APRIL/JUNE 1983
VOL. 6, NO. 2

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

CONTENTS

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

HP 3000 International Users Group
289 S. San Antonio Road, Suite 205
Los Altos, California 94022 USA
415/941-9960

HP 3000 INTERNATIONAL USERS GROUP BOARD OF DIRECTORS

Chairman
Sandra S. Bristow
Liberty Communications, Inc.
2225 Coburg Road
Eugene, Oregon 97401 USA
503/485-5611

Vice-Chairman
John True
Computer Center
Universtiy of Tennessee at Chattanooga
Chattanooga, Tennessee 37402 USA
615/755-4551

Secretary
Lana D. Famery
Quasar Systems Ltd.
275 Slater Street, 10th Floor
Ottawa, Ontario K1P 5H9 Canada
613/237-1440

Treasurer
Michael Lasley
Hinderliter Industries, Inc.
4524 E. 67th, Bldg. #9
Tulsa, Oklahoma 74135 USA
918/494-0992, ext. 303

Jane A Copeland
Tymlabs
211 East 7th Street
Austin, Texas 78701 USA
512/378-0611 (Austin)
512/340-6101 (San Antonio)

N.M. (N ck) Demos
Demos Computer Systems, Inc.
12 Hillsview Drive
Catonsville
Baltimore, Maryland 21228 USA
301/468-5693

Association Manager (ex officio)
Bill Crow
HP 3000 International Users Group
289 S. San Antonio Road, Suite 205
Los Altos, California 94022 USA
415/941-9960

Hewlett-Packard Representatives (ex officio)
Jan Stambaugh
Hewlett-Packard Company
Cupertino, California USA

Editor's Note

The second issue of the HP3000 International Users Group Journal features articles on Systems Management and Systems Security. These two issues are paramount for many of our users, and the articles included in this issue represent a cross section of the most recent work in these areas. As always the editor of the Journal is interested in your comments about the quality of these articles and about suggestions that you may have for other issues.

The third issue of the 1983 Journal will be devoted to Education and Educational Applications. Dr. Lloyd Davis, Associate Editor of the Journal and Chairperson of the Special Interest Group on Education will be guest editor for that issue. The fourth issue of the Journal will be devoted to HP3000 applications in Medicine. Dr. Ragnar Nordbert of the Department of Clinical Chemistry, The University of Gothenburg, Sahlgren's Hospital in Sweden, member of the Publications Committee and Chairperson of the Medical Special Interest Group will be guest editor for this issue.

The MPE Memory Dump; or How to Make a Statue of an Elephant

Jason M. Goertz
Hewlett-Packard
Bellevue, Washington

Introduction

In the past several years, the number of HP3000 sites has increased in number dramatically. The 3000 has been called into service to perform more and more complex and demanding applications. Applications that use Privileged Mode, Process Handling, Message files and other advanced features and capabilities are becoming almost commonplace. Along with this increasing application sophistication there has developed, necessarily, an increasingly sophisticated user community, who require more complex debugging aids and tools to facilitate development of these applications. It is for this class of user that this paper is written.

While developing this type of application, particularly ones using Privileged Mode, MPE integrity is sometimes compromised. Many times this results in some kind of system interruption, usually a system failure or hang. In almost all cases, in order to determine the cause of the problem, the MPE memory dump is the most concise, economical, and easy tool available. However, information on how to read and interpret the mountain of paper produced is virtually non-existent. Even within Hewlett-Packard, only recently has organized training on the subject been available.

This paper is an attempt to fill this information gap. Please note that MPE dump reading and (especially) interpretation requires in-depth knowledge of MPE and subsystem internals, not to mention a lot of practice and experience. This point cannot be stressed enough. This paper is not intended to replace any of these things, but to give a capsule summary of some of the more basic and important facts and methods.

An Explanation

Before beginning, the title of this paper must be explained. In the two years, and particularly the last year, that I have been reading dumps on a regular basis, I have evolved an answer to the two questions most often asked by people in my office, namely "What do all those ones and zeros mean?" and "What do you look for?" (The answer to the third question, "Do you really enjoy doing that?" is worthy of another paper, or at least a few hours of discussion accompanied by several doses of liquid refreshment.) The dialog which ensues after either of the above queries is something like:

> I: Do you remember elephant jokes?
>
> They: Sure.
>
> I: Remember the one about the statue?
>
> They: No
>
> I: Well, its like this: How do you make a statue of an elephant?
>
> They: I give. How?
>
> I: You take a hammer, a chisel, and a block of marble, and you knock away everything that doesn't look like an elephant. Reading a dump is basically the same idea. You take the dump, a Tables Manual, and PMAP's, and you find the part of the dump that doesn't look like MPE. At that point, you've found the problem.

It's then that the poor sod who "had to ask" usually walks away shaking his head in bewilderment. With this thought in mind, let us proceed.

Fundamentals

Before starting to read and interpret a Memory dump, it is necessary to understand exactly what one is.

When the system stops, for whatever reason, the contents of memory are "frozen" at that instant. In addition, the microcode of the machine dumps the value of the CPU registers (DB, Q, S etc.) into a special area of low memory. A serial medium, usually tape, is mounted and the contents of memory, starting at low addresses and proceeding through the highest word, are dumped serially to the medium. This is accomplished by either microcode (Series II/III, herewith referred to generically as SIO machines) or by software (SDFLOAD on Series 30/33/40/44/64 machines, herein referred to generically as HP-IB machines). After the machine is brought up, a program called DPAN4 is run under MPE control that reads the tape and formats the contents in a meaningful form. The resulting listing is what is commonly referred to as "the dump".

It is important to realize what this listing represents. It is basically a "picture" of MPE in memory. In essence, it is MPE, as much as any physical thing can "be" software. In order to interpret this "picture" of MPE it is critical that the interrelationships of the various parts be understood. Therefore, the very first thing that must be acquired to read a dump is a thorough understanding of the workings of MPE. This is not possible to do in the scope of this paper, but a few key facts and concepts will be presented.

The memory of the HP3000 is divided into sections or "banks" of 64KW each. Banks of memory are treated equally within MPE, with one exception, and that is bank 0. This is where MPE (specifically INITIAL) places most of its critical system tables. The reason for this is that, originally, the HP3000 was a 64KW machines, and all of MPE and user code were in this memory. All of the memory resident (nonswappable) tables are in this bank, especially the Code Segment Table, Data Segment Table, Process Control Block, IO and Disc Request Queues, and Memory Allocation Manager (MAM) tables. A great deal of information can be gained from understanding just the first 5 above, plus the format of the user stack.

