"Generosity will be rewarded."

43 Device Driver Programming

"*Device driver*" and "*Driver*" are interchangeably used in Programming world. Device drivers are the programs that control the functioning of peripherals. According to me, writing device driver is one of the easier things in programming. What all you need to know for device driver programming is good knowledge of hardware components. You may also need to know, how to access those hardware components through programs. In this chapter let's see how to write our own device driver.

43.1 Secrets

As I said earlier, device drivers are the programs that control the functioning of peripherals like keyboard, printer, etc. More specifically, they are the modules of an operating system.

MS DOS device drivers are with .SYS extensions. Since drivers drive peripheral devices, they get loaded into the memory when we bootup the system. So obviously, they remain resident in memory, but they are not considered as normal TSRs.

As drivers are the modules of an Operating System, one has to modify the OS whenever he adds new device to his system. Fortunately the *installable device drivers* technology available with MS DOS gives more flexibility to the user. It avoids direct operations or modifications of Operating System. The user can simply install a new device in a system, copy the driver files to boot disk and edit the system configuration file. Thus it clearly avoids complexity.

43.2 Types of MS DOS device drivers

1. Character device drivers
2. Block device drivers

43.2.1 Character device drivers

Character device drivers correspond to single byte. That is, these device drivers controls peripheral devices that perform input and output one character (i.e., one byte) at a time. The example for such devices are terminal, printer etc.

43.2.2 Block device drivers

Block device drivers correspond to block rather than byte. Even though they can be used with other devices, they are usually written to control random access storage devices such as floppy drives.

43.3 Writing our own device driver

Writing device driver is not a tough job as one may think. But nowadays device driver programming is not needed as the peripheral device vendors provide powerful drivers along with their products. So I avoid indepth explanation about the device driver programming. In a nutshell, device drivers are the COM (BIN) files with .SYS as their extensions. Our new device driver should be added with CONFIG.SYS file. Drivers also have headers. MS DOS 5+ versions support EXE file (renamed to .SYS extension) as drivers too. But it is a good practice to have COM file as drivers.

43.4 BUF160

BUF160 is a device driver for expanding the default keyboard buffer from 16 bytes to 160 bytes. 16 bytes restriction of default keyboard buffer might be strange to the people who are unnoticingly using keyboard buffer expansion program. If you don't use any keyboard buffer expansion utility and if your keyboard buffer is still 16 bytes in size (i.e., it can hold only 16 character when you work under command prompt), you may try this BUF160.

BUF160 is a good device driver. The recent version is 1.6a. Many people including **D J Delorie**, **David Kirschbaum** & **Robert M. Ryan** contributed to BUF160.

It works by installing itself as the standard keyboard buffer in the BIOS. It can only do this if it is in the same segment as the BIOS, so you are advised to install it as the first device driver. While it installs itself into the BIOS, it also installs a device driver called KBUFFER. Anything written to KBUFFER ends up in the keyboard buffer. I suggest you to look into the memory map found with Ralf Brown's Interrupt List for understanding BIOS data area.

