dump und ein- bis mehrwoechig ein Level 1 Dump durchgefuehrt. (Ein Dump ist nur von /dev/z und weiteren Benutzer-Dateisystemen erforderlich. Fuer /dev/root und /dev/usr liegen die Vertriebsdisketten vor. Diese sind nicht im Dump-Format beschrieben.)

Mit der Neuinitialisierung eines Dateisystems wird die Freiblockliste fuer optimale Plattenzugriffe angelegt. Mit der Zeit verliert sich der optimale Zustand durch Loeschen und Neuanlegen von Dateien (Richtwert: ca. 3 bis 6 Monate). Mit dump(M), mkfs(M) und restor(M) kann ein Dateisystem fuer optimale Zugriffe neu installiert werden.

4.6. Fehler in Dateisystemen

Das Programm fsck(M) ueberprueft Dateisysteme auf Konsistenz (Fehlerfreiheit). Es wird automatisch beim Start in den Multi-User-Mode ausgefuehrt. Natuerlich kann fsck auch manuell aufgerufen werden. In den meisten Faellen beseitigt fsck die Fehler selbstaendig. Manchmal ist jedoch eine manuelle Reparatur erforderlich. Die Programme icheck(M) und dcheck(M) werden zur Fehlerbeseitigung verwendet (fsck ist eine Zusammenfassung von icheck und dcheck). Das Programm icheck ueberprueft die Block-Konsistenz eines Dateisystems. Fuer jedes Dateisystem wird folgende Mitteilung ausgegeben (Beispiel):

```
files   149 (r=128, d=13, b=3, c=5)
used    1721 (i=57, ii=0, iii=0, d=1664)
free    3245
missing 0
```

Diese Ausschriften besagen folgendes:

file: Im Dateisystem sind 149 Dateien vorhanden, 128 davon sind regulaere (normale) Dateien, 13 sind Directorys, 3 sind Block-Geraetedateien (block special files) und 5 sind Character-Geraetedateien (character special files).

used: 1721 Bloecke des Dateisystems sind benutzt. Es gibt 57 indirekt adressierte und keine doppelt und dreifach indirekt adressierten Bloecke.

free: 3245 Bloecke des Dateisystems sind frei.

missing: Es werden keine Bloecke vermisst. (Bloecke gelten als vermisst, wenn sie weder einer Datei zugeordnet noch in der Freiblockliste enthalten sind.)

Das Programm dcheck ueberprueft die Anzahl der Links in jedem Inode mit der Anzahl der Eintraege in den Directorys, die auf das Inode verweisen. Stellt dcheck keine Fehler fest, wird nur der Name des ueberprueften Dateisystems ausgegeben. Die Programme icheck und dcheck koennen verschiedene Fehlerarten erkennen. Die Beseitigung von Fehlern sollte nur in nicht mit mount(M) verbundenen Dateisystemen erfolgen. Das System ist dazu in den Single-User-Mode zu bringen.

Ein inkonsistentes root-Dateisystem erfordert besondere Massnahmen. Zur Reparatur muss /dev/rmd2 bzw. /dev/rroot benutzt werden. Nach der Reparatur ist sofort die Reset-Taste zu druecken. Das Kommando sync(1) darf nicht benutzt werden, damit der reparierte Superblock des Dateisystems nicht ueberschrieben wird. Aus diesem Grund erfolgt die Ueberpruefung auf Fehlerfreiheit mit icheck und dcheck erst nach Neustart von WEGA.

4.6.1. Vermisste Bloecke

Vermisste Bloecke (missing blocks) werden mit icheck(M) ermittelt. Die Beseitigung der entsprechenden Fehler erfolgt mit icheck und der Option "-s", die Freiblockliste wird neu erstellt:

```
icheck -s /dev/...
```

Anschliessend sollte das Dateisystem nochmals ueberprueft werden:

```
icheck /dev/...
```

4.6.2. Doppelte Bloecke

Doppelte Bloecke (duplicate blocks) werden von icheck erkannt. Folgende Ausschrift weist darauf hin (Beispiel):

```
10311 dup; inode = 81 class = free
```

Block 10311 ist einer Datei zugeordnet und in der Freiblockliste enthalten. Der Fehler wird wie oben beschrieben mit

```
icheck -s /dev/...
icheck /dev/...
```

beseitigt. Der doppelte Block wird aus der Freiblockliste ausgetragen. Die Beseitigung des Fehlers sollte sofort erfolgen.

4.6.3. Fehlerhafte Freiblockliste

Das Programm icheck ermittelt eine fehlerhafte Freiblockliste (bad freeblock). In der Freiblockliste ist eine Blocknummer enthalten, die ausserhalb des verfuegbaren Bereiches des Dateisystems liegt. Der Fehler wird mit

```
icheck -s /dev/...
icheck /dev/...
```

beseitigt.

4.6.4. Fehlerhafte Bloecke

Ein fehlerhafter Block wird durch icheck mit der Ausschrift

```
53112 bad; inode = 58, class = free
```

(Beispiel) angezeigt.
Block 53112 liegt ausserhalb des Dateisystems, "class = free" zeigt an, dass der Block in der Freiblockliste enthalten ist. Der Fehler sollte sofort mit

```
icheck -s /dev/...
```

beseitigt werden. Im Anschluss daran ist das Dateisystem mit icheck und dcheck zu ueberpruefen:

```
icheck /dev/...
dcheck /dev/...
```

4.6.5. Fehlende Directory-Eintraege

Der Fehler "keine oder zu wenig Directory-Eintraege" wird von dcheck erkannt. Die Anzahl der Links in einem Inode ist groesser als die Anzahl der Directory-Eintraege, die auf dieses Inode verweisen. Ein Teil der Ausgaben von dcheck ist (Beispiel):

```
/dev/...
        entries   link cnt
348       0           1
```

In dem Inode 348 ist ein Link eingetragen, es ist keine Directory vorhanden, in der ein Eintrag auf dieses Inode bzw. diese Datei verweist. Mit dem Kommando clri(M) kann das Inode frei gegeben werden:

```
clri /dev/... 348
```

Mit dem icheck Kommando werden die freigewordenen Bloecke der ehemaligen Datei der Freiblockliste zugefuehrt:

```
icheck -s /dev/...
```

Zum Schluss erfolgt eine Ueberpruefung mit:

```
icheck /dev/...
dcheck /dev/...
```

Diese Reparatur ist auch auszufuehren, wenn "entries" und "link count" den Wert 0 besitzen. Ist der Wert fuer "entries" ungleich Null und "link count" groesser "entries" gibt es zwei Moeglichkeiten:

1. Es wird keine Reparatur vollzogen. Wenn alle Directory-Eintraege geloescht sind, wird das Inode nicht freigegeben. Der Zustand ist nicht gefaehrlich, es geht nur etwas Platz im Dateisystem verloren. Der Fehler kann nun, wie oben beschrieben, beseitigt werden.

2. Der Fehler wird analog Abschnitt 4.6.6. beseitigt.

4.6.6. Zu viele Directory-Eintraege

Es gibt mehr Directory-Eintraege als Links in dem Inode, "link count" ist kleiner als "entries". Eventuell weisen ein oder mehrere Directory-Eintraege auf nichtbelegte Inodes oder schlimmer auf andere Dateien.

Fehlerbeseitigung:

1. Mit dem Programm ncheck(M) werden alle Dateinamen (Directory-Eintraege) ermittelt, die auf das entsprechende Inode verweisen.

   Beispiel:

   ```
   ncheck -i 205 /dev/...
   ```

2. Dateisystem mit mount(M) verbinden, z.B.:

   ```
   mount /dev/z /z
   ```

3. Mit rm(1) Datei loeschen bzw. zuvor mit cp(1) in ein konsistentes Dateisystem kopieren.

4. Verbindung des Dateisystems mit umount(M) loesen, z.B.:

   ```
   umount /dev/z
   ```

5. Folgende Kommandos ausfuehren (Beispiel):

   ```
   clri /dev/z 205
   icheck -s /dev/z
   icheck /dev/z
   dcheck /dev/z
   ```

   Die Fehler sollten nun beseitigt sein.

4.6.7. Dateibloecke ausserhalb des Dateisystems

Analog Abschnitt 4.6.4. ermittelt icheck auch fehlerhafte Blockadressen in Dateien.

Beispiel:

```
53113 bad; inode = 25, class = data (small)
```

Block 53113 liegt ausserhalb des Dateisystems. Der Block ist Teil einer Datei. Mit "small" wird ein direkt adressierter Block bezeichnet. Die Ausschriften "large", "huge" und "garg" kennzeichnen einfach, zweifach bzw. dreifach indirekt adressierte Bloecke.

Fehlerbeseitigung:

1. Datei wie in Abschnitt 4.6.6. loeschen

2. Die Freiblockliste uebernimmt beim Loeschen einer Datei die Bloecke und enthaelt nun die fehlerhaften Blocknummern. Diese werden mit

   ```
   icheck -s /dev/...
   ```

   beseitigt.

3. Ueberpruefen des Dateisystems mit:

   ```
   icheck /dev/...
   dcheck /dev/...
   ```

4.6.8. Doppelte Bloecke in Dateien

Bei doppelt in Dateien vorhandenen Bloecken gibt icheck folgende Ausschrift (Beispiel) aus:

```
3114 dup; inode = 25, class = data (large)
```

Block 3114 ist in zwei oder mehreren Dateien enthalten. Das Inode einer dieser Dateien ist 25. Zur Ermittlung der weiteren Dateien, wird das icheck Programm mit der Option "-b" benutzt:

```
icheck -b 3114 /dev/...
```

Die Inodes aller Dateien, die diesen Block enthalten, werden ausgegeben. Mit ncheck(M) werden die zugehoerigen Dateinamen ermittelt (siehe Abschnitt 4.6.6).

Zur Beseitigung des Fehlers werden alle Dateien - bis auf eine - geloescht, mit cp(1) eventuell vorher in ein konsistentes Dateisystem kopiert. Es werden

```
icheck -s /dev/...
icheck /dev/...
dcheck /dev/...
```

ausgefuehrt.

5. Systemgenerierung

Die Generierung eines neuen WEGA-Kerns ist notwendig, wenn:

1. die Systemkonstanten oder

2 verschiedene Namen geaendert werden sollen,

3. eine neue Platte mit veraenderter Kapazitaet oder eine zweite Platte angeschlossen werden soll oder sich aus bestimmten Gruenden eine Neuaufteilung der Dateisysteme auf einer vorhandenen Platte notwendig macht

4. ein neuer Treiber in den Kern eingebunden werden soll.

Die zur Generierung des WEGA-Kerns erforderlichen Dateien sind in dem Directory /usr/sys und deren Subdirectorys enthalten. Der Systemverantwortliche muss sich als Superuser unter dem Namen "wega" im System anmelden und in das Directory /usr/sys/conf wechseln. Das Programm sysgen (siehe Abschnitt 5.5.) realisiert interaktiv alle moeglichen Aenderungen und generiert anschliessend einen neuen WEGA-Kern und bei Bedarf auch ein neues sekundaeres Bootstrap-Programm.

5.1. Aendern der Systemkonstanten

Die Systemkonstanten sind in der Datei wpar.c vereinbart. Die Werte der Systemkonstanten sind in der Datei /usr/sys/h/sysparm.h definiert. Die Systemkonstanten sollten nur bei unbedingter Notwendigkeit und dann mit aeusserster Vorsicht geaendert werden. Es besteht die Gefahr, einen nicht funktionsfaehigen Kern zu generieren. Die Konstanten DSTFLAG und TIMEZONE koennen ohne Bedenken geaendert werden.

```
CANBSIZ  256     /* max size of input line from terminal  */
DSTFLAG  0       /* Daylight Savings Time flag            */
MAXMEM   512     /* maximum memory per process            */
MAXUPRC  20      /* maximum number of processes per user  */

NBUF1    40      /* number of buffers in pool 1 (max 127) */
NBUF2    1       /* number of buffers in pool 2 (max 127) */
NBUF3    1       /* number of buffers in pool 3 (max 127) */
NBUF4    1       /* number of buffers in pool 4 (max 127) */
NBUF5    1       /* number of buffers in pool 5 (max 127) */
NHBUF    17      /* number of slots in buffer hash        */
                 /* should be a prime number              */
```

```
NCLIST   100     /* number of small buffers for term io   */
NFILE    175     /* size of system open file table        */
NINODE   200     /* number of in core i-nodes             */
NMOUNT   20      /* number of mountable file systems      */
NPROC    70      /* max number of active processes        */
NTEXT    40      /* max number of shared text segments    */
TIMEZONE (-1*60) /* minutes westward from GMT              */
NFLOCK   100     /* number of lockable regions in a file  */
```

5.2. Aendern der Plattenaufteilung

Die Aufteilung der Dateisysteme auf ein oder mehrere Winchester-Laufwerke wird in der Datei wpar.c festgelegt. Jede physischen Platte kann bis zu 10 logische (virtuelle) Platten enthalten. Fuer jede virtuelle Platte existiert eine Struktur vd_size (siehe Datei /usr/include/sys/conf.h):

```
struct vd_size {           /* Virtual disk specification */
        int     vd_unit;    /*  Unit v.d. starts on;     */
        daddr_t vd_blkoff;  /*  Offset on unit of v.d.;  */
        daddr_t vd_nblocks; /*  Size of v.d.             */
};
```

In der Datei wpar.c sind Eintraege fuer 2 physische Platten (20 Dateisysteme) vorhanden. Die Groesse der virtuellen Platten bzw. der Dateisysteme wird in der Datei /usr/sys/h/mdsize.h definiert:

```
#define USR     13000
#define SWAP    3000
#define ROOT    7000
#define TMP     4000
#define Z       60732
#define MD5     0
#define MD6     0
#define MD7     0
#define MD8     0
#define MD9     0

#define MD10    0
#define MD11    0
#define MD12    0
#define MD13    0
#define MD14    0
#define MD15    0
#define MD16    0
#define MD17    0
#define MD18    0
#define MD19    0
```

Im folgenden ist der entsprechende Teil der Datei wpar.c dargestellt:

```
int ndrives = 2;

#define NMD 20

int nmd = NMD;

struct vd_size md_sizes[NMD] = {
 0,  0,     USR,
 :   :      :
 :   :      :
 1,  0,     MD10,
 :   :      :
 :   :      :
};

int nswap = SWAP-1;

int  rootdev = makedev(0,2);
int  swapdev = makedev(0,1);
int  pipedev = makedev(0,2);
```

In nswap wird die Laenge des swap-Bereiches analog der Plattenaufteilung als Anzahl der Bloecke -1 angegeben. Im Anschluss daran, sind die (logischen) Geraete (Major-, Minornummer) fuer das root-Dateisystem, den swap-Bereich sowie das Geraet fuer Pipelines definiert. Letzteres sollte dem Root-Dateisystem entsprechen, um diese Funktion im Single-User-Mode mit nicht verbundenen Dateisystemen zu ermoeglichen.

Beachten Sie bitte !

Aendert sich die Lage des Dateisystems root auf der Systemplatte, muss auch das Startkommando fuer den WEGA-Kern "md(0,14000)wega" in der Datei bstart.c korrigiert werden.

5.3. Einbinden neuer Treiber

Der Treiber ist nach den fuer WEGA geltenden Richtlinien als Block-, Character- oder Terminaltreiber zu schreiben. Abschnitt 8 "Schreiben von WEGA-Geraetetreibern" beschaeftigt sich ausfuehrlich mit diesem Thema. Die einzige Schnittstelle zwischen dem Kern und dem Treiber sind definierte Sprungverteiler. Der Aufbau dieser Verteiler ist in der Datei /usr/include/sys/conf.h beschrieben.

Im Feld bdevsw[] sind Strukturen

```
struct bdevsw
{
        int             (*d_open)();
        int             (*d_close)();
        int             (*d_strategy)();
        struct buf      *d_tab;
};
```

als Sprungverteiler fuer Blocktreiber enthalten. Die struct buf d_tab wird als Header fuer die Auftragswarteschlange benutzt. Die Sprungverteiler fuer Charactertreiber sind im Feld cdevsw[] in der Form

```
struct cdevsw
{
        int             (*d_open)();
        int             (*d_close)();
        int             (*d_read)();
        int             (*d_write)();
        int             (*d_ioctl)();
        int             (*d_stop)();
        struct tty      *d_ttys;
};
```

enthalten. Fuer den Treiber stehen nur diese Eintrittspunkte zur Verfuegung. Wird ein Eintrittspunkt fuer den Treiber nicht benoetigt, sind diese Routinen als Leerroutinen im Modul zu belassen, oder es ist der Fehler ENODEV in die User-Struktur einzutragen.

Um den Benutzer die Einbindung eines eigenen Geraetetreibers zu erleichtern, sind bereits Dummy-Treibermodule fuer 6 Treiber vorbereitet, die Namen lauten

/usr/sys/dev/udev1.c bis
/usr/sys/dev/udev6.c

Weiterhin stehen "Rahmenprogramme" fuer Block-, Character- und Terminaltreiber zur Verfuegung:

/usr/sys/dev/blockdev.c
/usr/sys/dev/chardev.c
/usr/sys/dev/ttydev.c

5.4. Aendern der Systemnamen

Am Ende der Datei wpar.c sind 3 Namen vereinbart:

```
char utsysname[UTS_LENGTH] = SYSNAME;

char utsnodename[UTS_LENGTH] = NODENAME;

char utsversion[UTS_LENGTH] = VERSION;
```

SYSNAME, NODENAME und VERSION sind in der Datei /usr/sys/h/ver.h definiert und koennen dort (mit sysgen) geaendert werden:

```
#define SYSNAME         "P8000"
#define NODENAME        "WEGA"
#define VERSION         " 3.0"
```

1. Systemname (utssysname)

   Ueber den Systemnamen kann der Kern mittels des uname(2) System-Calls identifiziert werden. Dieser Name kann bis zu 8 Zeichen lang sein. Er kann Leerzeichen und spezielle Zeichen enthalten. Der Systemname erscheint in einigen Ausschriften.

