IDAT: For Dump Analysis and More...

Neil Ferguson
Boeing Computer Services
P.O. Box 7730
M/S K79-30
Wichita, KS 67277-7730

Introduction

IDAT is the name given to the Interactive Dump Analysis Tool. This utility is currently used, mainly by Response Center engineers, to analyze memory dumps. It functions, as its name implies, in an interactive manner, which differs from other methods of analyzing dumps where a paper listing is the only output available. This paper is intended to give users an overview of how the program works, and how you, as an end-user, can use this utility to do some *rudimentary* dump analysis on your own. If you are already skilled enough to do competent dump analysis, then this paper may not present new information for you or develop those skills any further. If, however, you are a skilled programmer, a system supervisor or a system manager, and have always wondered what an engineer looks for when reading a dump, then this paper will probably provide you with some useful information.

Your system has just failed with a SF310. Now what? Do you have an adequate plan to deal with the recovery? Will you have to wait for minutes or even hours for the system to be brought back up? Do you need to take a memory dump? If so, then when you get one, what do you do with the tape? Do you need to phone the Response Center? If you do, what can you be doing proactively while you wait for their return call? Do you know how to perform a rudimentary dump analysis?

These are some of the questions which undoubtedly have been considered by many a System Manager, Operations Manager or even the late night operator. This paper will attempt to enlighten you about one tool which can be used to speed up the process of analyzing a memory dump. This paper *will not* attempt to train you to read *any* memory dump and determine *exactly* what went wrong. That skill (or more aptly, that art) should be left to the personnel at the Response Center. The Response Centers are staffed with specialists who do dump analysis every day. Their knowledge of MPE and their ability to analyze dumps will probably be more honed to perfection than yours.

However, there are a few things which you can do which will speed the process along. After we cover those things, we can consider using IDAT to seek out additional information about your system, while it is still up an running. But first, let's return to the above scenario: the system has failed with a SF310.

Having a total of six years working with the HP3000, my first recommendation is that you always, *always* take a memory dump. Even if you are the only person on the machine and you are testing a new privileged-mode program, I would still take a dump. A Series 70 with 12 Meg of memory will only take about ten minutes to dump, depending on how much virtual memory may be copied to tape. Failure to take a dump could delay you in your efforts to prevent a recurrence of the problem. There is no way to predict whether this failure is a lone freak occurrence or the first in a string of failures which may happen over the course of many weeks or even months. For multiple failures, sometimes the only way to determine the cause is to perform comparisons between the dumps to look for common characteristics.

First, let us make sure that we get a *valid* memory dump. In order to start the dump procedure, at the system prompt, you would enter: DUMP RETURN. Follow the directions that will appear on the system console. The Software Dump Facility (SDF) will then take over the operation of the machine and will create the dump tape. If this process fails, then you may have a more serious problem and you should phone the Response Center for further assistance. The Response Center *may* be able to help you get a valid dump, but don't expect much, especially if the SDF banner has been displayed on your console terminal. This means that the contents of low memory have been copied to disc and that the SDF software has been loaded in its place. If you try to perform another dump, the *original* contents of low memory will be lost during the loading of the *second* copy of the SDF.

Assuming the SDF performs successfully, the dump tape will contain a copy of memory as it existed when you entered DUMP at the system prompt. If you are running an MPE revision of U-Mit or beyond, then you should also have a copy of all data segments which were swapped out to disc at the time of the failure (these segments make up virtual memory) and copies of several system files. Console messages will inform you about the contents of the tape.

After the dump tape is created, the SDF will issue a WARMSTART command. This does not mean that a WARMSTART is being initiated! This command is internal to the SDF software, and functions as though you switched on the machine and entered: START RETURN. After a short pause, you will see the familiar banner requesting which option you wish to use to start the machine. At this point you will have the option to choose a WARM or COOL start. Which option you choose will depend on your particular situation.

After taking the memory dump and restarting MPE, each site will have its own method for system recovery, depending on a wide variety of configuration options and application dependencies. This discussion will not attempt to elaborate on these variations, but instead will focus on what you can do after normal startup procedures have been completed.

You should now have the memory dump tape at hand. If your site can recover without dismounting the memory dump tape, you are that much further ahead of other sites, and you can now make the tape online again, in preparation for use by IDAT. Otherwise, mount the memory dump tape again and make it ready. If your site is such that a night operator has taken the dump, s/he may or may not have the ability and/or permission and/or user capability to run IDAT. Just make sure that the tape is safe until either you or the Response Center has had a chance to analyze it. I would recommend that the tape be labeled with the date and time, MPE revision, and the failure type.

Now we should be ready to run IDAT. The important point is to find the most recent version of IDAT that you can. Most systems will have a version of IDAT in PUB.SYS. It may be named IDAT or IDAT5. Many systems will also have another version somewhere in the TELESUP account. The distribution of the utilities in the TELESUP account is not as uniform as the distribution of MPE, so you may have to look around some. In general, you should find the version whose program file is the largest, since it will probably have the most formatting features.

If necessary, or if you ask someone from HP, you may be able to get a recent version of IDAT so that it will be there when you need it. Having been a former HP SE, and having spent time at the Response Center myself, I used to occasionally download a fresh copy of IDAT on the customer's machine if I could not find a very recent version. However, at 1200 baud, the present size of the IDAT program file is such that this can take as long as forty-five minutes to download, so don't count on HP being able to download a fresh copy to you. Having one already available will speed the dump analysis process along.

NOTE

As an aside, if you happen to own a Series 5x, you should type Ctrl-B after the SDF has issued its WARMSTART and the system is waiting for you to choose the WARM or COOL start option. Then, enter HALT RETURN at the system prompt. If you allow a Series 52 or 58 to be started from within the SDF subsystem, you will be subject to a bug in the SDF which prevents the Series 5x from enabling the CPU board option which distinguishes a Series 5x from a Series 4x. The result of starting a Series 58 with the SDF WARMSTART command will be a machine that accesses memory *slower than a Series 44.* After entering HALT, you can restart the machine by entering: START RETURN, whereupon you will again see the request for which type of startup you wish to perform. This process ensures that you will have a Series 5x which functions at its full speed.

At this point, the most helpful thing to do is to now run IDAT and copy the contents of the memory dump from tape to disc. I will attempt to give you a cookbook method for the procedure:

```
(sign on to the appropriate user)

:RUN IDAT[.grp[.acct]]

Interactive MPE-IV/V Dump Analysis Tool - 10/27/87
(C) HEWLETT-PACKARD COMPANY 1985

Type 'H' for Help
```

Notice that IDAT contains a HELP facility. It looks for a separate file named IDATHELP, residing in the same group that IDAT resides. If you do not have a file by that name, you may wish to contact someone at HP in order to obtain the proper file. The HELP facility will provide you with the appropriate syntax for the various commands that IDAT uses. Also, the abbreviations used in the output from various formatting and display commands are expanded upon. However, you should be aware that entering the proper commands will not guarantee that you will understand the output you receive. This paper will cover some, but not all, of the capabilities of IDAT.

IDAT's prompt character is a single dash (-). When following the examples below, be aware that you would enter everything *after* the dash.

We can now begin to discuss copying the dump from tape to disc. You can make the analysis process a little easier if you choose meaningful names for the copies of your memory dump(s). If, as in our case, you have a SF310, you can enter the following command:

```
     -T SF310,TAPE RETURN
```

IDAT will now attempt to read the memory dump tape that was created earlier. At the console, you will need to reply to the tape request. An example is listed below:

```
     :RECALL
     THE FOLLOWING REPLIES ARE PENDING:
     ?10:34/#S842/185/LDEV# FOR "DUMPTAPE" ON TAPE (NUM)?
     :REPLY 185,7
```

IDAT will now read the contents of the memory dump tape, transferring the data to a disc file named SF310. If you have a dump tape created by the SDF on U-Mit or beyond, and you are running a sufficiently recent copy of IDAT, the contents of virtual memory will also be copied to a disc file. It will have the name VSF310. (The letter 'V' will be concatenated to the front of the file name provided in the IDAT -T command)

Once the contents of the tape is copied to disc, you can use IDAT to examine the contents of memory by entering the command:

```
     -T SF310 RETURN
```

If IDAT sees a corresponding file with the name VSF310, it will attempt to open that file and process its contents as virtual memory. If successful, you will see a message on your screen similar to:

```
          *** VIRTUAL STORAGE IN EFFECT ***

     MPE VERSION: HP32033G.A3.01.   (BASE G.A3.01).

     ********* HALT  17
     ****SYSTEM FAILURE #310 ; STATUS %140052; DELTA P %013074
```

At this point, you have already helped the Response Center with dump analysis preparation, so you may wish to stop here and wait for the HP Response Center to return any phone calls that you may have placed. When someone does call, s/he can be informed as to the location of the IDAT program and the memory dump file(s) and the engineer will be able to start analyzing your dump immediately.