Code and data is separated in memory, and are accessed in variable length "chunks" called segments. It is necessary that MPE, as well as the hardware (microcode) keep track of where in memory or on disc these segments are located. The CST and DST are used for this purpose. The Code Segment Table is really divided into two parts, the CST and the XCST (Extended Code Segment Table). The latter was introduced in the Series II when the increased memory size necessitated a larger storage of Code segment information. Each entry is four words of memory, and contains information on location (either bank and offset or disc address), whether it is in memory or not, and its length. Other data is also stored, such as whether it is memory resident, or (in the case of a Data segment), whether or not the segment is a processes's stack.

The CST is used to point to Code segments that are resident in an SL file. Program file segments are kept in the XCST. For each process, there is a bank of XSCST entries, each entry with the same format as the CST. CST's currently are numbered from 1-%277, and XCST's from %301 to %377. It is this numbering range that is used by the microcode to represent logically contiguous code space, as well as by DPAN4 and the dump reader to determine the origin of the segment. These tables provide data to determine exactly what was executing during and prior to the failure.

The Process Control Block (PCB) is used by the dispatcher to keep track of the various processes on the machine, and which one will run (be dispatched) next. (A process is defined as a unique execution of a program at a point in time.) A process will always have at least one Code and Data segment, plus a PCB entry which ties the whole thing together. The PCB also contains extremely valuable information to the dump reader, such as why a process was waiting (and what event it was waiting for), as well as whether the process was attempting to abort, where the DB register was, etc.

The primary IO tables, the IOQ and the DRQ, are a list of those IO's that are waiting to occur or have just occurred. The structure of the two tables is almost identical, although there is a bit more information in the DRQ. In order to fully interpret the IOQ it is necessary to have a good understanding of ATTACHIO (the software interface to the IO system), and the individual device driver. However, these two tables can be invaluable to the dump reader who is facing the analysis of a system hang.

The data structure which gives the best "history" of what lead to a failure is the process's stack. The stack data area is delimited at various places by the CPU registers DL, DB, Q,S, and Z. Below DL is the PCBX (PCB Extension) which is used by MPE to store noncritical scheduling information and is not accessable to user code. This area contains some relevant data structures and information, most notably file control block pointers, as well as pointers to two other important process tables, the Job Directory Table and Job Information Table (JDT and JIT).

When the program executes, it issues PCAL instructions which cause control to be transferred to another procedure, most often to a system segment, such as IMAGE or the filesystem. The PCAL instruction, as part of its normal operation, places a four word marker on the stack (at the current S pointer). This marker contains data which allows the environment at the time of the call to be preserved so that a proper return can be executed. The data includes the current value of the X, P, and Status registers, as well as the number of words between that marker and the previous one. We can see that if we start at the topmost marker and work backward, we would have a history of what code the process had executed until the time of the failure (if the process in question was the cause of the failure), or what the process was doing before it gave up the CPU. This is called a Stack Marker Trace, and DPAN4 formats it twice, once by itself in the formatted portion, and again when the whole stack is formatted.

A similar structure to the process stack is the Interrupt Control Stack. This is a stack that resides in low memory, and is used primarily by the IO drivers and the dispatcher. In the case of a Memory Manager or IO system failure, the ICS is examined the same way a normal stack is to determine what code was executed before the failure. Typically, if this data structure is involved in the examination of a dump, the problem is most likely an MPE problem, and therefore up to HP to analyze.

The formats for most of these tables can be found in the System Reference guide, in addition to a very detailed description of the interrelationships of the tables. The MPE tables manual (PN 32002-90003) provides a detailed description of the formats of the various tables, and a description of the various data elements stored in them.

Dump Format

The actual dump listing is divided roughly into two parts, commonly called the "formatted" portion and "unformatted" portion. In reality, both portions are formatted, the first part more elaborately and with more detail and intelligence than the second. The formatted portion consists of selected tables which

DPAN4 prints with the various fields labeled. Additionally, most of the various fields are printed with mneumonics (such as C for Core-resident, or S for Stack). The unformatted portion is just an octal dump of memory, starting at bank 0. The various tables are labeled if DPAN4 can determine their identity. All tables used for memory management that are in memory, such as the region trailers and headers, are printed in such a manner as to allow the reader to separate them from the corresponding segment. For each segment, the left hand side has not only the bank relative address, but the segment relative address also. For most data segments, the right hand side has the ASCII equivalent of the contents printed, with periods representing nonprintable characters. We will now discuss how DPAN4 formats the various tables mentioned above.

The first page (Figure 1) is called the Register Page. This gives a listing of all the CPU registers at the time of the system halt. The stack registers are on the left, followed by the Code Segment registers. Next are the X, Current Instruction Register (CIR), and other registers that vary among the different hardware types. An interpretation of the various bits and fields in the Status Register formatted in the next column, followed by the other hardware dependent registers. While the latter is sometimes of interest, especially when diagnosing hardware diagnosing hardware problems, the first three are more commonly used. Below the box containing the registers are contents of low memory. These words of memory are used by the microcode to mark the beginning of the various critical tables, such as the CST, DST, XCST, and the PCB. The ICS limits are stored here also.

One very useful piece of data stored in low memory is a pointer to the PCB entry for the currently running process. If this is nonzero, then a process was running, and it is usually the process which caused the failure, although it is possible to have a current process and also have something, such as an IO driver, running on the ICS. Code running on the ICS is indicated by several things on the register page. On all machines except the Series 64, the DL register (far left) being set to -1 (%177777) indicates that the current stack being used was the ICS. On the Series 64, as well as the other machines, there is a bit in the CPX1 register which DPAN4 formats in the last column of the register box. DPAN4 labels this the ICS FLAG, and is either on or off.

Following this, the CST (Figure 2) is formatted. When DPAN4 runs, it interrogates the file LOADMAP.PUB.SYS to determine the names of the segments. These are printed out on the line, along with all of the other data from the CST. Next is the XCST (Figure 3), which is formatted by groups, each group representing the XCST entries for a particular process.

The DST (Figure 4) is listed in almost identical format to the CST, with the names of the various sytem tables being printed on the appropriate lines.

Next is the PCB. This is divided into two halves, as there is too much data in each entry to be formatted on one line. The first half of the PCB (Figure 5) shows the process tree information, the wake and event masks (used by the suspend and activate mechanism within MPE), plus the psuedo interrupts that the process has accumulated, such as from a break, control-y, or an :ABORTJOB executed on that process's job/session. The second half (Figure 6) has scheduling information, used primarily by the dispatcher, bits which show what resources (SIRs, SETCRITICAL) the process holds. In addition, various pointers and other data are formatted.

The IOQ and DRQ (Figures 7 and 8) are similar in format. DPAN4 formats each in two parts, an "in-use" list and an "available" list. The inuse list for the DRQ is additionally divided into a list for each disc configured on the system. For the dump reader, the available list is a recent history of IO activity on the system, sometimes giving a clue to the cause of the failure, or at least to what the failing process was doing. The inuse list can give invaluable data as to what IO's were pending and why they had or had not completed, as well as the relative order in which they had been queued.