2. Network-Node-Name (utsnodename)

   Ueber den Network-Node-Namen kann der Kern mittels uname(2) System-Call identifiziert werden. Der Name kann bis 8 Zeichen lang sein und Leerzeichen und spezielle Zeichen enthalten. Der Network-Node-Name wird von uucp(1) zur eigenen Identifizierung und zur Identifizierung anderer Systeme benutzt. Er wird auch bei Login ausgegeben.

3. Versionsname / -nummer

   Ueber den uname(2) System-Call kann die Version des Kerns ermittelt werden. Der Name kann bis zu 8 Zeichen lang sein und Leerzeichen und spezielle Zeichen enthalten.

5.5. Generierung des Kerns

Die Generierung eines neuen Kerns erfolgt im Directory

/usr/sys/conf

durch die Eingabe des Kommandos:

sysgen

Sysgen ist ein Shell-Programm. Es arbeitet interaktiv und stellt folgende Fragen, die mit "y" (ja) oder "n" (nein) beantwortet werden koennen:

"---- sysgen ----"

"Version X.Y"

"Do you want to change system constants ?:"

Bei Eingabe von "y" wird der Editor vi fuer die Datei /usr/sys/h/sysparm.h aufgerufen und die Systemkonstanten koennen geaendert werden. Der Editor wird durch die Kommandos "ZZ" oder ":x" beendet.

"file system layout:"
(usr/sys/h/mdsize.h wird angezeigt)

"Do you want to change file system layout ?:"

Bei Eingabe von "y" wird die Laenge der einzelnen Plattenbereiche abgefragt. Anschliessend wird gefragt, ob eine zweite Platte eingebunden werden soll:

"size of /usr:"
:
:
"Do you want to add a second harddisk ?:"

"aktual version parameters:"
(usr/sys/h/ver.h wird angezeigt)

"Do you want to change version parameters ?:"

Nach Eingabe von "y" koennen die Namen in /usr/sys/h/ver.h geaendert werden.

"Do you want to add new driver to the system ?:"

Nach Beantwortung dieser Frage mit "y" kann der Name eines eigenen Geraetetreibers eingegeben werden. Dieser muss bereits als Objektkode vorliegen.

Das Programm sysgen benutzt zur Erstellung des Kerns das make(1) Programm. In der Datei /usr/sys/conf/make.wega sind alle erforderlichen Anweisungen zur Generierung des Kerns enthalten. Zusaetzlich wird ein neues sekundaeres Bootstrap-Programm erzeugt (make.boot), falls die Lage des Root-Dateisystems auf der Systemplatte geaendert wurde.

Der neue Kern besitzt den Namen wega, das neue Boot-Programm den Namen boot. Beide Programme befinden sich in dem Directory /usr/sys/conf. Die Funktion des neuen Kerns sollte zuerst mit einem manuellem Start getestet werden.

Der Start von WEGA erfolgt ueber das alte sekundaere Bootstrap-Programm mit der Eingabe von:

```
md(0,0)sys/conf/wega
```

Funktioniert der Kern, koennen Kern und Boot-Programm in die richtigen Dateisysteme kopiert werden:

```
cp /wega /wega.old
cp /usr/sys/conf/wega /wega
cp /usr/boot /usr/boot.old
cp /usr/sys/conf/boot /usr/boot
chmod 400 /wega /usr/boot
```

6. Systemstoerungen

6.1. Abstuerze des Betriebssystems

Das Betriebssystem kann durch fatale Fehler in einen unerlaubten Zustand kommen und "abstuerzen". Im WEGA-Kern sind jedoch Vorkehrungen getroffen, um solche Zustaende zu erkennen und die weitere Arbeit im System zu verhindern. Das System gibt dann die Ausschrift "panic", gefolgt von weiteren Informationen, auf dem Systemterminal aus. Die Panik-Ausschriften sind im folgenden Abschnitt erlaeutert. Sowohl Software- als auch Hardware-Fehler koennen der Grund eines Absturzes sein. Wurde zuvor der WEGA-Kern neu generiert, kann angenommen werden, dass ein fehlerhafter Kern erstellt wurde, anderenfalls wird der Fehler in der Hardware liegen.

6.2. Panik-Ausschriften

blkdev

> Die getblk-Routine (eine interne WEGA-Kern-Routine) wurde mit einer nichtexistierenden Major-Geraetenummer als Argument aufgerufen.
> (WEGA neu starten, Betriebssystem ueberpruefen)

devtab

> Die getblk-Routine erhielt eine Major-Geraetenummer, fuer die ein Null-Eintrag in der Device-Tabelle existiert.
> (WEGA neu starten, Betriebssystem ueberpruefen)

EPU inst.

> Ein CPU-Instruction-Trap trat im System-Mode auf.

init

> Ein E/A-Fehler trat waehrend der Initialisierung des Kerns beim Lesen des Superblocks des Dateisystems root auf.
> (WEGA neu starten, Dateisystem root kann defekt sein.)

IO err in swap

> Ein E/A-Fehler trat bei swap auf.

Kernel seg. violation
Kernel/fp seg. violation

> Eine der MMUs stellte eine Zugriffsverletzung fest.

Kernel syscall

Ein System-Call trat im System-Mode auf.

<md>: fatal error

Ein nichtbehebbarer E/A-Fehler trat im Dateisystem root auf.

no fs

Die getfs-Routine (eine interne WEGA-Kern-Routine) kann ein E/A-Geraet nicht in der mount-Tabelle finden.

no int

Die iget-Routine (eine interne WEGA-Kern-Routine) kann ein E/A-Geraet nicht in der mount-Tabelle finden.

no procs

In der Prozesstabelle ging ein Prozesseintrag verloren. Die newproc-Routine (eine interne WEGA-Kern-Routine) vermisst den Eintrag nach dem sie feststellte, dass der Prozess existiert.
(WEGA neu starten, eventuell ist Dateisystem root fehlerhaft.)

NVI

NVI ist im System nicht zugelassen.

Out of swap

Die swap-Tabelle (im WEGA-Kern) ist voll.

Out of swap space

Ein Programm kann nicht ausgelagert werden, da der swap-Bereich der Platte voll ist.

NMI (parity)

Ein Paritaetsfehler trat im RAM-Bereich des Kerns auf.

Privileged inst.

Ein privilegierter Instruction-Trap trat im System-Mode aus.

restart

Der WEGA-Kern sollte neu gestartet werden.

Running a dead proc

Prozesswechsel zu einem schon beendeten Prozess.

Unexpected VI

Ein nichtzugelassener vektorisierter Interrupt trat auf.

zero wchan

Ein fehlerhafter Parameter (0) wurde der sleep-Routine (eine interne WEGA-Kern-Routine) uebergeben.
Das System muss neu gestartet werden. Ist das nicht moeglich (die /etc/INIT ist meist defekt), muss das Dateisystem root neu installiert werden.

6.3. Systemwarnungen

Der WEGA-Kern gibt, neben den Panik-Ausschriften, weitere Fehlerausschriften aus. Diese Fehler stellen wichtige Systemwarnungen dar.

bad block on dev x/y

Ein Block einer Datei befindet sich nicht zwischen der i-Liste und dem Ende des Dateisystems.
(x - Major-, y - Minor-Geraetenummer)

bad count on dev x/y

Der Konsistenztest fuer den Superblock eines mit mount verbundenen Dateisystems fuehrt zu Fehlern. Das Dateisystem muss neu installiert werden (Dateien retten, mkfs ausfuehren und Dateien rueckspeichern).

bad free count on dev x/y

Der Superblock des Dateisystems x/y ist voll bzw. fehlerhaft. Die Laenge der Freiblockliste, die im Superblock gespeichert ist, ist falsch. Zur Beseitigung des Fehlers sollten einige Dateien geloescht werden. Wird damit nicht der Fehler beseitigt, muss das Dateisystem neu installiert werden.

err on dev x/y bn=n er=0xm,0xo

Im E/A-Geraet x/y trat ein Fehler auf. Block n kann nicht gelesen bzw. geschrieben werden (m, o interner Fehlerkode).

Out of inodes on dev x/y

Im Dateisystem des E/A-Geraetes x/y sind keine freien Inodes vorhanden. Nichtbenoetigte Dateien sollten geloescht werden.

Out of space on dev x/y

Das Dateisystem des E/A-Geraetes x/y ist voll. Es muss gesaeubert werden. 10% des Speicherplatzes des Dateisystems sollten stets als Reserve frei bleiben. Sind es weniger, sollte der Systemverwalter die Benutzer zum Loeschen der nichtbenoetigten Dateien auffordern. Die Programme df(M) und fsck(M) zeigen den freien Speicherplatz eines Dateisystems an. Mit den Kommandos find(1) und quot(M) koennen diejenigen Benutzer herausgefunden werden, die den meisten Platz beanspruchen.

Inode table overflow

Die Inode-Tabelle im Kern ist voll. Es muss die Beendigung einiger Prozesse abgewartet werden. Unkritische Prozesse, die Dateien eroeffnet haben, koennen auch mit kill(1) abgebrochen werden.
Tritt dieser Fehler oefter auf, kann der WEGA-Kern mit einem vergroesserten NINODE-Parameter neu generiert werden.

<md>: unexpected VI 0xn ignored
<md>: fatal error

In allen Faellen erkennt der Plattentreiber einen Fehler, der nicht beseitigt werden kann. Das System sollte neu gestartet werden.

no file

Die Dateitabelle im Kern ist voll. Es muss abgewartet werden, bis einige Prozesse beendet sind. Unkritische Prozesse koennen auch mit kill(1) abgebrochen werden.
Tritt dieser Fehler oefter auf, kann der WEGA-Kern mit einem vergroesserten NFILE-Parameter neu generiert werden.

NMI (manuell)
NMI (parity)

Ein NMI trat auf. Im System gibt es zwei NMI-Quellen, die NMI-Taste und NMI bei Paritaetsfehler im RAM.

no procs

Die Prozess-Tabelle im Kern ist voll. Das Kommando ps(1) zeigt alle Prozesse an. Mit kill(1) koennen Prozesse beendet werden.
Tritt der Fehler oefter auf, kann der WEGA-Kern mit einem vergroesserten NPROC-Parameter neu generiert werden.

proc on q

Das Betriebssystem wollte einen Prozess in die Run-Warteschlange eintragen. Der Prozess war dort aber schon vorhanden.
(WEGA neu starten)

Out of text

Die Texttabelle im Kern ist voll.
Tritt der Fehler oefter auf, kann der WEGA-Kern mit einem vergroesserten NTEXT-Parameter neu generiert werden.

Warning: spurious int

Ein fehlerhafter Interrupt trat auf. Die Ursache kann in statischen Entladungen liegen.

6.4. Beseitigung von Fehlern

Die Kommandos kill(1) und ps(1) sind wichtige Hilfsmittel zur Fehlerbeseitigung. Verriegelt ein Benutzer durch ein fehlerhaft arbeitendes Programm sein Terminal oder laesst sich ein Programm nicht mit der "DEL"-Taste beenden, kann der Superuser das Kommando

ps -leaf

von einem anderen Terminal benutzen. Dieses gibt einen detallierten Status aller augenblicklich im System laufendenden Prozesse aus. Jeder Prozess wird durch eine Prozessidentifikation (pid) gekennzeichnet. Der Abbruch eines Prozesses erfolgt mit dem Kommando kill(1) und der pid-Nummer als Argument, z.B:

kill 213

Ein Benutzer wird durch das Beenden seiner Login-Shell aus dem System ausgetragen. Nach einem neuen Login wird eine neue Login-Shell gestartet. Dieser Vorgang kann erforderlich werden, wenn die Shell zerstoert ist und nicht mehr mit dem Terminal kommuniziert.

Die Wirkung des Kommandos kill kann mit der Option -9 (siehe signal(2)) sicherer gemacht werden:

```
kill -9 <pid>
```

6.5. Systemstatus anzeigen

Nach dem Druecken der NMI-Taste gibt der WEGA-Kern verschiedene Status-Informationen aus. Sie zeigen den Zustand des Systems zum Zeitpunkt der Tastenbetaetigung an. Diese Informationen sind nur fuer den Systemprogrammierer interessant.

Das Format ist wie folgt:

```
<STATE xx0xxxx eventid=xxxx, fcw=xxxx, pcseg=xxxx, pcoff=xxxx>
<CPU-Register>
<System-SP>
<letzter bearbeiteter Interrupt>
<anhaengende Interrupts>
<Scheduler Status>
<unterbrochener Prozess und sein Eigner> oder
<Scheduling or idle>
<Status der verschiedenen Segmente
 des unterbrochenen Prozesses>
```

7. WEGA-Uebersicht

7.1. Drucker-Spooler

7.1.1. Funktion

Der Drucker-Spooler steuert die Ausgabe von Daten zu einem oder mehreren Druckern. Ein Programm kann jederzeit zu druckende Daten ausgeben. Der Spooler merkt sich die Auftraege, wenn der Drucker besetzt oder nicht bereit ist. Sind mehrere Drucker im System vorhanden, koennen die Ausgaben zu jedem dieser Drucker gerichtet werden.

Der Drucker-Spooler von WEGA besteht aus den Programmen:

nq(1) zur Generierung der Druckeranforderung,

lpr(1) einem Link auf nq(1),

xq(1) zur Anzeige und zur Modifizierung der verschiedenen Auftraege und

dqueuer(M) fuer die Ausgaben an den Drucker mit den Backend-Programmen

/usr/lib/lp fuer normale Drucker und

/usr/lib/text fuer Drucker mit Textqualitaet,

xq(M) fuer zusaetzliche Funktionen des Systemverwalters.

Das Directory

/usr/spool/queuer

enthaelt die notwendigen Konfigurations-, Status- und Request-Dateien fuer die Funktion des Spoolers.

Der Prozess dqueuer(M) wird durch /etc/rc_csh gestartet. Die Programme nq(1), lpr(1) und xq(1) richten ueber Signale Auftraege an diesen Prozess. Der Prozess dqueuer(M) setzt das Prozess-Id des entsprechenden Programms in dem zugeordneten Feld der "aktiven Konfigurationsdatei" /usr/spool/queuer/activeconfig. Mit dieser Methode und mit verschiedenen "record locking"-Techniken koennen alle Spooler-Programme ohne Konflikte miteinander kommunizieren. Zusaetzlich deckt dqueuer(M) mehrfache Kopien seines Programms auf.

Das Programm dqueuer(M) fragt staendig die Bereitschaft der entsprechenden E/A-Geraete (Drucker) ab und startet die Backend-Programme zur Ausgabe an ein spezielles Geraet. Der Prozess dqueuer wird ueber pause(2) ausgesetzt, wenn keine Ausgabegeraete bereit sind oder keine Auftraege vorliegen. Nach einem Signal von einem anderen Prozess setzt er seine Arbeit fort. Die Programme nq(1) und lpr(1) senden Signale fuer Auftraege aus. Das Programm xq(1) kann ein Signal beim Wechsel des Systemstatus aussenden. Die Backend-Programme senden Signale bei Beendigung einer Druckausgabe. Ausgabegeraete im Offline-Zustand werden von dqueuer(M) bis zur erneuten Bereitschaft einmal pro Minute abgefragt.

7.1.2. Einige Superuser-Spooler-Kommandos

Eine Druckerausgabe kann z.B. bei Defekten des Druckers mit

```
xq -q que:dev -sd
```

angehalten werden ("-s"). Jeder weitere Zugriff auf das Geraet "dev" wird verhindert. Das Kommando

```
xq -q que:dev -Ud
```

erlaubt die weitere Benutzung des Ausgabegeraetes "dev" (siehe xq(M) und dqueuer(M)).
Ein augenblicklich nicht ausgefuehrter Auftrag wird mit

```
xq -d xxx (xxx - Auftragsnummer)
```

gestrichen.

Das Programm xq(1) listet mit

```
xq
```

alle Auftragsnummern und Dateien aus.

Der Drucker-Spooler kann auch Drucker ueber den Geraetetreiber tty steuern. Der entsprechende tty-Kanal muss in der Datei /etc/inittab mit "k" fuer ein Login deaktiviert sein. Der Eintrag

```
2:00:c:/etc/GETTY tty0 2
```

wird in

```
2:00:k:
```

geaendert, wenn an Kanal tty0 ein Drucker angeschlossen ist.

Mit dem Kommando

kill -2 1 oder
INIT 2

wird die Datei /etc/inittab neu gelesen und der Kanal fuer ein Login gesperrt. Das neue Ausgabegeraet muss auch in die Datei /usr/spool/queuer/config (siehe folgender Abschnitt) eingetragen werden.

7.1.3. /usr/spool/queuer/config

Fuer den Drucker-Spooler ist in der Datei /usr/spool/queuer/config die Konfiguration des Systems abgelegt. Drei Arten von Eintraegen sind moeglich:

- Kommentare,
- die Beschreibung der Ausgabewarteschlangen und
- die Beschreibung der Ausgabegeraete.

Das folgende Beispiel zeigt die Datei /usr/spool/queuer/config fuer ein System mit zwei Druckern in der ersten Ausgabewarteschlange und einem Drucker mit Textqualitaet in einer zweiten Ausgabewarteschlange. Dieser Drucker ist an tty6 angeschlossen.

```
#
# Sample Configuration File
#
#       This file describes a system with 2 line
#       printers and one text quality printer.
#
#       first queue ... lpr has two line printers
#
Qlpr,R,N,S
#       line printer 1
D1,R,/dev/lp,/usr/lib/lp
#       line printer 2
D2,R,/dev/lp2,/usr/lib/lp
#
#       second queue ... text has one text quality printer
#
Qtext,R,N,S
#       text quality printer (typical option
#       shown ... )
D1,R,/dev/tty6,/usr/lib/text -T "9600 -nl"
#
#
```

Die Komponenten dieses Beispiels sind:

1. Kommentare, sie beginnen mit dem Zeichen "#".

2. Queue-Deskriptor, beginnend mit "Q".

   Jeder Queue-Deskriptor enthaelt folgende Informationen:

Queue-Name	1-8 Zeichen
Status	R = bereit (ready) D = nicht bereit (down)
minimale Prioritaet	R = hoechste (rusch) N = normale (normal) D = niedrigste (defered)
Auswahlprinzip	F = first in first out S = kleinste Datei zuerst

   Fuer jeden Geraetetyp sollte eine eigene Ausgabewarteschlange vorhanden sein, z.B "lpr" fuer alle "normalen" Drucker und "text" fuer alle Drucker mit Textqualitaet.

