Process Sensing and Control

Nancy Kolitz
Hewlett-Packard Company
Cupertino, California

I. INTRODUCTION

Various MPE intrinsics on the HP3000 allow a user to create processes, to obtain information about them, and to control them. This paper will describe the process sensing and control capabilities available to a user, through illustrations and examples. The paper will also introduce a new intrinsic, PROCINFO, currently being developed by the MPE lab.

II. WHAT IS A PROCESS?

All user programs run as processes under MPE. A process is the unique execution of a program by a particular user at a particular time, and is the entity within MPE which can accomplish work. A process is also the mechanism which allows system resources to be shared and a user's code to be executed. Each process consists of a private data stack and code segments, shared by all processes executing the same program.

As the system is brought up, the Progenitor (PROGEN) is the first process created by MPE. One of the various system processes that Progen creates is the User Controller Process (UCOP), which creates a User Main Process (UMAIN) as a session or job logs on. Then when a user (or job) runs a program, a User Son of Main (USONM) process is created. If other processes are subsequently created from this program, User processes are established. (See Figure 1.)

A process will be in one of two states once it has been created: Wait or Active. If it is in a wait state, it is waiting for some event (I/O, RIN acquistion, etc.) to occur before it will run again. If it is in an active state, the process is running or ready to run.

A standard MPE user has no control over his processes. The operating system creates, controls, and kills the processes for the user. However, if the user's program has Process Handling (PH) capability, it can, to some degree, manage its own processes. In fact, it can even control processes in its family tree.

```
              PROGEN
                |
               UCOP
              / | \
             /  |  \
            /   |   \
           /    |    \
    UMAIN     UMAIN     UMAIN
                |
              USONM
              / | \
             /  |  \
        USER   USER   USER .....
```

Figure 1

III. PROCESS CREATION

In MPE, there are two intrinsics that a user with PH capability can use to create a process: CREATE and CREATEPROCESS.

The intrinsic CREATE will load a program into virtual memory, create a new process, initialize the stack's data segment, schedule the process to run, and return the process identification (PIN) number to the process requesting the creation. Once the process is established, it will have to be activated by the creating process. The command syntax is:

```
           BA        BA        I   IV    LV      IV      IV
CREATE(progname,entryname,pin,param,flags,stacksize,dlsize,
         IV        LV        IV      O-V
         maxdata,priorityclass,rank).
```

The last parameter RANK (in the CREATE intrinsic) is not used by the intrinsic and is only there for compatibility with previous versions of MPE.

CREATEPROCESS is the other intrinsic that can be used for creating processes. Its format is:

```
                          I    I     BA        IA        LA       O-V
        CREATEPROCESS(error,pin,progname,itemnums,items).
```

The parameter ITEMNUMS indicates the options to be applied in creating the new process, and the parameter ITEMS provides the necessary information to be used for each option specified in ITEMNUMS.

With CREATEPROCESS, a son may be activated immediately upon creation or may be activated as a process is with CREATE (via the ACTIVATE intrinsic). A user may also specify an entry point into a program, define $STDIN and $STDLIST to be any file other than the defaults (the defaults are the creating father's $STDIN and $STDLIST), control stack size, and control the process' priority queue. Some of these can also be done with CREATE.

The example that follows illustrates the intrinsic CREATEPROCESS. It will create a process, indicate that the father should be awakened upon completion of the son, and then activate the new process.

****

```
BEGIN
          <<CREATEPROCESS example>>

INTEGER ERROR, PIN;
BYTE ARRAY EXAMPLE(0:7) := "EXAMPLE ";
INTEGER ARRAY OPTNUMS(0:10);
LOGICAL ARRAY OPTIONS(0:10);

intrinsic CREATEPROCESS,TERMINATE;

          <<set up options>>
OPTNUMS(0) := 3;           OPTIONS(0) := 1;
OPTNUMS(1) := 10;          OPTIONS(1) := 3;
OPTNUMS(2) := 0;     <<terminator>>

CREATEPROCESS(ERROR,PIN,EXAMPLE,OPTNUMS,OPTIONS);

if <> then TERMINATE;
```

****

When calling MPE intrinsics, a good programming practice is to check the condition code returned, and the error parameter, if one is used. In the case of CREATEPROCESS, if the condition code is less than zero the process was created, but some event occurred to cause the operating system to give a warning to the creator. If the condition code is greater than zero, an error has occurred and the process was not created. If the error occurred because of a file system problem (error number returned is 6), a user can use the intrinsic FCHECK with a parameter of zero to obtain more information as to why the process creation failed.