The data structure which DPAN4 does the most work on is probably the data stack. As DPAN4 is dumping main memory, (the "unformatted" portion of the dump), it checks each data segment that it encounters to see whether or not it is a data stack. If it is, it formats several pieces of data from the bottom of the stack, an area known as the PXGLOB area. This data is very useful to quickly identify several things about the process, such as what $STDIN/$STDLIST device was assigned, what Job/Session number was assigned, and what the JIT and JDT dst numbers are for that process. After this, the stack markers are repeated, and the PXGLOB, PXFIXED, and PXFILE areas are printed. (See Figure 9). DPAN4 delimits and labels the various file control blocks, in the PXFILE area. When a location of memory is reached which is pointed to by a CPU register (for that process), DPAN4 prints a line of asterisks and labels this register. This is also done for each stack marker as it is encountered. Alongside the marker, the segment name is printed, just as it was in the stack marker trace, above.

Additionally, DPAN4 prints a "table of contents" at the beginning of each bank of memory. At the end of the dump, it produces a list of the main tables, and the page numbers on which that table appears in both the formatted and unformatted portions.

We now have at least a passing familiarity with the format of the dump and with the functions of the tables that the dump represents. Let us now discuss how to use this information.

System Interruptions

There are five types of system interruptions, and are defined as follows:

1. *System Failure.* This is caused when some code, usually MPE, detects some error condition, and calls a procedure called SUDDENDEATH. This procedure prints the all too familiar system failure banner, and halts the machine. These failures can be caused by a hardware problem which the software detects at a later time, a system table that has been altered in a way that causes integrity loss in MPE, or sometimes by an invalid parameter passed to a system primitive.

2. *System Hang.* This is when the system is in a pause state, but no response can be obtained from terminals. Many times, the system will hang when users try to logon or logoff, or run a program. In the case of a hang, the hardware is running, but the software cannot run for some reason. It is important that the exact symptoms of the hang be known. Without this knowledge, it is often difficult to know where to start looking in the dump for the cause.

3. *System Loop.* This occurs when a high priority process, such as a system process or datacomm monitor process (or user process in linear queue) gets into a tight loop, and does not allow another process to run. Another possible cause is a process which PDISABLE'd (turned off process dispatching), and has not PENABLE'd properly.

4. *Silent Halt.* This occurs when the microcode detects an "impossible" condition, such as an ICS overflow. These types of halts are silent only on SIO and Series 30/33 machines. On SIO machines, this usually will cause the System Halt light to come on. Other HP-IB machines will print a HALT n message, where n is a number which indicates the type of halt encountered. Most often, this indicates a hardware problem.

5. *Port lockout.* A particular port will not respond. Usually, this is an application problem. Most often, this is associated with a process handling application, or a problem with a specific peripheral.

We will examine each of these dump types, and summarize what to look for in the dump.

Some Tools

Before delving into the actual analysis of the dump, it is necessary to accumulate a few tools which make the dump reading process easy. Besides the dump, it is necessary to have the PMAPs of the various MPE modules, and a current MPE Tables Manual. The Tables manual can be ordered from HP, PN 32002-90003. A true PMAP listing of the MPE modules is only attainable by doing a PREP on the various MPE modules. Since this is not possible for most users, besides being very difficult and time consuming, an easier method is necessary. A program is available called SLPMAP, which reads an SL file (usually the system SL) and produces a PMAP-like listing for each segment of the SL, in alphabetical order. While the segment locations are not totally accurate, they are close enough to locate the procedure which was executing from a stack marker trace. If a particular application is involved, especially one that utilizes Privileged Mode, the source code and PMAPS for the code involved needs to be gathered as well.

Analysis

The first thing that must be done when analyzing a dump is to determine if the dump was even valid. Sometimes the contents of memory is so corrupt that DPAN4 cannot determine where certain tables are in memory. When this occurs, a diagnostic message is printed out, and a list of the exact tables that could not be formatted is given. DPAN4 will then say that it is suspending the formatted portion of the dump, and that memory will be printed in an unformatted manner. At this point, DPAN4 prints an octal dump of memory starting at bank 0 and proceeding to the end of the data that was on the tape. This dump is virtually worthless. It would be extremely tedious and time consuming to try to analyze this dump, and it is even a waste to print it.

The next thing that must be done is to determine what type of failure occurred. Typically, if a user is analyzing a dump that occurred on his or her system, then the type of failure will be painfully known. Assuming the type is not known, or the type is uncertain, the following analysis should be done:

1. Look at the current CST number, and determine if the segment HARDRES was executing. This can be determined by checking the formatted CST table or the file LOADMAP.PUB.SYS. If so, then SUDDENDEATH was probably called. The PMAP can be checked to confirm this. Then check the Current Process Pointer. If non-zero (and the other CPU registers do not indicate that some code was running on the ICS) then the failure was most likely caused by a particular process. If the registers indicate that something was on the ICS, then an IO driver or the dispatcher/Memory Manager probably caused the failure.

2. If the Current Process Pointer is non-zero and the current CST register is not HARDRES, or if it is and the current P register is not in SUDDENDEATH, then the dump is probably a system loop. Information from the site is invaluable in this case.

3. If there is no current process and the CIR register contains the PAUS instruction, then the dump is either a hang or a port lockout. Information from the site makes the analysis much easier for this type of dump.

4. If the CIR register contains an instruction other than a HALT or PAUS, then this is most likely a silent halt. It is very difficult, however, to tell this dump from a system loop. Halts are usually caused by a few machine instructions, and analysis of the specific instruction is necessary in this case. Site specific infor-

ation, such as configuration, is usually necessary to analyze this type of failure.

Determining whether the problem is hardware or software is sometimes very difficult, since hardware problems can often manifest themselves as software. There are a few failures which are readily identified as hardware just from the description. For example, SF15 is typically caused by a nonresponding hardware module, such as memory controller or GIC on HP-IB machines. SF201 is the same thing on SIO machines. Many types of hangs are caused by the disc sub-system or perhaps by communications boards. In any case, there are no hard and fast rules for determining whether something is hardware or software. The common types of hardware failures will be mentioned in the following discussion.

Figures 1 and 10–12 show an example of the first type of failure. The System Failure number is 311, which indicates process aborting while critical. In this case, the program is one called BADLABEL. The source code was modified to produce this failure. Figure 1 is the register page from the corresponding dump. Notice that the current process pointer is non-zero, and the DST registers are pointing to a stack, and that the DL register is not -1 (%177777). All of these things point to the fact that a specific process was the cause of the failure. Figure 10 shows the stack markers from that DPAN4 found to be the current process. If the process's name was not known, a simple technique can be used to find the name. The PCB entry is found, and the father's pin number determined. Assuming the father is a UMAIN (Command Interpreter) process, and that its stack is in memory, the :RUN or subsystem command can be found at DB+1 in that stack. We see from the current stack markers that the last user segment to execute was %301 at address %1510. Looking at the PMAP for this "application" (Figure 11) we see that the procedure that was executing was PROCESS'ENTRY. Subtracting the address of %1510 from the starting address in the PMAP of %1024 we get a net result of %464 which points to the actual line of code that caused the failure. Looking at the source (Figure 12), we see that the problem was that the QUIT intrinsic was called. Notice that the call to RESETCRITICAL is commented out. (We contrived the failure, remember?) Thus, the problem. The process was still SETCRITICAL when the process quit, hence the SF311.