43.4.1 Source code

Following is the source code of BUF160. It is written in assembly. As the code is more clear, I don't want to port it to Turbo C. I hope this real code will help you to understand the concepts behind device drivers. Refer the comment line for explanations.

```
      title BUF160
      page  58,132
;
; BUF160.ASM
;
;*****************************************************************
; Compilation flags
;*****************************************************************
```

```
TRANSFER    equ   1     ;Enables keyboard buffer transfer   v1.4
                        ;  procedure if enabled (1)         v1.4
USE286              equ   0     ;Should we use 286 (and later)
      v1.5
                        ;  CPU specific instructions?       v1.5
PRIVATESTACK        equ   1     ;Use own stack?               v1.6

PROGNAME    equ   'BUF160'
VERSION             equ   'v1.6a, 29 January 1992'

;*************************************************************************
; General equates
;*************************************************************************

BUFSIZE     equ   160         ;What is the size of the keyboard buffer
STACKSZ     equ   100h        ;What is the size of the private buffer
SUCCESS     equ   0100h
ERROR equ   8100h
BUSY  equ   0300h
CR    equ   13          ;Carriage Return
LF    equ   10          ;Line Feed
TERM  equ   '$'         ;DOS printing terminator character

;*************************************************************************
; Data structures
;*************************************************************************

dqq   struc
ofs   dw    ?
segw  dw    ?           ;changed from 'seg' to keep MASM 5.0 happy v1.4
dqq   ends

rqq   struc             ;Request header structure
len   db    ?           ;length of request block (bytes)
unit  db    ?           ;unit #
code  db    ?           ;driver command code
status      dw    ?           ;status return
q1    dd    ?           ;8 reserved bytes
q2    dd    ?
mdesc db    ?           ;donno
trans dd    ?
count dw    ?
rqq   ends

;*************************************************************************
; Pointers to BIOS data segment, v1.4
```

```
;************************************************************************
BIOS_DATA_SEG     equ 40H           ;MASM had prob using BIOS_DATA in
calculations,
                        ;   so this typeless constant introduced.  v1.6

BIOS_DATA   SEGMENT AT BIOS_DATA_SEG
      org   1AH
BUFFER_GET  dw    ?     ;org  1ah
BUFFER_PUT  dw    ?     ;org  1ch
      org   80H
BUFFER_START      dw    ?     ;org  80h
BUFFER_END  dw    ?     ;org  82h
BIOS_DATA   ENDS

;************************************************************************
; The actual program
;************************************************************************

Cseg  segment     byte
      assume      cs:Cseg,ds:Cseg,es:Cseg,ss:Cseg
      org   0                 ; no offset, it's a .SYS file
start equ   $                 ; define start=CS:0000

IF USE286                     ;                       v1.5
      .286
      %OUT Compiling 286 code ...
ELSE
      %OUT Compiling generic 8086 code ...
ENDIF
IF PRIVATESTACK
      %OUT Using private stack ...
ELSE
      %OUT Not using private stack ...
ENDIF
IF TRANSFER
      %OUT Including keyboard transfer code ...
ELSE
      %OUT Not including keyboard transfer code ...
ENDIF

      public      header
header      label near
      dd    -1                ;pointer to next device
      dw    8000h             ;type device
      dw    Strat             ;strategy entry point
      dw    Intr              ;interrupt entry point
      db    'KBUFFER '        ;device name
```

```
      public      req
req   dd    ?                ;store request header vector here

      public      queue_start,queue_end
queue_start dw    BUFSIZE dup (0)   ;our expanded keyboard buffer
queue_end   equ   $ - start         ;calculate offset as typeless
constant

IF PRIVATESTACK                     ;                       v1.6

stack_end   db   STACKSZ dup (0)    ;use our own private data stack
stack_start equ  $
oldss dw    0
oldsp dw    0
oldax dw    0

ENDIF

;*************************************************************************
; Strategy procedure
;     Save the pointer to the request header for Intr in the req area.
;     Enters with pointer in es:bx
;*************************************************************************

      public      Strat
Strat proc  far
      mov   cs:[req].ofs,bx
      mov   cs:[req].segw,es  ;                       v1.4
      ret
Strat endp

;*************************************************************************
; The main interrupt (driver)
;     This is the actual driver.  Processes the command contained in the
;     request header.    (Remember, req points to the request header.)
;*************************************************************************

      public      Intr
      ASSUME      ds:Cseg, es:NOTHING     ;                         v1.4
Intr  proc  far