3. Geraetedeskriptor, beginnend mit "D".
   Fuer jedes physische Geraet ist ein Geraetedeskriptor vorhanden. Diese Deskriptoren folgen dem Queue-Deskriptor. Ein Geraetedeskriptor hat folgendes Format:

Geraetename	1-5 Zeichen
Status	R = bereit (ready) D = nicht bereit (down)
Geraetedatei	Eintrag in /dev fuer das Geraet
Backend-Programm	Eintrag in /usr/lib
Optionen	fuer das Backend-Programm (siehe backend(M))

   Das Kommando

   /etc/dqueuer -n

   gibt eine Uebersicht der Installationen in der /usr/spool/queuer/config aus. Nach Aenderungen in dieser Datei muss das Kommando

   /etc/dqueuer -r

eingegeben werden. Dieses Kommando aktiviert die Aenderungen. Aenderungen im Status, der Prioritaet und der Auswahlkriterien werden damit jedoch nicht wirksam. Es kann aber xq(M) benutzt oder WEGA neu gestartet werden.

7.2. Vom Systemstart zum Login

7.2.1. Initialisierung des Kerns

Der WEGA-Kern wird ueber den U8000-Softwaremonitor sowie ein primaeres und sekundaeres Bootstrap-Programm gestartet. Nach verschiedenen internen Initialisierungen kopiert der Kern aus sich heraus ein kleines Programm und laesst es als Prozess 1 laufen. Dieses Programm entspricht folgender C-Shell-Prozedur:

```
main() {
exec ("/etc/INIT", 0);
while (1);
}
```

Der Prozess /etc/INIT erzeugt danach weitere Prozesse, die zum Login der Benutzer fuehren.

7.2.2. /etc/INIT

INIT als Urprozess (Prozess 1) kann dazu benutzt werden, alle anderen Prozesse im System zu steuern. INIT besitzt 9 Zustaende, 1 bis 9. Zustand 1 ist dem Single-User-, Zustand 2 dem Multi-User-Mode zugeordnet. Bei einem manuellen WEGA-Start wird Zustand 1, bei einem automatischen Start Zustand 2 eingenommen. (Der Zustand des Systems kann auch ueber eine Kommandoeingabe gewechselt werden, z.B. Eingabe von INIT 2). INIT liest bei jedem Zustandswechsel seine Initialisierungsdatei /etc/inittab. Diese Datei definiert die Aktivitaeten bei einem Zustandswechsel. Sie besitzt Eintraege fuer jeden Terminalkanal in der Form:

```
state:id:flags:command
```

state: Zustand des Systems, Integerwert zwischen 1 und 9
1 = Single-User-Mode
2 = Multi-User-Mode

id: 2 Byte Identifikator zur Zuordnung der Prozessgruppe, entspricht dem Terminalkanal, z.B 00 fuer tty0 und co fuer console.

flags: Die Zeichen "t", "k" und "c" sind zugelassen. Sie dienen zur Steuerung der Prozesse, siehe unten.

command: Kommando, das von der Standard-Shell /bin/sh ausgefuehrt werden kann. Optional koennen Argumente fuer das Kommando folgen.

Die Flags "t" (terminate, Signal 15) und "k" (kill, Signal 9) bewirken, dass alle Prozesse, die der Prozessgruppe id zugeordnet sind, vor dem Wechsel des Zustands beendet bzw. abgebrochen werden. Beide Flags "t" und "k" koennen gleichzeitig in dieser Reihenfolge angegeben werden. Anschliessend startet INIT die Shell-Prozedur /etc/rc. Die Prozedur /etc/rc erhaelt von INIT 3 Argumente, augenblicklicher Zustand, Anzahl der Wechsel in diesen Zustand und vorheriger Zustand. Die Prozedur /etc/rc fuehrt verschiedene Aufgaben aus, siehe Abschnitt 2.3.1.

Nach den Prozeduren /etc/rc werden die in inittab angegebenen Kommandos ausgefuehrt, aber erst wenn alle Prozesse der Prozessgruppe beendet sind. Das Zeichen "c" bewirkt, dass das Kommando immer wieder neu gestartet wird, sobald die Prozesse der Prozessgruppe beendet sind. Ohne "c" wird das Kommando nur einmal ausgefuehrt.

Nach einem manuellen Start erzeugt INIT ueber fork(2) den Prozess /bin/csh. Die Standardeingabe, -ausgabe und die Fehlerausgabe sind an das Systemterminal gerichtet. Das System befindet sich im Single-User-Mode und wechselt mit INIT 2 in den Multi-User-Mode. Die Prozeduren /etc/rc und /etc/rc_csh werden ausgefuehrt.

Bei einem automatischen Start ueberspringt INIT den Single-User-Mode und beginnt gleich mit der Abarbeitung der Prozedur /etc/rc.

Typische Eintraege in der /etc/inittab fuer Zustand 1 sind:

```
1:00:k:/etc/GETTY tty0 !
```

und fuer Zustand 2:

```
2:00:c:/etc/GETTY tty0 2
```

Fehlt ein Eintrag fuer einen Identifikator id, wird nichts ausgefuehrt. Aktive Prozesse bleiben erhalten, neue werden nicht gestartet.

Zum Deaktivieren eines seriellen Kanals wird im Feld "flags" ein "k" eingetragen und das Kommando geloescht, z.B. fuer Kanal TTY3:

```
2:03:k:
```

Beachten Sie bitte !

Es gibt keine Beschraenkung der Eintraege auf Login-Vorgaenge. Neue Systemzustaende koennen definiert und Applikationsprogramme gestartet werden.

7.2.3. /etc/GETTY

Das erste Argument von /etc/GETTY legt die Baudrate und verschiedene andere Parameter des Terminals fest. GETTY kann variable Baudraten einstellen. Das ist fuer "dealup"-Kanaele wichtig. Nach jedem Break-Zeichen (Benutzer drueckt Break- oder DEL-Taste) stellt GETTY die naechste Baudrate ein.

Folgende Terminalarten sind in GETTY vereinbart:

Typ	Baudrate(n)	Bedeutung
0	300,1200,150,110	dealup
-	110	Teletype (Mod. 33/35)
1	150	Teletype (Mod. 37)
2	9600	Terminal
3	1200,300	dealup
5	300,1200	dealup
4	300	Terminal
6	2400	Terminal
7	19200	Terminal

!	Aktualisierung der Accounting-Dateien und Beendigung von GETTY

Das Programm GETTY gibt die Login-Aufforderung auf dem Terminal aus. Die Uebernahme des vom Benutzer eingegebenen Login-Namens erfolgt im raw-Modus. Empfaengt GETTY den Login-Namen ausschliesslich in Grossbuchstaben, erfolgen alle weiteren Ein- und Ausgaben in Grossschreibung. Ein faelschlich grossgeschriebener Login-Name wird durch Eingabe eines ungueltigen Passwortes und der Eingabe von Control-D auf das neue Login korrigiert. Mit Control-D startet ein neues GETTY.

7.2.4. Login

Das Programm login wird von GETTY mit dem Login-Namen als Argument aufgerufen. Beginnt dieser Name mit dem Zeichen "-", erscheint die "message of the day" nicht. /etc/login ist mit /bin/login identisch. Login fragt nach dem Passwort, das mit dem Eintrag in der Datei /etc/passwd verglichen wird. Existiert diese Datei nicht, kann sich niemand im System anmelden. Zur Beseitigung des Fehlers wird das System neu in den Single-User-Mode gebracht und die Datei /etc/passwd angelegt bzw. vom Backup-Medium kopiert. Ist das Passwort richtig, aktualisiert login die Login-Accounting-Datenbasis, vorausgesetzt die beiden Dateien /usr/adm/wtmp und /etc/utmp existieren.

Anschliessend wechselt login das Arbeitsdirectory in das Benutzer-Home-Directory. Existiert dieses Directory nicht, erscheint die Ausschrift "No directory" und ein neues Login wird angefordert. Dieser Fehler tritt z.B. auf, wenn die Dateisysteme noch nicht mit mount(M) verbunden wurden. Als naechstes wird der Eigner fuer Terminal und Prozesse auf den Benutzer gesetzt. Das Programm login initialisiert Home-Directory, Shell und Terminaltyp. Der Terminaltyp wird z.B. von vi benutzt und ist in der Datei /etc/ttytype enthalten. Es existieren dort Eintraege in der Form

cc <device name>

zum Beispiel

P8 console

Dieser Eintrag besagt, dass /dev/console ein Terminal vom Typ P8000-Terminal ist. Die Terminaltypen sollten in der Datei /etc/termcap definiert sein. Ist das nicht der Fall, wird der Typ zu "xx" (unbekannt) gesetzt.

Das Programm login gibt den Inhalt der Datei /etc/motd, der "message of the day", aus. Es ueberprueft die Datei /usr/spool/mail/<name>, falls die Mailbox fuer den Benutzer existiert, auf Nachrichten. Zum Schluss startet login die Shell, die in der Datei /etc/passwd eingetragen ist, mit dem Argument "-". Dieses Argument bewirkt, dass die Shell die Kommandos von der Datei $HOME/.profile (/bin/sh) oder $HOME/.cshrc (/bin/csh) einliest. Die C-Shell interpretiert zusaetzlich die Datei $HOME/.login. Existiert die Shell nicht, erscheint die Ausschrift "No Shell". Ein neues Login wird angefordert.

7.2.5. Logout

Meldet sich ein Benutzer, der die C-Shell benutzte, ab (logout), fuehrt /bin/csh die Prozedur $HOME/.logout aus. Anschliessend wird das Programm beendet. Die Standard-Shell /bin/sh endet bei einem Logout ohne weitere Aktivitaeten. Das Programm hat INIT ein wait(2) offen. Stammt wait von einem unmittelbaren Folgeprozess (GETTY) und sind die Aktion "c" fuer diesen Prozess und der entsprechende Systemstatus in /etc/inittab vereinbart, wird das Programm GETTY neu gestartet.

7.3. WEGA-Kern und Shell

Dieser Abschnitt beschreibt den WEGA-Kern. Vier Teile werden behandelt:

1. Prozesssteuerung,

2. E/A-System,

3. Dateisystem und

4. WEGA-Kommandointerpreter (Shell).

7.3.1. Prozesssteuerung

Benutzer fuehren Programme in einer Umgebung aus, die als Benutzerprozess (user process) bezeichnet wird. Ein Prozess ist einfach ein Programm, das gerade ausgefuehrt wird und verschiedene Komponenten, die die Programmausfuehrung ermoeglichen, enthaelt. Diese beinhalten ein Speicherabbild, die CPU-Register-Werte und den Status der eroeffneten Dateien. Benoetigt ein Prozess eine Systemfunktion, wird der Kern wie ein Unterprogramm aufgerufen (Systemruf). Die Umgebung des Prozesses wechselt. Der Prozess laeuft im System-Mode weiter und fuehrt Instruktionen im WEGA-Kern aus. Benutzer- und Systemprozesse sind Phasen des gleichen Prozesses, die niemals gleichzeitig vorkommen koennen. Um das System zu schuetzen, hat der Systemteil des Prozesses einen eigenen Stackbereich.

Ein Prozess hat bis zu 7 Segmente:

1. ein User-Mode-Text(Programm)-Segment, das nur Instruktionen und unmittelbare Operanden enthaelt,

2. ein User-Mode-Datensegment,

3. ein User-Mode-Stacksegment,

4. ein System-Mode-Textsegment,

5. ein System-Mode-Datensegment, das alle Daten des Betriebssystems enthaelt, die zu allen Prozessen global sind,

6. ein lokales System-Mode-Stacksegment,

7. ein lokales System-Mode-Datensegment, oft als u-Vektor (Struktur u) bezeichnet.

Als Segment wird hier eine logische Einheit bezeichnet, nicht ein Speichersegment. Das User-Mode-Text- und User-Mode-Datensegment koennen zu einem Segment zusammengefasst werden. Sind sie getrennt, koennen alle Prozesse, die das gleiche Programm ausfuehren, das gleiche Textsegment benutzen. Diese Textsegmente sind Nur-Lese-Segmente. Die Trennung von Text- und Datensegment erfolgt waehrend der Uebersetzung des Programms mit der Option "-i" des Linkers.

Alle augenblicklich im System vorhandenen Nur-Lese-Segmente werden in der Texttabelle des Kerns verwaltet. Ein Eintrag in der Texttabelle enthaelt die Speicheradresse im sekundaeren Speicher (swap-Bereich), die Speicheradresse im primaeren Speicher (RAM) und einen Zaehler fuer die Anzahl der Prozesse, die sich diesen Eintrag teilen. Ein Eintrag wird erzeugt, wenn der erste Prozess das Textsegment benoetigt. Gleichzeitig wird das Textsegment in den sekundaeren Speicher geladen. Folgende Prozesse erhoehen nur den Zaehler im Eintrag der Texttabelle. Erreicht der Zaehler wieder den Wert 0, wird der Eintrag geloescht und der primaere und sekundaere Speicherbereich werden freigegeben.

Das Datensegment kann bei getrenntem Text- und Datenbereich 64 KByte gross sein. Sind die Segmente nicht getrennt, steht fuer Kode und Daten nur der gleiche Bereich zur Verfuegung. Im Datenbereich ist auch das User-Stacksegment enthalten. Das User-Datensegment enthaelt keine Systemdaten und keine E/A-Puffer des Systems.

Der User-Datenbereich hat zwei sich veraendernde Grenzen. Der User-Stack wird automatisch vergroessert, sobald sein Speicherbereich ueberschritten wird (Trap der MMU). Das User-Datensegment wird nur ueber eine entsprechende Anforderung an den Kern vergroessert oder verkleinert (siehe brk(2)). Der Inhalt des neu reservierten primaeren Speichers wird mit 0 initialisiert.

Der System-Datenbereich ist ein kleines Segment mit einer festen Laenge. Es enthaelt alle Daten ueber den Prozess, die vom System benoetigt werden, sobald der Prozess aktiv ist. Er wird mit dem Prozess ausgelagert (swap). Diese Daten sind zum Beispiel die geretteten CPU-Register, Deskriptoren fuer eroeffnete Dateien, Accounting-Informationen, Bereiche fuer temporaere Daten und der Stackbereich fuer die Systemphase des Prozesses. Ein Benutzerprozess kann nicht auf diesen Bereich zugreifen.

Prozesse werden in die Prozesstabelle eingetragen. Fuer jeden Prozess existiert ein Eintrag. Jeder Eintrag enthaelt Daten, die das System benoetigt, wenn der Prozess nicht aktiv ist. Diese Daten sind z.B. der Name des Prozesses, der Speicherplatz der anderen Segmente und Informationen fuer den Scheduler. Ein Eintrag entsteht, sobald der Prozess erzeugt wird und wird geloescht, wenn der Prozess endet. Der Kern kann auf einen Prozesseintrag direkt zugreifen. Die Eintraege in der Prozesstabelle sind eigentlich die Definition aller Prozesse. Von diesen Eintraegen aus, wird auf alle Daten der Prozesse zugegriffen.

7.3.1.1. Prozesserzeugung und Programmausfuehrung

Prozesse werden mit dem fork(2)-Systemruf erzeugt. Dieser Systemruf bewirkt, dass eine Kopie des Prozesses angelegt wird, der den Systemruf ausfuehrte. Ein zweites Abbild des gleichen Prozesses entsteht. Die Ausfuehrung beginnt in beiden Abbildern. Die beiden Prozesse haben die hierarchische Beziehung von Vater und Sohn. Sie sind in ihrem Speicherabbild, in den eroeffneten Daten usw. vollkommen identisch.

Liefert der fork-Systemruf den Return-Wert -1, war fork nicht erfolgreich. Jeder Wert groesser -1 kennzeichnet die erfolgreiche Ausfuehrung, der Wert 0 wird dem Sohnprozess und das Prozess-Id des Sohnprozesses dem Vaterprozess uebergeben. Der fork-Systemruf allein ist noch nicht sehr nuetzlich. Wenn er mit dem exec(2)-Systemruf kombiniert wird, entsteht ein vollstaendiger Mechanismus zur Erzeugung und Ausfuehrung von Prozessen bzw. Programmen. Der Systemruf exec ersetzt das Abbild des Prozesses durch ein Abbild eines neuen Programmes, dessen Dateiname mit dem exec(2) uebergeben wird. Obwohl ein expliziter Prozesswechsel nicht erfolgt, wird das alte Programm ueberschrieben, das neue bekommt den Prozess und die Ausfuehrung beginnt am Anfang des neuen Programms. Mit anderen Worten, sobald ein Prozess einen exec-Systemruf zum Kern richtet fuehrt dieser Prozess nicht mehr den gleichen Programmkode aus, Text- und Datensegmente werden durch die Segmente des neuen Programms ersetzt und ausgefuehrt.

Wenn ein Programm (z.B. Pass 1 des Compilers) die Instruktion enthaelt, sich selbst durch ein anderes Programm (z.B. Pass 2) zu ersetzen, fuehrt es den exec-Systemruf fuer das neue Programm aus. Moechte aber das Programm nach der Ausfuehrung des zweites Programms wieder die Steuerung erhalten, fuehrt es den fork-Systemruf fuer einen Sohnprozess, der das exec durchfuehrt, aus und geht mit dem wait(2)-Systemruf in einen Wartezustand. Er wartet bis der Sohn-Prozess endet.

7.3.1.2. Swap-Mechanismus

Die meisten Daten eines Prozesses (das User-Datensegment, das System-Datensegment uns das Textsegment) werden bei Bedarf zu einem sekundaeren Speicher ausgelagert bzw. von dort wieder in den primaeren Speicher geladen. Das User-Datensegment und das System-Datensegment sind im primaeren Speicher zur Reduzierung der swap-Zugriffe dicht hintereinander angeordnet. Die Reservierung von primaerem und sekundaerem Speicher erfolgt nach dem gleichen einfachen "first-fit" Algorithmus. Waechst ein Prozess, wird ein neuer primaerer Speicherbereich reserviert. Der Inhalt des alten wird zum neuen Bereich kopiert. Der alte Speicherbereich wird freigegeben. Steht nicht genuegend Speicher zur Verfuegung, erfolgt die Reservierung im sekundaeren Speicher. Der Prozess wird zum sekundaeren Speicher ausgelagert, bereit zum Einlagern mit der neuen Groesse.

