LN #D4

North American Response Centers

HP 3000 APPLICATION NOTE #31

Calling the CREATEPROCESS Intrinsic

Sample Programs

HEWLETT PACKARD

July 1, 1987
Document P/N 5958-5824R2727

HP 3000 APPLICATION NOTES *are published by the North American Response Centers twice a month and distributed with the Software Status Bulletin. These notes address topics, where the volume of calls received at the Centers indicates a need for addition to or consolidation of information available through HP support services. You may obtain previous notes (single copies only, please) by returning the attached Reader Comment Sheet listing their numbers.*

Note #	Published	Topic
4	4/15/86	*HP 3000 Printer Configuration Guide - Revised*
5	5/01/86	*MPE System Logfile Record Formats*
6	5/15/86	*HP 3000 Stack Operation*
7	6/01/86	*COBOL II/3000 Programs: Tracing Illegal Data*
8	6/15/86	*KSAM Topics: COBOL's Index I/O; File Data Integrity*
9	7/01/86	*Port Failures, Terminal Hangs, TERMDSM*
10	7/15/86	*Serial Printers - Configuration, Cabling, Muxes*
11	8/01/86	*System Configuration or System Table Related Errors*
12	8/15/86	*Pascal/3000 - Using Dynamic Variables*
13	9/01/86	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	9/15/86	*Laser Printers - A Software and Hardware Overview*
15	10/01/86	*FORTRAN Language Considerations - A Guide to Common Problems*
16	10/15/86	*IMAGE: Updating to TurboIMAGE & Improving Data Base Loads*
17	11/01/86	*Optimizing VPLUS Utilization*
18	11/15/86	*The Case of the Suspect Track for 792X Disc Drives*
19	12/01/86	*Stack Overflows: Causes & Cures for COBOL II Programs*
20	1/01/87	*Output Spooling*
21	1/15/87	*COBOLII and MPE Intrinsics*
22	2/15/87	*Asynchronous Modems*
23	3/01/87	*VFC Files*
24	3/15/87	*Private Volumes*
25	4/01/87	*TurboIMAGE: Transaction Logging*
26	4/15/87	*HP 2680A, 2688A Error Trailers*
27	5/01/87	*HPTrend: An Installation and Problem Solving Guide*
28	5/15/87	*The Startup State Configurator*
29	6/01/87	*A Programmer's Guide to VPLUS/3000*
30	6/15/87	*Disc Cache*

CALLING THE CREATEPROCESS INTRINSIC

With the introduction of the CREATEPROCESS intrinsic in MPE V/E it became possible to create and activate a son process that uses a $STDLIST and $STDIN different from those used by the father process. However, in order to use this capability it is necessary to assign a "pointer to a byte array" to an element of a logical array. This task is easy in SPL, which provides convenient syntax for indirect addressing, in COBOLII which provides a ".LOC." pseudo- intrinsic, and in Pascal, which provides a BADDRESS function that returns a pointer to a byte array. But suppose you have none of these languages on your system and you need to use this feature with FORTRAN/3000, FORTRAN/77, BUSINESS BASIC, TRANSACT or some language acquired from an independent software house? This note supplies a description of the necessary underlying principles and examples, written and tested at the Response Center, of code for doing this on every language in which it is possible on the HP3000. You will note the omission of RPG, which cannot call intrinsics and BASIC/3000 which cannot call any external that expects a parameter to be passed by value (which includes all OPTION VARIABLE externals; they expect a bit map to be passed by value).

POINTERS ARE ADDRESSES

A future application note will discuss the topic of addressing and parameter passing in further detail. This note will address only the material necessary for an understanding of calling the CREATEPROCESS intrinsic.

What is a pointer to a byte array? It is simply a 16 bit number that tells how many BYTES to count from the DB register to the first BYTE of storage allocated for a byte array. This is also known as the "byte address" of the byte array.

The HP3000 uses word addresses as well as byte addresses. Word addresses,too, are 16 bit numbers and they tell how many WORDS to count from DB to the first WORD of storage of the object in question.