IV. SENSING PROCESSES

Each process in MPE has a large amount of information about it that can be useful, providing a process can access it. There are various intrinsics that will return this information once a process has been created. However, a program must have PH capability to use these intrinsics.

A user may determine the PIN number of the process that created it via the intrinsic FATHER. Its syntax is:

```
       I
       pin := FATHER.
```

Once again, a programmer should check the condition code that was returned. In this case, it will tell what type of process the father is. Through specific codes, it will specify whether the father is a system process, a user main process, or a user process.

To obtain the PIN number of any of his son processes, a program may use the intrinsic GETPROCID. The command is:

```
 I                IV
pin := GETPROCID(numson).
```

The parameter NUMSON is a integer value that specifies which son a father wants to know the PIN number. For example, if a father has created three sons and wants to know the PIN number of the second son, he will supply GETPROCID with a parameter of two.

The WHO intrinsic provides the access mode and attributes of the user running a program. The file access capabilities (save file (SF), ability to access nonsharable devices (ND), etc.), user attributes (OP, SM, etc), and user capabilities (PH, DS, etc) can be obtained. Also information about the user, his logon group name and account name, his home group, and the logical device of his input file may be returned. The command syntax for WHO is:

```
      L      D          D       BA    BA     BA    BA     L     O-V
WHO(mode,capability,lattr,usern,groupn,acctn,homen,termn).
```

The intrinsic GETORIGIN will return, to a reactivated process, the origin of its activation. The value returned will specify if the PIN was activated from a suspended state by a father, a son, or another source (interrupt or timer). GETORIGIN looks like:

```
   I
source := GETORIGIN.
```

Other information about a son or father may be obtained from the intrinsic GETPROCINFO. Its format is:

```
   D                     IV
statinfo := GETPROCINFO(pin).
```

A double word is passed back giving the process' priority number and priority queue, its activity state (active or waiting), its suspension condition and source of next activation, and the origin of its last activation. The process number, passed as a parameter, specifies which process you want information about. If PIN=0, then information is returned for the father; otherwise, the information is for a son process.

A new intrinsic currently under development in the MPE lab is called PROCINFO. This intrinsic returns general information about processes that is currently unavailable, unless you have privileged mode capability. It should simplify some of the uses of process related intrinsics because a large amount of information may be retrieved in one call to PROCINFO. Its command syntax is:

```
            I      I      IV       I      BA          O-V
PROCINFO(error1,error2,pin[,itemnum1,item1]
                          [,itemnum2,item2]
                          [,itemnum3,item3]
                          [,itemnum4,item4]
                          [,itemnum5,item5]
                          [,itemnum6,item6]).
```

This intrinsic is formatted similar to FFILEINFO in order to maintain ease of use and extensibility. It can return to a program the process number of the process itself, its father, all its sons, and all its descendants. It can also supply information about the number of descendants and generations in a family tree, the name of a program that a specified process is running, the process' state, and the process' priority number.

The first error word is used to return the type of error incurred when executing the intrinsic. The second error word returns the index of the offending item number. The program name is returned in a byte array that is a minimum of twenty eight bytes long. It is in the format of <filename.group.account>.

The following example will help to illustrate the use of the PROCINFO intrinsic:

```
***************************************************

BEGIN  <<procinfo example>>

  INTEGER ERROR1, ERROR2, PIN;

  BYTE ARRAY ITEMVAL1 (0:1),
             ITEMVAL2 (0:1),
             ITEMVAL3 (0:1),
             ITEMVAL4 (0:1),
             ITEMVAL5 (0:1);

  INTEGER ITEMNUM1,ITEMNUM2,ITEMNUM3,ITEMNUM4,ITEMNUM5;


  INTRINSIC PROCINFO;

  PIN := 0;             <<seek information about ourselves>>
  ITEMNUM1 := 1;     <<request our pin #>>
  ITEMNUM2 := 3;     <<how many sons we have>>
  ITEMNUM3 := 4;     <<how many descendants we have>>
  ITEMNUM4 := 2;     <<pin number of our father>>
  ITEMNUM5 := 5;     <<how many generations we have>>

  PROCINFO (ERROR1, ERROR2, PIN, ITEMNUM1, ITEMVAL1,
                                 ITEMNUM2, ITEMVAL2,
                                 ITEMNUM3, ITEMVAL3,
                                 ITEMNUM4, ITEMVAL4,
                                 ITEMNUM5, ITEMVAL5);

  IF <> THEN GO PROCERROR;


  .  .  .  .

  PROCERROR:
    <<print message and error number>>
    RETURN;

END.  <<procinfo example>>


***************************************************
```