Der Scheduler ist ein separater Prozess im Kern, der andere Prozesse ein- und auslagert. Er sucht in der Prozesstabelle Prozesse, die ausgelagert und bereit zur Ausfuehrung sind. Er reserviert primaeren Speicher fuer den Prozess und laedt die Segmente in diesen Speicher. Dort bewirbt sich der Prozess mit allen anderen Prozessen im primaeren Speicher um CPU-Zeit. Ist kein primaerer Speicher verfuegbar, prueft der Scheduler die Prozesstabelle auf Prozesse, die ausgelagert werden koennen. Er waehlt einen Prozess aus und kopiert ihn zum sekundaeren Speicher. Anschliessend sucht er wieder Prozesse, die eingelagert werden koennen.

Der Scheduler benutzt zwei Algorithmen, jeweils einen zum Ein- und Auslagern der Prozesse. Der Prozess, der am laengsten ausgelagert ist, wird als erster eingelagert. Grosse Prozesse werden etwas benachteiligt. Der zweite Algorithmus bestimmt den Prozess, der ausgelagert wird. Prozesse, die nicht laufen oder auf langsame Ereignisse warten, werden nach der laengsten Verweildauer im primaeren Speicher ausgewaehlt. Es erfolgt wieder eine Benachteiligung grosser Prozesse. Die anderen Prozesse werden nach dem gleichen Algorithmus ueberprueft, aber nur ausgelagert, wenn keine anderen Prozesse gleicher Verweildauer mehr vorhanden sind. Damit wird die totale Blockierung des Systems verhindert.

Das Ein- und Auslagern beeinflusst nicht die Ausfuehrung der residenten Prozesse. Das System wird natuerlich langsamer, wenn das gleiche E/A-Geraet zum Speichern der Dateien benutzt wird.

7.3.1.3. Synchronisation und Prozesswechsel

Prozesse werden synchronisiert, indem Prozesse, die auf Ereignisse warten, schlafen (sleep). Ereignisse werden durch Adressen dargestellt. Das sind gewoehnlich Eintraege in Tabellen, die mit dem Ereignis in Beziehung stehen. Ein Prozess, der z.B. auf die Beendigung eines seiner Sohnprozesse wartet, wartet auf ein Ereignis, das die Adresse seines eigenen Eintrags in der Prozesstabelle ist. Endet ein Prozess, signalisiert er das Ereignis, das durch die Adresse des Prozesseintrags seines Vaters dargestellt wird. Die Signalisierung eines Ereignisses, auf das kein Prozess wartet, hat keine Wirkung. Die Signalisierung eines Ereignisses, auf das mehrere Prozesse warten, weckt diese Prozesse auf. Aber nur ein Prozess erhaelt die benoetigte Ressource, alle anderen Prozesse gehen wieder schlafen.

Zu jeder Zeit warten - bis auf den gerade in Ausfuehrung befindlichen Prozess - alle Prozesse auf Ereignisse. Muss der laufende Prozess auf ein Ereignis warten, wird dafuer ein anderer Prozess, fuer den das Ereignis schon auftrat, zur Ausfuehrung gebracht.

Die Prozessprioritaet hilft den Prozess zu bestimmen, der als naechster laufen soll. Jeder Prozess besitzt eine Prioritaet. Die Prioritaet eines Benutzerprozesses wird durch das Verhaeltnis zwischen CPU-Zeit und Gesamtlaufzeit bestimmt. Ein Prozess, der schon viel CPU-Zeit erhielt, wird auf eine niedrigere Prioritaet gesetzt. Damit werden interaktive Prozesse, die i. allg. ein kleines Verhaeltnis zwischen CPU- und Gesamtzeit haben, ohne weitere Vorkehrungen bevorzugt.

Das Verhaeltnis CPU- zur Gesamtlaufzeit wird jede Sekunde aktualisiert. Benutzerprozesse werden i. allg. jede Sekunde gewechselt. Ein Prozess mit einer hohen Prioritaet verdraengt einen Prozess mit einer niederen Prioritaet. Die Prioritaet eines laufenden Prozesses verringert sich und erhoeht sich fuer einen Prozess mit niedriger Prioritaet, der schon lange nicht ausgefuehrt wurde.

7.3.2. E/A-System

In WEGA gibt es zwei Arten der Ein-/Ausgabe, Blockeingabe/-ausgabe und Zeicheneingabe/-ausgabe, letztere oft auch als "raw"-Ein-/Ausgabe bezeichnet. Die Blockeingabe/-ausgabe bewirkt, dass die Daten erst ueber 512 Byte E/A-Puffer des Systems von bzw. zum E/A-Geraet uebertragen werden. Bei der "raw"-Eingabe/Ausgabe erfolgt die direkte Uebertragung einer gewuenschten Anzahl von Bytes zwischen Speicher und E/A-Geraet.

E/A-Geraete besitzen eine Major- und eine Minor-Geraetenummer, sowie eine Unterscheidung zwischen Block- und Zeichenuebertragung. Fuer jede Art (Block oder Zeichen) existiert eine separate Tabelle, die fuer jedes Geraet einen Eintrag besitzt. Jeder Eintrag enthaelt die Startadressen der verschiedenen Routinen des Geraetetreibers. Die Major-Geraetenummer entspricht der Nummer des Eintrags in der Tabelle. Die Minor-Geraetenummer wird dem Treiber uebergeben. Gewoehnlich benutzt der Treiber diese Nummer zur Unterscheidung mehrerer gleicher physischer Geraete, z.B. Auswahl des Hard-Disk-Laufwerks.

7.3.2.1. Blockeingabe/-ausgabe

Das Modell eines Block-E/A-Geraetes besteht aus sekundaeren Speicherbloecken von je 512 Byte, die in beliebiger Reihenfolge adressiert werden koennen. Die Adressierung erfolgt mit Nummern (0, 1, 2, ... bis zum Speicherende). Der Blockgeraetetreiber hat die Aufgabe, dieses Modell auf einem physischen Geraet zu emulieren.

Das Blockgeraet wird von einer Softwareschicht benutzt, die den Puffermechanismus realisiert. Das System verwaltet eine Anzahl von Puffern, die Blockgeraeten zugeordnet sind. Diese Puffer stellen einen Cache-Speicher fuer die Blockgeraete dar. Bei einen Lesezugriff wird zuerst der Cache-Speicher durchsucht. Existiert dieser fuer den gewuenschten Block, erfolgt die Bereitstellung der Daten ohne einen Zugriff auf das physische Geraet.

Existiert der Block im Cache-Speicher nicht, wird der am laengsten unbenutzte Puffer neu zugeordnet. Der Geraetetreiber wird aufgerufen, um diesen Speicher mit dem gewuenschten Block zu fuellen. Anschliessend werden die Daten bereitgestellt. Schreibzugriffe erfolgen analog den Lesezugriffen. Der Puffer wird ermittelt bzw. neu zugeordnet. Das Schreiben erfolgt, indem der Puffer einfach entsprechend markiert wird. Das physische Geraet wird fuer den Block erst wieder benutzt, wenn der Puffer neu zugeordnet wird. Der Puffer wird zum physischen Geraet uebertragen. Beim undefinierten Beenden des Systems (Reset-Taste), koennen die Bloecke auf dem E/A-Geraet noch nicht aktualisiert sein.

Im System gibt es den Systemruf sync, der alle ausstehenden Puffer zum E/A-Geraet schreibt. Im allgemeinen erfolgt dieser Systemruf alle 30 Sekunden durch den Prozess update(M). Er wird gestartet, wenn das System in den Multi-User-Mode geht. Dieser Prozess garantiert jedoch nicht die Konsistenz der Dateisysteme, wenn das System fehlerhaft angehalten wird. Zur Reparatur inkonsistenter Dateisysteme gibt es das Programm fsck(M) zur automatischen Beseitigung der Fehler. Die Reparatur kann aber auch manuell nach Abschnitt 4.6. erfolgen.

7.3.2.2. Zeicheneingabe/-ausgabe

Die Zeicheneingabe/-ausgabe schliesst die klassischen Zeichengeraete, wie Terminals und Drucker, ein. Aber auch die Disk-Laufwerke und Magnetbandgeraete fallen darunter, wenn sie direkt benutzt werden. Der Cache-Speicher wird umgangen, z.B. fuer das spurweise Kopieren von Platten. Benutzer-E/A-Anforderungen werden im grossen und ganzen unveraendert zum Treiber geleitet.

Es besteht eine Konvention, dass der Name der raw-E/A-Geraete mit dem Zeichen "r" beginnt und ansonsten mit dem Namen fuer das Blockgeraet uebereinstimmt. Der Zugriff auf ein raw-Geraet ist gewoehnlich schneller als ein Zugriff auf das Blockgeraet. Das raw-Geraet darf nur benutzt werden, wenn das zugeordnete Dateisystem nicht mit mount(M) verbunden ist.

7.3.3. Dateisysteme

In WEGA ist eine Datei ein eindimensionales Feld von Bytes. Das System legt keine weiteren Dateistrukturen fest. Dateien werden irgendwo in der Hierarchie der Directorys eingetragen, mehrfache Eintraege der gleichen Datei sind moeglich. Directorys sind auch Dateien, der Benutzer kann sie jedoch nicht direkt beschreiben.

7.3.3.1. Struktur von Dateisystemen

Die Disk ist ein wahlfrei adressierbares Feld von 512 Byte Bloecken. Dieses "Feld" wird gewoehnlich in mehrere Bereiche geteilt. Ein swap-Bereich wird benoetigt. Die anderen Bereiche sind Dateisystemen vorbehalten. Ein Dateisystem (Zugriff erfolgt ueber Block- oder raw-E/A-Geraet) ist in 4 Regionen aufgeteilt:

1. Block 0 ist fuer das primaere Bootstrap-Programm reserviert.

2. Block 1 ist der Superblock. Der Superblock enthaelt, neben anderen Informationen, die Groesse des Dateisystems, die Grenzen der anderen Regionen und eine Liste von Zeigern zu freien Inodes.

3. Der 3. Bereich ist eine Liste von Inodes. Inodes sind Strukturen von 64 Byte Laenge, die eine Datei definieren (Blockadressen, Dateityp, Zugriffsberechtigung, Eigner usw.). Die Inode-Nummer (i-number) waehlt das n. Inode in der Liste aus. Eine Datei wird durch die Angabe des E/A-Geraetenamens (Major- und Minor-Geraetenummer) und der Inode-Nummer selektiert.

4. Der 4. Bereich beginnt nach den Inodes und geht bis zum Ende des Dateisystems. Es ist der freie Speicherbereich fuer die Dateien.

Der freie Speicherbereich wird mit der Freiblock-Liste verwaltet. Jeder Block dieser Kette enthaelt die Adresse des naechsten Blocks der Kette. Jeder Block enthaelt die Adressen von bis zu 50 freien Bloecken. Mit einem E/A-Zugriff erhaelt das System 50 freie Bloecke und einen Zeiger zu weiteren freien Bloecken.

7.3.3.2. Directory-Dateien

Eine logische Directory-Hierarchie (Baumstruktur) im Dateisystem wird durch Directory-Dateien gebildet. Eine Directory enthaelt Eintraege von 16 Byte Laenge, 2 Byte Inode-Nummer gefolgt von 14 Byte fuer den Dateinamen. Das erste Directory (root) in der Baumstruktur besitzt die Inode-Nummer 2. Auf Directorys wird genauso wie auf normale Dateien zugegriffen.

7.3.3.3. Inode

Ein Zugriff auf eine Datei erfolgt ueber die Inode-Nummer, die das zugehoerige Inode auswaehlt. Das Inode ist der Deskriptor einer Datei und enthaelt eine Reihe von Informationen:

1. das User- und Gruppen-Id des Benutzers,

2. die Schutzbits (Zugriffsrechte),

3. die Disk-Adressen der Datei (Blocknummern),

4. die Groesse der Datei,

5. Datum, Zeit der Erzeugung, des letzten Zugriffs und der letzten Modifizierung der Datei,

6. die Anzahl der Links zu der Datei (Anzahl der Eintraege in den Directorys) und

7. einen Kode zur Unterscheidung zwischen normaler Datei, Directory- und Geraetedatei.

Das System verwaltet eine Tabelle aller aktiven Inodes im Kern. Wird auf eine neue Datei zugegriffen, ermittelt das System das Inode und kopiert den Inhalt in einen freien Eintrag der Tabelle im primaeren Speicher. Das ist das aktuelle Inode. Alle Aenderungen erfolgen zuerst hier. Erst nach dem letzten Zugriff, wird das Inode zurueck zum sekundaeren Speicher kopiert und der Eintrag im primaeren Speicher freigegeben.

Alle E/A-Operationen zu Dateien werden mit Hilfe des korrespondierenden Eintrags in der Inode-Tabelle durchgefuehrt. Inodes und Inode-Nummern sind fuer den Benutzer transparent. Zugriffe im Dateisystem erfolgen ueber die Pfadnamen der Directory-Baeume. Die Konvertierung des Pfadnamens in einen Inode-Eintrag beginnt mit einem bekannten Inode, das Inode des root- oder des aktuellen Directorys. Die naechste Komponente des Pfadnamens, die (naechste) Inode-Nummer, ermittelt das System durch Lesen des Directorys. Gleichzeitig ist damit das E/A-Geraet, das dem Dateisystem mit der Datei entspricht, bekannt. Nun kann auf den naechsten Eintrag in der Inode-Tabelle zugegriffen werden. Das setzt sich fort, bis das Inode fuer die letzte Komponente des Pfadnamens ermittelt wurde.

Ein Inode enthaelt 13 Blockadressen. Die ersten 10 adressieren die ersten 10 Bloecke einer Datei direkt. Die 11. Blockadresse zeigt zu einem Block, der die Adressen fuer die naechsten 128 Bloecke der Datei enthaelt (einfach indirekte Adressierung). Voraussetzung ist natuerlich immer, dass die Datei wirklich so lang ist. Die 12. Blockadresse zeigt zu einem Block, der die Adressen von 128 Bloecken enthaelt, die jeweils 128 Blockadressen der Datei enthalten (doppelt indirekte Adressierung). Fuer noch groessere Dateien ist mit der 13. Blockadresse eine dreifach indirekte Adressierung moeglich. Damit kann die maximale Groesse einer Datei

(10 + 128 + (128 * 128) + (128 * 128 * 128)) * 512 Byte

betragen.

Das Inode einer Geraetedatei enthaelt in der 1. Blockadresse den Geraetenamen, ausgedrueckt durch die Major- und Minor-Geraetenummern. Die weiteren zwoelf Adressen sind unbenutzt.

Ein Benutzerprozess greift ueber verschiedene Systemrufe auf das Dateisystem zu. Diese Systemrufe sind open(2), creat(2), read(2), write(2), lseek(2) und close(2). Im System-Datensegment fuer einen Benutzer existiert eine Tabelle fuer 20 eroeffnete Dateien. Diese enthaelt Zeiger zu den Eintraegen in der Inode-Tabelle. Fuer jede eroeffnete Datei existiert ein E/A-Zeiger. Dieser gibt den Offset in der Datei fuer den naechsten Lese- oder Schreibzugriff an. Fuer einen wahlfreien E/A-Zugriff kann der Offset mit dem lseek(2) Systemruf geaendert werden.

Fuer den geteilten Dateizugriff (mehrere Prozesse haben die gleiche Datei eroeffnet) ist es notwendig, dass voneinander abhaengige Prozesse einen gemeinsamen E/A-Zeiger, voneinander unabhaengige Prozesse getrennte E/A-Zeiger benutzen. Dadurch kann der E/A-Zeiger weder in der Inode-Tabelle noch in der Tabelle der eroeffneten Dateien enthalten sein. Eine weitere Tabelle, die "open file table", enthaelt die E/A-Zeiger. Prozesse, die sich die gleiche Datei als Resultat des Systemrufs fork teilen, teilen sich einen gemeinsamen E/A-Zeiger. Bei einer separaten Eroeffnung der gleichen Datei, wird nur der Inode-Eintrag geteilt, ansonsten eigene Eintraege in der Tabelle der eroeffneten Dateien (E/A-Zeiger) benutzt.

Die wichtigsten Systemrufe fuer Zugriffe auf das Dateisystem sind:

open(2) Der Systemruf open konvertiert einen Pfadnamen, der vom Benutzer vorgegeben wurde, in einen Eintrag in der Inode-Tabelle. Ein neuer Eintrag wird auch in der Tabelle der eroeffneten Dateien (open file table) angelegt. Dort wird ein Zeiger zum Eintrag in der Inode-Tabelle eingetragen. In das System-Datensegment fuer den Prozess wird ein Zeiger zum Eintrag in der Tabelle der eroeffneten Dateien eingetragen.

creat(2) Der Systemruf creat legt eine neue Datei an, indem ein neuer Eintrag in der Inode-Tabelle erzeugt wird. Der weitere Ablauf entspricht dem Systemruf open.

read(2) Lesen von der Datei, die eroeffnet wurde.

write(2) Schreiben zur Datei, die eroeffnet wurde.

lseek(2) Der Systemruf lseek manipuliert den E/A-Zeiger. Eine physische Positionierung erfolgt nicht.

close(2) Die Eintraege in den Tabellen, die durch open und creat angelegt wurden, werden wieder frei gegeben.

In den Eintraegen der Tabelle der eroeffneten Dateien und der Inode-Tabelle sind Zaehler vorhanden, in denen die Anzahl der gleichzeitigen Referenzen notiert ist. Die Eintraege werden erst frei gegeben, wenn die Zaehler den Wert 0 erreicht haben. Bei einem Link zu einer existierenden Datei, wird ein Directory-Eintrag mit dem neuen Namen angelegt, die Inode-Nummer vom Eintrag der Originaldatei kopiert und der Link-Zaehler in dem Inode erhoeht. Das Loeschen einer Datei erfolgt durch Decrementieren des Link-Zaehlers in dem Inode und Loeschen des Directory-Eintrags. Wird der letzte Verweis auf das Inode geloescht, wird die Datei geloescht und das Inode frei gegeben. Ist die Datei beim Loeschen eroeffnet, wird der Link-Zaehler erst nach dem Schliessen der Datei auf Null gesetzt, da neben dem Link-Zaehler auch noch der Zaehler fuer die Anzahl der gleichzeitigen Eroeffnungen der Datei existiert.