SOME PRINCIPLES OF PARAMETER PASSING

There are two methods of parameter passing that are important in discussing how to obtain a pointer to a byte array; passing by value and passing by reference. Passing a parameter by value means that a copy of the data is received by the called subprogram. Passing by reference means that the called subprogram receives the byte address or word address at which the data are actually stored.

When calling MPE intrinsics a programmer usually need not be concerned with the details of how a parameter is passed *as long as the intrinsics are declared in the program as system intrinsics.* If they are declared as system intrinsics, the compiler (for most languages) will open SPLINTR.PUB.SYS to find out how the parameters are to be passed and then generate a calling sequence that passes them as expected. If the intrinsic is OPTION VARIABLE it will also generate the bit map to be passed by value that is required by these intrinsics. All of this is transparent to the programmer.

There are two cases where it is necessary to explicitly control how the parameters are passed to an MPE intrinsic. First is if you need to "fool" the intrinsic in order to achieve some special result. (In this case don't declare it as an intrinsic and use the language specific syntax to force the method of passing.) Second is if you are using TRANSACT. TRANCOMP will not automatically generate the correct parameter passing methods for MPE intrinsics, even if the intrinsics are declared by DEFINE(INTRINSIC) in the program.

HOW TO GET A POINTER TO A BYTE ARRAY

In those languages which do not provide a function to return a byte address or an explicit method of indirection in addressing, you must use the principles sketched above and seek a way get a byte address into a 16 bit word. If you can do this, you can assign the contents of this word to an element of the array required by CREATEPROCESS.

The ASCII and BINARY intrinsics provide you with the necessary tools to do this. The ASCII intrinsic expects to receive three parameters: WORD, a 16 bit word passed by value; BASE, a 16 bit word passed by value; STRING, a 16 bit word containing a byte address. The intent is to take the WORD parameter as a number which is to be converted into an ASCII numeric representation. The BASE parameter represents the base (octal, decimal, etc.) in which the ASCII representation is to appear. The STRING parameter points to a storage location where the ASCII string will be returned.

The BINARY intrinsic has two parameters: STRING, a 16 bit word containing a byte address; LENGTH, a 16 bit word containing a byte address. The intent is the reverse of the ASCII intrinsic. A numeric ASCII string, stored at the byte address contained in STRING, and of length LENGTH is to be converted to its corresponding binary value.

Both of these intrinsics are functions, and return a result to a specified variable of the calling routine.

Now comes the need to "fool" the ASCII intrinsic in order to achieve a special result. You have a program in which you have declared a byte array. You have initialized this array to contain the name of a file to be used as $STDIN for the son process to be created, being careful to terminate the file name with a carriage return as required. You need to assign a pointer to this byte array to an element of the array to pass to CREATEPROCESS. You can do this as follows:

1. DO NOT DECLARE THE ASCII INTRINSIC AS A SYSTEM INTRINSIC! Instead, declare it as a short integer external function written in SPL.

2. Call the ASCII intrinsic using the declared name of the byte array. Note that the byte array needs a pointer as its first parameter.

3. Use the BINARY intrinsic (it is O.K. to declare it as a system intrinsic) to convert the string returned by the ASCII intrinsic into a binary number. This number is then assigned into an element of the array to be passed to CREATEPROCESS.

Since you have not declared the ASCII intrinsic as a SYSTEM intrinsic, the compiler will use the default method of parameter passing (ie, pass by reference). In the first parameter, the ASCII intrinsic expects to receive a word by value. The word it actually received was passed by reference, but it has no way of knowing this. It simply assumes it contains a number to be converted and converts it into an ASCII string. The BASE parameter must be passed by value. The method of doing this is language specific and can be found in the appropriate language reference manual. It is also illustrated in the languages for which it is applicable in the following code examples.