IF PRIVATESTACK                     ;If using private stack, process
      mov   cs:oldax, ax            ;                       v1.6
      cli                     ; turn ints off
      mov   ax, ss
      mov   cs:oldss, ax
```

```
        mov   cs:oldsp, sp
        mov   sp, offset stack_start
        mov   ax, cs
        mov   ss, ax
        sti                     ; turn ints back on
        mov   ax, cs:oldax
    ENDIF

        push  ds                ;save everything in sight
        push  es
    IF USE286
        pusha                   ;                       v1.5
    ELSE
        push  ax
        push  bx
        push  cx
        push  dx
        push  di
        push  si
    ENDIF

        mov   ax,cs
        mov   ds,ax             ;DS=code segment

        les   bx,req                    ;point to request hdr         v1.4a
        mov   si,offset cmd_table       ;our function table
        mov   cl,es:[bx].code           ;get command
        xor   ch,ch             ;clear msb              v1.4
        shl   cx,1              ;*2 for word addresses
        add   si,cx             ;add to table base

        call  word ptr [si]             ;call our function            v1.4a
        les   bx,cs:req         ;get back request hdr vector
        mov   es:[bx].status,ax ;return status
    IF USE286
        popa                    ;                       v1.5
    ELSE
        pop   si                ;clean everything up
        pop   di
        pop   dx
        pop   cx
        pop   bx
        pop   ax
    ENDIF
        pop   es
        pop   ds
```

```
IF PRIVATESTACK
      mov   ax, cs:oldss            ;                       v1.6
      cli                     ; turn ints off
      mov   ss, ax
      mov   sp, cs:oldsp
      mov   ax, cs:oldax
      sti                     ; turn ints on
ENDIF
      ret

      public      cmd_table
cmd_table:                    ;command routing table
      dw    Cmd_Init          ;0=initialization (we do that)
      dw    Cmd_None          ;1=media check (always SUCCESS)
      dw    Cmd_None          ;2=build BIOS param block (ditto)
      dw    Cmd_None          ;3=IO control input (ditto)
      dw    Cmd_None          ;4=input from device (ditto)
      dw    Cmd_None          ;5=nondest input no-wait (ditto)
      dw    Cmd_None          ;6=input status (ditto)
      dw    Cmd_None          ;7=flush input queue (ditto)
      dw    Cmd_Output        ;8=output to device (we do that)
      dw    Cmd_Output        ;9=output with verify (same thing)
      dw    Cmd_Output_Status ;A=output status (we do that)
      dw    Cmd_None          ;B=flush output queue (always SUCCESS)
      dw    Cmd_None          ;C=IO control output (ditto)

;*************************************************************************
; Cmd_Output procedure
;*************************************************************************

      public      Cmd_Output
Cmd_Output  proc  near
      mov   ax,BIOS_DATA
      mov   ds,ax             ;BIOS data area
      ASSUME      ds:BIOS_DATA            ;keep MASM happy        v1.4

      mov   cx,es:[bx].count
      les   bx,es:[bx].trans
Output_Loop:
      mov   al,es:[bx]
      inc   bx
      cli
      mov   di,BUFFER_PUT           ;next free space        v1.4
      call  Buf_Wrap          ;add 2, check for wraparound
      cmp   di,BUFFER_GET           ;is the buffer full?          v1.4
      sti                     ;ints back on                 v1.4
      je    Output_Error            ;buffer is full, error        v1.4
```

```
        xchg  BUFFER_PUT,di           ;save the old, get the new    v1.4
        xor   ah,ah
        mov   [di],ax                 ;                         v1.4
        loop  Output_Loop

        public      Cmd_None          ;                         v1.4
Cmd_None:                       ;share this code        v1.4
        mov   ax,SUCCESS
        ret

Output_Error:
        mov   ax,ERROR
        ret
Cmd_Output  endp

;*************************************************************************
; Buf_Wrap procedure
;*************************************************************************

        public      Buf_Wrap
Buf_Wrap    proc  near
        inc   di
        inc   di
        cmp   di,BUFFER_END           ;hit end yet?                 v1.4
        je    Wrap                ;>=, wrap around        v1.4
        ret
Wrap:
        mov   di,BUFFER_START         ;force ptr to start           v1.4
        ret
Buf_Wrap    endp

;*************************************************************************
; Cmd_Output_Status procedure
;*************************************************************************

        public      Cmd_Output_Status
Cmd_Output_Status proc  near
        mov   ax,BIOS_DATA
        mov   ds,ax
        mov   di,BUFFER_PUT           ;ptr to next free space       v1.4
        call  Buf_Wrap          ;wraparound if necessary
        cmp   di,BUFFER_GET           ;same as next char to get?    v1.4
        jne   Cmd_None          ;ok, return SUCCESS           v1.4a
        mov   ax,BUSY