7.3.3.4. Mount-Verbindung

Ein Dateisystem ist eine Verbindung von mehreren Block-E/A-Geraeten, die eine Directory-Hierarchie enthalten. Ein Dateisystem enthaelt immer das Wurzel-Directory (root). Es wird als Dateisystem "root" bezeichnet. Ein anderes Dateisystem kann an jeden Knoten der bestehenden Hierarchie angebunden werden und damit die logische Hierarchie erweitern. Dieser Vorgang wird als "mount" bezeichnet.

Das Anlagern bzw. Verbinden der Dateisysteme erfolgt ueber den Systemruf mount(2). Dieser benoetigt 3 Parameter, den Namen der Geraetedatei, den Namen des Directorys und ein Flag. Dieses zeigt einen eventuellen Nur-Lese-Status fuer das zu verbindende Dateisystem an. Nach der Verbindung zeigt ein Pfadname, der den Namen des Directorys enthaelt, auf das verbundene E/A-Geraet. Fruehere Eintraege in dem Directory sind unsichtbar.

Der Speicherbereich jedes verbundenen E/A-Geraetes wird getrennt verwaltet, um ein leichtes Loesen der mount-Verbindung zu ermoeglichen.

7.3.4. Shell

Die Shell oder der Kommandointerpreter ist ein Programm, das den Benutzer nach dem Login interaktiv mit dem System verbindet. Die meiste Zeit wartet die Shell auf Kommandoeingaben des Benutzers. Nach dem Abschluss einer eingegebenen Zeile mit Return wird der Systemruf read(2) der Shell beendet, und sie analysiert die Kommandozeile. Die Argumente werden fuer den Systemruf exec(2) aufbereitet, anschliessend der Systemruf fork(2) ausgefuehrt. Der Sohnprozess (mit dem gleichen Kode) fuehrt den Systemruf exec(2) aus. Wenn er erfolgreich ist, wird das Programm mit dem eingegebenen Namen geladen und gestartet. Der Vaterprozess wartet bis der Sohnprozesses endet. Die Shell weiss nun, dass das Kommando beendet ist, gibt ihr Prompt-Zeichen aus und fuehrt einen neuen Systemruf read(2) aus, um ein neues Kommando einzulesen.

Mit diesem Algorithmus ist es leicht moeglich, Hintergrundprozesse zu implementieren. Endet eine Kommandozeile mit dem Zeichen "&", wartet die Shell lediglich nicht auf den Prozess, der das Kommando ausfuehrt.

Der Mechanismus passt gut zur Idee der Standardeingabe und -ausgabe. Wird ein Prozess mit fork(2) erzeugt, erbt er nicht nur das Speicherabbild, sondern auch alle eroeffneten Dateien des Vaterprozesses einschliesslich der Dateien mit den Dateideskriptoren 0, 1 und 2. Die Shell benutzt natuerlich diese Dateien zum Einlesen des Kommandos, zur Ausgabe ihrer Mitteilungen und fuer Diagnosen. Der Sohnprozess (das Kommando) erbt diese gewoehnlich. Ist aber das Argument "<" oder ">" in der Kommandozeile angegeben, legt der Sohnprozess kurz vor dem Systemruf exec(2) den Deskriptor (0 oder 1) fuer die Standardein/-ausgabe auf die angegebene Datei. Das ist einfach moeglich, da durch eine Festlegung der kleinste Dateideskriptor zugewiesen wird, wenn ein open(2)- oder creat(2)-Systemruf erfolgt. Die Datei mit dem Deskriptor 0 oder 1 muss nur geschlossen und die neue Datei eroeffnet werden. Endet der Prozess (das Programm), wird die Verbindung zwischen Datei und Deskriptor 0 oder 1 automatisch geloest. Die Shell kennt deshalb niemals die Dateinamen ihrer aktuellen Standardeingabe und -ausgabe, sie muss diese niemals neu eroeffnen.

Unter normalen Umstaenden wird die Hauptschleife der Shell (des Vaterprozesses) niemals beendet. Eine "end of file"-Bedingung in der Eingabedatei beendet jedoch die Shell. Es wird z.B. die Shell wie ein normales Kommando mit einer gegebenen Eingabedatei ausgefuehrt:

```
sh < comfile
```

Die Kommandos in "comfile" werden so lange ausgefuehrt, bis die "end of comfile"-Bedingung erreicht ist. Die Shell endet. Da sie jedoch der Sohnprozess einer anderen Shell ist, endet das Warten des Vaterprozesses, und ein neues Kommando kann ausgefuehrt werden.

In den meisten Systemen wird die C-Shell (siehe csh(1)) benutzt. Die C-Shell ist zwar groesser, bietet aber mehr Moeglichkeiten und ist einfacher zu benutzen als die Standard-Shell sh(1).

7.3.5. Ersetzen der Shell

Die Shell ermoeglicht, wie oben beschrieben, den vollen Zugriff der Benutzer zu den Faehigkeiten des Systems, da jedes Programm in einem geeigneten Schutzmodus ausgefuehrt werden kann. Manchmal wird ein anderes Interface zum System gewuenscht. Die Realisierung ist im System leicht moeglich.

Gewoehnlich ruft INIT(M) nach einem erfolgreichen Login des Benutzers die Shell auf. Der Programmname kann jedoch im Benutzereintrag der Datei /etc/passwd geaendert werden. Das neue Programm kann vollkommen freizuegig den Dialog mit dem Benutzer realisieren. Fuer ein Textverarbeitungssystem kann in der /etc/passwd z.B. der Name des Editors angegeben sein. Der Benutzer dieses Systems befindet sich sofort im Editor und kann mit der Arbeit beginnen. Damit koennen diese Benutzer auch am Aufrufen von Programmen gehindert werden, die nicht fuer sie bestimmt sind.

8. Schreiben von WEGA-Geraetetreibern

8.1. Einleitung

In diesem Kapitel wird dargestellt, wie Geraetetreiber-Software fuer das Betriebssystem WEGA zu schreiben ist. Es werden damit alle Systemprogrammierer angesprochen, deren Aufgabe darin besteht, neue Geraetetreiber in WEGA einzubinden.

Die Ablaeufe fuer die Integration neuer Geraetetreiber wurden in den Mechanismus der Systemgenerierung eingearbeitet. Bis zu 6, vom Benutzer geschriebene, Geraetetreiber koennen in den WEGA-Kern eingebunden werden. Die einzige Bedingung besteht darin, bestimmte Namen fuer die Prozeduren und bestimmte Namen fuer Interrupt-Vektoren zu benutzen. Diese Interface werden weiter unten detailliert behandelt.

In den folgenden Abschnitten werden "typische" Geraetetreiber beschrieben. In Wirklichkeit ist jedes E/A-Geraet und deshalb jeder Geraetetreiber unterschiedlich. Ein bestimmtes anzuschliessendes E/A-Geraet kann ein vom "typischen" Fall voellig unterschiedliches Interface erfordern. Deshalb sollten die folgenden Abschnitte als Hinweis und nicht als Absolutdarstellung betrachtet werden.

Beachten Sie bitte !

Fehler in einem Geraetetreiber koennen zu katastrophalen Systemproblemen fuehren. Deshalb sollte die Entwicklung der Geraetetreiber mit grosser Sorgfalt erfolgen. Solange ein Treiber nicht umfassend getestet wurde, sollten keine Benutzer mit dem System arbeiten. Wenn das nicht moeglich ist, sind zumindest die Datenbestaende haeufig zu sichern.

Die Effektivitaet der Implementierung eines Geraetetreibers hat direkten Einfluss auf die Gesamteffektivitaet des E/A-Geraetes. Die Treiber muessen so "knapp" wie moeglich geschrieben werden. Dabei sollte voller Gebrauch von Registervariablen und anderen kompakten Programmiermethoden gemacht werden.

8.2. Geraetetreiber

Ein Geraetetreiber stellt ein Softwaremodul dar, das die logischen E/A-Anforderungen des WEGA-Betriebssystems in physische Befehle fuer ein E/A-Geraet uebertraegt. Typische Geraetetreiber sind von relativ geringer Groesse (3 - 10 Seiten in C-Quellkode). Ein erfahrener Systemprogrammierer benoetigt etwa einen Monat, um einen Geraetetreiber fuer ein ziemlich einfaches Geraet zu schreiben, dessen Schnittstelle gut definiert ist. Wenn es sich um eine sehr komplexe oder noch nicht komplett definierte Schnittstelle handelt, kann das Schreiben und Austesten des Geraetetreibers betraechtlich laenger dauern.

Ein Geraetetreiber besteht aus einer Anzahl von Prozeduren, die von verschiedenen Teilen des Betriebssystems aufgerufen werden. Die Schnittstellen koennen in zwei Klassen eingeteilt werden. Die erste stellt die Interrupt-Routinen dar. Diese Routinen werden asynchron als Antwort auf den Interrupt des Hardware-Geraetes angesprochen. Es ist typisch, dass die Interrupt-Routine den Status des Geraetes ueberprueft und wenn alles in Ordnung ist, Prozesse weckt, die auf den Interrupt gewartet haben. Es wird eine weitere interne Prozedur des Geraetetreibers aufgerufen, um die naechste Uebertragung zu starten. Die zweite Klasse von Prozeduren wird synchron aufgerufen, um E/A-Uebertragungen aus und an das E/A-Geraet zu veranlassen oder seinen Status zu aendern.

Es gibt 3 Typen von Geraeteschnittstellen:

1. Blockgeraete, bei denen die Ein-/Ausgabe in Bloecken mit fixierter Groesse vollzogen wird. Die Uebertragung erfolgt in/aus einem internen Pufferspeicher.

2. Character-Geraete, bei denen die Ein-/Ausgabe in Einheiten variabler Groesse direkt von/zu Adressen im User-Prozess vollzogen wird.

3. Terminal-Geraete, bei denen die Ein-/Ausgabe durch einen ziemlich komplizierten Mechanismus durchgefuehrt wird, der verschiedene Funktionen (z.B. erase/kill, canonicalization), eine Pufferung der Zeichen (type ahead) und die timeout-Funktion (fuer physische Verzoegerungen) realisiert. Terminal-Geraete werden als ein Spezialfall von Character-Geraeten behandelt.

Die synchron angesprochenen Prozeduren, die den E/A-Beginn veranlassen, werden ueber zwei Sprungtabellen, eine fuer Blockgeraete und eine fuer Character-Geraete aufgerufen. Die Aufrufe haben die Form:

```
(*bdevsw[major(dev)].d_strategy)(bp);
```

oder

```
(*cdevsw[major(dev)].d_read)(dev);
```

wobei dev das interne Geraetekennzeichen ist. Es besteht aus 2 Byte, den Geraetenummern major/minor. Die beiden Makros major(dev) und minor(dev) geben die Major- und Minor-Geraetenummer zurueck. Die Major-Geraetenummer ist ein Index in die Sprungtabelle fuer Block- oder Character-Geraete. Die Minor-Nummer ist ein Kennzeichen fuer eine besondere Funktion oder ein Subgeraet. Einige Geraetetreiber benutzen die Minor-Nummer zur Unterscheidung von verschiedenen Moeglichkeiten des Geraetetreibers, z.B. zur Realisierung von verschiedenen Diskettenformaten.

8.3. Anpassung und Installation

Um einen Geraetetreiber zu erstellen, entwickelt der Systemprogrammierer ein Objektmodul mit den Schnittstellen fuer Prozeduraufrufe, die weiter unten beschrieben werden, und beauftragt das Systemgenerierungsprogramm diesen Treiber anstatt den normalerweise vorhandenen gleichnamigen Dummy-Routinen aufzunehmen. Im allgemeinen sollten alle Variablen und Prozedurnamen in Geraetetreibern den Treibernamen als eine Vorsilbe enthalten (z.B. benutzt man "usr1current" anstelle von "current" als Namen einer Variablen im usr1-Geraetetreiber). Dadurch werden Kollisionen mit Namen verhindert, die an anderen Stellen im Kern benutzt werden.

Das Systemgenerierungsprogramm geht davon aus, dass es im Arbeitsdirectory /usr/sys/conf ablaeuft. Der Zugriff auf die verschiedenen Dateien, die fuer die Systemgenerierung benoetigt werden, ist in der Regel nur durch den Superuser moeglich.

Es folgt ein Beispiel fuer die Systemgenerierung:

```
# scc -O -c mydriver.c
# sysgen
```

(siehe auch Abschnitt 5.)

Nach einer erfolgreichen Systemgenerierung sollte der neue WEGA-Kern ueber die Boot-Programme gestartet und sorgfaeltig getestet werden.

Der naechste Schritt besteht unter Nutzung von mknod(M) in der Erstellung des WEGA-Pfadnamens fuer das neue E/A-Geraet. Das Erstellen dieser Namen mit mknod(M) hat keine Auswirkungen auf den Geraetetreiber, solange nicht E/A-Aufrufe, die den neuen Namen als Argument enthalten, erfolgen. Das Programm mknod gestattet dem Benutzer die Angabe, ob sich der gegebene Name auf ein Block- oder Character-Geraet bezieht und erlaubt die Angabe der Major- und Minor-Geraetenummern. Die Auswahl des Pfadnamens erfolgt im Prinzip voellig willkuerlich. Nach Vereinbarung sind jedoch alle E/A-Geraete im Directory /dev enthalten.

Sechs Interrupt-Vektoren sind fuer vom Anwender geschriebene Geraetetreiber reserviert. Es gibt Dummy-Routinen fuer sechs Treiber. Die folgende Tabelle fasst die Schnittstellen zusammen:

Interrupt Vector	Char dev Major #	Block dev Major #	Interrupt Routine	cdevsw entries	bdevsw entries
0xF0	10	10	usr1int	u1xxx	usr1xxx
0xF2	11	11	usr2int	u2xxx	usr2xxx
0xF4	12	12	usr3int	u3xxx	usr3xxx
0xF6	13	13	usr4int	u4xxx	usr4xxx
0xF8	14	14	usr5int	u5xxx	usr5xxx
0xFA	15	15	usr6int	u6xxx	usr6xxx

Die Spalten cdevsw und bdevsw zeigen die Anfangsnamen der Routinen fuer die Block- bzw. Character-Geraete, z.B. lautet die Leseroutine fuer das zweite Anwendergeraet usr2read.

Es ist zu beachten, dass nur sechs Interrupt-Vektoren fuer vom Anwender geschriebene Treiber zur Verfuegung stehen. Es wird davon ausgegangen, dass der Anwender nur ein oder zwei Anwendergeraete und zugehoerige Treiber hat.

8.4. Betrachtungen zur Hardware

Der grundlegende Hardware-Entwurf wird durch zwei Schnittstellen bestimmt, die von der Hardware unterstuetzt werden muessen. Die eine Schnittstelle ist durch das P8000, die andere durch das E/A-Geraet vorgegeben. Der uebrige Hardware-Entwurf ist relativ frei von Beschraenkungen. Der Prozessor des P8000 (CPU U8001) spricht die Hardware ueber E/A-Befehle an. Gegenwaertig werden nur die "normalen" (im Gegensatz zu den "speziellen") E/A-Befehle in Geraetetreibern benutzt. Die Hardware muss vollstaendig die vom Prozessor gelieferte 16-Bit-E/A-Adresse dekodieren. E/A-Geraete werden von WEGA grundsaetzlich wie DMA-Geraete behandelt. Die DMA-Funktion muss vom Geraetetreiber simuliert werden, wenn ein DMA-Betrieb vom E/A-Geraet nicht unterstuetzt wird.

Die Ein-/Ausgabe kann an jede (physische) Speicheradresse uebertragen werden. Segmentgrenzen koennen ueberschritten werden. Im allgemeinen liegt die angeforderte E/A-Uebertragung zwischen 1 Byte und 64 KByte. Durch viele E/A-Geraete werden zusaetzliche Einschraenkungen fuer die minimalen und maximalen Uebertragungslaengen auferlegt. Geraete, die mit U880-DMA-Schaltkreisen aufgebaut sind, muessen mit besonderer Sorgfalt das "wrap around" der Adresse behandeln, da das P8000 eine 24-Bit-Adresse hat. Der Geraetetreiber kann einen umfangreichen E/A-Auftrag in mehrere kleinere physische Uebertragungen aufteilen. Im Interesse der Gesamtleistung des Systems, sollten Uebertragungen nicht in zu kleine Teile aufgeteilt werden.

Wenn realisierbar, sollte die Peripheriekarte einen Speicher (FIFO oder normaler RAM) zur Zwischenspeicherung der Daten enthalten. Es kann nicht vorausgesetzt werden, dass die Daten von WEGA immer schnell genug abgenommen werden.

Besitzt die Hardware einen Dual-Port-RAM, der auch vom P8000-Prozessor angesprochen werden kann, muss die Anfangsadresse des Speichers ueber 4 MByte liegen. Damit wird verhindert, dass WEGA den Speicher des Geraetes User-Prozessen zuweist. WEGA ermittelt den vorhandenen Systemspeicher selbst, wobei am ersten "Loch" im RAM angehalten wird.

Periphere Steuerungen auf der Basis von Mikroprozessoren bringen einige zusaetzliche Entwurfsprobleme mit sich. Zuerst gilt der Schnittstelle zwischen der Firmware des Geraete-Controlers und dem Treiber besondere Aufmerksamkeit. Die Hauptaufgabe besteht darin, sogenannte Softwarewettlaeufe zu verhindern. Wenn sich z.B. das "Geraete-Busy"-Bit unter der Kontrolle der Firmware befindet, kann es noetig sein, dass der WEGA-Geraetetreiber eine kurze Verzoegerung in den Kommandofolgen an das E/A-Geraet realisiert. Erfolgt dies nicht, kann der WEGA-Treiber das Busy-Bit untersuchen und es frei finden, bevor die Firmware in der Lage war, es zu setzen. Diese Situation fuehrt zu einer katastrophalen Desynchronisation zwischen dem E/A-Geraet und dem Betriebssystem. Es sollte deshalb auch ein Reset in der Geraete-Firmware realisiert werden.