USING THE EXAMPLE PROGRAMS

The rest of this note consists of:

1. a sample $STDIN file which contains instructions for TDP.PUB.SYS or EDITOR/3000
2. a sample TEXTFILE for TDP or EDITOR
3. sample programs which can be entered in an editor, compiled, prepped with PH capability and run

Sample $STDIN File

```
TEXT TEXTFILE
LIST ALL
EXIT
```

Sample TEXTFILE for TDP.PUB.SYS

```
This is a test to verify
the call of the CREATEPROCESS
intrinsic with $STDIN
redirected to another file.
```

SAMPLE PROGRAM FOR BUSINESS BASIC

```
 10 OPTION BASE 1
 20 INTRINSIC Createprocess,Binary
 30 EXTERNAL SPL SHORT INTEGER Ascii(Fname$, SHORT INTEGER VALUE Base,
    ST$)
 40 REM
 50 REM
 60 REM For process handling applications Business BASIC provides the
 70 REM SYSTEMRUN statement which eliminates the need to write
 80 REM this code.  It is included in this application note in order
 90 REM to document a method of getting a pointer to a byte
100 REM array in case it should be needed for some other kind
110 REM of application.
120 REM
130 REM
140 SHORT INTEGER Error,Pin,Itemnums(10),Items(10),Length
150 DIM Progname$[36],St$[10]
160 Progname$="TDP.PUB.SYS "
170 Stdin$="STDIN  "
180 REM
190 REM This filename must terminate with a carriage return (CHR$(13)).
200 REM
210 Stdin$[8]=CHR$(13)
220 Items(1)=1
230 Itemnums(1)=3
240 Items(2)=3
250 Itemnums(2)=10
260 Itemnums(3)=8
270 Itemnums(4)=0
280 Base=10
290 Length=FNCALL(Ascii(Stdin$,Base,St$))
300 Items(3)=FNCALL(Binary(St$,Length))
310 CALL Createprocess(Error,Pin,Progname$,Itemnums(*),Items(*))
320 IF Error<>0 THEN PRINT Error
330 END
```

SAMPLE PROGRAM FOR COBOLII

```
001000 IDENTIFICATION DIVISION.
001000 IDENTIFICATION DIVISION.
001100 PROGRAM-ID.  COBOL-EXAMPLE.
001200 AUTHOR.  RESPONSE CENTER.
001300 REMARKS.  This program is calls the CREATEPROCESS intrinsic
001400           with $STDIN redirected.
001500 DATA DIVISION.
001600 WORKING-STORAGE SECTION.
001700 01  CPERR                    PIC S9(4) COMP.
001800 01  PIN                      PIC S9(4) COMP.
001900 01  PROGNAME                 PIC X(36) VALUE "TDP.PUB.SYS ".
002000 01  STDIN.
002100     05  NAME                 PIC X(8) VALUE "STDIN    ".
002200     05  CR                   PIC S9(4) COMP VALUE 3328.
002300 01  ITEMNUMS.
002400     05  ITEMNUM OCCURS 4 TIMES PIC S9(4) COMP.
002500 01  ITEMS.
002600     05  ITEM OCCURS 3 TIMES PIC S9(4) COMP.
002900 PROCEDURE DIVISION.
003000 C-P.
003300     MOVE 3 TO ITEMNUM(1).
003400     MOVE 1 TO ITEM(1).
003700     MOVE 10 TO ITEMNUM(2).
003800     MOVE 3 TO ITEM(2).
004100     MOVE 8 TO ITEMNUM(3).
004200     CALL INTRINSIC ".LOC." USING @STDIN GIVING ITEM(3).
004300     MOVE 0 TO ITEMNUM(4).
004400     CALL INTRINSIC "CREATEPROCESS" USING CPERR, PIN, PROGNAME,
004500           ITEMNUMS, ITEMS.
004600     IF CPERR NOT EQUAL TO 0 THEN DISPLAY CPERR.
004700     STOP RUN.
```