Of course, one of the advantages to having the dump now residing on disc is that it can be handled the same as any other disc file. It can be :STOREd to tape and sent to HP for further analysis. It can be SQUISHed by any one of several compression programs so that it does not occupy as much disc space (and it can take *a lot* of disc to hold 12 Meg plus virtual memory). You can place other memory dumps in the same group for comparison purposes. You can even get DPAN4/5 to read the dump, if you choose. For the most part, however, there is little that DPAN can tell you that IDAT won't, and IDAT uses *a lot* less paper. A lot of coordinated effort on HP's part has gone into the present system of being able to perform dump analysis over the phone, many times within minutes after a failure has occurred. (See the text covering a brief history of IDAT at the end of this paper for further information)

What we will now consider is: What do you do if you want to know more about the failure before HP calls? The following pages are laid out in an attempt to categorize some of the IDAT commands and they follow a makeshift course. Depending on the output from one command, you may perform other commands. Again, I would like to reiterate that what follows are examples, and they are not intended to be a substitute for being trained to read a dump.

Also, even HP Response Center engineers will vary the choices that they make while reading a dump. There are a few guidelines that can be followed, but, for the most part, you have to resist the temptation to read dumps in a cookbook method. While being trained in dump reading, I was told to remember one rule: Anything, *absolutely anything* could be wrong when the system fails. This means that it could be a hardware problem, an MPE software problem, an application software problem, a hacker who corrupted memory on purpose or even a freak occurrence only attributable to a cosmic ray, or something else equally as obscure.

For each example command which follows, I have listed an objective or action at the beginning, followed by the command to enter, followed by some example output and finally an explanation of what we found and where we should proceed next. For the most part, these examples were taken from actual dumps. Only the names have been changed to protect the innocent.

STEP #1

Format the system registers:

```
-F REG RETURN


                    *****   REGISTERS   *****

 3/12/88,  7:20PM     MPE 5 (G.A3.01) (BASE G.A3.01) * SERIES 4x/5x  *
************************************************************************
*   DATA SEGMENT   *  CODE SEGMENT  *MISCELLANEOUS* STATUS = 103027  *
************************************************************************
* DB BANK = 000013 * PB     = 062130 * X   = 001271 * MODE       = PRIV *
* DB      = 127630 * P      = 071277 * CIR= 030377 * INTERRUPTS= OFF  *
* S  BANK = 000013 * PL     = 106433 * NIR= 000377 * TRAPS      = OFF  *
* DL      = 127500 * PBBANK= 000001 *              * STACK OP   = LEFT *
* Q       = 132200 * (P-PB)= 007147 *             * OVERFLOW  = OFF  *
* S       = 132210 *                * MAP= ON     * CARRY     = ON   *
* Z       = 136240 *                *             * COND CODE = CCE  *
*                  *                *             * SEGMENT # =  27P *
**********************************************************************
      # - SIGNIFIES THAT VALUE SHOWN IS DIFFERENT
          FROM ORIGINAL VALUE OF REGISTER.
***************************************************************************
*                            S I R = 140017                               *
***************************************************************************
* SYSTEM RESET            = ON        * NON RESPOND. DEVICE    = OFF      *
* SYSTEM CLOCK            = ON        * RUN/HALT FOR CMP       = OFF      *
* CHANNEL SERVICE REQ.    = OFF       * DISABLE ATTN. FLAG     = OFF      *
* EXTERNAL INTERRUPT      = OFF       * DATA NOT VALID ON IMB  = OFF      *
* POWER ON                = OFF       * DISPATCHER FLAG        = OFF      *
* POWER FAIL WARNING      = OFF       * ICS FLAG               = OFF      *
* INTEGER OVERFLOW        = OFF       * SPLIT STACK MODE       = OFF      *
* MEMORY PARITY ERROR     = OFF       * RUN/HALT               = HALT     *
***************************************************************************

***** FIXED LOW MEMORY *****
  (@% 0) CST  PTR       022140      (@% 5) ICS QI              050240
  (@% 1) XCST PTR       030324      (@% 6) ICS ZI              060236
  (@% 2) DST  PTR       002140      (@% 7) INTERRUPT MASK      157120
  (@% 3) NOT USED(MPEVE)000000      (@%10) DRT BANK            000000
  (@% 4) CPCB INDEX REL 000671      (@%11) DRT ADDR            000000

***** SYSGLOB (%1000)  *****
  (+%0) SGLOB-SBASE     000000      (+% 5) IOQBASE-REL         007141
  (+%1) CST BASE-REL    021140      (+% 6) SBUF-REL            177041
  (+%2) DST BASE-REL    001140      (+% 7) ICS-QI REL          047240
  (+%3) PCB BASE-REL    121700      (+%10) LPDT BASE-REL       163540
  (+%4) SWAPTAB BASE-REL134340      (+%11) SMON  BASE-REL      166700
```

This output is usually a good place to start looking at a dump. It tells you when the dump was taken, what version of MPE the dump was taken from and what type of CPU it was. This output will show you what the contents of the registers were when the memory dump was taken. It also shows you the contents of some of the more important memory locations in the system. IDAT performs some integrity checking during the production of this output, so if you see error messages relating to the contents of memory then you may have difficulty continuing with further analysis.

IDAT, when you 'Text' in a dump file, also looks for the tell-tale signs that someone "took a dump of the dump system." If you had a problem taking a memory dump, but thought you could try it again by entering DUMP RETURN twice, then this fact will be displayed by IDAT. Usually, the second dump has wiped out nearly every important piece of information about the system registers, which makes these types of memory dumps of little value.

STEP #2

Determine whether or not we have an active process:

Check the value in absolute location %4. You will find it displayed under the heading:

```
     ***** FIXED LOW MEMORY *****
```

In this case, we find:

```
        (@% 4) CPCB INDEX REL 000671
```

If the contents of %4 are non-zero, then we *probably* have a valid active process. To be sure, check the contents of the DB BANK and S BANK registers. If they are *both* non-zero *and* equal, then we are *not* processing on the ICS (Interrupt Control Stack). As a further check, look for the status of the ICS FLAG. If it is OFF, then we are not processing on the ICS. When the system is "on the ICS," this means that there may be no active user process discernible from the contents of the system's registers.

Absolute location %4 is a PCB (Process Control Block) relative address that points to the current process' entry. We refer to each process in the system by its PIN (Process Identification Number). The PIN corresponds to the number of the entry in the PCB table. If you are running any version of MPE V/E, each entry in the PCB table contains %25 words of data. By doing some simple arithmetic, we can determine which PIN was active when the system failed.

Divide the contents of absolute location %4 by %25.

Within IDAT, you can do this easily by entering:

```
        -=671/25 RETURN
```

IDAT responds with the answer of: 25. (All arithmetic is octal unless you specify decimal numbers). This is the PIN which was active when the system failed. The contents of the stack for PIN 25 is what we are now interested in, so we will proceed to Step #3.

STEP #3

Format the current PCB entry that we just found:

```
  -F A4:,PCB RETURN
```

```
   PROCESS ID                                                      SCHEDULE INFO
  --------------                     RESOURCES     PSEUDO INT      -------------
   PIN: % 25              EVENT    -------------  -----------       PRI: 225
   (CURRENT)              FLAGS       CRIT:         PSIM: NORM    WSOFT:
```

```
PTYPE: SYST                       -----      HSIR:        SI:        DISPQ: YES
 NAME:              M:                        SC:         HK:           LQ: YES
                   RG:               NEXT IMP:            SK:           CQ:
                   RL:               PREV IMP:            ST:           DQ:
DATA SEGMENTS      MA:                                    HB:           EQ:
--------------    BIO:               MISCELLANEOUS        CY:        INTER:
    XDS:           IO:               -------------        BK:        CORER:
 ABS DB:         UCOP:                   BMS: SNF     RITBK:         ASOFT:
                 JUNK:                   PPC: NUL     PIOVR:         HIPRI:
   STACK: 157   TIMER:                OBJID1:                        USEDQ:
 SOV ALC:         MSG:        YES    PBX PTR: 12      DSSRVR:          TRW:
                  SON:                SL PTR: 2624                      SW:
                FATHR:                BPLINK:         LIFE/DEATH        LW:
  FAMILY INFO     IMP:                                -----------    DSOFT:
 --------------   SIR:                QUEUE LINKS       LIVE: YES       PC:
   FATHER: 24   TMOUT:               -------------      DEAD:        IPEXP:
      SON:    MEM/WWS:               NQPIN:              FAC:        HSPRI:
  BROTHER:                           PQPIN: 37                          SAR:
        OA: F   PROG FILE LABEL LDEV: 1  DISC ADDR: %00000166215    P SOV:
```

We could have obtained the same output by specifying the PCB entry specifically. In this case, the following command would also have worked:

```
    -F PCB25 (RETURN)
```

This output shows just about everything you would ever want to know about the current status of the process requested. When looking for the current process, check to make sure that the word (CURRENT) appears underneath the line: PIN: %xxx. If the designation (CURRENT) does not appear, then you should check to make sure that the command that you entered is correct, or that your octal arithmetic is correct.

For our purposes, we should first determine which data segment is the stack for this process. In the left column of the display there is a line which shows us the stack data segment number. In this case, it is %157. Having found what we want, we can proceed to Step #4.

