Professionell programmieren

Per XMS-Treiber kontrolliert das Extended-RAM benutzen

Eines ist seit Windows sicher: der Vormarsch der Grafikapplikationen ist unaufhaltsam. Dadurch wird der Platz im Speicher immer enger. EMS ist die eine Möglichkeit, der Speichernot aus dem Wege zu gehen. Leider kann Windows damit nichts anfangen. Seit einiger Zeit gibt es allerdings auch eine festgelegte Schnittstelle für das normale Extended-Memory: Die XMS-Schnittstelle.

Gerade bei Assembler-Programmen ist der Umgang mit dem erweiterten Speicherraum schwierig. Das Ablaufen im Protected Mode eröffnet völlig neue Perspektiven – und damit völlig neue Probleme. Welcher Debugger kann schon im Protected Mode tracen? Wie kann man aus dem Protected Mode einen ganz normalen DOS-Aufruf absetzen?

Glücklicherweise sind die meisten Programme klein genug, um im konventionellen Hauptspeicher Platz zu finden. Doch Pufferspeicher kann man oft gar nicht genug haben. Schon sind VGA-Karten mit 1 MByte Bildschirmspeicher für um die 300 Mark zu haben. Aber wohin mit all den Pixeln, wenn man mal eben eine kleines Fenster einblenden will?

Prinzipiell könnte man jeweils einen größeren Datenblock in den Speicherbereich über 1 MByte verschieben und bei Bedarf zurückladen. Allerdings obliegt in dem Fall der Applikation die alleinige Verantwortung über die Verwaltung dieses Speichers. Von Kompatibilität und Verträglichkeit kann dann keine Rede mehr sein. Deshalb ist es besser, die Zuteilung des Speichers einem speziellen Treiber zu überlassen.

Dies kann entweder ein EMS-Treiber sein, wenn es sich um Expanded Memory handelt, oder ein XMS-Treiber wie HIMEM.SYS, wenn auf Extended Memory aufgebaut wird. Über die Hardwarestrukturen und Unterschiede dieser beiden Systeme ist bereits genug geschrieben worden. Woran es noch mangelt, ist die Bedienung für die Einbindung in eigene Programme. Man kann sich zwar ausführliche Funktionsbeschreibungen besorgen, doch die sind meist viel zu ausführlich und zu kompliziert. Für die meisten Fälle genügen drei sehr einfache Funktionen:

- Bereitstellen eines Speicherblocks (Allocate)
- Kopieren eines Speicherbereichs (aus, in oder innerhalb des Speicherblocks)
- Freigeben des Speicherblocks nach der Benutzung.

Betrachten wir die entsprechenden Funktionen des XMS-Treibers. Bevor von ihm ein Speicherblock angefordert werden kann, muß natürlich sichergestellt sein, daß der Treiber auch installiert ist. Dies kann mit dem Multiplex-Interrupt 2Fh festgestellt werden.

```
mov ax,4300h ; XMS Installation
             ; check
int 2Fh      ; multiplex INT
cmp al,80h   ; XMS installed?
jne noxms
```

Die Prüfung, ob der Interrupt 2Fh definiert ist, kann entfallen. Bereits seit DOS Version 2 wird er für die Kommunikation mit Hintergrundprozessen (damals nur PRINT.COM) verwendet. Allgemein wird dabei der Identifikationscode des gesuchten Treibers im AH-Register eingetragen (siehe Tabelle 1 am Ende des Artikels) und die Nummer der gewünschten Funktion in AL. Üblicherweise wird die Funktion Nummer Null verwendet, um festzustellen, ob der Treiber vorhanden ist. Der Treiber ändert den Wert in AL auf -1 und hat damit seine Schuldigkeit bereits getan. Fühlt sich kein Treiber der Vektorkette 2Fh mit dem übergebenen Code in AH angesprochen, bleibt der Wert in AL Null und das aufrufende Programm weiß, daß der gesuchte Treiber nicht

dabei war. Der XMS-Treiber weicht von dieser Konvention insofern ab, daß er ausschließlich den Rückgabewert 80h als gültige Antwort auf die Installationsanfrage definiert.

Über den Multiplexkanal gibt es nur noch eine weitere XMS-Funktion: die Adresse des XMS-Handlers selbst erfragen.

```
xmsvec dd ?
mov  ax,4310h
int  2Fh
mov  word ptr xmsvec,bx
mov  word ptr xmsvec + 2,es
```

Prinzipiell hätte man zwar alle XMS-Funktionen gleich über INT 2Fh abwickeln können. Doch je mehr Programme sich in diese Vektorkette einschleifen, desto länger kann es dauern, bis ein Funktionsaufruf den Adressaten erreicht. Die eigentlichen XMS-Funktionen werden daher mit dem Funktionscode in AH durch einen FAR CALL an die hier erhaltene Adresse angestoßen.

