Troubles with Procedure Parameters

Recently I had some trouble understanding the problems with the MANORBOY program. This has to do with procedure parameters (that is: passing procedures or functions as parameters to other procedures or functions).

Remember:

In Pascal (in contrast to C, for example), it is possible to create procedures inside of other procedures. The internal procedures can access local variables of the surrounding procedures, as long as their definitions are not hidden (because of name collisions, for example).

To support this efficiently, the compiler has to keep track of the base addresses of the local storage of the surrounding procedures. It does this normally in a so-called display vector. The display vector contains the base addresses of the variables of every static level (1 to n).

Now, what happens, if a procedure at static level 6, for example, wants to call a procedure at static level 3 ?

In the general case, there is more than one procedure at static level 3 which can be called. First, there is the procedure which surrounds the level 6 procedure and which is at level 3. Calling it would be possible; this would be a recursive call. But there are also other procedures at level 3 that can be called. There is a level 2 procedure which surrounds the level 6 procedure, and it has more than one sub-procedures in the general case, and all of them can be called, because their definitions are in the scope of the level 2 procedure.

So, if the level 6 procedure calls one of these level 3 procedures, the display vector entry for level 3 has to be changed. The level 1 and 2 entries are still valid. The level 4 and 5 (and 6) entries are invalid, but not of interest at the moment. They remain unchanged.

Now, very important: the new called level 3 procedure has to save the old level 3 display value - and it has to restore it after completion.

If the new level 3 procedure calls a level 4 procedure, this level 4 procedure has to do the same thing: saving and restoring the display level of its static level.

If every procedure saves and restores the display vector entry of its static level, the display vector on return from the level 3 procedure to the level 6 procedure from the beginning will be unchanged.

I had some doubts, if that would work for external modules, too. Furthermore, I discovered, that Stanford Pascal only reserves 40 bytes for the display vector, and I wanted to know, what happens if I wrote a test program with more than 10 levels, for example.

The second answer first:

- Stanford Pascal is limited to 9 levels at the moment (which is not much IMO), and the compiler itself is at this limit. If you use one more, the compiler stops immediately with a fatal error message (P251)

- External modules have no problem; the entry procedure of the external module is at level 2, and it acts like an internal procedure at level 2. So you can have 8 levels of procedures in every module and call another external function (residing in a third module) from the lowest level procedure in the module. No problem so far. The one and only display vector is sufficient.

This all is related to the MANORBOY program as follows:

when a procedure or function is passed as a parameter to another procedure or function, its environment should be passed, too (that is: the variables that it can see). This is crucial for the MANORBOY program to work.

Stanford Pascal does this by copying the whole display vector together with the entry point of the procedure, when passing procedure parameters (it is only 40 bytes at the moment, after all). This way, when calling the procedure, which is passed as a parameter, the environment at the time of the function call (when the procedure parm was passed) is temporarily restored.

The only remaining problem for me now is:

this all works on the mainframe version of Stanford Pascal, but not on the PC version; the P-Code interpreter stores the display vector at another place, and so procedure parameters don't work at the moment.

BTW:

I added some comments on the ASMOUT file (which shows the generated 370 instructions), especially in the function prefix and suffix area. This way you can easily examine the generated code and see the saving and restoring of the display vector entries.