STEP #4

Format the stack markers for the current process:

```
-PMAP (RETURN)
```

The above command, if available with your version of IDAT, can be entered when you are examining a dump which came from the same system where you are now running IDAT. This command can also be entered if you are looking at a dump that came from a version of MPE that is *exactly* the same revision as the one you are running. This command tells IDAT to look up the procedure names for the stack markers associated with MPE.

```
-F DA157,STACK (RETURN)


FORMATTED STACK.    DST 157

***PXGLOBAL***

013 126430: 001050  001200  177777  000000  177777  000000
013 126436: 006000  000000  000024  000024  000063  000045

SEG REL DL: 001050 SEG REL DB: 001200 JMAT INDEX:   000000 JPCNT INDEX:  177777
JOB IP LDN: 000024 JOB OP LDN: 000024 JDT DST INDX: 000063 JIT DST INDX: 000045
JOB TYPE:  UNDEF   DUP:          YES    INTERACT:     YES    JCUT INDEX:   000000

* CURRENT PROCESS *

BANK ADDR.    X    DELTA'P STATUS DELTA'Q SEGMENT         PROCEDURE       OFFSET

013 132175: 000702 053074 140052 000152 MORGUE'ABORT (15 ABORT              00522
013 132023: 032057 044731 140001 000012 ININ
013 132011: 000112 001615 142404 000025 USER SEGMENT
013 131764: 000010 001461 142004 000011 USER SEGMENT
013 131753: 000004 001403 142004 002100 USER SEGMENT
013 127653: 000000 000265 160002 000006 USER SEGMENT
013 127645: 000000 040000 140052 000004 MORGUE'ABORT (15 TERMINATE          00000
```

The output received from the this formatting command is now what we need to examine. IDAT has flagged this stack as the stack of the current process. Each stack marker for this process is then traced and displayed for us, starting with the *most recent* stack marker. Under the column labeled SEGMENT we can trace the MPE segment which was being executed, along with several 'USER SEGMENT' markers. The USER SEGMENT markers represent calls between procedures within the program itself. The rest of the segments listed are the MPE system segment stack markers. For each MPE segment, the prior PMAP command has caused IDAT to look up the name of the MPE procedure within the segment which was called. These names appear under the column labeled: PROCEDURE.

In this case, there is a segment identified as ININ in the list. This is important, because an ININ stack marker means that the system had to process an INternal INterrupt. When the system processes an internal interrupt, it keeps a record in the stack of the active process of the type of interrupt that was processed.

One guideline that I was informed of while looking at dumps is that an ININ stack marker should probably be the first stack marker that you examine. In this case, we have had a SF310. Your reference manuals will tell you that this is caused by a system process aborting. Processes can abort from a variety of causes, some of which include stack over/underflows or bounds violations. Exactly which type has occurred here is what we need to determine, so let's proceed to Step #5.

STEP #5

Display the stack in the vicinity of the ININ stack marker:

```
-D EA13+132000,40,B (RETURN)

013 132000 (000000): 000200 000001 000000 000001  ........ (000000)
013 132004 (000004): 000000 000305 000021 000016  ........ (000010)
013 132010 (000010): 141420 000112 001615 142404  ...J.... (000020)
013 132014 (000014): 000025 100401 032057 000671  ....4/.. (000030)
013 132020 (000020): 000400 000030 000000 032057  ......4/ (000040)
013 132024 (000024): 044731 140001 000012 000002  I....... (000050)
013 132030 (000030): 056136 141104 000010 001213  \^.D.... (000060)
013 132034 (000034): 000100 000042 110414 001213  .@.".... (000070)
```

The area of memory that you want to display is determined by the stack marker trace that we saw after the -F DA157,STACK command. Under the column labeled BANK ADDR. we see the memory locations that we will be looking at. When you look at memory around stack markers, choose enough so that you can see the data that you need to, but not so much that you have to wade through pages of listings or several screens of output. In this case, there is enough memory displayed to include the previous USER SEGMENT stack marker and several words beyond the location of the ININ stack marker.

The bank and address offset of each stack marker displayed in Step #4 is the address of the *first* word of the marker. Therefore, if we are interested in the stack marker *previous* to the ININ marker, it is at location EA13+132011 and extends for four words. They are shaded below:

```
013 132000 (000000): 000200 000001 000000 000001  ........ (000000)
013 132004 (000004): 000000 000305 000021 000016  ........ (000010)
013 132010 (000010): 141420 000112 001615 142404  ...J.... (000020)
013 132014 (000014): 000025 100401 032057 000671  ....4/.. (000030)
013 132020 (000020): 000400 000030 000000 032057  ......4/ (000040)
013 132024 (000024): 044731 140001 000012 000002  I....... (000050)
013 132030 (000030): 056136 141104 000010 001213  \^.D.... (000060)
013 132034 (000034): 000100 000042 110414 001213  .@.".... (000070)
```

Just beyond the end of this stack marker is where the ININ process laid down its information. The word laid down was the type of interrupt being processed. This is shown shaded below:

```
013 132000 (000000): 000200 000001 000000 000001  ........ (000000)
013 132004 (000004): 000000 000305 000021 000016  ........ (000010)
013 132010 (000010): 141420 000112 001615 142404  ...J.... (000020)
013 132014 (000014): 000025 100401 032057 000671  ....4/.. (000030)
013 132020 (000020): 000400 000030 000000 032057  ......4/ (000040)
013 132024 (000024): 044731 140001 000012 000002  I....... (000050)
013 132030 (000030): 056136 141104 000010 001213  \^.D.... (000060)
013 132034 (000034): 000100 000042 110414 001213  .@.".... (000070)
```

The interrupt codes can be looked up in any of the older MPE pocket guides in the ASCII/Instruction Set section. The one I happened to use listed them on page 11-14. There you can find that the interrupt code of %100401 is caused by a bounds violation. When the system detected that one of its *system processes* aborted, a call was generated to the MPE procedure SUDDENDEATH with a parameter of 310. This produced the failure message that appeared on the console.

NOTE

With more recent releases of MPE, the Internal Interrupt routine may not save a record of certain types of interrupts. Also, the older brown MPE pocket guides may not list *all* the possible interrupt codes. If you try this method of analysis on a dump of your own and you do not seem to find any meaningful information in the stack, please let the Response Center assist you in your analysis.

Now, how can we find out more specifically which process aborted? Let's go back to the formatted display of the current process' PCB entry (Step #3). At the bottom of the display is a line containing the disc address of the file label for the program which was running at the time of the failure. IDAT can sometimes look up the process' file name in memory, in which case it will be displayed in this area of the output. In this case, however, IDAT was not able to determine the process' file name, so as a next-best effort, it looked up the disc address of the program file, which was also stored away in one of the MPE tables.

Since, in our scenario, we are looking at our own dump, unless the program file was purged after the failure *and* overwritten with new data, we can use one of the other TELESUP utilities to look up this sector on the appropriate disc and display the contents of the file label. DISKED5.PUBSYS can do that for you. In this case, the program file happened to be NMLOGMON.PUBSYS.

The program NMLOGMON.PUBSYS is one of the programs run by HP communications software. This program is activated when some HP communications subsystems are brought up.

If I were still working for HP, and if I happened to be one of the "on-loan" engineers working this problem for a customer, I would now begin to start asking questions about what was going on at the time of the failure. Perhaps more importantly, I would also ask about anything that may have changed recently. Were any Communication System devices reconfigured recently? What communication products exist on the system? Were any changes made to MPE recently?

In this case, let us see what else we can determine about the status of the the machine at the time of the failure. Proceed to Step #6.

STEP #6

Determine who was signed on and what they were doing:

```
-F JOB (RETURN)

JOBNUM STATE UMAIN JIN  JLIST    INTRODUCED JOB NAME
             PIN#
#S2     EXEC   26   20  20       SAT  7:15P MANAGER.SYS,PUB

  1   SESSIONS, 0   JOBS
 JOB LIMIT = 1   SESSION LIMIT = 1
 JOBFENCE = 7
 JOBSECURITY = HIGH
```

The output generated by this command shows us who was signed on at the time of the failure, similar to what a :SHOWJOB would do for us while the system is up. We can see that there was only one user on the system at the time. By entering the command below we can see what this user's last command was:

```
-F CI (RETURN)

JOBNUM UMAIN STACK JOBNAME          LAST COMMAND
        PIN#  DST#
#S2      26   150  MANAGER.SYS      NETCONTROL START
```

We have now determined that the only user on the system had just entered a :NETCONTROL START command. Since NMLOGMON.PUB.SYS is one of the programs which would be activated during this process, it stands to reason that the problem is related to the use of the Network Services on this system.

In order to determine an exact cause of the failure at this point, it may be necessary to leave the realm of IDAT and pursue other courses of action. If I had access to the source code for the NMLOGMON.PUB.SYS program file, I could use it to trace the 'USER SEGMENT' stack markers and attempt to determine what procedures were being called, what data was passed to those procedures, etc. But since I am no longer privy to that kind of information, this would be the time to let the Response Center take over and further research the problem.