SAMPLE PROGRAM FOR FORTRAN/3000

```
$CONTROL USLINIT
      PROGRAM FTN3000
      SYSTEM INTRINSIC CREATEPROCESS
      CHARACTER*36 PROGNAME, FILENAME
      INTEGER ERROR, PIN, ITEMNUMS(12)
      LOGICAL ITEMS(12), SUSP, LOC
      PROGNAME = 'TDP.PUB.SYS '
      FILENAME = 'STDIN '
C
C     THIS FILENAME MUST TERMINATE WITH A CARRIAGE RETURN
C
      FILENAME[7:1] = %15C
      ITEMNUMS(1) = 3
      ITEMS(1) = %1L
      ITEMNUMS(2) = 10
      ITEMS(2)=%3L
      ITEMNUMS(3)=8
      ITEMS(3)=LOC(FILENAME)
      ITEMNUMS(4)=0
      CALL CREATEPROCESS(ERROR,PIN,PROGNAME,ITEMNUMS,ITEMS)
      IF (.CC.) 10,20,10
10    CONTINUE
      WRITE(6,100) ERROR
20    CONTINUE
      STOP
100   FORMAT(' CREATEPROCESS FAILURE - ERROR NUMBER: ',I3)
      END
C
C
      LOGICAL FUNCTION LOC(FILENAME)
      SYSTEM INTRINSIC BINARY
      INTEGER ASCII,LENGTH
      CHARACTER*6 STR
      CHARACTER*36 FILENAME
      LENGTH=ASCII(FILENAME,\10\,STR)
      LOC=BINARY(STR,LENGTH)
      RETURN
      END
```

SAMPLE PROGRAM FOR FORTRAN 77

```
$STANDARD_LEVEL SYSTEM
$USLINIT
$SHORT
      PROGRAM FTN77
      IMPLICIT NONE
      SYSTEM INTRINSIC CREATEPROCESS
      CHARACTER PROGNAME*36,STDIN*36
      INTEGER*2 ERROR,PIN,ITEMNUMS(10),ITEMS(10),LOC
      STDIN = 'STDIN          '
      PROGNAME = 'TDP.PUB.SYS     '
C
C     THIS FILE NAME MUST TERMINATE WITH A CARRIAGE RETURN,
C     WHICH IS ASCII CHARACTER 13 (DECIMAL).
C
      STDIN(7:7) = CHAR(13)
      ITEMNUMS(1) = 3
      ITEMS(1) = 1
      ITEMNUMS(2) = 10
      ITEMS(2) = 3
      ITEMNUMS(3) = 8
      ITEMS(3) = LOC(STDIN)
      ITEMNUMS(4) = 0
      CALL CREATEPROCESS(ERROR,PIN,PROGNAME,ITEMNUMS,ITEMS)
      IF (ERROR .NE. 0) THEN
          WRITE(6,100) ERROR
      ENDIF
      STOP
100   FORMAT(' CREATEPROCESS ERROR NUMBER =',I3)
      END
C
C
$SHORT
      INTEGER*2 FUNCTION LOC(NAME)
$ALIAS ASCII SPL (%REF, %VAL, %REF)
      IMPLICIT NONE
      SYSTEM INTRINSIC BINARY
      CHARACTER*36 NAME
      CHARACTER*6 STRING
      INTEGER*2 LENGTH,ASCII
      LENGTH = ASCII(NAME,10,STRING)
      LOC = BINARY(STRING,LENGTH)
      RETURN
      END
```

SAMPLE PROGRAM FOR PASCAL

```
$STANDARD_LEVEL 'HP3000'$
$USLINIT$

PROGRAM PASCAL(INPUT,OUTPUT);
TYPE
     SMALL = -32768..32767;
     SMALARAY = ARRAY [1..5] OF SMALL;
     BYTES = PACKED ARRAY[1..36] OF CHAR;

     INUMS = RECORD
               FLAGS: SMALL;
               SUSP: SMALL;
               STDIN:SMALL;
             END;

VAR ERROR,                  {  FIRST PARAMETER OF INTRINSIC  }
    PIN: SMALL;             {  SECOND PARAMETER  }
    PROGNAME: BYTES;        {  THIRD PARAMETER  }
    ITEMNUMS: SMALARAY;     {  FOURTH PARAMETER  }
    ITEMS: INUMS;           {  FIFTH PARAMETER  }
    STDIN: BYTES;


PROCEDURE CREATEPROCESS; INTRINSIC;
BEGIN

  STDIN := 'STDIN ';
  STDIN[7] := CHR(13);
  PROGNAME := 'TDP.PUB.SYS ';
  ITEMNUMS[1] := 3;
  ITEMNUMS[2] := 10;
  ITEMNUMS[3] := 8;
  ITEMNUMS[4] := 0;
  ITEMS.FLAGS := 1;
  ITEMS.SUSP := 3;
  ITEMS.STDIN := BADDRESS(STDIN);
  CREATEPROCESS(ERROR,PIN,PROGNAME,ITEMNUMS,ITEMS);
END.
```