If the previous program was executed by pin 45 in the process tree of figure 2, the following information would be returned:

```
item number  information
     1            45
     3            2
     4            5
     2            12
     5            3
```

```
               Pin 12
              /    \
             /      \
            /        \
        Pin 23     Pin 45
                   /    \
                  /      \
                 /        \
            Pin 22      Pin 34
             /  |  \
            /   |   \
           /    |    \
          /     |     \
         /      |      \
     Pin 38  Pin 21  Pin 30
```

Figure 2

****

V. CONTROLLING PROCESSES

Once a program has created a process, it can control its activity. As mentioned before, it can activate its sons via the intrinsic ACTIVATE. However, only a father can activate a newly created process. ACTIVATE is called with the following parameters:

```
            IV    IV       O-V
ACTIVATE (pin, susp).
```

The process' pin number is required, but the susp parameter is not. If susp is provided and not equal to zero, then the calling process will be suspended and the specified process will be activated. Otherwise, the father process continues to run and the activated process becomes ready to run. The activated process will execute when the dispatcher selects it as the highest priority process to launch.

A process may also suspend itself. Via the intrinsic SUSPEND, a process may place itself in a wait state and state its expected origin of activation. The intrinsic calling sequence is:

```
          LV   IV        O-V
SUSPEND (susp,rin).
```

The RIN parameter is the Resource Identification Number that will be locked for the process until it suspends again. The RIN allows a process to have exclusive access to a particular resource at a particular time. This is one way to synchronize processes and their resources running under the same job.

One other process control intrinsic is GETPRIORITY. When a process is created, it is given the same priority as its father. This intrinsic allows a program to change its own process' priority or that of a son. The process cannot, however, request a priority outside of its allowable priority class. GETPRIORITY is called as follows:

```
             IV        LV          IV        O-V
GETPRIORITY(pin,priorityclass,rank).
```

The priorityclass parameter is a 16-bit word that contains two ASCII characters. Depending on the priority queue desired, the parameter is "AS," "BS," "CS," "DS," or "ES." (If a user has privileged mode, he can supply an absolute number for the priority parameter instead of the ASCII characters. It is done by supplying the parameter "xA where "x" is an integer value and "A" is the ASCII character A.) The rank parameter, once again, is not used except for compatibility with old versions of MPE.

The last two intrinsics to be discussed are used for process termination. When a process is terminated, it must return all the system resources that it is holding, stop its sons from running and start their termination sequence, and then request that its father take away its stack. The two intrinsics used for termination are:

```
     IV
KILL(pin)      and         TERMINATE.
```

The parameter in the KILL intrinsic is the pin number of the process' son that it wants deleted. TERMINATE can only be used for the calling process.

The following is another example using these various intrinsics. This example illustrates the CREATE, ACTIVATE, GETPRIORITY and TERMINATE intrinsics:

```
*************************************************

BEGIN

ARRAY NAME(0:15) := "EXAMPLE.PUB.SYS ";
BYTE ARRAY BNAME (*) = NAME;
INTEGER PIN;

INTRINSIC CREATE,ACTIVATE,TERMINATE,GETPRIORITY;

CREATE(BNAME,,PIN,,1);        <<create the new process. reactivate >>
                              <<the father when this one finishes.>>

IF <> THEN TERMINATE;         <<kill process because of error in>>
                              <<creation sequence              >>

ACTIVATE(PIN,2);              <<activate process and then reactivate>>
                              <<calling process by the son         >>

IF <> THEN TERMINATE;         <<process not activated due to error>>

GETPRIORITY(PIN,"DS");        <<change priority of son process>>

IF <> THEN TERMINATE;         <<new priority not granted>>

END.

*************************************************
```

VI. SUMMARY

This paper has summarized various intrinsics that can be used to create new processes, obtain information about them, control them, and then terminate them. A new intrinsic, PROCINFO, was also introduced which can provide the user with more information about processes without requiring privileged mode capability. MPE is a process oriented operating system, and with a better knowledge and understanding of how processes operate, a user can enhance his applications and their performance on the HP3000.