When this failure actually happened, however, I elected to do one other task while waiting on the Response Center. I used a utility called CHECKSUM to examine the NMLOGMON.PUB.SYS program file. This utility can calculate a unique checksum for a program and can verify this checksum against a checksum stored permanently as part of a program. I realize that I am digressing away from the topic of this paper, but in order to satisfy those who may be curious, I will elaborate just a little more on this example.

The CHECKSUM utility found a discrepancy between the calculated checksum and the checksum stored permanently in the program file. It happened that another Boeing site in Seattle had already updated several machines to this version of MPE, and had no problems bringing up their networks. Dialing into one of their systems, I again used the CHECKSUM utility to verify that NMLOGMON.PUB.SYS in Seattle was the same as the copy that I had. It was *not*.

Running CHECKSUM against NMLOGMON.PUB.SYS in Seattle told me that their copy was not corrupt, but mine was. Later, it was confirmed by HP that something had corrupted the file on a few of the MIT tapes that had been distributed to the field. By using DSCOPY to retrieve a *good* copy of the program, I was able to bring up the network without any problems.

What you can see from this example is that IDAT may be only one of several tools that you may have to use to determine the cause of a failure or system hang. In an actual scenario such as this SF310, however, there are still other items of interest in the contents of the memory dump that may be worth checking, just to be sure that you haven't overlooked something.

The following are examples of other IDAT commands which will aid you in determining the integrity (or lack of it) of MPE at the time of the failure. These commands are provided as examples of various commands and are not meant to imply that they are the most appropriate commands for use in analyzing a SF310.

Addenda

Check the integrity of system code segments in memory:

```
    -F CKSUM [RETURN]


COMPUTED   RETRIEVED            SEG  SEG              PROG
CHECKSUM   CHECKSUM    COMPARE  NUM  NAME             NAME
--------   ---------   -------  ---  -----------      --------
 127614     164241     NORMAL     1  ININ             P10P033C
 017412     017412      OK        6  CLIB'03
 074344     074344      OK       15  NLS02'3
 022724     022724      OK       16  NLS01'4
 052771     052771      OK       17  FILESYS4'6
 173125     173125      OK       20  FILESYS1'8
 024414     024414      OK       21  FILESYS7
 144235     144235      OK       22  FILESYS6A
 141476     141476      OK       23  FILESYS5
   .          .          .        .   .
   .          .          .        .   .
   .          .          .        .   .
```

```
162514       162514       OK       215  NMADSFMTSEG1
010744       010744       OK       216  NMSEGF
044646       044646       OK       220  NETUI
001123       001123       OK       221  SEG'            P43P002C
001144    *NO PATCH AREA*          222
174302       174302       OK       223  SEG'            P44P033C
121760    *NO PATCH AREA*          224
075511       075511       OK       225  IOLANO'MPE5     P21P131A
```

In one word of each MPE system code segment, the systems programmers place the checksum of the segment *as it was originally created.* IDAT finds this original checksum and also computes a new checksum using the same algorithm. This command displays the two checksums for you so that you can instantly determine if there was any corruption of memory that dealt with MPE. If you find any, then you need to determine if the copy of the corrupt segment(s) in SL.PUB.SYS is/are also corrupt. If the contents of memory are corrupt, but the copy on disc is *not*, then restarting the system will replace the corrupt segment with one that is valid. If the segment on the disc is *also* corrupt, then you will need to perform a COLDLOAD or UPDATE from a *known-good* tape. That is why you *always* need to know where you keep a valid COLDLOAD tape, since, if your disc has been corrupted, then one or more backup tapes are also probably corrupt. If you COLDLOAD or UPDATE from one of these corrupt backup tapes, you will not be free of the problem.

NOTE

The placement of a checksum in an MPE code segment or in a program file depends on the use of an undocumented option in the Segmenter which tells it to calculate a checksum and save it in the proper place. There may be one or more MPE segments which were not *PREP*ed with this option in certain versions of MPE starting with V-Mit. I was informed that one such segment is named: ASVTSEG2. This segment is part of the Network Services software (NS/3000). Before you panic at finding a "corrupt" segment, at least according to IDAT, check with the Response Center to see if, in fact, you *do* have something to be concerned about.

How do you tell if the segments on disc are corrupt? One way would be to use the CHECKSUM utility previously mentioned. The other way is to use IDAT in "live" mode. After you have restarted your machine, new copies of the MPE code segments will have been loaded into memory. IDAT has an option which merely switches its function from that of formatting the contents of a disc file to that of formatting the *live* system. The command is: `-L` RETURN. If we switch IDAT into "live" mode, then we can use the same `-F CKSUM` command as we used above to format what is currently in memory while the system is up and running. If IDAT shows the same corruption as was contained in the memory dump, then you will need to COLDLOAD or UPDATE from a good tape in order to remove the corruption.

Check for any privileged mode programs:

```
    :F PMUSERS RETURN

     PIN %227    STK %636    OPT.PUB.SYS
     PIN %230    STK %363    QUAD.PUB.XTOOLS
```

If you see *any* entries listed, they are candidates for examination concerning the failure. Even programs which are supplied and supported by HP could possibly cause system failures under the proper circumstances.

As we found out in our first scenario, a corrupt program file was at the root of the cause for the SF310. Given other conditions, this program, which runs in privileged mode, could just as easily have modified memory in some location which would have caused a totally different system process to abort, or caused random memory corruption to other users' programs or data segments. Failures caused by this type of corruption are usually *very* difficult to trace.

Let us consider a few other integrity-checking commands within IDAT.

Verify various memory structures:

```
    :F ARL RETURN

    TESTING LINKS OF THE ARL. #REGIONS in LIST = 61
    FINISHED CHECKING ARL. NO ERRORS FOUND.
                                  MPE 5/E FORMAT
                  REG.  PREVIOUS      NEXT      A R A C S L F I M  OBJ. OBJ. CSTX
     HDR ADDRESS  SIZE HDR ADDRESS HDR ADDRESS  S S V L C K Z O I  TYPE NUM. INDEX
     022 177600     1  000 000000  023 055400       Y               DST 751
     023 055400     1  022 177600  027 133400       Y               DST
     027 133400     1  023 055400  024 056600       Y               DST 1110
                               .                    .                .
                               .                    .                .
                               .                    .                .
     016 144200    60  026 042400  014 135000       Y               DST
     014 135000    62  016 144200  035 000000       Y               DST 226
     035 000000    65  014 135000  024 000000       Y              CSTX 14    165
     024 000000    67  035 000000  020 155400       Y               CST 170
     020 155400    74  024 000000  034 127000       Y              CSTX 33    165
     034 127000    74  020 155400  034 047000       Y               DST 213
     034 047000    79  034 127000  022 000600       Y              CSTX 15    165
     022 000600    81  034 047000  011 043200       Y               CST 66
     011 043200    85  022 000600  015 070200       Y               CST 141
```

```
015 070200    86  011 043200  034 000000        Y                DST 221
034 000000    87  015 070200  021 146600        Y                DST 172
021 146600   101  034 000000  000 000000        Y               CSTX 27    165
```

This command traces and checks the Available Region List (ARL) of your system's memory. Note that the region sizes proceed from the smallest to the largest. IDAT performs some integrity checking as it produces the list so, for example, any backward pointer which did not point to the previous region in the list would be flagged as corrupted.

In the right-most column you will see a representation of what the segment identifier says that this segment is. In this case, the identifier for an available region is mostly meaningless. There are other memory-formatting commands which *will* display meaningful information here.

One of these commands would be:

```
-F MEM (RETURN)
```

This command starts at the low memory addresses and formats information contained in each memory region header and trailer. Again, IDAT performs some integrity checking along the way. This command can produce a good deal of output, especially on larger machines. Space being limited here, I have not included an output example. The format of the output is the same as that of the -F ARL command shown above, however.

If MPE disc caching was in use, check the Cache Directory Table (CDT):

```
-F CDT (RETURN)
```

```
Cache Directory Table Header.   (DST %206 Address %004 100230
***************************************************************
Num entries = 001611 | Entry size = 000032 | Num free = 001605
First free  = 021242 | Last free  = 021210 | Max used = 000007
Ldevs cached= 000002 | First entry= 000001 | DST size = 055752
Stop pending= 000000 | Sectors seq= 000140 | Sec. rand= 000020
Force post  = 000000 | Head imp q = 000000 |
***************************************************************

Cache Directory Table - Device Entry #1
***************************************************************
Next ldev = 000077 | Prev ldev = 000000 | This ldev = 000002
Num pages = 001121 | # domains = 000000 | Map head  = 000000
Map tail  = 000000 | # regions = 000043 |
...............................................................
Head domain pntr = 015 017623 | Tail domain pntr = 022 060623
Number read hits = 611        | Number write hits= 406
Number read miss = 47         | Number write miss= 20
Number proc stops= 147        | Scan pointer     = 002 177223
***************************************************************
```

```
Cache Directory Table - Device Entry #63
**************************************************************
Next ldev = 000000 | Prev ldev = 000001 | This ldev = 000004
Num pages = 001055 | # domains = 000000 | Map head  = 000000
Map tail  = 000000 | # regions = 000035 |
--------------------------------------------------------------
Head domain pntr = 012 162623 | Tail domain pntr = 007 137423
Number read hits = 153        | Number write hits= 61
Number read miss = 74         | Number write miss= 1
Number proc stops= 72         | Scan pointer     = 010 112623
**************************************************************
```

If MPE disc caching was turned on for any of your disc drives, then this command will tell you which devices were cached. You can then use another command to format the Cached Domain Regions for each disc. Again, integrity checking is done on the disc domain regions. If the system had failed with an error indicating some type of caching problem, then tracing the cache domain regions might be very helpful in finding the cause.

