Franz Schomacher

MS-DOS-Format unter CP/M 3.0

Wie man einzelne Sektoren unter CP/M 3.0 auf der Diskette lesen und schreiben kann, wurde in MC 3/85 beschrieben. Die im folgenden Beitrag vorgestellten Programme dienen zum Übertragen ganzer Dateien unter diesem Betriebssystem auf bzw. von MS-DOS-formatierten Disketten.

Will man Dateien von einer MS-DOS-Diskette lesen oder auf diese schreiben, so muß man natürlich wissen, wie Disketten unter MS-DOS organisiert sind: Zunächst einmal gibt es vier verschiedene Formate, nämlich einseitig mit acht Sektoren pro Spur, doppelseitig mit acht Sektoren pro Spur und beides noch einmal mit neun Sektoren pro Spur. Dies bereitet schon die erste Schwierigkeit, denn unter CP/M 3.0 kann man neun Sektoren pro Spur nicht lesen. Deshalb wird bei neun Sektoren eine Datei möglicherweise nur zum Teil übertragen; dies wird aber vom Programm festgestellt und gemeldet.

Format-Erkennung

Um zu erkennen, in welchem Format eine Diskette formatiert wurde, wie auch für die Berechnung der sogenanten Gruppen, braucht man die „File Allocation Table" (FAT), denn in dieser Tabelle stehen die Informationen über das Diskettenformat und die Belegung der Diskette. Für das Format ist immer das erste Byte der Tabelle zuständig, und zwar entspricht:

FFh doppelseitig mit acht Sektoren pro Spur,

FEh einseitig mit acht Sektoren pro Spur,

FDh doppelseitig mit neun Sektoren pro Spur und

FCh einseitig mit neun Sektoren pro Spur.

In der FAT stehen ausserdem die Gruppennummern, die jeweils einem Bereich auf der Diskette zugeordnet sind. Steht hier der Wert 000, so ist die betreffende Gruppe frei und kann benutzt werden. FF8h...FFFh bedeutet, daß es sich um die letzte Gruppe einer Datei handelt. Jede andere Zahl gibt die nächste Gruppe einer Datei an.

Dateien

Die Verkettung dieser Gruppen zu einer kompletten Datei läuft nach folgendem Schema ab: Die Startgruppe einer Datei steht im Inhaltsverzeichnis (Directory).

Will man aus einer Gruppennummer die nächste Gruppennummer bestimmen, so muß zunächst ein Offset in der FAT bestimmt werden, der sich aus der ganzzahligen Multiplikation der alten Nummer mit 1,5 ergibt, denn jeder Eintrag ist 1,5 Byte lang. Dann liest man aus der FAT mit diesem Offset ein Wort (= 2 Byte). War die alte Gruppennummer gerade, so bilden die niederwertigen 12 Bit die neue Gruppennummer, sonst die höherwertigen. Ergibt sich als Gruppennum-

```
 1: CONST Blocksize  = 512; {physikalische Blockgroesse unter MS-DOS }
 2:       BufGr      =  86; {86 fuer COM-File, sonst 24               }
 3: {*********  Achtung BufGr muss eine gerade Zahl sein!   *********}
 4:                        {Gilt so nur fuer Commodore PC128          }
 5: TYPE  DPB = RECORD                              {Disk-Parameter-Block}
 6:                SPT                 :Integer; {vgl. mc Heft 3/85   }
 7:                BSH, BLM, EXM       :Byte;
 8:                DSM, DRM            :Integer;
 9:                AL0, AL1            :Byte;
10:                CKS, OFF            :Integer;
11:                PSH, PHM            :Byte;
12:            END;
13:       tDisk = (doppelacht, einsacht, doppelneun, einsneun);
14:       tmode = (Suche, Zeigen);
15:       tBuff = ARRAY[1..Blocksize] OF Byte;
16:       String12 = STRING[12];
17:       Anystring = STRING[255];
18:
19: VAR   Psector, Psize,
20:       SekpGr, LSekpTr, TrFak,
21:       Eintraege, i, StartGruppe,
22:       Datstart, benutzt             : Integer;
23:       ptr                           : ^DPB ABSOLUTE i;
24:       DPH                           : ^Integer ABSOLUTE i;
25:       Diskart                       : tDisk;
26:       FATBuf                        : ARRAY[1..2] OF tBuff;
27:       DatBuf                        : ARRAY[1..BufGr] OF tBuff;
28:       Suchname, CPMName                     : String12;
29:       sel                           : Byte;
30:       Dateiende, erst, gefunden,
31:       Abbruch                       : Boolean;
32:       Antwort                       : Char;
33:
34: PROCEDURE Meldung(Meldetext: Anystring);
35: {Dient zur Ausgabe einer Meldung in der Menuzeile 23                 }
36: BEGIN
37:   GotoXY(1,23);ClrEol;
38:   Write(#27,'G4');              {Schaltet auf inverse Darstellung   }
39:   Write(#7,' ',Meldetext,' ');
40:   Write(#27,'G0');              {Schaltet zurueck auf normale Darstellung}
41:   Write('  ')
42: END; {Meldung}
43:
44: FUNCTION Ubios(FN, PA, PBC, PDE, PHL: Integer): Integer;
45: {Da unter CP/M 3.0 BIOS-Funktionen ueber BDOS-Funktion 50 aufgerufen}
46: {werden, wird diese Funktion benutzt. vgl. mc Heft 3/85             }
47: TYPE ParameterBlock = RECORD
48:                         func, Areg          :Byte;
49:                         BCreg, DEreg, HLreg :Integer;
50:                       END;
```