For the user with an application problem that directly causes a failure, the above steps summarize the basic steps necesary to diagnose this failure from a dump. As long as the application source is available, it should be relatively simple to find the exact line of code which is causing the problem.

Figures 13 and 6 show an example of the second type of problem, a system loop. This dump was generated by writing a simple program which entered the linear queue, then went into an infinite loop. The halt button was pressed. Notice on the register page that the Current Process pointer is non-zero, the current CST is %301 (program segment), and the current instruction (CIR register) is %140000, which is a BR P-O instruction. (You can't get a loop much tighter than that!) Looking at the PCB (Figure 6), we see the current (starred) PCB is at priority %30, indicating a linear queue. From there, all that would be necessary would be to look at the stack markers to determine the exact nature of the loop.

The third example was generated by starting several programs doing disc IO's, then taking the disc offline. This simulated a hardware disc hang. The CIR on the register page (Figure 14) is %030020 which is the PAUS instruction. If this instruction is present, then that means the hardware was not being asked to perform any work. If it is known that the application was indeed trying to run, then the obvious problem was that it could not for some reason. Determining the reason is sometimes very difficult, but there are a few things that can be checked:

1. Check the PCB. Check what wait bits are set. If the SW bit is on for several processes (Short Wait) then they were trying to perform disc IO's, and the DRQ and disc subsystem should be checked.

2. Check the SIR tables for SIR deadlocks. If this is so, then the problem is most likely an involved MPE bug, unless PM code is involved.

3. If the PCB shows the processes to be waiting for Global Rins (RG bit on) then the problem is most likely a file locking deadlock. This occurs when multiple files are being locked. (MR capability granted).

If we examine the Disc Request Queue (Figure 15) we see that there are several processes waiting to perform disc IO's to LDEV 1. Looking at the Device Information Table for that LDEV (Figure 16), we can see that the device is offline. This is determined by examining the hardware status words and interpreting them from the CE handbook.

The port lockout is considered a subset of the hang. A separate example will not be given.

The fourth and last example shows what happens when Priv Mode Debug is used to modify absolute memory location 0 to the value 0. (This was used to generate this failure). This particular memory location is used by the microcode to delimit the beginning of the CST table. If this points to a place that does not look like the CST, the microcode will halt the system. The main thing to look for in this type of dump is the CIR value. The type of instruction being executed would indicate what exactly was wrong. This is done by understanding exactly what the particular instruction does, and knowing what data in memory is used by that instruction. This tells what the actual instruction was that halted the system. Here, in Figure 17, we see that the current instruction was %31051, which is a PCAL 51. The PCAL instruction

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

must examine the CST in order to determine the location in memory to branch to. We can see on the register page that low memory location for the CST pointer was zero. If we were to further examine the TBUF's (not shown), we would see the MAO command, plus the DEBUG PRIV . . . message. This would be a sure fire tipoff as to the exact source of the problem.

Summary

This paper has attempted to give a very simple description of a very complex area of the HP3000. It should be apparent that the most important attribute that one can have when attempting to read and interpret a memory dump is a great understanding of the workings of MPE. The greater the understanding, the easier the dump is to read. Beyond this, familiarity with the exact format and listings produced by DPAN4 is an invaluable aid. Finally, a great deal of experience must be acquired before one can truly be considered an effective dump reader and interpreter.

Figure 1.

```
HP3000 III MEMORY DUMPC 0) 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83.  5 08AM
(C) HEWLETT-PACKARD CO  1980

                              ******   REGISTERS   ******

**************************************************************************************************************
*  DATA SEGMENT       * CODE SEGMENT     * MISCELLANEOUS * STATUS = 103033    * ISR = 100000                         SERIES 30,33 *
**************************************************************************************************************
*                     *                  *               *                    *                                            *
*  DB BANK  = 000004  * PB      = 073020 * X   = 000000  * MODE       = PRIV  * RUN/HALT   = RUN   STACK OVR     = OFF    *
*  DB       = 010623  * P       = 103674 * CIR = 030377  * INTERRUPTS = OFF   * IRQ        = ON    BNDS OVR/UNF  = OFF    *
*  S BANK   = 000004  * PL      = 115347 * NIR = 000000  * TRAPS      = OFF   * CSRQ       = OFF   VIOL CODE     = NONE   *
*  DL       = 010467  * PBBANK= 000000   *               * STACK OP   = LEFT  * PARITY     = OFF   DISABLE ATN   = OFF    *
*  Q        = 016525  * (P-PB)= 010654   *               * OVERFLOW   = OFF   * POWERFAIL  = OFF                          *
*  S        = 016534  *                  *               * CARRY      = ON    * POWERON    = OFF                          *
*  Z        = 021206  *                  *               * COND CODE  = CCE   * DISP FLAG  = OFF                          *
*                     *                  *               * SEGMENT #  =  33   * ICS FLAG   = OFF                          *
**************************************************************************************************************

********** HALT  17

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

(ADDR  0)  CODE SEGMENT TABLE POINTER                  030110
(ADDR  1)  EXTENDED CODE SEGMENT TABLE POINTER         100000
(ADDR  2)  DATA SEGMENT TABLE POINTER                  024110
(ADDR  3)  PROCESS CONTROL BLOCK BASE                  037510
(ADDR  4)  CURRENT PCB POINTER                         040210
(ADDR  5)  INTERRUPT STACK BASE                        041610
(ADDR  6)  INTERRUPT STACK LIMIT                       042606
(ADDR  7)  INTERRUPT MASK                              067600
(ADDR 10)  DRT BANK                                    000000
(ADDR 11)  DRT ADDR                                    000000
```

Figure 2.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/06/83. 2.28AM
(C) HEWLETT-PACKARD CO 1980