This output shows that two discs, logical device numbers two and four, respectively, were cached when the system failed. The logical device number is displayed after the heading: `This ldev =`. The heading: `Device Entry #` does *not* refer to the logical device number. This number corresponds to the entry number in the Cache Directory Table, instead.

Check the cached domains for each drive cached:

```
-F CDR 2 RETURN
```

```
CACHED DOMAIN REGION HEADERS FOR Ldev 2
# OF REGIONS IN THE LIST = 134
TESTING THE LINK POINTERS OF THE CDR
FINISHED CHECKED CDR. NO ERRORS FOUND.
                              MPE 5/E FORMAT
              REG.  PREVIOUS      NEXT     A R A C S L F I M  OBJ. OBJ. CSTX
 HDR ADDRESS  SIZE HDR ADDRESS HDR ADDRESS S S V L C K Z O I  TYPE NUM. INDEX
 010 171600      2 000 000000  025 113400  Y     Y             CDT
 025 113400      2 010 171600  114 134200  Y     Y             CDT
 114 134200      2 025 113400  075 152200  Y     Y             CDT
 075 152200      2 114 134200  133 122000  Y     Y             CDT
 133 122000      2 075 152200  055 072000  Y     Y             CDT
 055 072000      2 133 122000  137 172600  Y     Y             CDT
       .         .       .           .           .              .
       .         .       .           .           .              .
       .         .       .           .           .              .
 124 136400     17 013 064400  061 066000  Y     Y             CDT
 061 066000      2 124 136400  121 035600  Y     Y             CDT
```

```
121 035600    61  061 066000  101 145000  Y      Y           CDT
101 145000    71  121 035600  065 156200  Y      Y           CDT
065 156200    71  101 145000  125 012000  Y      Y           CDT
125 012000    17  065 156200  043 043600  Y      Y           CDT
043 043600    17  125 012000  137 034600  Y      Y           CDT
137 034600    17  043 043600  000 000000  Y      Y           CDT
```

(I'll only include one disc for this example)

When you format the Cached Domain Regions (CDR) for a particular disc drive, what you should look for at first is to make sure that no integrity errors were detected. Also, each object type appearing under the heading of OBJ. TYPE should be labeled a CDT. If you see items in the list which are identified as something else (CST, DST, CSTX) then you may have cause to suspect something is wrong. Let me reiterate, however, that I can only provide "rules of thumb" for most of what you might see. Don't forget that IDAT is officially an unsupported utility, even though the Computer Systems Division (CSY) at HP *does* maintain it and also develops new capabilities for IDAT. As newer versions of MPE are released, however, there may be structural changes which even recent versions of IDAT may not handle correctly. If you have any doubts at all about what IDAT tells you, please let the Response Center assist you.

Let us consider another type of failure, for example. This time, it will be a system hang. Again, this example came from an actual system hang, but the names have been changed to protect the innocent.

In order to get a valid dump of a system that is hung, you must HALT the system execution and immediately start the SDF. Most of the time, you can do this by entering Ctrl-B at the system console. When the system returns with its prompt, you would enter HALT RETURN. After the machine halts, you can invoke the SDF software to take the dump. If, for some reason, you cannot get the system to respond after entering a Ctrl-B at the console, then you will have to try using physical means to halt the machine. On Series 4x/5x machines, you can open the front panel of the CPU and press the button labeled HALT. On Series 6x/7x machines, and also the newer Series 37's and Micro 3000's, you may have to resort to attempting to trick the system by causing a power failure. If that procedure fails, you may be out of luck. Phone the Response Center for further assistance, in that case.

STEP #1

Format the system registers:

```
-F REG RETURN
```

```
                    *****   REGISTERS   *****

 4/01/88,  4:28PM     MPE 5 (G.A3.01) (BASE G.A3.01) * SERIES 4x/5x  *
***********************************************************************
*   DATA SEGMENT   *  CODE SEGMENT  *MISCELLANEOUS* STATUS = 100513  *
***********************************************************************
* DB BANK = 000003 * PB     = 140430 * X  = 001271 * MODE      = PRIV *
* DB      = 035030 * P      = 157365 * CIR= 000700 * INTERRUPTS= OFF  *
* S  BANK = 000000 * PL     = 173753 * NIR= 000000 * TRAPS     = OFF  *
* DL      = 177777 * PBBANK= 000002 *              * STACK OP  = LEFT *
* Q       = 050456 * (P-PB)= 016735 *              * OVERFLOW  = OFF  *
* S       = 050555 *                * MAP= ON      * CARRY     = OFF  *
* Z       = 060236 *                *              * COND CODE = CCL  *
*                  *                *              * SEGMENT # = 113P *
***********************************************************************
      # - SIGNIFIES THAT VALUE SHOWN IS DIFFERENT
          FROM ORIGINAL VALUE OF REGISTER.
*******************************************************************************
*                              S I R = 140011                                 *
*******************************************************************************
* SYSTEM RESET           = ON       * NON RESPOND. DEVICE    = OFF            *
* SYSTEM CLOCK           = ON       * RUN/HALT FOR CMP       = OFF            *
* CHANNEL SERVICE REQ.   = OFF      * DISABLE ATTN. FLAG     = OFF            *
* EXTERNAL INTERRUPT     = OFF      * DATA NOT VALID ON IMB  = OFF            *
* POWER ON               = OFF      * DISPATCHER FLAG        = OFF            *
* POWER FAIL WARNING     = OFF      * ICS FLAG               = ON             *
* INTEGER OVERFLOW       = OFF      * SPLIT STACK MODE       = ON             *
* MEMORY PARITY ERROR    = OFF      * RUN/HALT               = HALT           *
*******************************************************************************

***** FIXED LOW MEMORY *****
  (@% 0) CST  PTR        022140       (@% 5) ICS QI               050240
  (@% 1) XCST PTR        031004       (@% 6) ICS ZI               060236
  (@% 2) DST  PTR        002140       (@% 7) INTERRUPT MASK       157120
  (@% 3) NOT USED(MPEVE)000000        (@%10) DRT BANK             000000
  (@% 4) CPCB INDEX REL 007304        (@%11) DRT ADDR             000000

***** SYSGLOB (%1000)  *****
  (+%0) SGLOB-SBASE     000000        (+% 5) IOQBASE-REL          007141
  (+%1) CST BASE-REL    021140        (+% 6) SBUF-REL             177041
  (+%2) DST BASE-REL    001140        (+% 7) ICS-QI REL           047240
  (+%3) PCB BASE-REL    121600        (+%10) LPDT BASE-REL        164240
  (+%4) SWAPTAB BASE-REL134240        (+%11) SMON  BASE-REL       170000
```

Notice that we have a non-zero value in absolute location %4, but the DB BANK and S BANK values are *not* equal. Also, notice that the status of the ICS FLAG is ON. When processing certain interrupts, the system environment uses a stack area called the Interrupt Control Stack (ICS). In order to find out what is going on when we are "on the ICS," we can ask IDAT to format the ICS information for us. Continue to Step #2.

STEP #2

Format the ICS information:

```
·F ICS (RETURN)

         ---SCHEDULING INFORMATION---
CURR E   CURR D                      CURR C   MAX C    MIN C
FILTER   FILTER   CWTNUM   CWTDENOM  FILTER   FILTER   FILTER
001750   001750   000143   000144    000140   000144   000000

E BASE   D BASE   C BASE   E LIMIT   D LIMIT  C LIMIT
000360   000276   000230   000375    000356   000310

         ---CURRENT STACK INFO---
STACK                      TRACE
DST      PISTAT   PIADDR   FLAG      PFAILPCB JCUT     CPCB INDX
000233   100114   044142   177777    000000   163000   007304

         LAUNCH   LAUNCH   LAUNCH             STACK
PCBX     Z        DL       S         SBANK    DB
121630   014730   000000   012627    000007   122500

         ---MISCELLANEOUS INFORMATION---
CAND     LAST     LIST     PMBC      PMBC     PMBC     PMBC     PSDB
PIN      WEIGHT   STATE    BNDS FLG  XDS SIZE XDS BASE XDS BANK COUNTER
000264   000001   000000   000000    000000   000000   000000   000000

PAUSE TIME
00336335742

         ---STACK MARKERS---
ADDR.   X-REG   DELTA-P STATUS  DELTA-Q SEGMENT
------  ------  ------- ------  ------- -------
050453  000004  050435P 101113  000034  TERMDRIVER (223)
050417  000001  057062P 100071  000026  TERMHANDLR (200)
050371  000007  045121P 102071  000040  TERMHANDLR (200)
050331  000001  041575P 102113  000037  TERMDRIVER (223)
050272  000004  042715P 142113  000035  TERMDRIVER (223)
050235  000000  042350P 100111  100000  ***DISP MARKER***
```

Perhaps one of the more important things we gain from this output is an indication of which PIN was processing interrupts. This information can be found under the heading CAND PIN near the middle of the displayed output. Along with the PIN, its stack data segment number is also listed under the heading STACK DST. At the bottom of the displayed output will be any other stack marker information which is available to IDAT. Here, we can see several MPE segment names listed under the column labeled SEGMENT. The last stack marker displayed will always be labeled: ***DISP MARKER***. This is a special stack marker laid down by the MPE Dispatcher. If you should find no stack markers displayed other than the dispatch marker, then you may wish to proceed to the candidate PIN's stack to see if more information can be learned there. In this case, let us do just that, even though we have other stack markers displayed here. Proceed to Step #3.