Bild 1. Mit diesem Programm können Dateien von MS-DOS-formatierten Disketten gelesen werden

```
51: VAR BiosPB         :ParameterBlock;
52:     Result         :Integer;
53: BEGIN
54:    WITH BiosPB DO
55:    BEGIN
56:       func  := FN;
57:       Areg  := PA;
58:       BCreg := PBC;
59:       DEreg := PDE;
60:       HLreg := PHL;
61:    END;
62:    Result := 0;
63:    CASE FN OF
64:       2,3,7,13,14,15,17,18,19,24: Result := Bdos(50, Addr (BiosPB));
65:       9,16,20,22,25:              Result := BdosHL(50, Addr (BiosPB));
66:       ELSE                        Bdos(50, Addr (BiosPB));
67:    END;
68:    Ubios := Result;
69: END; {Ubios}
70:
71: PROCEDURE Init;
72: BEGIN
73:   i := Ubios(9, 0, 0, 0, 0);          {Laufwerk -hier 0 = aktuell-}
74:   Bdos(14,0);                         {auswaehlen.                }
75:   i := BdosHl(31);                    {Adresse des DPB ermitteln. }
76:   Psize := 1 SHL ptr^.PSH;
77: END; {Init}
78:
79: PROCEDURE RWSektor(Dr, Tr, lSec : Integer; Schreib: Boolean;
80:                    VAR SecBuf : tBuff);
81: { Liest einen physikalischen Sektor von Diskette in SecBuf,          }
82: { bzw. schreibt SecBuf auf Diskette, je nach Wert von Schreib.       }
83: BEGIN
84:    lSec := lSec DIV Psize;                  {log. in phys. Sektor    }
85:    i := Ubios (9, 0, Dr, 1, 0);             {Drive auswaehlen        }
86:    Psector := Ubios (16, 0, lSec, DPH^, 0);{phys. Sektor berechnen }
87:    i := Ubios (23, 0, 1, 0, 0);             {nur einen Sektor lesen  }
88:    i := Ubios (10, 0, Tr, 0, 0);            {Spur waehlen            }
89:    i := Ubios (11, 0, Psector, 0, 0);       {Sektor waehlen          }
90:    i := Ubios (12, 0, Addr(SecBuf), 0, 0); {DMA-Adresse waehlen     }
91:    i := Ubios (28, 1, 0, 0, 0);             {Bank waehlen            }
92:    IF Schreib
93:    THEN i := Ubios (14, 0, 0, 0, 0)         {phys. Sektor schreiben  }
94:    ELSE i := Ubios (13, 0, 0, 0, 0)         {phys. Sektor lesen      }
95: END;{RWSektor}
96:
97: PROCEDURE LiesFAT;
98: {Liest die File-Allocation-Table in einen Buffer und setzt je nach }
99: {Diskart (einseitig -doppelseitig bzw. neun - acht Sektoren) Werte }
100: {fuer Directoryeintraege und zur Umrechnung von Gruppen in Track   }
101: {und Sektor fest.                                                 }
102: BEGIN
103:   RWSektor(0,0,4,false,FATBuf[1]);
104:   Diskart := tDisk(Abs(FatBuf[1,1] - $ff));
105:   IF Ord(Diskart) > 1
106:   THEN RWSektor(0,0,8,false,FATBuf[2]);
107:   IF Diskart IN [doppelacht, doppelneun]
108:   THEN SekpGr := 2
109:   ELSE SekpGr := 1;
110:   IF Diskart IN [doppelacht, einsacht]
111:   THEN LSekpTr := 32
112:   ELSE LSekpTr := 36;
113:   IF Diskart IN [einsacht, einsneun]
114:   THEN BEGIN
115:          TrFak := 2;
116:          Eintraege := 64
117:        END
118:   ELSE BEGIN
119:          TrFak := 1;
120:          Eintraege := 112
121:        END
122: END; {LiesFAT}
123:
124: PROCEDURE Datenstart_bestimmen;
125: {Setzt je nach Diskart den Wert fuer Datenstart fest.}
126: BEGIN
127:   CASE Diskart OF
128:     doppelacht: Datstart := 40;