Kehren wir nun zu den angesprochenen drei Grundfunktionen zurück: Allocate, Move und Free. Bevor man seine Daten im XMS loswird, muß man sich einen entsprechend großen Speicherbereich zuweisen lassen. Dies geschieht in der Funktion Nummer 9, Allocate Memory.

```
mov  dx,kbytes ; benötigte Größe
mov  ah,9
call xmsvec    ; anfordern
dec  ax
jnz  keinram
mov  handle,dx
```

Die meisten XMS-Funktionen geben in AX ein Erfolgsflag zurück. Konnte die Funktion ordnungsgemäß durchgeführt werden, ist AX Eins, bei einem Fehler ist AX Null, und der Fehlercode wird dann in BL hinterlegt.

Oft ist es hilfreich, zu wissen, wieviel Platz im Speicher noch zur Verfügung steht. Wenn man zehn verschiedene Blöcke über zusammen 100 KByte benötigt, und es sind nur noch 98 KByte frei, braucht man mit dem Zuweisen vielleicht gar nicht erst anzufangen. Die XMS-Funktion 8 übergibt im DX-Register den gesamten freien XMS-Speicher in KByte und in AX den größten, noch freien zusammenhängenden XMS-Block.

Mit dem Handle, daß beim Zuweisen in Funktion 9 zurückgegeben wird, kann man den Speicherblock ansprechen. Da der XMS-Speicher physikalisch oberhalb 1 MByte liegt, ist er im Real Mode nicht direkt zugänglich. Das Übertragen von Daten in den oder aus dem XMS-Block geschieht deswegen auch über einen Funktionsaufruf. Ihm werden die drei benötigten Parameter (Quelladresse, Zieladresse, Anzahl Bytes) in einer Struktur bei DS:SI übergeben (siehe Tabelle 2).

Die Handles für Quelle und Ziel sind entweder die von Funktion 9 übergebenen XMS-Handles oder Null. Bei einer Handle bezeichnet die dazugehörige Adresse den Offset in dem betreffenden Datenblock. Eine Null als Handle betrifft den Adreßraum unterhalb 1 MByte. In dem Fall werden im Adreßfeld normale Real-Mode Adreßpärchen (Segment:Offset) abgelegt, also ein 10-Bit Offset im unteren Word und das Segment im oberen.

```
mov  si,offset daten  ; Parameterfeld
mov  ax,laenge_L      ; Anzahl Bytes
mov  dx,laenge_H
add  ax,1
adc  dx,0
and  al,not 1         ; gerade Zahl erforderlich
mov  [si],ax
mov  2[si],dx
mov  word ptr 4[si],0 ; Real-Mode Handle
mov  word ptr 6[si],offset quelle
mov  word ptr 8[si],seg data
mov  ax,Handle        ; XMS-Handle
mov  10[si],ax
xor  ax,ax            ; Offset Null im XMS-Block
```

```
mov  12[si],ax
mov  14[si],ax
mov  ah,0Bh
call xmsvec           ; Daten ins XMS kopieren
dec  ax
jnz  fehler
```

Das Zurückholen der Daten aus dem XMS geht natürlich ganz genauso, man muß lediglich Quell- und Zielangaben vertauschen. Wichtig ist, vor Beendigung des Programms alle zugewiesenen Handles wieder freizugeben, weil sonst Teile des XMS-Speichers blockiert bleiben. Anders als bei Dateihandles, wird der XMS-Speicher, den ein Programm angefordert hat, beim Verlassen nicht automatisch wieder freigegeben.

```
mov  dx,handle
mov  ah,0Ah
call xmsvec
```

Auch diese Funktion gibt ein Flag in AX zurück. Allerdings kann diese Funktion wohl nur dann scheitern, wenn ein gravierendes Hardware-Problem vorliegt. Da das XMS im Protected Mode verwaltet wird, kann tatsächlich eine Störung im Zusammenhang mit dem Mode Switching oder dem Gate A20 auftreten. Die Verwaltungsinformationen selbst liegen allerdings beim Treiber im Hauptspeicher, daher sind Fehler dieser Art bei den Funktionen 8, 9 und 0Ah sehr unwahrscheinlich.

Anhand dieser vier Funktionen kann man große Datenblöcke ins XMS auslagern und später zurückladen (Swapping). Der Vollständigkeit halber soll aber auch eine Übersicht über die anderen XMS-Funktionen gegeben werden. Eine weitere, sehr wichtige Aufgabe besteht nämlich in der Kontrolle über Upper Memory und High Memory.