****** CST TABLE ******

SEGMENT NUMBER	SEGMENT NAME	MODE	REFERENCE BIT	TRACE	SEGMENT LENGTH	ABSOLUTE ADDRESS	BANK/ /LDEV	DISC ADDRESS	ROC	IMI	SYS	CRES
1	ININ	PRIV	ON	OFF	4674	164524	0				S	C
2	FILESYS1 (0)	PRIV	ON	OFF	10774	006623	7				S	
3	FILESYS4 (1)	PRIV	ON	OFF	3574	166623	1				S	
4	FILESYS5 (2)	PRIV	ON	OFF	4470	162023	1				S	
5	FILESYS6 (3)	PRIV	ON	OFF	5430	154223	1				S	
6	FILESYS6A (4)	PRIV	ON	OFF	12504	034023	7				S	
7	FILESYS7 (5)	PRIV	ON	OFF	6100	157223	7				S	
10	CIALTORG (6)	PRIV	ON	OFF	10570	104023	5				S	
11	CICOMSYS (7)	PRIV	OFF	OFF	4220		1	36573			S	
12	CIERR (10)	PRIV	ON	OFF	2700	113023	1				S	
13	CIFILEB (11)	PRIV	OFF	OFF	10750	141623	7				S	
14	CIFILEM (12)	PRIV	ON	OFF	3304	174223	1				S	
15	CIINIT (13)	PRIV	ON	OFF	7644	157623	5				S	
16	CILISTF (14)	PRIV	OFF	OFF	6500	017423	2				S	
17	CIMISC (15)	PRIV	OFF	OFF	4504		1	37064			S	
20	CIORGMAN (15)	PRIV	ON	OFF	6310	044423	3				S	
21	CIPREPRUN (17)	PRIV	ON	OFF	6424	166423	2				S	
22	CISUBS (20)	PRIV	ON	OFF	4520	142023	5				S	
23	CISYSMGR (21)	PRIV	OFF	OFF	7624		1	37240			S	
24	CIUSERUTIL (22)	PRIV	ON	OFF	4540	065023	5				S	
25	CXSTOREST (23)	PRIV	OFF	OFF	6440	062623	3				S	
26	RESTORE (24)	PRIV	OFF	OFF	6640	025223	1				S	
27	STORE (25)	PRIV	OFF	OFF	11744	000023	5				S	
30	DIRC (26)	PRIV	ON	OFF	7510	046623	7				S	
31	ALLOCATE (27)	PRIV	ON	OFF	6454	125023	5				S	
32	ALLOCUTIL (30)	PRIV	ON	OFF	6434	000023	7				S	
33	HARDRES (31)	PRIV	ON	OFF	22330	073020	0				S	C
34	TERMRES (32)	PRIV	ON	OFF	20050	115350	0				S	C
35	ABORTDUMP (34)	PRIV	ON	OFF	7060	012223	5				S	
36	MESSAGE (35)	PRIV	ON	OFF	5000	021423	6				S	
37	PROCSEG (36)	PRIV	ON	OFF	7670	024023	7				S	
40	SOFTIO (37)	PRIV	OFF	OFF	20260		1	40267			S	
41	HPIO (40)	PRIV	ON	OFF	14664	137223	1				S	
42	PCREATE (41)	PRIV	ON	OFF	10140	152223	6				S	
43	MORGUE (42)	PRIV	ON	OFF	4600	155423	7				S	
44	BIPC (43)	PRIV	ON	OFF	3524	152523	7				S	
45	IPC (44)	PRIV	ON	OFF	11710	042423	5				S	
46	CHECKER (45)	PRIV	ON	OFF	1764	127623	1				S	
47	UTILITY1 (46)	PRIV	ON	OFF	5014	132023	1				S	
50	UTILITY2 (47)	PRIV	OFF	OFF	6664		1	40717			S	
51	RINS (51)	PRIV	ON	OFF	3644	020023	7				S	
52	JOBTABLE (52)	PRIV	ON	OFF	5150	105623	4				S	
53	DEBUG (53)	PRIV	ON	OFF	20554	021423	5				S	
54	NURSERY (54)	PRIV	OFF	OFF	7570		1	41171			S	
55	SPOOLING (57)	PRIV	OFF	OFF	23150	105423	6				S	
56	SPOOLCOMS1 (60)	PRIV	OFF	OFF	10350	163223	6				S	
57	SPOOLCOMS2 (61)	PRIV	OFF	OFF	12010		1	41513			S	
60	PVCOMSEG (62)	PRIV	OFF	OFF	3610		1	41566			S	

Figure 3.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/06/83. 2:28AM
(C) HEWLETT-PACKARD CO 1980

326	14	USER	OFF	OFF	5620		1	1123754
327	14	USER	OFF	OFF	4100		1	1124004
330	14	USER	OFF	OFF	5704		1	1124025
331	14	USER	OFF	OFF	11610		1	1124055
332	14	USER	OFF	OFF	13230		1	1124125
333	14	USER	OFF	OFF	6670		1	1124203
334	14	USER	OFF	OFF	12510		1	1124237
335	14	USER	OFF	OFF	15760		1	1124312
336	14	USER	OFF	OFF	4704		1	1124402

SEGMENT NUMBER	CSTBLK/PROCESS INDX	MODE	REFERENCE BIT	TRACE	SEGMENT LENGTH	ABSOLUTE ADDRESS	BANK/ /LDEV	DISC ADDRESS	ROC	IMI	SYS	CRES
301	15	USER	OFF	OFF	17734		1	1102157				
302	15	USER	OFF	OFF	17520		1	1102257				
303	15	USER	OFF	OFF	17714		1	1102356				
304	15	USER	OFF	OFF	17740		1	1102456				
305	15	USER	OFF	OFF	17700		1	1102556				
306	15	USER	OFF	OFF	5424		1	1102656				
307	15	USER	OFF	OFF	16544		1	1102705				
310	15	USER	OFF	OFF	17010		1	1103000				
311	15	USER	OFF	OFF	16644		1	1103075				
312	15	USER	OFF	OFF	16714		1	1103171				
313	15	USER	OFF	OFF	14744		1	1103265				
314	15	USER	OFF	OFF	17104		1	1103351				
315	15	USER	OFF	OFF	16030		1	1103446				
316	15	USER	OFF	OFF	15420		1	1103537				
317	15	USER	OFF	OFF	16514		1	1103626				
320	15	USER	OFF	OFF	17604		1	1103721				
321	15	USER	OFF	OFF	17424		1	1104021				
322	15	USER	OFF	OFF	16754		1	1104120				
323	15	USER	OFF	OFF	17434		1	1104214				
324	15	USER	OFF	OFF	17714		1	1104313				
325	15	USER	OFF	OFF	17550		1	1104413				
326	15	USER	OFF	OFF	17524		1	1104512				
327	15	USER	OFF	OFF	16434		1	1104611				
330	15	USER	OFF	OFF	12504		1	1104704				

SEGMENT NUMBER	CSTBLK/PROCESS INDX	MODE	REFERENCE BIT	TRACE	SEGMENT LENGTH	ABSOLUTE ADDRESS	BANK/ /LDEV	DISC ADDRESS	ROC	IMI	SYS	CRES
301	16	USER	OFF	OFF	17314		2	454706				
302	16	USER	OFF	OFF	17624		2	455004				
303	15	USER	OFF	OFF	17410		2	455104				
304	16	USER	OFF	OFF	17534		2	455203				
305	16	USER	OFF	OFF	17510		2	455302				
306	16	USER	OFF	OFF	17564		2	455401				
307	16	USER	OFF	OFF	17474		2	455500				