STEP #3

Format the candidate PIN's PCB entry:

```
-F PCB264 RETURN

     PROCESS ID                                                 SCHEDULE INFO
   --------------                      RESOURCES   PSEUDO INT   -------------
     PIN: % 264               EVENT  -------------  -----------     PRI: 230
     (CURRENT)                FLAGS      CRIT: YES   PSIM: BK     WSOFT:
   PTYPE: UMAIN               -----      HSIR:         SI:        DISPQ: YES
    NAME:             M:                   SC: YES     HK:           LQ:
                     RG:             NEXT IMP:         SK:           CQ: YES
                     RL:             PREV IMP:         ST:           DQ:
   DATA SEGMENTS     MA:                               HB:           EQ:
   -------------    BIO:       YES   MISCELLANEOUS     CY:        INTER: YES
       XDS:          IO:             -------------     BK:        CORER:
    ABS DB: YES    UCOP:                BMS: SNF   RITBK:         ASOFT:
                   JUNK:                PPC: NUL   PIOVR:         HIPRI:
     STACK: 233   TIMER:              OBJID1:                     USEDQ:
   SOV ALC:         MSG:             PBX PTR:      DSSRVR:          TRW:
                    SON:              SL PTR: 4202                   SW:
                  FATHR:              BPLINK:      LIFE/DEATH        LW:
    FAMILY INFO     IMP:                           -----------    DSOFT:
   -------------    SIR:              QUEUE LINKS   LIVE: YES        PC:
      FATHER: 11  TMOUT:             -------------  DEAD:        IPEXP:
        SON: 213 MEM/WWS:            NQPIN: 345      FAC:        HSPRI:
    BROTHER: 310                     PQPIN: 303                     SAR:
         OA: S                                                    P SOV:
```

This output displays the status of PIN %264 on this system at the time of the failure. HP engineers and SE's commonly interchange the term PIN with PCB when referring to entries in the Process Control Block. The type of process that this PCB entry refers to is noted after the heading: PTYPE:. As you can see, it is specified as the type: UMAIN. This term stands for 'USER MAIN.' This means that this process corresponds to some user's Command Interpreter. When a user enters a command such as :SHOWME or :RUN IDAT, it is the Command Interpreter process which handles this command and performs the appropriate action necessary to complete it. In the case of a :RUN command, a son process to the Command Interpreter will be started, and this son process will be the program that the user has requested.

In the case of this dump, we now might be interested in which session or job corresponded to this Command Interpreter process. In order to find this out, proceed to Step #4.

STEP #4

Format the Job Master Table (JMAT):

```
·F JOB RETURN

JOBNUM STATE UMAIN JIN  JLIST    INTRODUCED JOB NAME
             PIN#
#S1817  EXEC  264   23  23       FRI 12:22P MATINFO.TBA,PUB
#S803   EXEC   60   22  22       WED  9:05A OPER.SYS,OPER
#S1877  EXEC  314   45  45       FRI  3:44P OLVA,MGR.WIPDEV,DATA
         .           .                .             .
         .           .                .             .
         .           .                .             .
#J771   EXEC  214   10S LP       FRI  3:58P CABW15,RALPH.TBD,SALLY
#S1803  EXEC  112   95  95       FRI 11:18A SCOTTY.NEAT,SCOTTY
#S1862  EXEC   21   30  30       FRI  3:07P INFO.TBA,PUB

  36  SESSIONS, 8   JOBS
 JOB LIMIT = 10  SESSION LIMIT = 50
 JOBFENCE = 0
 JOBSECURITY = LOW
```

Fortunately, the session which has PIN %264 as its UMAIN PIN # is the first one listed. What we can learn here is that the user's name is MATINFO.TBA. This user is signed on to Logical Device 23.

One item which may now be of interest is to find out what this user was doing at the time the dump was taken. Proceed to Step #5.

STEP #5

Format the Command Interpreter stacks:

```
·F CI RETURN
```

```
JOBNUM UMAIN STACK JOBNAME          LAST COMMAND
        PIN#  DST#
#S1817  264   233  MATINFO.TBA      ::::::::::::::::::::::::::::::::::::::::
#S803    60   602  OPER.SYS         ENDIF
#S1877  314   151V MGR.WIPDEV       RUN QUAD.PUB.XTOOLS;INFO="Q;T TBUKS
         .          .                .             .
         .          .                .             .
         .          .                .             .
#J771   214  1472V RALPH.TBD        RUN QUERY.PUB.SYS
#S1803  112   403V SCOTTY.NEAT
#S1862   21  1415V INFO.TBA         RUN TBAINFO.TBALM;LIB=P
```

The user MATINFO.TBA has, according to IDAT, a long string of colon characters as his last command. This is definitely not a normal occurrence. Let us remember this, but continue on with the task of gathering information.

STEP #6

Format the CI stack:

```
-F DA233,STACK (RETURN)

 FORMATTED STACK.   DST 233

 ***PXGLOBAL***

 007 121630: 000650  000650  000002  000001  000102  000000
 007 121636: 016000  000000  000027  000027  000402  001533

 SEG REL DL: 000650 SEG REL DB: 000650 JMAT INDEX:   000001 JPCNT INDEX:  000102
 JOB IP LDN: 000027 JOB OP LDN: 000027 JDT DST INDX: 000402 JIT DST INDX: 001533
 JOB TYPE:  SESSION DUP:         YES    INTERACT:     YES    JCUT INDEX:   000000

 * CURRENT PROCESS *

 BANK ADDR.    X    DELTA'P STATUS DELTA'Q SEGMENT          PROCEDURE       OFFSET

 007 135322: 000022 063074 142105 000033 TERMANAGER (215) IM'READ           01122
 007 135267: 000011 055652 140054 000075 TERMONITOR (161) TERM'LOG'MON      01174
 007 135172: 000001 047052 140427 000010 HARDRES (132)    AWAKEIO           00070
 007 135162: 000001 057736 141027 000032 HARDRES (132)    P'ATTACHIO        00713
 007 135130: 000137 057020 140027 000033 HARDRES (132)    ATTACHIO          00402
 007 135075: 000427 047252 140517 000174 FILESYS1A (227)  IOMOVE            03231
 007 134701: 000000 042355 142517 000116 FILESYS1A (227)  FREAD             00377
 007 134563: 007315 050603 141057 000013 UTILITY1 (164)   READ              00130
 007 134550: 165671 041315 140441 000603 CIINIT (144)     COMMANDINTERP     01271
 007 133745: 007304 043664 140114 000037 MISCSEGC'CHECK ( PSEUDOINT         00642
007 133706: 177756 070033 101111 000016 KERNELC (221)    WAIT             00655
007 133670: 005564 066765 101111 000014 KERNELC (221)    AWAKE            00272
007 133654: 000005 047174 141035 002077 CIPREPRUN (140)  CXPREPRUN        03333
007 131555: 007304 044754 140441 000602 CIINIT (144)     COMMANDINTERP    04730
007 130753: 000045 047063 140074 000261 UDC (203)        FEEDCI           00457
007 130472: 000000 040555 140074 000572 UDC (203)        UDC              00260
007 127700: 007304 044140 142041 000603 CIINIT (144)     COMMANDINTERP    04114
007 127075: 000007 047063 140074 000261 UDC (203)        FEEDCI           00457
007 126614: 000000 040555 140074 000572 UDC (203)        UDC              00260
007 126022: 000000 045610 141074 002006 UDC (203)        INITUDC          03233
007 124014: 000000 043605 140041 000605 CIINIT (144)     COMMANDINTERP    03561
007 123207: 000000 040000 140052 000004 MORGUE'ABORT (15 TERMINATE        00000
```

The *most recent* stack marker shows us executing in the MPE segment named TERMANAGER. Considering that the formatted stack markers on the ICS were TERMDRIVER and TERMHANDLR, it may be assumed that the system was trying to process some sort of I/O for this user. Even knowing this much can help us perform some detective work before the Response Center returns our call. It would probably be a good idea to find out who was actually using logical device 23 at the time the system hung. Once found, you could ask him/her a few questions about anything that s/he may have seen on the terminal that was out of the ordinary. There might have been some sort of application error message that appeared, or even some sort of MPE error message. Finding out as much as you can before HP begins their analysis will help speed the dump analysis along, in most cases. For this particular dump, there were thirty-six sessions signed on when the system hung. Knowing which user you should contact first is obviously a time-saver.