Upper Memory Block (UMB) ist der Bereich zwischen 640 KByte und 1 MByte (lineare Adresse des PCs von 0A0000h und 0FFFFFh). In diesen 384 KByte liegen Bildschirmspeicher und BIOS sowie alle Zusatzspeicher (RAM oder ROM) eventueller Erweiterungskarten. Wo sich dazwischen noch Lücken ergeben, läßt sich theoretisch RAM einblenden und im Real Mode nutzen. Natürlich müssen dazu Hardware-Voraussetzungen gegeben sein. Entweder RAM muß in diesem Bereich schon vorhanden sein, oder er muß sich in irgendeiner Form einblenden lassen – zum Beispiel bei einem 80386 durch das in den Prozessor integrierte Paging.

Der High-Memory-Bereich (HMA) sind knapp 64 KByte über dem ersten MByte des Computers. Auch dieser Teil des Speichers läßt sich im Real Mode ansprechen. Durch einen Offset von mindestens 16 greift der Prozessor im Segment 0FFFFh auf eine Adresse am unteren Rand des zweiten MBytes zu. Nicht jedes Programm kann allerdings dort laufen, denn zum einen darf das Programm selbst die unteren 16 Byte seines Speichersegments nicht benutzen, und zum anderen darf man mit diesen Adressen nicht so einfach rechnen, wie zum Beispiel Datensegment ist Stacksegment +16, wenn man 256 Byte Stack freihalten möchte.

Zum Zugriff auf das High Memory muß das Gate A20 offen sein. Das Gate kann die zwanzigste Adreßleitung des Prozessors abschalten, damit sich ein 80286 oder späterer Prozessor genauso verhält wie seine Vorgänger, die nur 1 MByte adressieren können. Ist das Gate geschlossen, greift der Prozessor anstatt ins High Memory auf das untere Segment zu. Dort liegt die Vektortabelle für die Interrupts und eine unbeabsichtigte Änderung kann so ziemlich jedes Ergebnis produzieren, das niemand gebrauchen kann.

Harald Piske/hf

Tabelle 1. Identifikationscodes beim Interrupt 2F	
Code	Beschreibung
0,1	PRINT.COM
2	PC LAN intern
5	ab DOS 3 Critical Error Handler
6	ASSIGN
8	Unterstützung für DRIVER.SYS
10	ab DOS 4 SHARE.EXE intern
11	Network Redirector bei DOS 4: IFSFUNC.EXE
12	DOS intern
13	Disk Interrupt Handler einstellen
14	NLSFUNC.COM
15	MSCDEX (Microsoft CD-ROM Extensions)
16	MS-Windows Funktionen DPMI (DOS Protected Mode Interface)
17	MS-Windows WinOldAp
19	DOS 4 DOS-Shell und COMMAND.COM
1A	ab DOS 4 ANSI.SYS intern
1B	DOS 4 XMA2EMS.SYS intern
40	OS/2 compatibility box
43	XMS
46	MS-Windows Version 3
54	TesSeRact RAM-resident program interface
7A	Novell Netware
80	Easy-Net
AD	DISPLAY. SYS
AD	KEYB.COM
AE	ab DOS 3 installierbarer Befehl (intern)
B0	GRAFTABL.COM
B4	IBM PC 3270 EMUL PROG v3
B7	APPEND
B8	Network LANtastic
B9	PC Network RECEIVER.COM
BF	PC LAN intern
CB	Communicating Applications Spec
CD	Intel Image Processing Interface
D7	Banyan VINES ab V4
DE	DESQview 2.26 Interface für externe Devices
FB	Borland international

Tabelle 2. Struktur für Datenübertragung des XMS		
Offset	Typ	Inhalt
0	DWord	Länge (muß gerade sein)
4	Word	Quelle, Handle
6	DWord	Quelle, Adresse
10	Word	Ziel, Handle
12	DWord	Ziel, Adresse

Tabelle 3. XMS-Funktionen	
0	XMS Versionsnummer holen
Ausgabe AX: Version in BCD BX: interne Revisionsnummer DX: Flag, ob HMA verfügbar	
1	HMA belegen
Eingabe DX: benötigte Größe in Bytes Das High Memory kann nur von einem Programm belegt werden, vor allem, weil es nur über Segment FFFF zu erreichen ist.	
2	HMA freigeben
3	A20 generell freigeben
Für Zugriff auf HMA zu verwenden.	
4	A20 generell sperren
Für Zugriff auf HMA zu verwenden.	
5	A20 temporär freigeben
Für direkten Zugriff ins Extended Memory zu verwenden.	
6	A20 temporär sperren
Wird tatsächlich nicht gesperrt, wenn generell freigegeben.	
7	A20 Zustand erfragen
Ausgabe BL: Fehlercode Wenn BL Null ist, trat kein Fehler auf. In dem Fall enthält AX ein Wert. AX: Flag, ob A20 freigegeben	
8	freien XMS-Speicher erfragen
Ausgabe AX: größter freier XMS-Block in KB DX: Summe freier XMS-Speicher	
9	XMS-Block belegen
Eingabe DX: benötigter Speicher in KB Ausgabe DX: Handle	
0Ah	XMS-Block freigeben
Eingabe DX: Handle	