Von WEGA werden keine Einschraenkungen bezueglich der Geraete-Firmware gemacht. Die meisten Einschraenkungen resultieren entweder aus Hardware- oder Zeitzwaengen, die von den peripheren Geraeten ausgehen. Gleichermassen ist es sehr kompliziert, eine Reaktionszeit von WEGA zu garantieren. Im Betriebssystem gibt es kritische Abschnitte, die durch das Abschalten von Interrupts implementiert werden. Ein E/A-Geraet kann eine Millisekunde oder laenger warten muessen, bevor seine Interrupt-Anfrage bearbeitet wird. Fuer E/A-Geraete mit hohen Datenraten, z.B. Disk-Laufwerke und Datennetze mit hoher Geschwindigkeit, sollten Firmware und Hardware so ausgelegt sein, dass alle Daten zwischengespeichert werden.

Die dem Geraetetreiber zugaengliche Schnittstelle ist von relativ niedrigem Niveau. Die Grundfunktionen des Treibers realisieren den Datentransfer von/nach einem physischen Speicherplatz. Gewoehnlich "wissen" Geraetetreiber nichts ueber den Aufbau der hoeheren Ebene des Betriebssystems, z.B. darueber, welcher Prozess eine E/A-Uebertragung angefordert hat. Daraus folgt fuer die Geraete-Firmware, dass sie nicht von Informationen der hoeheren Ebene abhaengig sein darf.

8.5. Interface der Geraetetreiber

In diesem Abschnitt werden genauere Einzelheiten zu den internen Schnittstellen zwischen dem Geraetetreiber und dem uebrigen Teil von WEGA behandelt. Diese Schnittstelle ist ziemlich "sauber". Deshalb ist es im Vergleich zu vielen anderen Betriebssystemen leichter, Geraetetreiber fuer WEGA zu schreiben.

8.5.1. Interrupt-Schnittstelle

In einem Geraetetreiber wird die Interrupt-Routine aufgerufen, wenn das E/A-Geraet einen Interrupt verursacht. Die Interrupt-Routine hat die Aufgabe, den Hardware-Status zu untersuchen, den erfolgreichen Abschluss einer Anforderung zu pruefen und die naechste Uebertragung zu starten. Wenn Fehler auftraten, versucht der Treiber diese zu beheben und die Uebertragung neu zu beginnen. Traten keine Fehler auf, so entfernt die Interrupt-Routine den Auftrag aus der Warteliste des Geraetetreibers und startet den naechsten Auftrag.

Die Interrupt-Routine wird mit einem Argument aufgerufen, das einen Zeiger auf die Zustandsstruktur darstellt:

```
usr1int(s)
register struct state *s;
```

Die Zustandsstruktur ist der Maschinenzustand, der in der Interrupt-Routinen sichergestellt wurde.

Beachten Sie bitte !

Der Geraetetreiber darf keine Daten in dieser Struktur aendern.

Die einzige Variable in der Zustandsstruktur, die fuer die Interrupt-Routine gewoehnlich von Interesse ist, ist s_eventid. Diese Variable stellt eine Kopie des Interrupt-Vektors dar, der vom E/A-Geraet geliefert wurde. Das untere Byte enthaelt den eigentlichen Interrupt-Vektor (Offset). Es wird nur benoetigt, wenn die gleiche Interrupt-Routine fuer mehrere Geraete genutzt wird. Mit diesem Byte wird herausgefunden, welches E/A-Geraet tatsaechlich den Interrupt angefordert hat. Das hoehere Byte haengt vom E/A-Geraet ab, z.B. kann der Status des Geraetes enthalten sein, um eine zusaetzliche Fehlerabfrage zu vermeiden.

Die Interrupt-Routine ist fuer den Start der naechsten Uebertragung verantwortlich, falls weitere Aufgaben zu erledigen sind. Die Wirkungsweise ist fuer Character- und Blockgeraete unterschiedlich.

8.5.1.1. Interrupt-Routinen fuer Blockgeraete

Bei einem Blockgeraet wird die E/A-Uebertragungsanforderung durch die Strategieroutine asynchron in eine doppelt verknuepfte Anforderungsliste (Warteschlange) eingereiht. Wenn die Uebertragung erfolgreich war, muss die Interrupt-Routine den Auftrag aus der Warteschlange entfernen und den naechsten Auftrag der Warteschlange beginnen. Der Treiber enthaelt gewoehnlich eine Subroutine fuer das Starten des E/A-Geraetes, damit eine Uebertragung an ein bereites E/A-Geraet durch die Strategieroutine selbst eroeffnet werden kann. Diese Subroutine wird sowohl von der Interrupt-Routine als auch von der Strategieroutine aufgerufen.

Viele E/A-Geraete sind hinsichtlich der maximalen Uebertragungslaenge begrenzt. Der Geraetetreiber muss diese Grenzen beachten, indem er eine umfangreiche Uebertragungsanforderung in mehrere kleinere Auftraege zerlegt. Der Geraetetreiber unterhaelt eine interne Variable, die auf die Uebertragungsanforderung hinweist, die gerade abgearbeitet wird. Wenn es sich um eine umfangreiche Anforderung gehandelt hat, werden die Parameter der laufenden Anforderung modifiziert und die Anforderung wird ueber die Startroutine fortgesetzt. Eine umfangreiche Anforderung wird so verarbeitet, als ob es sich um mehrere Uebertragungen mit der vom E/A-Geraet maximal erlaubten Groesse handelt.

Wenn das E/A-Geraet gemeldet hat, dass die Uebertragung nicht erfolgreich war, ist die Interrupt-Routine fuer die Wiederholung der Uebertragung verantwortlich. Die Interrupt-Routine sollte einen Zaehler fuer Wiederholungen enthalten, damit endlose Schleifen bei einem defekten E/A-Geraet vermieden werden. Die Wiederholungen sollten auf eine geringe Anzahl begrenzt werden. Kann der Fehler nicht beseitigt werden, wird die gesamte Uebertragung als fehlerhaft gekennzeichnet, indem B_ERROR in b_flags fuer den angeforderten Zwischenspeicher gesetzt wird:

```
bp->b_flags |= B_ERROR;
```

Die Interrupt-Routine sollte durch Aufruf von

```
deverror(bp, error1, error2);
```

anzeigen, dass Fehler auftraten. Der Pointer bp zeigt auf den Zwischenspeicher, mit dem gearbeitet wurde, error1 und error2 sind willkuerliche ganzzahlige Variable, deren Inhalt in der Fehlermitteilung ausgegeben wird. Die Routine deverror gibt die Meldung am Systemterminal aus:

```
err on dev MAJ/MIN
bn=Z er=0xXXXX, 0xYYYY
```

wobei MAJ und MIN die Major- und Minor-Nummern des E/A-Geraetes, Z die Blocknummer (bp->b_blkno) und XXXX und YYYY die Hexadezimaldarstellungen von error1 und error2 sind. Wenn die Schwierigkeiten durch die Wiederholungen nicht beseitigt werden koennen, kann die Interrupt-Routine ueber die Kernfunktion printf eine weitere Fehlermeldung auf dem Systemterminal auszugeben. In schwerwiegenden Faellen, bei denen die gesamte Funktion des Systems nicht mehr moeglich ist, kann die Interrupt-Routine die Kernfunktion panic aufrufen, um das gesamte System anzuhalten. Der Aufruf von panic sollte nicht leichtfertig geschehen, da moeglicherweise inkonsistente Dateisysteme entstehen koennen.

Die Routinen der hoeheren Ebene muessen ueber den Abschluss der Uebertragungen, egal ob ein Fehler auftrat oder nicht, benachrichtigt werden. Die Anforderung muss aus der Warteschlange entfernt werden. Man hebt die Verknuepfung mit einem Zwischenspeicher, auf den bp zeigt, wie folgt auf:

```
(bp->av_forw)->av_back = bp->av_back; /* Rueckverbindung */
(bp->av_back)->av_forw = bp->av_forw; /* Vorwaertsverbindung */
```

Eine solch doppelt verknuepfte Liste ist nicht erforderlich. Der Entwickler kann, wenn gewuenscht, eine einfachere Struktur benutzen. Von Bedeutung ist, dass eine Symmetrie zwischen der Strategieroutine, die die Anforderung in eine Warteschlange einreiht und der Interrupt-Routine, die sie aus der Warteschlange wieder entfernt, besteht. Nachdem der Auftrag aus der Warteschlange entfernt ist, wird die Routine iodone(bp) aufgerufen, um die Uebertragung als beendet zu kennzeichnen:

```
iodone(bp);
```

Die Interrupt-Routine ruft schliesslich die Geraetestartroutine auf, um den naechsten Auftrag zu starten.

Welcher Zwischenspeicher fuer die naechste Uebertragung ausgewaehlt wird, ist Aufgabe des Geraetetreibers. WEGA stellt keine Bedingungen fuer die Reihenfolge der Abarbeitung. Die Disktreiber nutzen z.B. einen "modifizierten SCAN"-Algorithmus, um auf die Diskbloecke nach aufsteigendem (oder fallendem) Zylinderindex zuzugreifen. Es kann eine Zusammenarbeit zwischen der Strategieroutine und der Interrupt-Routine geben, um eine optimal geordnete Warteschlange aus den Anforderungen zu erstellen. Fuer die meisten vom Anwender geschriebenen Geraetetreiber ist moeglicherweise eine Abarbeitungsreihenfolge auf der Basis "First-come, First-served" (FCFS) angebracht.

8.5.1.2. Interrupt-Routinen fuer Character-Geraete

Diese Interrupt-Routinen sind in der Funktion aehnlich denen der Blockgeraete. Sie sind etwas einfacher, da keine umgruppierbaren Anforderungswarteschlangen existieren. Die Interrupt-Routine des Character-Geraetes beschaeftigt sich nur mit der Verarbeitung einer Anforderung. Die meisten Character-Geraete koennen fuer jedes uebertragene Zeichen einen Interrupt generieren. Somit besteht die Primaerfunktion der Interrupt-Routine darin, ein weiteres Zeichen vom/zum E/A-Geraet zu uebertragen.

Die Aufgabe der einfachsten Interrupt-Routine besteht darin, das naechste Zeichen vom User-Prozess zu lesen, das E/A-Geraet wieder zu aktivieren und das Zeichen auszugeben bzw. in analogen Aktionen fuer die Eingabe. Von dieser Realisierung eines Geraetetreiber wird abgeraten, da es erforderlich ist, dass der User-Prozess bis zum Abschluss der gesamten E/A-Uebertragung im Speicher verbleibt. Besser ist es, einen Geraetezwischenspeicher zwischen dem User-Prozess und der Geraete-Interrupt-Routine anzuordnen. Dies erlaubt die Rueckkehr zum User-Prozess, sobald das letzte Zeichen in den Geraetezwischenspeicher geschrieben wurde, anstatt warten zu muessen, bis das letzte Zeichen tatsaechlich zum/vom E/A-Geraet uebertragen wurde.

8.5.1.3. Interrupt-Routinen fuer Terminal-Geraete

In diesen Treibern stehen die eigentlichen Geraetetreiber in einer komplizierten Beziehung zu einem allgemeinen logischen Terminal-Treiber tty.c. Die Routinen in tty.c fuehren Funktionen wie Loeschen (erase/kill) und "type ahead" der Eingabe aus. Die Routinen in tty.c benutzen Variablen in der Struktur struct tty. Das Interrupt-Programm fuer solche E/A-Geraete muss diese Variablen fuer den aktuellen Kanal bearbeiten und die zugeordneten Routinen wecken.

Fuer Terminal-Geraete gibt es 3 Interrupt-Typen:

1. Empfaenger-Interrupts, die anzeigen, dass ein Zeichen verfuegbar ist.

2. Sender-Interrupts, die anzeigen, dass ein weiteres Zeichen fuer die Uebertragung angelegt werden kann.

3. Interrupts bei Statusaenderungen, die anzeigen, dass die Modem-Signale den Zustand veraendert haben.

Das Interrupt-Programm des Empfaengers reaktiviert das E/A-Geraet und ruft eine Routine in tty.c auf, um die weitere Behandlung des Zeichens zu veranlassen. Diese Routine benutzt die Sprungtabelle linesw (line switch table), um zu verschiedenen Programmen zu verzweigen. Die Aufgabe von linesw besteht darin, die Moeglichkeit von Mehrfachprotokollen zuzulassen. Fuer jedes Terminal-Geraet enthalten die tty-Strukturen eine Variable t_line, die einen Index in diese Tabelle darstellt. Leitungsprotokolle sind zum Beispiel:

linesw Index	Protokoll
0	asynchrone Uebertragung
1	NET (Netz-Protokoll)
2	Bisync

Beim P8000 wird gegenwaertig nur das erste Protokoll unterstuetzt.

Die Empfangs-Interrupt-Routine realisiert also einen zeigergesteuertem Aufruf der entsprechende Protokollroutine.

Die Sende-Interrupt-Routine ist genauso einfach. Hier wird ebenfalls die Tabelle linesw benutzt, um die Geraetestartroutine ueber einen Zeiger auszuwaehlen. Das Interrupt-Programm reaktiviert das E/A-Geraet und kehrt aus dem Interrupt-Programm zurueck. Die Startroutine, die ueber linesw aufgerufen wird, ruft schliesslich die Startroutine des E/A-Geraetes auf, um die Uebertragung zu beginnen.

Die Geraete-Open-Routine traegt in tp->t_proc der tty-Struktur des Kanals die Adresse der Geraetestartroutine ein:

```
     tp->t_proc = usr1xstart;
```

in der Sende-Interrupt-Routine:

```
     (*linesw[tp->t_line].l_start)(tp);
```

ist fuer Leitungsprotokoll 0 aequivalent zu:

```
     ttstart(tp);
```

in ttstart:

```
     (*tp->t_proc)(tp);
```

ist aequivalent zu:

```
     usr1xstart(tp);
```

Durch dieses Verhaeltnis zwischen dem Geraetetreiber und den Protokollroutinen wird eine beachtliche Flexibilitaet ermoeglicht. Der gleiche Geraetetreiber kann mehrere unabhaengige Protokolle bedienen. Gleichermassen kann eine gegebene Protokollroutine Zeichenstroeme von mehreren unterschiedlichen E/A-Geraeten verarbeiten. Der zusaetzliche Aufwand ist relativ gering.

Die letzte Interrupt-Routine fuer tty-Geraetetreiber bearbeitet Interrupts, die durch Aenderung der Modem-Signale hervorgerufen werden. (Beim U880-SIO-Schaltkreis gibt es noch einen Interrupt "spezielle Empfangsbedingungen", d.h., fuer Framing- und Paritaetsfehler. Diese Interrupts werden wie regulaere Empfangsinterrupts behandelt, in denen ein Fehler aufgetreten ist.) Interrupts bei Aenderung der Modem-Signale realisieren zwei Funktionen. Sie bestehen im Wecken von Prozessen, die auf die Aenderung des Zustands der Signalleitungen gewartet haben und dem Aussenden eines Signals an andere Prozesse bei Auftreten eines unerwarteten Ereignisses. Aus beiden Gruenden ueberwacht der gegenwaertige Treiber fuer SIO-Schaltkreise die Data-Set-Ready-Leitung. Die Open-Routine einer Terminalleitung schlaeft auf der Adresse

```
&tp->t_rawq
```

und warten auf Data-Set-Ready (wenn es nicht schon beim Aufruf der Open-Routine aktiv war). Wenn das Data-Set-Ready-Signal den Zustand veraendert und einen Interrupt verursacht, ruft die Interrupt-Routine die Funktion wakeup auf. War Data-Set-Ready jedoch aktiv und das E/A-Geraet eroeffnet, so zeigt der Interrupt einen Traegerausfall (loss of carrier) an. Das Interrupt-Programm ruft

```
signal(tp->t_pgrp, SIGHUP);
```

auf, um zur entsprechenden Prozessfamilie ein Signal zu senden. Natuerlich koennen vom Anwender geschriebene Geraetetreiber andere Signale ueberwachen und andere Aktivitaeten veranlassen.

8.5.2. Schnittstellen fuer E/A-Anforderungen

8.5.2.1. Blockgeraeteschnittstelle

Die Interface-Routinen fuer das Anwenderblockgeraet 1 lauten:

usr1open — wird aufgerufen, wenn das E/A-Geraet selbst eroeffnet wird. Ein Teil des Programmkodes wird typisch so ausgelegt, dass er nur einmal abgearbeitet wird, um das E/A-Geraet fuer nachfolgende Anforderungen vorzubereiten.

usr1close — wird aufgerufen, wenn das letzte aktive Open des E/A-Geraetes geschlossen wird. Die Routine kann wiederum einen "Nur-einmal-Kode" fuer die Beendigung der Arbeit mit dem E/A-Geraet enthalten.

usr1strategy — wird aufgerufen, um eine E/A-Anforderung fuer das E/A-Geraet in eine Warteschlange einzureihen. Diese Routine bewirkt, dass das E/A-Geraet die entsprechende Uebertragung ausloest. Die Anforderung selbst zeigt an, ob es sich um Lesen oder Schreiben handelt. Die Strategieroutine behandelt beide Arten.

usr1tab — die Adresse einer Pufferstruktur (/usr/include/sys/buf.h). Das ist ein Header einer Zwischenspeicherkette fuer eine Liste von Zwischenspeichern, die durch die Strategieroutine in eine Warteschlange eingereiht wird.

usr1int — Interrupt-Routine

Die meisten Blockgeraete verfuegen ebenfalls ueber eine Character-Geraeteschnittstelle. Diese wird gewoehnlich als "raw"-Geraet bezeichnet, z.B. wuerden /dev/md4 und /dev/rmd4 jeweils das Block- bzw. das raw-Geraet bezeichnen. Die Schnittstelle des raw-Geraetes gestattet Uebertragungen direkt an den User-Prozess (d.h., im Kern erfolgt keine Zwischenspeicherung). Die raw-Geraeteschnittstelle unterstuetzt den ioctl-Aufruf. Ein gegebenes E/A-Geraet kann die Transfergroesse auf verschiedene Weise begrenzen. So gestatten z.B. die meisten raw-Disk-Geraete nur Uebertragungen von ganzen Bloecken. Der Zugriff auf das raw-Geraet ist oft schneller, als wenn man durch den gesamten Zwischenspeichermechanismus des Kerns geht. So werden fuer die Konsistenzpruefungen des Dateisystems beim Systemstart die raw-Geraete benutzt.