129:     einsacht:   Datstart := 28;
130:     doppelneun: Datstart := 48;
131:     einsneun:   Datstart := 36
132:   END
133: END; {Datenstart_bestimmen}
134:
135:
136: PROCEDURE Gruppe_in_Track_Sektor(Gr: Integer; VAR Tr, Sec: Byte);
137: {Berechnet aus der Gruppennummer Spur und Sektor auf der Diskette.}
138: VAR weiter      : Char;
139: BEGIN
140:   Gr := Gr - 2;
141:   Gr := Gr * SekpGr;
142:   Gr := Gr * 4;
143:   Gr := Gr + DatStart; {Start der Daten}
144:   Tr := Gr DIV LSekpTr;
145:   Sec := Gr MOD LSekpTr;
146:   IF Sec > 31
147:   THEN BEGIN
148:          Meldung('Der folgende Sektor kann nicht gelesen werden. '
149:                  + 'Weiter? <j/n>');
150:          Read(KBD,weiter); Write(weiter);
151:          Abbruch := Upcase(weiter) = 'N'
152:        END;
153:   Tr := Tr * TrFak;
154: END; {Gruppe_in_Track_Sektor}
155:
156: PROCEDURE Nameneingeben(VAR Name: String12);
157: {Dient zur Eingabe des Namens der zu uebertragenden Datei     }
158:   PROCEDURE Mache_Gross(VAR Str: String12);
159:   {Wandelt alle Buchstaben in Str in Grossbuchstaben um       }
160:   VAR i : Integer;
161:   BEGIN
162:        FOR i := 1 TO 12 DO
163:        IF Str[i] <> ' ' THEN Str[i] := Upcase(Str[i])
164:   END;{Mache_Gross}
165:
166: BEGIN
167:   Meldung('Bitte Name der zu uebertragenden Datei angeben <Typ> ');
168:   Read(Name);
169:   CPMname := Name;
170:   Name := Copy(Name,1,Pos('.',Name)-1)+Copy(Name,Pos('.',Name)+1,3);
171:   IF Length(Name) < 11
172:   THEN Name := Copy(Name,1,Length(Name)-3)
173:                +Copy('         ',1,11-Length(Name))
174:                + Copy(Name,Length(Name)-2,3);
175:   Mache_Gross(Name);
176: END; {Nameneingeben}
```

```
1: PROGRAM Schreib_MS_DOS_Disketten;
2: {Entwickelt auf Commodore PC 128 unter CP/M+ mit einem Disketten-}
3: {laufwerk Commodore 1571                                  -----  }
4:
5: {$i diskproc.inc}
6:
7: VAR AnzSektoren : Integer;
8: PROCEDURE DirectoryEintrag(Gr: Integer);
9: {Dient zum Eintrag von Namen und Startgruppe in die Directory der}
10: {MS-DOS-Diskette                                               }
11: VAR DirBuf                          : tBuff;
12:     DirTr, DirSec, Zaehl, ZeilZaehl,
13:     EinZaehl, k                     : Byte;
14:     Offset                          : Integer;
15:     Ende                            : Boolean;
16:     Vglname                         : String12;
17: BEGIN
18:   DirTr := 0;
19:   DirSec := 12;
20:   RWSektor(0,DirTr,DirSec,false,DirBuf);
21:   gefunden := False;
22:   Ende := False;
23:   EinZaehl := 0;
24:   WHILE  NOT Ende AND NOT gefunden DO
25:   BEGIN
26:     Offset := 0;
27:     REPEAT
28:       EinZaehl := Succ(EinZaehl);
29:       IF (DirBuf[Offset+1] = 0) OR (DirBuf[Offset+1] = $e5)
30:               THEN BEGIN {Freier Platz gefunden}
31:                     FOR k := 1 TO 11 DO
32:                         DirBuf[Offset+k] := Ord(Suchname[k]);
33:                     FOR k := 12 TO 26 DO
34:                         DirBuf[offset+k] := 0;
35:                     gefunden := True
36:                    END;
37:       IF NOT gefunden