As with the first failure example, we can also perform some preliminary checks of MPE integrity. Some possible areas for exploration are shown below.

Addenda

Check for possible causes of deadlocks:

```
-F SIR RETURN

 FORMATTING...

 SIR # 45 LOCKED BY PIN #251              FILE INTEGRITY
 NO IMPEDED PROCESSES

 NO DEADLOCKS FOUND
```

This command checks the status of the SIR table. It will display the name of any SIR that was locked at the time of the failure and also display the PIN number of the locking process. Subsequently, it checks the SIR table for any deadlocks. If we had found a deadlock situation for this example memory dump, it might have meant that something else was at the root of the system hang.

NOTE

A deadlock caused by two processes in a SIR lock is not the only way that a system can become hung. File locking problems, known as RIN locks, and IMAGE/TurboIMAGE locking problems can also cause a system to hang. IMAGE/TurboIMAGE locking problems usually have to be tracked down "manually," i.e., IDAT does not have a special command to track down and display an IMAGE/TurboIMAGE deadlock.

Format the RIN table:

```
·F RIN (RETURN)

Total number of RINs configured:        1024
Total number of GLOBAL RINs configured: 64

Current number of LOCAL RINs in use:  63
Current number of GLOBAL RINs in use: 1
Current number of FILE RINs in use:   67
Current number of RINs AVAILABLE:     893

GLOBAL RIN information:

RIN#    CREATOR              PASSWORD
====    =======              ========
3       WP      .HPOFFICE    WORDRIN

ACTIVE RINs STATUS:

RIN# 1      FILE     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 2      FILE     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 3     GLOBAL    Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 5     LOCAL     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 7     LOCAL     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 8     LOCAL     Holding PIN %647   Waiting PIN %0      LOCKED, NO WAITING
RIN# 10     FILE     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 11    LOCAL     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 12     FILE     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 13    LOCAL     Holding PIN %0     Waiting PIN %0      NOT LOCKED
             .             .                  .                  .
             .             .                  .                  .
             .             .                  .                  .
RIN# 180    FILE     Holding PIN %0     Waiting PIN %0      NOT LOCKED
RIN# 181   LOCAL     Holding PIN %0     Waiting PIN %0      NOT LOCKED
```

```
RIN# 182    FILE    Holding PIN %0    Waiting PIN %0    NOT LOCKED
RIN# 183    FILE    Holding PIN %0    Waiting PIN %0    NOT LOCKED
```

This command displays the status of the RIN table, listing RIN numbers, type of RIN (local or file), holding PIN (if there is one), waiting PIN (if there is one) and lock status. What you should look for, if you wish to research possible deadlock situations, is for a locked RIN *with* one or more other PINs waiting. Look for a chain of waiting PINs. If one is found, then further research may be necessary to determine if the system hang occurred because of this situation.

Check the integrity of MPE system code segments:

```
-F CKSUM (RETURN)
```

You should examine this output for any segment corruption. If any is found, it should be noted and this information should be made known to the Response Center engineer who may be helping you.

Check for any privileged mode programs:

```
-F PMUSERS (RETURN)
```

Make note of any user-written programs, TELESUP utilities and/or third-party software. Remember that a malfunctioning privileged-mode program can wreak havoc on your system.

Format the Monitor Table:

```
-F MON (RETURN)

                ******    MONITOR TABLE    ******
LOC     PIN  EVENT
171550   264 SIODM-EXIT       000002 000000 060013 000000 000000 147131
171540   264 SPECIALRQ        000000 000731 000003 000000 000000 000000
171530   264 SIODM-ENTRY      000002 033122 040213 000013 000000 147130
171520   264 GIPINTERUPT      000002 033122 040413 000017 000000 147127
171510   270 QUIESCE          000000 000000 110356 000000 000000 147114
171500   270 SIODM-EXIT       000004 000000 060013 000000 000000 147106
171470   270 SPECIALRQ        000000 000314 000003 000000 000000 000000
171460   270 SIODM-ENTRY      000004 006531 040213 000013 000000 147104
171450   270 GIPINTERUPT      000004 006531 040413 000017 000000 147104
171440   117 QUIESCE          020040 004000 110356 000000 000000 147073
171430   117 SIODM-EXIT       000002 033122 060400 000013 000000 147072
171420   117 STARTIO          102461 000002 000132 133122 060000 147072
171410   117 SPECIALRQ        000000 000731 000000 000001 000000 000000
171400   117 SIODM-ENTRY      000002 033122 040000 000000 000000 147070
171370   271 QUIESCE          000140 000200 110676 000000 000000 147041
171360     0 SWAPIN           000271 140004 000000 000000 000000 000000
171350     0 FETCHSEG         000000 001565 000271 000000 000000 000000
171340     0 DEALLOCM         000000 000026 126030 000000 000000 000000
171330     0 FETCHSEG         000000 000666 000271 000001 000000 000000
171320   352 QUIESCE          020010 002000 122272 000000 000000 147027
171310   352 SIODM-EXIT       000004 006531 060400 000013 000000 147025
171300   352 STARTIO          101235 000004 000113 106531 060000 147025
171270   352 SPECIALRQ        000000 000314 000000 000001 000000 000000
171260   352 SIODM-ENTRY      000004 006531 040000 000000 000000 147023
171250   264 QUIESCE          020040 004000 122230 000000 000000 147010
171240   352 QUIESCE          020010 000000 122272 000000 000000 146770
171230   352 SIODM-EXIT       000001 000000 060013 000000 000000 146770
171220   352 SPECIALRQ        000000 000135 000003 000000 000000 000000
171210   352 SIODM-ENTRY      000001 035474 040213 000013 000000 146766
171200   352 GIPINTERUPT      000001 035474 040413 000017 000000 146766
171170   264 QUIESCE          030040 004000 122230 000000 000000 146763
171160   264 SIODM-EXIT       000001 035474 060400 000013 000000 146762
171150   264 STARTIO          102155 000001 000131 135474 060000 146762
171140   264 SPECIALRQ        000000 000135 000000 000001 000000 000000
171130   264 SIODM-ENTRY      000001 035474 040000 000000 000000 146761
           .                          .                    .
           .                          .                    .
           .                          .                    .
171670   352 SPECIALRQ        000000 000135 000003 000000 000000 000000
171660   352 SIODM-ENTRY      000001 002310 040213 000013 000000 146631
171650   352 GIPINTERUPT      000001 002310 040413 000017 000000 146630
```

```
171640   264 QUIESCE          030120 004000 122230 000000 000000 146625
171630   264 SIODM-EXIT       000001 002310 060400 000013 000000 146625
171620   264 STARTIO          102155 000001 000131 102310 060000 146625
171610   264 SPECIALRQ        000000 000135 000000 000001 000000 000000
171600   264 SIODM-ENTRY      000001 002310 040000 000000 000000 146612
171570    15 QUIESCE          000000 000400 140062 000000 000000 146567
```

Certain events which occur during the execution of MPE are logged in a table referred to as the Monitor table. Sometimes, formatting this table after a system hang can give you an idea of what past events were happening on the system just prior to the hang. In this case, there are many entries for PINs other than our candidate process (PIN %264). This would seem to indicate that other processes were getting some chances to perform work. If the Monitor table were filled mostly with entries that belonged to PIN %264, then this would be an indication that this was the last process that had a *chance* to run just prior to the system hang, and it may have been caught in some sort of loop at a high priority. Or, it could mean that the process was waiting for the completion of some system-related event, which then could also have been impeding other processes.

At this point, we can surmise that the system was attempting to perform I/O to logical device 23, and seemed to be having some sort of problem. Again, however, there is perhaps more information that should be gathered regarding this logical device. Does this logical device connect to the system via an ADCC or an ATP? Is there anything else about the connection of this device that is unusual or non-standard?

A listing produced by the SYSINFO utility would be able to show us what type of hardware this logical device uses. IDAT also has a command which will display configuration information about the system. To answer other questions about the connection of this device to the system, however, you may have to rely on a visual inspection. At some sites, as it is at ours, there may even be a separate department and staff that maintains the datacomm/network hardware. For the moment, let us continue with our information gathering concerning logical device 23.