Bei den raw-Schnittstellen bestehen die Lese- und Schreibroutinen der Character-Geraeteschnittstelle aus dem Aufruf:

```
physio(usr1strategy, &rusr1buf, dev, direction);
```

wobei

rusr1buf einen Puffer-Header, der in einem Geraetetreiber lokal vorhanden ist,

dev die Geraetenummer, die als Argument weitergegeben wird und

direction die Transferrichtung (die Leserichtung lautet B_READ und die Schreibrichtung B_WRITE, zwei Konstanten, die in buf.h definiert sind),

darstellen. Der Aufruf an physio resultiert schliesslich in einem Aufruf an die Strategieroutine fuer das E/A-Geraet.

Die gesamte Character-Schnittstelle muss definiert sein, da der Dummy-Modul alle Routinen beinhaltet. Der Dummy-Modul muss durch den vom Anwender geschriebenen Treiber vollstaendig ersetzt werden. Die Open- und Close-Routinen fuer das Character-Geraet sollten einfach die Blockgeraeteroutinen bzw. umgekehrt sein.

Fuer Blockgeraete erfolgen alle Anforderungen ueber den Aufruf der Geraetestrategieroutine. (Der Name kommt daher, dass fuer Disklaufwerke die Strategieroutine festlegt, wohin der Schreib-/Lesekopf bewegt wird.) Die Geraetestrategieroutine wird mit einem Argument, dem Zeiger auf eine Puffer-Header-Struktur, aufgerufen. Die Grundfunktion der Strategieroutine besteht darin, die Fehlerfreiheit der Anforderung zu pruefen, sie in die Warteschlange einer doppelt verknuepften Anforderungsliste einzureihen und die Geraetestartroutine aufzurufen (wie in der Interrupt-Routine bereits beschrieben wurde).

Die Anforderung wird z.T. durch das E/A-Geraet bestimmt. Die folgende (unvollstaendige) Liste enthaelt einen Satz von ueblichen Forderungen:

1. Die Startadresse des Zwischenspeichers und die Byte-Anzahl muessen geradzahlig sein.

2. Die angeforderte Blocknummer muss im Bereich der fuer das E/A-Geraet gueltigen Blocknummern liegen. Dieser Bereich wird von internen Tabellen des Treibers bestimmt.

3. Die Minor-Geraetenummer muss gueltig sein.

4. Es kann erforderlich sein, dass die zu uebertragenen Bytes ein ganzzahliges Vielfaches der Geraeteblockgroesse sind.

Vor der Einordnung der Anforderung in die Warteschlange wird der Rest (bp->b_resid) auf die Anzahl der zu uebertragenden Bytes gesetzt. Wie oben beschrieben, wird der Rest in Auftraege, deren Groesse vom E/A-Geraet abhaengig ist, aufgeteilt, wenn er groesser als der maximale Uebertragungsumfang ist. Fuer Disklaufwerke kann eine bedeutende Verbesserung der Gesamtleistung erreicht werden, indem die Anforderungswarteschlange in aufsteigender bzw. fallender Zylinderreihenfolge geordnet wird. Die doppelt verknuepfte Anforderungsliste gestattet eine Sortierung nach dem "SCAN"-Prinzip. Um einen Zwischenspeicher, auf den bp zeigt, vor einem Zwischenspeicher, auf den ip zeigt, einzufuegen, schreibt man:

```
bp->av_forw = ip;            /* Vorwaertsverknuepfung vom Zwischenspeicher,
                                der eingefuegt werden soll */

bp->av_back = ip->av_back;   /* Rueckverknuepfung in den
                                neuen Zwischenspeicher */

(bp->av_back)->av_forw = bp; /* Vorwaertsverknuepfung in den Zwischen-
                                speicher, der in der Liste vor ip steht */

ip->av_back = bp;            /* Rueckverknuepfung in den
                                ip-Zwischenspeicher */
```

Die Strategieroutine entscheidet in jedem Fall, an welcher Stelle der Anforderungsliste die neue Anforderung eingereiht werden soll, verknuepft sie mit der doppelt verknuepften Anforderungsliste und ruft die Startroutine auf.

Beachten Sie bitte !

Die Einreihung der Anforderung in die Warteschlange muss bei deaktivierten Interrupts erfolgen.

Es koennen gefaehrliche Wechselwirkungen auftreten, wenn die Interrupt-Routine den Versuch unternimmt, eine Anforderung aus der Warteschlange auszugliedern, waehrend die Strategieroutine versucht, eine weitere Anforderung in die Warteschlange einzureihen. Da die Interrupts ausser Betrieb gesetzt sind, muss der Kode fuer die Eingliederung einer Anforderung sorgfaeltig geschrieben werden, um die Zeit zu minimieren, in der die Interrupts ausgeschaltet sind. Ebenso sollten Variablen, die in der Interrupt-Routine modifiziert werden koennten, erst benutzt werden, wenn die Interrupts ausgeschaltet sind. Eine Ignorierung dieser Forderung kann zu "Time of Check, Time of Use" (TOCTOU) Problemen fuehren, indem eine Variable von Anweisung zu Anweisung nicht gleich bleibt.

Wenn es keine vorherigen aktiven Anforderungen gab, muss das E/A-Geraet nach der Einordnung der Anforderung gestartet werden. Das erfolgt im allgemeinen durch das Initialisieren einer internen Variablen, die auf die laufende Anforderung hinweist, und dem nachfolgenden Aufruf der Geraetestartroutine. Die Geraetestartroutine kennzeichnet das E/A-Geraet als aktiv und gibt dann die Geraetekommandos und Parameter aus, um die Uebertragung zu beginnen.

Es gibt zwei weitere Routinen fuer die Blockgeraeteschnittstelle, die Geraete-Open- und -Close-Routinen. Die Open-Routine wird jedesmal aufgerufen, wenn eine Datei im E/A-Geraet eroeffnet wird. Sie verfuegt normalerweise ueber ein statisches Flag, um den Open-Kode zur Treiber- und Geraeteinitialisierung nur einmal abzuarbeiten. Die Initialisierung besteht darin, die Anforderungswarteschlange und aktuelle Anforderungszeiger zu setzen. Das E/A-Geraet selbst kann noch weitere Initialisierungsmassnahmen erfordern.

Die Close-Routine wird aufgerufen, wenn das letzte Open fuer das E/A-Geraet geschlossen wird. Weder von der Geraete-Open- noch von Geraete-Close-Routine wird verlangt, dass sie etwas ausfuehren. Die Routinen muessen jedoch im Treiber vorhanden sein, um die externen Referenzen im Kern zu realisieren.

8.5.2.2. Character-Geraeteschnittstelle

Character-Geraete unterscheiden sich in der Schnittstelle wesentlich von Blockgeraeten. Die Schnittstellen fuer das Anwender-Character-Geraet 1 sind:

u1open — Geraete-Open-Routine. Ihre Wirkungsweise aehnelt der usr1open-Routine.

u1close — Geraete-Close-Routine. Ihre Wirkungsweise aehnelt der usr1close-Routine.

u1read — wird aufgerufen, um ein Read anzufordern.

u1write — wird aufgerufen, um ein Write auszufuehren.

u1ioctl — wird aufgerufen, um Geraete-spezifische Operationen zur Statusaenderung auszufuehren.

Es sind wiederum 6 anwenderspezifische Character-Geraete vorgesehen. Die Open- und Close-Routinen sind mit ihren Blockgeraete-Gegenstuecken identisch und werden nicht weiter behandelt.

Die Lese- und Schreibroutinen werden mit einem Parameter aufgerufen:

```
u1read(dev);
dev_t dev;
```

wobei dev das E/A-Geraet ist, auf das die Uebertragung gerichtet ist. Die Parameter fuer die Uebertragung sind in der Struktur u enthalten (Definition in user.h). Die Byte-Anzahl ist in der Variablen u.u_count und die Adresse im User-Prozess ist in der Variablen u.u_base enthalten. Die Richtung der Uebertragung wird gesteuert, indem entweder die Lese- oder die Schreibroutine aufgerufen wird. Im allgemeinen wird das Argument dev dekodiert, um die Minor-Nummer zu bestimmen, damit die Ein-/Ausgabe an das entsprechende E/A-Geraet gerichtet werden kann.

Der einfachste Ausgabetreiber besteht darin, Zeichen nacheinander vom User-Prozess zu lesen, sie an das E/A-Geraet zu geben und auf den Geraete-Interrupt zu warten:

```
while (u.u_count--)
      if ((fubyte(u.u_base++, &c)) != 0) { /* weist auf eine
                                              ungueltige User-Adresse hin */
         u.u_error=ERAULT;
         break;
      } else {    /* wenn Zeichen richtig, gib es an das Geraet */

         /* hierher gehoert der Geraeteausgabekode */

         sleep (ulwrite, U1SLPPRI); /* warte auf Interrupt */
         if (ulxfrerr) {
         /* Geraetefehler-Flag wird durch Int-Routine gesetzt */
            break;
         }
      }
} /* Ende der While-Schleife */
```

Wie oben beschrieben, wird davon abgeraten, dieses Prinzip in Treibern einzusetzen, da der User-Prozess laenger als notwendig im Speicher verbleiben muss. Es werden Prinzipien bevorzugt, die eine Zwischenspeicherung nutzen, um eine fruehzeitigere Rueckkehr in den User-Prozess zu ermoeglichen. Der Treiber kann lokale Zwischenspeicher nutzen (am einfachsten, aber es geht Speicherplatz verloren) oder es koennen systemweite Zwischenspeicher eingesetzt werden (bessere Speicherauslastung, aber komplizierter).

Die Routinen putc und getc ermoeglichen das Einfuegen und Auslesen der Zeichen in/aus einen systemweiten gemeinsamen Zwischenspeichervorrat (getc und putc sind nicht die Routinen aus stdio(3)). In diesen Aufrufen ist c eine Zeichenvariable und queue eine Struktur struct clist. Diese Routinen nutzen die gemeinsamen Zwischenspeicher, die mit Drucker- und Terminal-Subsystemen geteilt werden. Die Benutzung von Zwischenspeichern verlangsamt die Uebertragung.

Ein Fragment fuer die Benutzung ist:

```
if (usrlqueue.c_cc->USR1HIWATER)
        sleep(usr1queue, USR1PRI);
else putc(c, usr1queue);
```

Sind mehr Zeichen als USR1HIWATER in der Warteschlange, geht der Prozess schlafen. In der Zwischenzeit leert die Interrupt-Routine die Warteschlange und fuehrt ein wakeup aus, wenn die Anzahl der Zeichen in der Warteschlange unter einer Geraete-abhaengigen "low water"-Markierung liegt. Das high-water/low-water-Prinzip verbessert die Gesamtleistung des Systems. Es kann eventuell guenstig sein, die Anzahl der systemweiten Zwischenspeicher zu erhoehen (indem NCLIST in sysparm.h mit sysgen geaendert wird). Jedes Element des Speicherfonds liefert 14 Zeichen Speicherkapazitaet.

Die Lese- und Schreibroutinen koennen eine Kanonisierung ihres Zeichenstroms ausfuehren. Im allgemeinen ist es oft vorteilhaft, eine solche Kanonisierung in den User-Prozessen zu verlagern, um den Kern klein zu halten. Ein Grund fuer die Kanonisierung im Treiber besteht darin, die Schnittstellenunabhaengigkeit fuer das E/A-Geraet zu wahren. Somit wird die Auswahl davon bestimmt, wieviele Programme das E/A-Geraet direkt nutzen werden. Ist die Anzahl gering, z.B. im Falle eines Druckers, der ueber einen Spooler angesteuert wird, kann die Verarbeitung an ein User-Programm uebergeben werden. In den anderen Faellen ist die Verarbeitung Teil des Geraetetreibers.

Die dritte Routine in der Character-Geraeteschnittstelle ist die ioctl-Service-Schnittstelle. Durch diese Routine ist es moeglich, einen Block von Steuerworten und Zustandsinformationen an das E/A-Geraet zu geben oder von diesem zu erhalten. Diese Schnittstelle ist voellig Geraete-abhaengig. Die Syntax und Bedeutung der Schnittstellen werden im Geraetetreiber bestimmt. Soweit wie moeglich sollte eine Geraete-spezifische ioctl-Schnittstelle vorhandene Parameterinterpretationen wieder benutzen. (siehe ioctl(2) und tty(4).)

8.5.2.3. E/A-Schnittstelle fuer Terminal-Geraete

Die Routinen der Schnittstelle fuer Terminal-Geraete sind mit den Schnittstellenroutinen des Character-Geraetes identisch. Die Unterschiede befinden sich innerhalb der Routinen. Um flexibel zu sein, rufen die Terminal-Treiber Protokollroutinen indirekt ueber eine Auswahltabelle auf (linesw-Tabelle). Der Index in diese Tabelle befindet sich unter Kontrolle des User-Prozesses. Er wird ueber die ioctl-Schnittstelle unter Nutzung der Funktion TIOCSETD gesetzt.

Die meisten Terminal-Treiber sind dafuer ausgelegt, verschiedene Kanaele (Leitungen) eines gegebenen E/A-Geraetes zu bearbeiten. Jeder Kanal wird durch die Minor-Geraetenummer bestimmt und besitzt eine eigene Struktur struct tty. (Es ist ein Feld struct tty vorhanden, die Auswahl erfolgt ueber die Minor-Geraetenummer.) Die Struktur tty enthaelt u.a. Zustandsinformationen des jeweiligen Kanals. Die in WEGA vorhandene tty-Struktur ist auf eine spezielle Hardware abgestimmt (U800-SIO, U880-CTC). Es ist moeglich, dass einige Zustandsinformationen fuer andere E/A-Geraete unterschiedlich interpretiert oder ignoriert werden muessen.

Im letzten Teil dieses Abschnitts wird jede Schnittstellenroutine beschrieben:

u1open

```
u1open(dev, flag)
dev_t dev;
int flag;
```

Die Open-Routine ist fuer die Initialisierung von Teilen der tty-Struktur des Kanals verantwortlich und wird fuer alle Geraete-abhaengigen Initialisierungen benoetigt. Wenn das E/A-Geraet bereits eroeffnet ist, sollten diese Initialisierungen umgangen werden. Das Argument dev ist die Geraetenummer, flag zeigt an, ob das E/A-Geraet fuer Lesen oder Schreiben eroeffnet werden soll. FREAD (0) bedeutet Lesen und FWRITE (02) bedeutet Schreiben. Die gegenwaertig vorhandenen Geraetetreiber ignorieren das Flag. Fuer Kanaele, die mit einem Modem verbunden sind, wartet die Open-Routine im Sleep-Zustand darauf, dass das "Data-Set-Ready"-Signal aktiv wird. Das wird in der Status-Interrupt-Routine erkannt, die ein wakeup erteilt, damit das Open fortgesetzt werden kann. Die Open-Routine ruft dann die Open-Routine des Protokollprogramms auf:

```
(*linesw[tp->t_line].l_open)(dev, tp);
```

wobei dev die Geraetenummer und tp ein Pointer auf struct tty fuer diese Leitung sind.

u1close

```
u1close(dev,flag)
dev_t dev;
int flag;
```

Diese Routine ruft das Protokoll-Close-Programm auf und deaktiviert die Geraete-Interrupts und enthaelt andere Geraete-abhaengigen Aktionen. Der linesw-Aufruf uebergibt nur den tty-Struktur-Pointer:

```
(*linesw[tp->t_line].l_close)(tp);
```

u1read

```
u1read(dev)
dev_t dev;
```

Diese Routine ruft das Protokoll-Leseprogramm auf:

```
(*linesw[tp->t_line].l_read)(tp);
```

u1write

```
u1write(dev)
dev_t dev;
```

Diese Routine ruft das Protokoll-Schreibprogramm auf:

```
(*linesw[tp->t_line].l_write)(tp);
```

u1ioctl

```
u1ioctl(dev, cmd, addr, flag)
dev_t dev;
int command;   /* ioctl-Kommando */
char *addr;    /* Adresse im User-Bereich, in dem sich der
                  Parameterblock befindet */
int flag;      /* Kopie des Open-Flags, zeigt an, ob das
                  Geraet fuer Lesen oder Schreiben eroeffnet ist */
```

Die Routine ruft das Protokoll-ioctl-Programm auf:

```
(*linesw[tp->t_line].l_ioctl)(cmd, tp, addr, dev);
```

Die Protokoll-ioctl-Routine fuehrt eine Geraete-unabhaengige Verarbeitung aus. Im allgemeinen besteht die Verarbeitung im Veraendern der internen Zustandsinformationen des Protokollprogramms. Die ioctl-Routine des Geraetetreibers verarbeitet dann die Geraete-abhaengigen Teile des Aufrufs, z.B. Setzen der Baudrate und anderer Parameter fuer die SIO/CTC-Schaltkreise. Die Protokollroutine gibt 0 zurueck, wenn es ihr nicht moeglich war, die Geraete-unabhaengigen Aktivitaeten auszufuehren. In diesem Falle sollten die Geraete-abhaengigen Teile nicht ausgefuehrt werden. Da in der Open-Routine ebenfalls das Geraete-abhaengige Parametersetzen erfolgen muss, ist es ueblich, eine Subroutine zum Parametersetzen zu benutzen.