38:       THEN BEGIN
39:              Ende := (DirBuf[Offset+1] = 0) OR (EinZaehl = Eintraege);
40:              Offset := Offset + 32
41:            END
42:     UNTIL gefunden OR Ende OR (Offset >= 512);
43:     IF NOT gefunden AND NOT Ende
44:     THEN BEGIN
45:            DirSec := Dirsec + 4;
46:            RWSektor(0,DirTr,DirSec,false,DirBuf)
47:          END
48:   END; {While} {Platz gefunden oder Directory zuende}
49:   IF gefunden
50:   THEN BEGIN
51:          DirBuf[Offset+27] := Lo(Gr);
52:          DirBuf[Offset+28] := Hi(Gr);
53:          RWSektor(0,DirTr,DirSec,True,DirBuf)
54:        END
55:   ELSE BEGIN
56:          Meldung('Directory voll');
57:          Halt
58:        END
59: END; {DirectoryEintrag}
60:
```

```
61: PROCEDURE SchreibeDatei;
62: {Schreibt die Datei auf die MS-DOS-Diskette und aktualisiert die FAT.}
63: VAR k, z, Gruppe             : Integer;
64:     Track, Sektor            : Byte;
65:     Start, oben              : Boolean;
66:
67: FUNCTION FreieGruppe(k :Integer; VAR ob :Boolean): Boolean;
68: {Dient zur Ueberpruefung, ob in der FAT an der Stelle k eine freie}
69: {Gruppe existiert, und zwar werden in Abhaengigkeit von ob die    }
70: {oberen (ob = true) oder unteren (ob = false) 12 Bits untersucht. }
71: VAR Hilf : Integer;
72:     h    : Boolean;
73: BEGIN
74:   h := False;
75:   IF ob
76:   THEN BEGIN {obere Bits untersuchen (von k und k+1)}
77:          Hilf := FATBuf[1,k+1] * 256 + FATBuf[1,k];
78:          IF Hilf AND 4095 = 0 THEN h := True
79:        END;
80:   IF NOT h
81:   THEN BEGIN {noch keine freie Gruppe gefunden,                }
82:              {also untere 12 Bits untersuchen (von k+1 u. k+2)}
83:          Hilf := FATBuf[1,k+2] * 256 + FATBuf[1,k+1];
84:          IF Hilf SHR 4 = 0
85:          THEN BEGIN h := true; ob := False END
86:        END;
87:   FreieGruppe := h
88: END; {FreieGruppe}
89:
90: PROCEDURE Trageein(k :Integer; ob :Boolean);
91: {Dient dazu, eine Gruppe in der FAT als belegt mit dem entsprechenden }
92: {Wert zu kennzeichnen.                                                }
93: VAR Ein, Hilf, j : Integer;
94:     h           : Boolean;
95: BEGIN
96:   j := k;
97:   IF Dateiende
98:   THEN Ein := $FFF
99:   ELSE BEGIN {naechste freie Gruppe suchen und eintragen}
100:          h := NOT ob;
101:          IF h THEN j := j + 3;
102:          WHILE NOT FreieGruppe(j,h) AND (j < 512) DO
103:                j := j + 3;
104:          IF j < 512
105:          THEN BEGIN
106:                 IF h THEN Ein := Trunc((j-1)/1.5)
107:                      ELSE Ein := Trunc((j-1)/1.5)+1
108:               END
109:          ELSE BEGIN
110:                 Meldung('Diskette voll');
111:                 Halt
112:               END;
113:        END;
114:   IF ob
115:   THEN BEGIN
116:          Hilf := FATBuf[1,k+1] *256 + FATBuf[1,k];
117:          Hilf := Hilf OR Ein;
118:          FATBuf[1,k+1] := Hi(Hilf);
119:          FATBuf[1,k]   := Lo(Hilf)
120:        END
```

Bild 2. Zum Schreiben auf MS-DOS-formatierte Disketten mit diesem Programm dürfen nur acht Sektoren pro Spur vorhanden sein