TITLE: Writing Device Drivers - Multiple Segments
IBM Developer Connection News Volume 7
by Steve Mastrianni

Occasionally I need to write a driver that occupies multiple code or multiple data segments. If you follow the guidelines presented here, you can avoid hours of frustrating development time.

The most fundamental change you will make is to use all far pointers and functions. This allows you reference functions and data across the 64KB segment boundaries, and requires the use of far calls to the DevHlp libraries. You can do this by modifying the DHCalls library on The Developer Connection Device Driver Kit for OS/2 or, if you are using the DevHlp library from Personal Systems Software, you can call (203) 693-0404 for an upgrade. Once you've converted the DevHlp library calls to FAR procedures, you must be sure to remove any NEAR references in your assembly language and C code. You'll need to change your compiler switches to use FAR calls as follows:

```
cl -c -Alfu -Gs /NT_TEXT -G2 -Zl -Zp yourfile.c
```

The /NT_TEXT parameter names the segment and allows you to order the segments at link time. As your first code segment approaches the 64KB boundary, you can name new code segments to a different name (for example, /NT_TEXT2), which will let you place the code in a second segment using the assembly language startup code and .DEF file. Of course, you must link with OS2286.LIB and the large model library, LLIBCEP.LIB, instead of SLIPCEP.LIB.

```
              .SEQ

 _DATA        segment    word public 'DATA'
 _DATA        ends

 CONST        segment    word public 'CONST'
 CONST        ends

 _BSS         segment    word public 'BSS'
 _BSS         ends

 FAR_BSS      segment    word public 'FAR_BSS'
 FAR_BSS      ends

 DGROUP       group FAR_BSS, CONST, _BSS, DATA, _DATA

 _TEXT        segment    word public 'CODE'
              assume     cs:_TEXT, ds:DGROUP, es:NOTHING, ss:NOTHING
              .286P
 ;
 _STRATEGY    proc  far
 __acrtused:             ; to satisfy EXTRN in C-generated modules
 ;
     .
     .
     ret
 ;
 _STRATEGY    endp

 _INT_HANDLER proc far

 _INT_HANDLER endp

 _TIM_HANDLER proc far
```

```
_TIM_HANDLER endp

_TEXT        ends

_TEXT2       segment word public 'CODE'
_TEXT2       ends

RMCode       segment word public 'CODE'
RMCode       ends

;  stick RM code in second segment

CGROUP group _TEXT2,RMCode

  end
```

Sample Code 1. Startup code

Next, you must initialize all of your global variables. If you don't, the linker will attempt to place them in a second data segment ahead of your first code segment. Initializing your variables forces them into the default data segment. Once you've made the changes, compile and link your device driver and examine the map file to be sure all of your variables are initialized and reside in the default data segment.

```
LIBRARY YOURLIB
PROTMODE
SEGMENTS
  _DATA       CLASS'DATA'    PRELOAD
  CONST       CLASS'CONST'   PRELOAD
  _BSS        CLASS'BSS'     PRELOAD
  FAR_BSS     CLASS'FAR_BSS' PRELOAD
  _TEXT       CLASS'CODE'    PRELOAD
  _TEXT2      CLASS'CODE'    PRELOAD IOPL
  RMCode      CLASS'CODE'    PRELOAD IOPL
```

Sample Code 2. .DEF file for extra code segment

You will have to change your .DEF file to order the segments correctly, and to mark the extra segments as IOPL. This will keep them around long enough to lock them down, which is your final step. In your driver's Init section, get the selector of any function which will reside in the upper segment, and call DevHlp Lock with that selector. Be sure to use the long-term lock.

```
fptr     = (PFUNCTION) SomeFunction;
codesel  = SELECTOROF(fptr);

// lock the second code segment down permanently

        if(LockSeg(
        codesel,                        // selector
        1,                              // lock long term
        0,                              // wait for seg lock
        (PLHANDLE) &lock_seg_han_code)) // handle returned
              return (RPDONE | RPERR | ERROR_GEN_FAILURE);

        return (RPDONE);
```

Sample Code 3. Locking the extra code segment

Your device header requires offsets to the strategy and IDC routines, so declare them as NEAR before the header, or use

another method of your choice.

Need more data space for those large buffers? No problem. Use the /ND option on the compiler command line to rename the data segments (for example, DATA2). Pick the name of a variable in the high data segment, get a pointer to it, extract the selector, and call LockSeg with the selector.

```
  cl -c -Alfu -Gs /NT_TEXT /ND_DATA2 -G2 -Zl -Zp yourfile.c
```

```
 LIBRARY YOURLIB
 PROTMODE
 SEGMENTS
   _DATA       CLASS'DATA'    PRELOAD
   CONST       CLASS'CONST'   PRELOAD
   _BSS        CLASS'BSS'     PRELOAD
   FAR_BSS     CLASS'FAR_BSS' PRELOAD
   _TEXT       CLASS'CODE'    PRELOAD
   _TEXT2      CLASS'CODE'    PRELOAD IOPL
   RMCode      CLASS'CODE'    PRELOAD IOPL
   _DATA2      CLASS'DATA'    PRELOAD IOPL
```

Sample Code 4. .DEF file for extra data segment

That's all there is to it. Well, almost--remember to mark your new data segment as IOPL.

Just because you've got all this new space, remember to use it sparingly!

Note: I write most of my device drivers in the small model, using Microsoft C 6.0.

Steve Mastrianni is an Industry Consultant specializing in device drivers and real-time applications for OS/2. The author of Writing OS/2 2.1 Device Drivers in C, Steve is regarded as one of the industry's leading experts in OS/2 and OS/2 device drivers. Steve can be reached on CompuServe@73354,746, or Internet at stevemas@vnet.ibm.com.