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

Figure 4.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/06/83. 2 28AM
(C) HEWLETT-PACKARD CO 1980

****** DST TABLE ******

SEGMENT NUMBER	SEGMENT DESCRIPTION	REFERENCE BIT	SEGMENT LENGTH	ABSOLUTE ADDRESS	BANK/ /LDEV	DISC ADDRESS	DCV	ROC	IMI	STK	MOD	FWP	SYS	CRES	WD	VM ALLOC
1	(CODE SEGMENT TABLE)	OFF	1400	030110	0								S	C		0
2	(DATA SEGMENT TABLE)	OFF	4000	024110	0								S	C		0
3	(PROCESS CONTROL BLOCK)	OFF	2000	037510	0								S	C		0
4	(CST EXTENSION)	OFF	6000	001510	0								S	C		0
5	(SYSTEM GLOBAL AREA)	OFF	200	001000	0								S	C		0
6	(FIXED LOW CORE)	ON	4000	000000	0								S	C		0
7	(INTERRUPT CONTROL STACK)	OFF	1100	041510	0								S	C		0
10	(SYSTEM BUFFERS)	ON	2020	057514	0								S	C		0
11	(UCOP REQUEST QUEUE)	ON	64	177623	7								S			1
12	(PROCESS-PROCESS COMMUNICATION TABLE)	ON	200	141423	5								S			1
13	(I/O QUEUE)	OFF	1310	042610	0								S	C		0
14	(TERMINAL BUFFERS)	OFF	17750	002130	0								S	C		0
15	(LOGICAL-PHYSICAL DEVICE TABLE)	ON	154	070210	0								S	C		0
16	(LOGICAL DEVICE AND CLASS TABLE)	ON	1234	172523	1								S			2
17	(DRIVER LINKAGE TABLE)	OFF	230	000440	0								S	C		0
20	(I/O RESOURCE TABLES)	OFF	14	000670	0								S	C		0
21	(SECONDARY MSG TABLE)	OFF	200	065320	0								S	C		0
22	(LOADER SEGMENT TABLE)	ON	3744	103223	2								S			30
23	(TIMER REQUEST LIST)	OFF	144	070364	0								S	C		0
24	(DIRECTORY)	ON	2000	103223	6								S			3
25	(DIRECTORY SPACE)	ON	600	177023	5								S			1
26	(PIN TABLE)	ON	504	030623	2								S	C		0
27	(SWAPTABLE)	OFF	2400	061534	0								S	C		0
30	(JOB PROCESS COUNT)	ON	20	000750	0								S			14
31	(JOB MASTER TABLE)	ON	200	162423	5								S			1
32	(TAPE LABEL TABLE)	ON	710	176223	4								S			0
33	(LOG TABLE)	OFF	650		1	6136	D						S			3
34	(REPLY INFORMATION TABLE)	OFF	2000		1	6237	D						S			1
35	(VOLUME TABLE)	ON	144	177023	0								S			1
36	(BREAKPOINT TABLE)	OFF	550		1	7117	D						S			1
37	(LOG BUFFER 1)	OFF	400	173023	3								S			1
40	(LOG BUFFER 2)	ON	400	057023	4								S			0
41	(LOG ID TABLE)	OFF	580		1	6132	D						S			1
42	(ASSOCIATION TABLE)	OFF	564	173623	6								S			0
43	(CST BLOCK)	OFF	120	064134	0								S	C		0
44	(JOB CUTOFF TABLE)	OFF	74	070530	0								S	C		1
45	(SYSTEM DIT)	ON	100	012023	5								S			0
46	(SPECIAL REQUEST TABLE)	OFF	144	064254	3								S	C		0
47	(VIRTUAL DISK SPACE TABLE)	OFF	344	065640	0								S	C		0
51	(ARSBM TABLE)	OFF	44	000704	0								S	C		0
52	(ILT)	OFF	11354	046140	0								S	C		0
53	(SIR TABLE)	OFF	170	070624	0								S	C		0
54	(FILE MULTI-ACCESS VECTOR)	ON	200	177223	4								S			2
55	(INPUT DEVICE DIRECTORY)	ON	400	174623	6								S			40
56	(OUTPUT DEVICE DIRECTORY)	ON	400	133623	5								S			40
57	(WELCOME MESSAGE #1)	OFF	1750		1	6767	D						S			2

Figure 5.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/06/83. 2 28AM
(C) HEWLETT-PACKARD CO 1980

****** PROCESS CONTROL BLOCK (1ST HALF) ******

WAIT STATE

PIN	DATA SEGMENTS: XDS	ADB	STK	OVALC	FAMILY TREE: FTHR PIN	SON PIN	BRO PIN	OA
1			105			2		S
2		A	74		1		3	
3			75		1		4	
4			76		1		5	F
5			77		1		6	F
6			100		1		7	F
7	11		101		1	37	10	F
10			102		1		11	
11			103		1		12	
12			104		1		14	
14			120		1		15	S
15			117		1			S
37			121		7	64		S
64 •			133		37			F

PIN	WAKEMASK: M	RG	RL	MA	BIO	IO	UCOP	JUNK	TIMER	MSG	SON	FATHR	IMP	SIR	TMOUT	MEM
1								J								
2																
3						I			T	M						
4								J								
5													I			
6								J								
7								J								
10								J								M
11						I		J								
12								J								
14											S	F				
15											S	F				
37											S					
64 •																

PIN	EVENTFLAGS: M	RG	RL	MA	BIO	IO	UCOP	JUNK	TIMER	MSG	SON	FATHR	IMP	SIR	TMOUT	MEM
1																
2								J								
3																
4																
5																
6																
7																
10																
11																
12																
14																
15																
37																
64 •																

PIN	PSEUDO INTERRUPTS: PSIM	HK	SK	ST	HB	CY	BK	RITBK	PIOVR	MISC: PTYPE	CRIT	HSTR
1	NORM									SYST		
2	NORM									SYST		
3	NORM									SYST		
4	NORM									SYST		
5	NORM									SYST	C	
6	NORM									SYST		
7	NORM									SYSTJ		
10	NORM									SYST		
11	NORM									SYST		
12	NORM									SYST		
14	NORM									SYST		
15	NORM									SYST		
37	NORM									UMAIN		
64 •	NORM									USONM		

Figure 6.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/06/83 2.28AM
(C) HEWLETT-PACKARD CO 1980