SAMPLE PROGRAM FOR SPL

```
$CONTROL USLINIT
BEGIN

   INTEGER ERROR;
   INTEGER PIN;
   BYTE ARRAY PROGNAME(0:15) := "TDP.PUB.SYS ";
   INTEGER ARRAY ITEMNUMS(0:8);
   ARRAY ITEMS(0:8);
   BYTE ARRAY INPUTFILE(0:8) := "STDIN ";
   BYTE ARRAY ERRNUM(0:8);
   ARRAY LBUF(*)=ERRNUM(0);

   INTRINSIC CREATEPROCESS,ASCII,PRINT;

   INPUTFILE(6) := %15;
   ITEMS(0) := 1;
   ITEMNUMS(0) := 3;
   ITEMS(1) := 3;
   ITEMNUMS(1) := 10;
   ITEMS(2) := @INPUTFILE;
   ITEMNUMS(2) := 8;
   ITEMNUMS(3) := 0;

   CREATEPROCESS(ERROR, PIN, PROGNAME, ITEMNUMS, ITEMS);
   IF ERROR <> 0 THEN
      BEGIN
        ASCII(ERROR,10,ERRNUM);
        PRINT(LBUF,-8,%40);
      END;

END.
```

SAMPLE PROGRAM FOR TRANSACT

```
SYSTEM TRNSCT;

DEFINE(ITEM) PROGNAME        X(36):
             STDIN           X(8):
             CCTL            I(4)=STDIN(8):
             ERROR           I(4):
             PIN             I(4):
             ITEMNUMS       4I(4):
             ITEMNUM            I(4)=ITEMNUMS(1):
             ITEMS          4I(4):
             ITEM           I(4)=ITEMS(1):
             STRING          X(6):
             BASE            I(4):
             LENGTH          I(4):
             MAP             I(4);

LIST  PROGNAME:
      STDIN:
      ERROR:
      PIN:
      ITEMNUMS:
      ITEMS:
      STRING:
      BASE:
      LENGTH:
      MAP;

  MOVE (PROGNAME) = "TDP.PUB.SYS  ";
  MOVE (STDIN) = "STDIN ";
  LET (CCTL) = 13*256;

  LET OFFSET(ITEMNUM) = 0;
  LET OFFSET(ITEM) = 0;
  LET (ITEMNUM) = 3;
  LET (ITEM) = 1;

  LET OFFSET(ITEMNUM) = 2;
  LET OFFSET(ITEM) = 2;
  LET (ITEMNUM) = 10;
  LET (ITEM) = 3;


  LET OFFSET(ITEMNUM) = 4;
  LET OFFSET(ITEM) = 4;
  LET (ITEMNUM) = 8;
  LET (BASE) = 10;
  PROC ASCII( %(STDIN), #(BASE) , %(STRING), &(LENGTH) );
  PROC BINARY( %(STRING), #(LENGTH), &(ITEM) );
  LET OFFSET(ITEMNUM) = 6;
  LET (ITEMNUM) = 0;
```

```
<<  Note that in TRANSACT you can DEFINE(INTRINSIC) CREATEPROCESS,
    but this does not have the effect of opening the SPLINTR file
    at compile time and generating the calling sequences in
    accordance with the declarations in it for you, as it will in
    the other languages.  You must explicitly pass the parameters
    as the external procedure expects to receive them, including
    commas for parameters omitted (two for double word parameters
    passed by value if omitted).  In the case of OPTION VARIABLE
    externals such as CREATREPROCESS this also means that you must
    pass a bit map by value.  See Chapter 7 of the SPL Reference
    Manual for detailed discussion of this matter.      >>


  LET (MAP) = 31;
  PROC CREATEPROCESS( (ERROR), (PIN), %(PROGNAME),
                      (ITEMNUMS), (ITEMS), #(MAP) );
  IF (ERROR) = 0 THEN GO TO STOP
  ELSE DISPLAY "CREATEPROCESS ERROR NUMBER = ":ERROR
STOP:
  EXIT;
END;
```