Display the configuration of logical device #23:

```
-F CONFIG,23 (RETURN)

LDEV DRT  UNIT TYPE SUB PIN STATE CPVAO   PLABEL        CLASS     DSTATE
-----------------------------------------------------------------------
 23  11    0    16   0  264 Owned 000000  TERMONITOR    ADCC        00
```

This output shows us that, indeed, PIN %264 was the "owner" of this device at the time of the hang. It also shows us that the device is connected to the system via an ADCC. In this case, that determination was easy since a wise system manager configured the port with the device class: ADCC. At most sites, this would probably not be the case, so you would have to know how to tell the difference between an ADCC and an ATP with other information that IDAT would provide. Or, probably the most common utility that could be used for this task is SYSINFO. If the driver name for logical device 23 is named HIOTERM2, then the device is an ADCC; if the driver name is HIOTERM1, then the device is an ATP.

The fact that logical device 23 is on an ADCC is interesting because of the way that an ADCC handles terminal I/O. When an ADCC is "interrupted" in order to do I/O, or even perform handshaking, then the CPU is also interrupted. This differs from an ATP, where the processor board on the ATP can handle I/O "interrupts" without disturbing the processing of the CPU. In the case of this dump, it was necessary to also inform the Response Center that the actual terminal was connected through a combination of broadband network and MUX devices. Physically, it was located about two miles from the CPU.

This fact helped the Response Center determine that the ADCC for this device was, in fact, processing handshake characters when the system hung. It seemed, however, to be processing so many of them that the CPU was in a constant state of handling I/O to this device. At other user group meetings I have heard discussions about how a modem or other datacomm device that "strobes" can actually interrupt the ADCC so often as to tie up the entire system. According to the final analysis by the Response Center, that is what seemed to be happening in this example.

Example Summary

These two brief examples are intended to show you that you can gain valuable information from looking at your own memory dumps, even if you are not as trained as a Response Center engineer might be. If this discussion has stirred your interest in doing some rudimentary dump analysis for yourself, then this paper has been successful. If there are those who still want more than what was covered here, then you may have to seek out some individuals who have the knowledge *and the time* to give you further assistance. Ask around at the next user's group meeting that you have a chance to attend. Find out if there are knowledgeable people in your area who would be willing to share their skills with you.

Occasionally, HP also provides an opportunity for customers to attend a special "internals" class. I have been informed that one such class will be held in mid-1988 in California. The instructor is a gentleman from a third-party software vendor who was brought in especially for this class. If you would like to find out more information about the availability of this class, you should probably contact your local HP SE, at least as a starting point. S/he may be able to find out when and/or if the next class will be conducted.

Be aware, however, that no matter how good you may become at finding out information from a memory dump, there are many occasions when the Response Center engineers must consult the *compiled source of MPE* in order to determine the cause of some failures. Unless you also happen to have purchased the source for MPE, *and* it conforms to the version that you are running, you will probably also find that there will be a limit to how much dump analysis you can do on your own.

More about "Live" mode

IDAT can perform some functions that are similar to some other common utilities, both unsupported and supported. What follows are a few examples of useful information that IDAT can display about your system while it is up and running.

As was seen before, IDAT can format the contents of the JMAT (Job Master Table) in much the same manner as that of a `:SHOWJOB`. It can also format the contents of the Command Interpreter stacks, which shows us the last command that a job or session issued. Seeing the last command that was entered by a user is a capability that other utilities such as OPT/3000 can do. IDAT can also display information about a process that is even more technical than the information in OPT/3000. IDAT can also format information about the SIR table and RIN table, similar to OPT/3000 and other utilities.

Calling IDAT a poor-man's OPT/3000 might be stretching things a little, but there are some interesting things that you can see with IDAT in "live" mode. There are other tables whose information can be displayed in live mode, including the Timer Request List (TRL). Also, you can format the PCBX area of a stack. This will include the PXGLOBAL, PXFIXED and PXFILE area of a stack. Learning to decipher this information can lead to a determination as to what files are in use by a process and what their current status is.

IDAT also has the ability to format the Global Available File Table (GAFT), which is where entries are kept for files that are being shared by more than one process. The Segment Locality List (SLL) is another table which can be formatted.

Teaching you how to decipher each of these items is probably beyond the scope and intent of this paper. Consulting a Tables Manual is perhaps a good place to start. Or, you can also do some experimenting in order to become familiar with the information that IDAT presents. *User beware,* however. IDAT's only task that requires privileged mode while reading a memory dump on disc is that of being able to call the procedure DEBUGUTIL in order to decipher words into assembler instructions. In "live" mode, it needs privileged mode in order to read some of the areas of memory that are outside of its own stack. Be careful of older versions of IDAT while using "live" mode. Some early versions *could* cause problems for you, if you, for example, tried to display a word of memory that is outside the range of memory on your system.

In addition, if you are using IDAT in "live" mode, it is possible to format a table and receive some strange output. This is usually because you have chosen to format something which is dynamic in nature. This will not normally cause any problems to the system. If it does, however, your only alternative is to note to yourself not to do that again.

If you have any problems with IDAT malfunctioning, contact your local HP SE or the Response Center. Since IDAT is unsupported, however, HP will only attend to fixing reported problems at a low priority. At last report, the latest version of IDAT is being distributed on the same tape as the TELESUP utilities. IDAT also is one of the utilities that is included with each new version of MPE. Wherever you find it, or however you obtain it, you *can* learn to use this tool and perform some useful, interesting and helpful tasks.

IDAT Development History

The following is a brief attempt to reconstruct some of the history of IDAT as it was developed from 1982 until today. This information is presented in chronological order, to the best of my recollection. At this time, I would also like to give credit to many other individuals who have contributed or are, even now, still contributing to the success of IDAT as a dump analysis tool. Apologies are also extended to those whom I do not include in the list. Their omission is not intentional, but is more likely the result of an imperfect memory or of my own poor documentation. Thanks go to: Brian Di Silvestro, Costa Hasapopoulos and Mark Cousins, Santa Clara Response Center; Mark Hatfield, HP CEC, Mountain View, CA; Simon Cutting and Donna Gracyk, HP CSY; Marie Weston, HP Roseville, CA; Nelson Hall, HP ITG; Craig Myles, Ron Helms and Bernie Staley, HP St. Louis, MO; Jim Inghram, HP Cedar Rapids, IA (my district manager who allowed me to spend some time on IDAT development); Bruce Hemminger, Suresh Ganu and Mary Hindman, HP Cedar Rapids, IA (who put up with my ability to crash the office system in truly spectacular ways); and, of course, Bob Mead, now a manager at HP Labs, Cupertino, CA, who had the original idea for IDAT.

1982	Bob Mead develops the first pass at IDAT. Its capabilities include being able to display memory in a DEBUG-like format, along with being able to format a PCB entry and a stack.
1983	New programmers in the lab group add small amounts of functionality, such as formatting the SIR table. Staff limitations do not allow a great deal of development. Problem analysis still depends heavily on the use of DPAN.
Feb. 1984	Work starts on a separate version to handle MPE V/E memory dumps. Minor bug fixes are incorporated.
Mar. 1984	The routine to format the SIR table is rewritten, saving stack space. Use of the help file is made easier. Command parser is modified to be more user-friendly.
Jun. 1984	Octal display of memory is included when decoding machine instructions.
Aug. 1985	My first contact with IDAT while at HP. I use it to assist dump analysis occasionally. A few other SE's in the Midwest also become aware of IDAT.
Sep. 1985	I acquire a copy of IDAT which handles MPE V/E dumps, but find that supporting customers who are still on MPE IV can get confusing. Several SE's in the Midwest begin contributing suggestions for improving IDAT so that remote support can be performed quicker.
Dec. 1985	I begin toying with a copy of IDAT source. I find it to be the preferred tool for analyzing dumps, in many cases. Forcing a customer to box up a tape and/or a DPAN listing seems to be much too time consuming.
May 1986	U-Mit enhancements to SDF allow virtual memory to be copied to tape. IDAT is also modified to take advantage of this feature. At

last, we hardly ever find a critical process' stack unreadable because it was swapped out to disc when the system failed.

Jun. 1986	Other routines from Europe are found and incorporated which format certain I/O and file information. CSY in California has again committed some staff development time to IDAT, in view of its increasing importance to the Response Centers.
Aug. 1986	Added the `-F LMAP` command, which can rebuild from the contents of memory the same information about segments that would normally be found in LOADMAP.PUB.SYS. `REDO` capability added. Added the `-F MEM` command. Fixed a bug in the routine that checks for SIR deadlocks. Fixed the formatting of the current process' stack information for certain states.
Sep. 1986	Minor clean-up done for some formatted output. Added the `-F CKSUM` command. Added data integrity checks to some display routines.
Oct. 1986	Added the `-F TRL` command. Expanded the information displayed from formatting the ICS. Added formatting commands to handle disc caching tables. Added the capability to detect the Mirco-3000 CPU types. Added the `-F CONFIG` and `-F ARL` commands.
Nov. 1986	Modified the routines for caching to handle UB-Mit CDT changes. Fixed minor bugs in formatting banks of memory.

At this point, I began working for Boeing Computer Services, but the development of additional IDAT features has continued. As this paper is being written, another enhancement to IDAT has been released: a macro facility has been incorporated into the command parser.

Hopefully, you have found this information useful, or at least interesting. IDAT will probably continue to be a very heavily used utility when it comes to analyzing dumps from any MPE-based system.