The following example shows a procedure at static level 8 (it was called LEVEL7 - the compiler starts counting at 1, which is the main program, but I started with the first procedure as PROCEDURE LEVEL1 ...)

```
-------------------- LOC  68 --------------------------------
     0000:  $PRV0008   ENT
     0000:  $PRV0008 ENT  P,8,L1    LEVEL7              ,
     0000:               T,T,F,F,2,8,,
                    BGN  $PRV0008,LEVEL7
@@ 0000: $PRV0008 CSECT
@@ 0000:       BC    15,52(0,15)
@@ 0004:       DC    AL1(29)
@@ 0005:       DC    C'$PRV0008 LEVEL7              '
@@ 0022:       DC    CL6'STPASC'    -- Compiler signature
@@ 0028:       DC    XL2'1803'      -- Compiler version
@@ 002A:       DC    AL2(0)         -- Stacksize
@@ 002C:       DC    AL2(2)         -- Debug-Level
@@ 002E:       DC    AL2(0)         -- Length of Proc
@@ 0030:       DC    A(0)           -- Static CSECT
@@ 0034: *                          -- save display level 8
@@ 0034:       L     0,112(12)
@@ 0038: *                          -- save registers and chain areas
@@ 0038:       STM   14,12,12(1)
@@ 003C:       ST    1,8(13)
@@ 0040:       ST    13,4(1)
@@ 0044:       LR    13,1
@@ 0046: *                          -- update current display
@@ 0046:       ST    13,112(12)
@@ 004A: *                          -- setup base registers
@@ 004A:       LR    10,15
@@ 004C:       LA    11,4092(10)
@@ 0050: *                          -- check for enough stack space
@@ 0050:       LA    1,248(1)
@@ 0054:       C     1,72(12)
@@ 0058:       BC    11,224(12)
-------------------- LOC  68 --------------------------------

...
```

```
-------------------- LOC  72 --------------------------------
     010E:           RET  P
@@ 010E: *                          -- clear stack frame using MVCs
@@ 010E:       MVI   80(13),129
@@ 0112:       MVC   81(167,13),80(13)
@@ 0118: *                          -- restore registers
@@ 0118:       LM    14,12,12(13)
@@ 011C:       L     13,4(13)
@@ 0120: *                          -- restore display level 8
@@ 0120:       ST    0,112(12)
@@ 0124: *                          -- clear the save area
@@ 0124:       MVC   0(80,1),80(1)
@@ 012A: *                          -- branch to return address
@@ 012A:       BCR   15,14
     012C:  L1       DEF  I,248
 PEND
```

you will see that the code loads the old value of the level 8 display entry into register 0 just before the STM. This way the display entry value is stored in the save area and reloaded at the end (into register 0 again), from where it is stored again into the display vector position 112(12).

Register 12 points to a global area provided by the runtime; the layout is as follows:

```
STACK    DS    18F          0000   BOTTOM OF RUNTIME STACK
CLOCK    EQU   STACK               CLOCK LOCATION
NEWPTR   DS    A            0072   PASCAL 'NEW' POINTER
HEAPLIM  DS    A            0076   UPPER LIMIT OF HEAP ( +1 )
*                                  ALSO POINTS TO DYN2STOR
DISPREGS DS    10F          0080   RUN TIME DISPLAY REGISTERS
DISPLAY  EQU   DISPREGS,*-DISPREGS
*
FL1      DS    D            0120   R/W  FIX/FLOAT CONVERSION HELPS
FL2      DS    D            0128   R ONLY
FL3      DS    D            0136   R/W
FL4      DS    D            0144   R ONLY
*
CHKSUBS  DS    0F                  ENTRY TO RUN TIME CHECK ROUTINES
INXCHK   DS    3F           0152   INDEX CHECK
RNGCHK   DS    3F           0164   SUBRANGE CHECK
PRMCHK   DS    3F           0176   PARAMETER VALUE CHECK
PTRCHK   DS    3F           0188   POINTER CHECK
PTACHK   DS    3F           0200   SET MEMBER CHECK
SETCHK   DS    3F           0212
STKCHK   DS    3F           0224
TRACER   DS    3F           0236
INPUT    DS    3F           0248
OUTPUT   DS    3F           0260
PRD      DS    3F           0272
PRR      DS    3F           0284
QRD      DS    3F           0296
QRR      DS    3F           0308
CLEARBUF DS    XL8          0320   BUFFER TO CLEAR ACTIVATION RECORDS
```

Because the Display vector is at fixed addresses known to the compiler and because it is followed by other known addresses, it is not easy to enhance the number of allowed levels.

Let's see ...

I hope you enjoyed this story of New Stanford Pascal development;
please send comments and suggestions to

berndoppolzer@yahoo.com

or

bernd.oppolzer@t-online.de