8.6. Service-Routinen

bcopy

```
char *skaddr;  /* Quelladresse im Kern */
char *dkaddr;  /* Zieladresse im Kern */
int n;         /* Anzahl der Bytes */
```

Beide Adressparameter sind logische 24-Bit-Adressen.

```
ret = bcopy(skaddr, dkaddr, n);
```

kopiert n Bytes von der Adresse skaddr zur Adresse dkaddr. Beide Adressen liegen im Kern. Es wird 0 zurueckgegeben, wenn das Kopieren fehlerfrei verlaufen ist, sonst -1.

bzero

```
char *kaddr;   /* Adresse im Kern */
int n;         /* Anzahl der Bytes */
```

Der Adressparameter ist eine logische 24-Bit-Adresse.

```
bzero(kaddr, n);
```

fuellt den angegebenen Speicherbereich mit 0.

copyiin

```
int uiaddr;   /* Adresse im Textbereich eines nichtsegmentierten
                 User-Programms */
char *kaddr;  /* Adresse im Kern */
int n;        /* Anzahl der Bytes */
```

Der erste Adressparameter ist eine logische 16-Bit-Adresse, der zweite Adressparameter eine logische 24-Bit Adresse.

```
ret = copyiin(uiaddr, kaddr, n);
```

kopiert n Bytes von der Adresse uiaddr aus dem Textbereich eines nichtsegmentierten User-Programms zur Adresse kaddr im Kern. Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1. Diese Routine kommt in Treibern wahrscheinlich nicht zum Einsatz.

copyin

```
char *uaddr;  /* Adresse im User-Programm */
char *kaddr;  /* Adresse im Kern */
int n;        /* Anzahl der Bytes */
```

Beide Adressparameter sind logische 24-Bit-Adressen.

```
ret = copyin(uaddr, kaddr, n);
```

kopiert n Bytes aus dem Daten-/Stackbereich ab Adresse uaddr eines segmentierten oder nichtsegmentierten User-Programms (oder aus dem Textbereich eines segmentierten User-Programms) zur Adresse kaddr im Kern. Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1. Die Routine wird benutzt, um Daten in den lokalen Treiberzwischenspeicher einzugeben.

copyiout

```
char *kaddr;  /* Adresse im Kern */
int uiaddr;   /* Adresse im Textbereich eines
                 nichtsegmentierten User-Programms */
int n;        /* Anzahl der Bytes */
```

Der erste Adressparameter ist eine logische 16-Bit-Adresse, der zweite Adressparameter eine logische 24-Bit Adresse.

```
ret = copyiout(kaddr, uiaddr, n);
```

kopiert n Bytes von der Adresse kaddr im Kern zur Adresse uiaddr im Textbereich eines nichtsegmentierten User-Programms. Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1. (Fuer Treiber kaum benutzt).

copyout

```
char *kaddr;  /* Adresse im Kern */
char *uaddr;  /* Adresse im User-Programm */
int n;        /* Anzahl der Bytes */
```

Beide Adressparameter sind logische 24-Bit-Adressen.

```
ret = copyout(kaddr, uaddr, n);
```

kopiert n Bytes von der Adresse kaddr im Kern zur Adresse uaddr im Daten-/Stackbereich eines segmentierten oder nichtsegmentierten User-Programms (oder zum Textbereich eines segmentierten User-Programms). Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1.

Die Routine wird benutzt, um Daten aus den lokalen Treiberzwischenspeichern zu erhalten.

copyseg

```
int fromaddr;  /* physische Startadresse */
int toaddr;    /* physische Zieladresse */
```

Beide Parameter sind 16-Bit-Werte, wobei das unterste Byte der 24-Bit-Adresse mit 0 angenommen wird.

```
copyseg(fromaddr, toaddr);
```

kopiert 128 Woerter (256 Bytes) von der physischen Adresse fromaddr zur physische Adresse toaddr. Die Adressen sind physische 24-Bit-Adressen, mit einem angenommenen Wert von 0 im untersten Byte. Das bedeutet, dass die Adressen an 256 Byte-Grenzen synchronisiert sind. Diese Routine wird in Treibern nur begrenzt benutzt, da die Synchronisationsbedingungen nicht einfach zu erfuellen sind.

cpass

```
int c;
c = cpass();
```

kopiert ein Byte aus dem User-Bereich und passt u.u_base, u.u_offset und u.u_count an. Es wird -1 zurueckgegeben, wenn das Fetchen aus dem User-Bereich nicht funktioniert hat. Obwohl ein Integer-Wert bei erfolgreicher Ausfuehrung zurueckgegeben wird, wird das zurueckgegebene Zeichen auf ein Byte herabgesetzt, um Probleme der Zeichenerweiterung zu umgehen. cpass/passc sind die normalen Routinen, um im User-Prozess Zeichen anzusprechen. Sie beinhalten all das, was fuer die Aktualisierung der Zaehler und Adressen in der u-Struktur notwendig ist.

deverror

```
struct buf *bp;
unsigned int error1, error2;
deverror(bp, error1, error2);
```

wird in Interrupt-Programmen fuer ein Blockgeraet aufgerufen, um einen Uebertragungsfehler anzuzeigen. bp ist ein Pointer auf den zu uebertragenden Block, von dem die Major- und Minornummern des E/A-Geraetes gewonnen werden. error1 und error2 werden als Hex-Zahlen ausgegeben und geben ueber Fehler Auskunft, z.B. beinhalten sie den Geraetestatus.

dvi

```
int s;
s = dvi();
```

schaltet vektorierte Interrupts aus und gibt die vorhergehende Inhalt des FCW-Registers der CPU U8001 zurueck. Die Routine wird in der E/A-Anforderung der Treiber aufgerufen, bevor die Variablen, die fuer die Interrupt-Schichten benoetigt oder gesetzt werden, geprueft oder veraendert werden. dvi gibt das laufende FCW zurueck, das in einem nachfolgenden rvi-Aufruf benutzt werden kann, um es wieder so zu setzen, wie es vorher war. Die meisten Treiber nutzen einfach dvi/evi-Paare, da sie den Status kennen und ihn nicht wieder erzeugen muessen.

evi

```
int s;
s = evi();
```

aktiviert vektorierte Interrupts. Es wird die vorhergehende Inhalt des FCW zurueckgegeben.

fubyte

```
char *uaddr;  /* Adresse im User-Programm */
char c;       /* Variable im Kern Treiber */

ret = fubyte(uaddr, &c);
```

kopiert 1 Byte aus dem Daten-/Stackbereich ab Adresse uaddr eines segmentierten oder nichtsegmentierten User-Programms (oder aus dem Textbereich eines segmentierten User-Programms) zur Variablen c im Kern. Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1. fubyte und verwandte Routinen (fuword, fuibyte, fuiword, subyte, suibyte, suword, suiword) werden in Geraetetreibern wahrscheinlich nie direkt angewandt. Es werden Funktionen der hoeheren Ebene (z.B. cpass), die ebenfalls die verschiedenen Statusinformationen beinhalten, bevorzugt.

fuword

```
char *uaddr;  /* Adresse im User-Programm */
int c;        /* Variable im Kern (Treiber) */

ret = fuword(uaddr, &c);
```

kopiert 1 Word, sonst analog fubyte.

fuibyte

```
int uiaddr;   /* Adresse im Textbereich eines
                 nichtsegmentierten User-Programms */
char c;       /* Variable im Kern (Treiber) */
```

Der erste Parameter ist eine logische 16-Bit-Adresse.

```
ret = fuibyte(uiaddr, &c);
```

kopiert 1 Byte von der Adresse uiaddr aus dem Textbereich eines nichtsegmentierten User-Programms zur Variablen c im Kern. Es wird 0 zurueckgegeben, wenn das Kopieren erfolgreich verlaufen ist, sonst -1. Diese Routine kommt in Treibern wahrscheinlich nicht zum Einsatz.

fuiword

```
int uiaddr;   /* Adresse im Textbereich eines
                 nichtsegmentierten User-Programms */
int c;        /* Variable im Kern (Treiber) */
```

Der erste Parameter ist eine logische 16-Bit-Adresse.

```
ret = fuiword(uiaddr, &c);
```

kopiert 1 Word, sonst siehe fuibyte.

getc

```
struct clist *queue; /* Pointer, um Header-Struktur
                        einzureihen */
int c;
c = getc(queue);
```

holt ein Zeichen von der Warteschlange. Die Warteschlange wird aus den systemweiten Zwischenspeichern gebildet.

in

```
unsigned int ioaddr;
int c;
ret = in(ioaddr);
```

fuehrt einen "in"-Befehl des U8000 aus, der ein Wort von der E/A-Adresse ioaddr liest.

inb

```
unsigned int ioaddr;
int c;
c = inb(ioaddr);
```

fuehrt ein "inb" von der angegebenen E/A-Adresse aus. Es wird ein Integer-Wert zurueckgegeben, das obere Byte ist auf 0 gesetzt.

iodone

```
struct buf *bp;
iodone(bp);
```

markiert einen E/A-Auftrag fuer den bezeichneten Zwischenspeicher als beendet und fuehrt ein wakeup aus, wenn noch Prozesse auf der Adresse des Zwischenspeichers warten. Diese Routine wird in Interrupt-Routinen fuer Blockgeraete angewandt.

max

```
int z;
unsigned int x,y
z = max(x,y);
```

gibt das Maximum von x und y zurueck. Genaugenommen sollte max vorzeichenlos deklariert werden, aber beim U8000 ist das bedeutungslos.

min

```
int z;
unsigned int x,y
z = min(x,y);
```

gibt das Minimum von x und y zurueck.

otirb

```
unsigned int ioaddr;
char *blkaddr;
int cnt;
otirb(ioaddr, blkaddr, cnt);
```

fuehrt

```
otirb @R7, @RR4, r3
```

aus, wodurch cnt Bytes, die ab Adresse blkaddr beginnen, an den Port ioaddr ausgegeben werden.

outb

```
unsigned int ioaddr;
char c;
outb(ioaddr, c);
```

fuehrt

```
outb @R7, RL6
```

aus, wodurch das Zeichen c an den Port ioaddr ausgegeben wird.

panic

```
char *string;
panic(string);
```

gibt "string" auf dem Systemterminal aus und haelt das System abrupt an. Von dieser Routine darf nur Gebrauch gemacht werden, wenn der Treiber feststellt, dass ein Gefahrenmoment aufgetreten ist, dass die Integritaet des Systems gefaehrdet. Wenn panic aufgerufen wird, koennen in den in den Dateisystem logische Schaeden entstehen. Diese koennen mit fsck behoben werden.

physio

```
long usr1strategy();    /* kein Return-Wert */
struct buf rusr1buf;    /* Puffer fuer raw-E/A */
dev_t dev;              /* E/A-Geraet */
int direction;          /* B_READ fuer Lesen, B_WRITE fuer Schreiben */
physio(usr1strategy, &rusr1buf, dev, direction);
```

liefert die Schnittstelle fuer das raw-Geraet in Block-Geraetetreibern.

printf

```
char *pattern;
printf(pattern, arg1, arg2, arg3, arg4, arg5);
/* Es sind verschiedene Formate moeglich,
aber nur bis zu 5 Argumente gestattet */
```

ist die interne Version von printf fuer die Ausgabe von Mitteilungen auf dem Systemterminal. Waehrend der Ausgabe der Zeichen sind die Interrupts gesperrt. In einem Treiber koennen schon kurze printf-Aufrufe zur Fehlertestung des Treibers soviel Zeit in Anspruch nehmen, dass zeitempfindliche Fehler verdeckt werden. Es werden nur die Formate %s, %u, %d (== %u), %o, %x und %D unterstuetzt. Es sind nur bis zu 5 Argumente erlaubt.

putc

```
char c;
struct clist *queue;
putc(queue,c);
```

plaziert das Zeichen c in die angegebene Warteschlange. Das wird fuer das Zwischenspeichern in Character-Geraeten angewandt. Die Warteschlangen werden aus Zwischenspeichern gebildet, die aus einem systemweiten gemeinsamen Vorrat bereitgestellt werden, dessen Groesse durch NCLIST in der Systemgenerierungsprogramm bestimmt wird. Geraetetreiber sollten Schritte enthalten, die absichern, dass ein E/A-Geraet nicht den gesamten Zwischenspeicher belegt.

reti

```
reti();
```

simuliert ein U880-reti, um U880-Peripherie zurueckzusetzen.

rvi

```
int s;
rvi(s);
```

setzt das FCW der CPU (siehe evi, dvi).

signal

```
int processgrp;
int SIGNALNAME;
signal(processgrp, SIGNALNAME);
```

sendet ein Signal SIGNALNAME (normalerweise SIGHUP) an alle Prozesse in der Prozessgruppe processgrp. Die Routine wird z.B. in tty-Treibern bei der Behandlung der Status-Interrupts benutzt. Die Prozessgruppe wird aus allen Prozessen, die mit einem Login verbunden sind, gebildet. In Terminal-Treibern wird die Prozessgruppe aus der Variablen t_pgrp in der fuer den Kanal gueltigen tty-Struktur gewonnen.

sleep

```
unsigned long chan;
int priority;
sleep(chan, priority);
```

gibt die Steuerung des Prozesse ab, der Prozess geht "schlafen". Wenn ein anderer Prozess wakeup(chan) aufruft, wird dieser Prozess reaktiviert. Alle auf diesem Kanal wartenden Prozesse werden auf diese Weise aktiviert. Jeder Prozess sollte ueberpruefen, ob der Grund, aus dem er schlafen geschickt wurde, aufgehoben wurde. Wenn nicht, muss er wieder schlafen. Mit "priority" wird die Prioritaet der Reaktivierung festgelegt. Prozesse mit Prioritaeten kleiner als PZERO koennen durch Signale von anderen Prozessen nicht unterbrochen werden. Im allgemeinen bezeichnet "chan" eine vereinbarte eindeutige Adresse, z.B. die Adresse eines Zwischenspeichers. In Interrupt-Programmen darf "sleep" nicht aufgerufen werden.

suibyte

```
int uiaddr;
char c;
suibyte(uiaddr, c);
```

speichert das Zeichen c auf der angegebenen Adresse im Textbereich des nichtsegmentierten User-Programms. Es wird 0 zurueckgegeben, wenn dies erfolgreich verlaufen ist, sonst -1. Diese Routine kommt in Treibern vermutlich nicht zum Einsatz.

suiword

```
int uiaddr;
int i;
suiword(uiaddr, i);
```

speichert das Wort i, sonst siehe suibyte.

subyte

```
char *uaddr;
char c;
subyte(uaddr, c);
```

speichert das Zeichen c auf der angegebenen Adresse im Daten-/Stackbereich eines nichtsegmentierten User-Programms oder in einem beliebigen Bereich eines segmentierten Programms. Es wird 0 zurueckgegeben, wenn dies erfolgreich verlaufen ist, sonst -1.

suword

```
char *uaddr;
int i;
suword(uaddr, i);
```

speichert das Wort i, sonst siehe subyte.

timeout

```
long fcn();      /* Funktionstyp ist unwichtig */
int *arg;
unsigned ticks;
timeout (&fcn, arg, ticks);
```

sorgt fuer das Auftreten eines Aufrufs der Form

```
(*fcn)(arg);
```

nach "ticks" 1/60 Sekunden. Timeout wird benutzt, um die Abarbeitung des Programms fuer eine bestimmte Zeit auszusetzen, z.B. nur Realisierung einer Verzoegerung. Die Zeit ist ein Minimalwert. Der tatsaechliche Aufruf kann spaeter auftreten. Wenn "ticks" mit sehr kleinem Wert (0 oder 1) angegeben ist, ist eine Moeglichkeit fuer Wettlaeufe gegeben.

wakeup

```
unsigned long chan;
wakeup(chan);
```

weckt alle auf dem Kanal chan schlafenden Prozesse. "sleep" und "wakeup" sind die grundlegenden Mechanismen fuer die Prozess-Synchronisierung im Kern.

8.7. Rahmenprogramme fuer Geraetetreiber

Fuer die Einarbeitung in die Problematik der WEGA-Treiber und als Basis zur Entwicklung von Anwendertreibern wurden drei Treiber in Form von "Rahmenprogrammen" entwickelt. Die Rahmenprogramme enthalten die logischen Routinen fuer Block-, Character- und Terminal-Treiber. Die Rahmenprogramme sind in der WEGA-Standardsoftware enthalten:

```
/usr/sys/dev/blockdev.c - Treiber fuer ein Blockgeraet
/usr/sys/dev/chardev.c  - Treiber fuer ein Character-Geraet
/usr/sys/dev/ttydev.c   - Treiber fuer ein Terminal-Geraet
```

Hinweise des Lesers zu diesem Dokumentationsband

Wir sind staendig bemueht, unsere Unterlagen auf einem qualitativ hochwertigen Stand zu halten. Sollten sie Hinweise zur Verbesserung dieser Dokumentation haben, so bitten wir Sie, diesen Fragebogen auszufuellen und an uns zurueckzusenden.

Titel des Dokumentationsbandes: WEGA-Systemhandbuch

Ihr Name / Tel.-Nr.:

Name und Anschrift des Betriebes:

Genuegt diese Dokumentation Ihren Anspruechen? ja / nein
Falls nein, warum nicht?

Was wuerde diese Dokumentation verbessern?

Sonstige Hinweise:

Fehler innerhalb dieser Dokumentation:

Unsere Anschrift: Kombinat VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Abteilung Basissoftware
Hoffmannstrasse 15-26
BERLIN
1193

KOMBINAT VEB
ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW
»FRIEDRICH EBERT«

Exporteur:
HEIM-ELECTRIC
EXPORT-IMPORT
Volkseigener Außenhandelsbetrieb
der Deutschen Demokratischen Republik
EAW-Automatisierungstechnik Export-Import
DDR - 1026 Berlin, Alexanderplatz 6
Haus der Elektroindustrie
Telefon 2180 · Telex 011 - 4557

VEB ELEKTRO-APPARATE-WERKE BERLIN-TREPTOW
»FRIEDRICH EBERT«
Stammbetrieb des Kombinats EAW
DDR - 1193 Berlin, Hoffmannstraße 15-26
Fernruf: 2760
Fernschreiber: 0112263 eapparate bln
Drahtwort: eapparate bln

Die Angaben über technische Daten entsprechen dem bei Redaktionsschluß vorliegenden Stand. Änderungen im Sinne der technischen Weiterentwicklung behalten wir uns vor.

Ausgabe Januar 1989