        ret
Cmd_Output_Status endp
```

```
      public      last_code
last_code   label near

;*****************************************************************
; Initialization (installation) procedure
;*****************************************************************

      public      Cmd_Init
Cmd_Init    proc  near
      mov   ax,cs
      mov   ds,ax
      mov   es,ax             ;                       v1.4a
      ASSUME      ds:Cseg,es:Cseg         ;                       v1.4a

; Is our new keyboard buffer within reach of the near pointers in
;BIOS_DATA?

      cmp   ax,(0fffh+BIOS_DATA_SEG-queue_end/10h);               v1.6
      ja    Init_Error        ;No, too far away

      mov   dx,offset banner  ;Yes, 'Buf160 loaded'
      mov   ah,9              ;DOS display msg
      int   21h
      mov   bx,0              ;Initialize size of buf       v1.5
      mov   cx,BIOS_DATA            ;PRESERVE THIS!               v1.4
      mov   ds,cx             ;BIOS data area
      ASSUME      ds:BIOS_DATA            ;                       v1.4

      cli                     ;turn off ints                v1.6a

IF    TRANSFER
      public      Transfer_Buffer
Transfer_Buffer:
      mov   si,BUFFER_GET           ;next key to read       v1.4
      mov   dx,BUFFER_PUT           ;next empty space       v1.4a

      mov   di,offset queue_start   ;gonna stuff here       v1.4a
      cld                     ;insure fwd             v1.4
Transfer_Loop:
      cmp   si,dx             ;hit empty yet?               v1.4a
      je    Transfer_Done           ;yep, transfer complete

      lodsw                   ;snarf the kbd word
      stosw                   ;stuff in OUR buffer          v1.4a
      inc   bx                ;increment counter            v1.5
      inc   bx                ;increment counter            v1.5
```

```
        cmp   si,BUFFER_END           ;hit kbd buffer's end yet?    v1.4
        jne   Transfer_Loop           ; nope, keep going
        mov   si,BUFFER_START         ;yep, wrap around to start    v1.4
        jmp   Transfer_Loop           ; and keep going

        public      Transfer_Done
    Transfer_Done:
    ENDIF

        mov   ax,cs             ;Code Segment
        sub   ax,cx             ; calculate difference b/w bios & this
    IF USE286
        shl   ax,4              ;                       v1.5
    ELSE
        shl   ax,1              ;remainder * 16 (paras to bytes)
        shl   ax,1
        shl   ax,1
        shl   ax,1
    ENDIF
        mov   cx,ax             ;CX = driver starting offset
        add   ax,offset queue_start   ;AX = queue_start offset
        mov   BUFFER_START,ax         ;init BIOS buffer pointers    v1.4
        mov   BUFFER_GET,ax           ;                         v1.4
        add   ax,bx             ;here'e next free space
        mov   BUFFER_PUT,ax           ;tell BIOS                v1.4

        mov   ax,cx             ;get back driver starting offset v1.4a
        add   ax,queue_end            ;code start + queue end       v1.4a
        mov   BUFFER_END,ax           ;tell BIOS                v1.4

        sti                     ;restore ints                 v1.6a

        les   bx,cs:[req]       ;complete driver header
        mov   es:[bx].trans.ofs,offset last_code  ;driver end
        jmp   short Stuff_Seg         ;share code, return success   v1.4a

        public      Init_Error
        ASSUME      ds:Cseg,es:Cseg         ;                     v1.4
    Init_Error:
        mov   dx,offset msg_err ;'Buf160 too far...'
        mov   ah,9              ;display msg
        int   21h

        les   bx,cs:[req]       ;complete driver header       v1.6

        IF    0                 ;not sure if it works.
        mov   es:[bx].trans.ofs,0
```

```
      ELSE
      mov   es:[bx].trans.ofs,offset last_code
      ENDIF

Stuff_Seg:                    ;                        v1.4a
      mov   es:[bx].trans.segw,cs   ;                      v1.4
      mov   ax,SUCCESS
      ret

Cmd_Init    endp

      public      banner, msg_err
banner      db    PROGNAME,' ',VERSION,' installed.',CR,LF        ;v1.4
      db    'Keyboard now has buffer of 160 characters.'
IF PRIVATESTACK
      db    '  Using private stack.'
ENDIF
      db    CR,LF,CR,LF,TERM

msg_err     db    PROGNAME,' too far from BIOS data area.'        ;v1.4
      db    CR,LF,CR,LF,TERM

Intr  endp

Cseg  ends

      end
```

43.4.2 Compiling BUF160

To compile with Turbo Assembler use:

```
tasm BUF160
tlink BUF160
exe2bin BUF160.exe BUF160.sys
```

To compile with Microsoft Assembler use:

```
masm BUF160
link BUF160
exe2bin BUF160.exe BUF160.sys
```

43.4.3 Installing BUF160

To install BUF160, insert the following line in your config.sys:

```
DEVICE=<path>BUF160.SYS
```

43.5 BGI Driver

As we know BGI drivers (one with .BGI extension) are used in Graphics Programming. We can also create our own BGI drivers. I omit the BGI driver programming here, because of the space constraint. More codes and documentations are found on CD  .