****** PROCESS CONTROL BLOCK (2ND HALF) ******

----- S C H E D U L I N G I N F O R M A T I O N -----																						---RESOURCES---					LIFE/ DEATH			-------------- MISCELLANEOUS --------------						
PIN	NOPIN	POPIN	DISPQ	LQ	CQ	DQ	EQ	INTER	CORER	PRI	HIPRI	USERQ	TRW	SW	LW	MP	PC	IPEXP	MSPRI	SAR	SOV	CRIT	CHSIR	PREV IMPO PIN	NEXT IMPO PIN	SC	LIVE	DEAD	FAC	BMS	PPC	PCST	PBXPTR	SLLPTR	BPT LNK	SYSTEM PROC NAME
1				L						81																	L			SNF	NUL		10	60673		PROGEN
2		64	D	L						82																	L			SNF	NUL			60541		SYSIO
3				L						170																	L			SNF	NUL			60553		IOMESS
4				L						62																	L			SNF	NUL		1	60565		LOG
5				L						175												C					L			SNF	NUL		2	60577		MEMLOG
6				L						175																	L			SNF	NUL		3	60611		
7				L						175																	L			SNF	NUL		4	60623		UCOP
10				L						12										S							L			SNF	NUL		5	60635		PFAIL
11				L						175																	L			SNF	NUL		6	60647		DEVREC
12				L						218																	L			SNF	NUL		7	60661		LOAD
14				L						230					L												L			SNF	NUL			61513		
15				L						230					L												L			SNF	NUL			61537		
37					C			I		230					L											S	L			SNF	NUL			62152		
54 *	2		D	L				I		130																	L		F	SNF	NUL		26	50712		

100 ENTRYS
61 UNASSIGNED ENTRYS
17 ASSIGNED ENTRYS

Figure 7.

HP3000 III MEMORY DUMPC 00 05 OF SYS VER C UPDATE E0 FIX 20 DUMP TIME 2/09/83. 6 32AM
(C) HEWLETT-PACKARD CO 1980

TABLE INDEX	LOGICAL DEVICE	PCB	ADDR REL	DST	BUFFER ADDRESS	FUNC	COUNT	PARM1	PARM2	MISC	FLAGS	STATUS DESCRIPTION	STATUS
357	20	23	+08	134	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
344	20	30	+08	141	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
331	20	30	+08	141	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
316	20	25	+08	137	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
303	20	25	+08	137	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
270	20	30	+08	141	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
10	20	30	+08	141	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
242	20	23	+08	134	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
227	20	23	+08	134	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
214	20	30	+08	141	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
153	20	30	+08	141	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
140	22	34	+08	152	0	000011	0W	000000	000000	000000	007000 IW BL CO	NORMAL COMPLETION	1
77	22	34	SBUF	10	0	000005	0W	000017	000000	000000	017000 SB IW BL	NORMAL COMPLETION	1
125	22	34	+08	152	0	000021	0W	000000	000000	000000	007000 IW BL CO	NORMAL COMPLETION	1
112	22	34	+08	152	0	000015	0W	000000	000000	000000	007000 IW BL CO	NORMAL COMPLETION	1
64	22	34	+08	152	337	WRITE	138	000320	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
23	22	34	+08	152	410	WRITE	1168	000320	000004	000032	007000 IW BL CO	NORMAL COMPLETION	1
51	20	30	+08	141	445	READ	0W	000001	000000	000043	007000 IW BL CO	NORMAL COMPLETION	1
36	20	30	+08	141	445	WRITE	1W	000053	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
1262	22	34	+08	152	337	WRITE	1228	000320	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1
1275	22	34	+08	152	532	WRITE	238	000320	000004	000000	007000 IW BL CO	NORMAL COMPLETION	1

****** I/O REQUEST TABLE (IN USE LIST) ******

TABLE INDEX	LOGICAL DEVICE	PCB	ADDR REL	DST	BUFFER ADDRESS	FUNC	COUNT	PARM1	PARM2	MISC	FLAGS	STATUS DESCRIPTION	STATUS
42755	22	34	+08	152	337	READ	18	100001	000000	000002	006000 IW BL	PENDING	0

Figure 8.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83   6:32AM
(C) HEWLETT-PACKARD CO  1980