0Bh	Speicherbereich kopieren
Eingabe DX: Handle Ausgabe DS:SI zeigt auf Kontrollstruktur (Tabelle 2)	
0Ch	XMS-Block sperren
Eingabe DX: Handle Ausgabe DX:BX 32-Bit lineare Adresse des Blocks	
0Dh	XMS-Block freigeben
Eingabe DX: Handle	
0Eh	Information über XMS-Block
Eingabe DX: Handle Ausgabe DX: Größe des Blocks in KByte BH: Anzahl Sperren auf den Block BL: Anzahl noch verfügbarer Handles	
0Fh	Größe eines Blocks ändern
Eingabe DX: Handle BX: neue Größe in KByte	
10h	Upper Memory Block belegen
Eingabe DX: benötigte Größe in 16-Byte-Paragraphs Ausgabe BX: Segment DX: tatsächliche Größe Im Fehlerfall DX: maximal verfügbare Größe	
11h	UMB freigeben
Eingabe BX: Segment	

Die Funktionen, bei denen AX nicht als Ausgaberegister beschrieben ist, geben eine Eins zurück, wenn die Operation erfolgreich durchgeführt werden konnte und eine Null, wenn ein Fehler aufgetreten ist. Bei einem Fehler wird der Fehlercode im BL-Register übergeben.

Tabelle 4. Fehlercodes des XMS-Treibers	
Code	Beschreibung
80	unbekannte Funktion
81	VDISK gefunden
82	Gate A20 Fehlfunktion
8E	unbestimmter Fehler des Treibers
8F	schwerwiegender Fehler, weitere Benutzung nicht empfehlenswert
90	kein HMA verfügbar
91	HMA belegt
92	DX kleiner als /HMAMIN= Parameter bei DEVICE=HIMEM.SYS
93	HMA nicht vergeben
94	A20 noch freigegeben
A0	XMS vollständig vergeben
A1	alle XMS Handles vergeben
A2	ungültige Handle
A3	ungültige Handle bei Quelladresse
A4	ungültige Quelladresse (Offset)
A5	ungültige Handle bei Zieladresse
A6	ungültige Zieladresse (Offset)
A7	ungültige Länge
A8	Kopierbereiche überlappen
A9	Parity-Fehler
AA	Block nicht gesperrt
AB	Block gesperrt
AC	zuviele Blocksperren
AD	Blocksperre fehlgeschlagen
B0	nur noch kleinere UMBs verfügbar
B1	kein UMB verfügbar
B2	ungültiges UMB Segment

Listing: xmstest.asm

```
stacks  segment para stack
        dw      80h dup (?)
stacks      ends

prog    segment word
        assume  cs:prog, ds:prog, ss:stacks

xmsvec  dd      ?
handle  dw      ?
movsiz  dd      SRCLEN
shand   dw      0
soffs   dw      source, seg prog
thand   dw      ?
toffs   dw      0, 0

source  db      'Das ist ein '
        db      'Speicherbereich, der '
        db      'ins XMS verschoben werden '
        db      ' soll.'
SRCLEN  equ     $ - source
target  db      SRCLEN dup (?)

entry:  sti
        cld
        mov     ax,4300h
        int     2Fh
        cmp     al,80h
        jne     exit
        mov     ax,cs
        mov     ds,ax
        mov     ax,4310h
        int     2Fh
        mov     word ptr xmsvec,bx
        mov     word ptr xmsvec + 2,es

        mov     dx,1
        mov     ah,9
        call    xmsvec
        dec     ax
        jnz     exit

        mov     handle,dx
        mov     thand,dx
        mov     si,offset movsiz
        mov     ah,0Bh
        call    xmsvec
        dec     ax
        jnz     abort

        xor     ax,ax
        mov     soffs,ax
        mov     soffs + 2,ax
        xchg    ax,thand
        mov     shand,ax
        mov     toffs,offset target
        mov     toffs + 2,seg prog
        mov     ah,0Bh
        call    xmsvec
        dec     ax
```

```
        jnz     abort

        mov     dx,offset target
        mov     cx,SRCLEN
        mov     bx,1
        mov     ah,40h
        int     21h

abort:  mov     dx,handle
        mov     ah,0Ah
        call    xmsvec

exit:   mov     ax,4C00h
        int     21h

prog    ends
end     entry
```

aus mc 12/91 – Seite 130-133