                    ******** DISC REQUEST TABLE ********  (AVAILABLE LIST)

                                                  STATUS  0 XX -> PENDING
                                                          1 XX -> SUCCESSFUL
                                                          2 XX -> END OF FILE
                                                          3 XX -> UNUSUAL CONDITION
                                                          4 XX -> IRRECOVERABLE ERROR
```

TABLE INDEX	LDEV	UNIT	PCB	S S	DST/ BANK	OFFSET/ ADDRESS	FUNC	XFER CNT	PARM1	PARM2	MISC	SEG IDENT	SEGDSP	NXTAVL	- F L A G S - MAIN	AUX	STATUS
002000	2	0	26	S	137	000023	WRITE	1	000000	060647	000001			0	007010	003010	1 0
000120	2	0	23	S	134	000023	WRITE	1	000000	060617	000001			2000	007010	001430	1 0
001560	2	0	26	S	137	000023	WRITE	1	000000	060646	000001			120	007010	003070	1 0
000250	2	0	23	S	134	000023	WRITE	1	000000	060616	000001			1560	007010	001570	1 0
000040	2	0	26	S	137	000023	WRITE	1	000000	060645	000001			250	007010	001350	1 0
000300	2	0	23	S	134	000023	WRITE	1	000000	060615	000001			40	007010	001610	1 0
001160	2	0	26	S	137	000023	WRITE	1	000000	060644	000001			300	007010	002470	1 0
000360	2	0	23	S	134	000023	WRITE	1	000000	060614	000001			1160	007010	001670	1 0
000700	2	0	26	S	137	000023	WRITE	1	000000	060643	000001			360	007010	002210	1 0
001540	2	0	23	S	134	000023	WRITE	1	000000	060613	000001			700	007010	003050	1 0
001700	2	0	26	S	137	000023	WRITE	1	000000	060642	000001			1540	007010	003210	1 0
001140	2	0	23	S	134	000023	WRITE	1	000000	060623	000001			1700	007010	002450	1 0
001720	2	0	26	S	137	000023	WRITE	1	000000	060641	000001			1140	007010	003230	1 0
001520	2	0	23	S	134	000023	WRITE	1	000000	060622	000001			1720	007010	003030	1 0
001000	2	0	26	S	137	000023	WRITE	1	000000	060651	000001			1520	007010	002310	1 0
001600	2	0	23	S	134	000023	WRITE	1	000000	060621	000001			1000	007010	003110	1 0
001020	2	0	26	S	137	000023	WRITE	1	000000	060650	000001			1600	007010	002330	1 0
000720	2	0	23	S	134	000023	WRITE	1	000000	060620	000001			1020	007010	002230	1 0
000240	2	0	26	S	137	000023	WRITE	1	000000	060647	000001			720	007010	001550	1 0
001200	2	0	23	S	134	000023	WRITE	1	000000	060617	000001			240	007010	002510	1 0
000420	2	0	26	S	137	000023	WRITE	1	000000	060646	000001			1200	007010	001730	1 0
000520	2	0	23	S	134	000023	WRITE	1	000000	060616	000001			420	007010	002030	1 0
001420	2	0	26	S	137	000023	WRITE	1	000000	060645	000001			520	007010	002730	1 0
000460	2	0	23	S	134	000023	WRITE	1	000000	060615	000001			1420	007010	001770	1 0
001060	2	0	26	S	137	000023	WRITE	1	000000	060644	000001			460	007010	002370	1 0
000200	2	0	23	S	134	000023	WRITE	1	000000	060614	000001			1060	007010	001510	1 0
000740	2	0	26	S	137	000023	WRITE	1	000000	060643	000001			200	007010	002250	1 0
001750	2	0	23	S	134	000023	WRITE	1	000000	060613	000001			740	007010	003270	1 0
000760	2	0	26	S	137	000023	WRITE	1	000000	060642	000001			1750	007010	002270	1 0
001240	2	0	23	S	134	000023	WRITE	1	000000	060623	000001			760	007010	002550	1 0
000020	2	0	26	S	137	000023	WRITE	1	000000	060641	000001			1240	007010	001330	1 0
000500	2	0	23	S	134	000023	WRITE	1	000000	060622	000001			20	007010	002010	1 0
000440	2	0	26	S	137	000023	WRITE	1	000000	060651	000001			500	007010	001750	1 0
001340	2	0	23	S	134	000023	WRITE	1	000000	060621	000001			440	007010	002650	1 0
000140	2	0	26	S	137	000023	WRITE	1	000000	060650	000001			1340	007010	001450	1 0
000160	2	0	23	S	134	000023	WRITE	1	000000	060620	000001			140	007010	001470	1 0
000640	2	0	26	S	137	000023	WRITE	1	000000	060647	000001			160	007010	002150	1 0
001320	2	0	23	S	134	000023	WRITE	1	000000	060617	000001			640	007010	002630	1 0
000560	2	0	26	S	137	000023	WRITE	1	000000	060646	000001			1320	007010	002070	1 0
001440	2	0	23	S	134	000023	WRITE	1	000000	060616	000001			560	007010	002750	1 0
000600	2	0	26	S	137	000023	WRITE	1	000000	060645	000001			1440	007010	002110	1 0
001260	2	0	23	S	134	000023	WRITE	1	000000	060615	000001			600	007010	002570	1 0
001100	2	0	26	S	137	000023	WRITE	1	000000	060644	000001			1260	007010	002410	1 0

Figure 9.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/08/83   2:28AM          BANK   1
(C) HEWLETT-PACKARD CO  1980

                  ******    PCBX AND STACK MARKERS FOR DST 104  (PCB 12 )    ******

SEG REL   SEG REL    JMAT     JPCNT    JOB INPUT   JOB OUTPUT  JDT DST  JIT DST                                           JCUT
DL        DB         INDEX    INDEX    LOG DEV#    LOG DEV #   INDEX    INDEX    JOB TYPE   DUPLICAT  INTERACT  INIT Q     INDEX
000444    001444         0        0       20          20         63        45    UNDEF         YES       YES    001145         0

ADDRESS   BANK         X      DELTA P   STATUS   DELTA Q    SEGMENT                      OFFSET/PROCEDURE        MOD/PRODUCT

075300      1       177756    020214    102075   000012      75 KERNELC (100)
075266      1       037470    017521    101075   000014      75 KERNELC (100)
075252      1       000013    000757    141301   000012     301  USER SEGMENT
075240      1       000000    000000    140043   000004      43 MORGUE (42)

$$$$$$$$ DST  104 (STACK)                        $$$$$$$$

******PCBX *************************************************************************************************
***PXGLOBAL
072423  000444 001444 177777 000024 000024 000063 006045 000000
***PXFIXED
072433  000120 001213 003474 001145 001000 177777 000000 000000  072443  000000 000000 000000 000000 000000 040000 000000 000000
072453  000000 000000 000000 000000 043000 000000 000000 007674  072463  000005 015302 000000 000000 000000 000000 000000 000000
072473  000000 000000 000000 000000 000000 000000 000000 000000  072503  000000 000000 000000 000000 000000 000000 000000 000000
LINES 072513 - 072532 SAME AS ABOVE

072533  000000 000000 000000 000000 000000 000000 000000 000000  072543  000000 000000 000000 000000 000000 000000 000000 000000
***PXFILE  (ZERO TABLE ENTRIES ARE NOT PRINTED)
072553  000310 000000 000000 000000 000000 000014 000000 000000  072563  000000 000000 000000 000000 000000 000000 000000 000000
072573  000246 000104 000100 000000 000000
------- FILE VECTOR TABLE                ENTRY  ADDRESS  LOCK  BRK  LOCK COUNT/PIN   HIPRI TAIL  HIPRI HEAD  LOPRI TAIL  LOPRI HEAD
072600  000106 000000 000000 000000        0      106                         0  0
------- CONTROL BLOCKS
072700(000105)  000001 100060 000001 051514 020040 020040 020040 000001 000756 000400 000200 000000  072700    0  SL
072714(000121)  000000 004000 000000 000000 000400 000000 000002 000000 000000 000000 015523 000000  072714                       S
072730(000135)  015523 177777 177777 000111 000000 000000 000001 000001 000000 000000 000000 000000  072730  S     I
072744(000151)  000000 000001 000000 000001 000000 000000 000000 000000 000000 000000 000000 000000  072744
072750(000155)  000000 000060 000003 043505 052120 051111 050040 002001 000767 000400 000200 000000  072750    0  GETPRIP
072774(000201)  000000 004000 000000 000000 000400 000000 000002 000000 000000 000000 000010 000000  072774
073010(000215)  000010 177777 177777 006111 000000 000000 001401 000001 000000 000000 000000 000000  073010       I
073024(000231)  000000 000002 000000 000002 000000 000000 000000 000000 000000 000000 000000 000000  073024
073040(000245)  000000                                                                               073040
073041  000000 000000 000000 000000 000000 000000
------- AVAILABLE FILE TABLE             FNUM    FTYPE   SNULL    PACB V     LACB V    IOQX
073053  000000 000112 000000 000000        2      FILE            0  112     0    0
073057  000000 000104 000000 000000        1      FILE            0  104     0    0
**PXPOINTERS
072063  000000 000314 000434 000444
****DL REGISTER ********************************************************************************************
073067(177000)  005507 042524 050122 044517 051111 006041 030001 001000 000000 044517 051111 052131  073067  GETPRIORI !0     IORITY
073103(177014)  030001 001000 000000 005507 042524 050122 044517 051111 052131 030001 001000 000000  073103 0       GETPRIORITY0
073117(177030)  000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000  073117
```

Figure 10.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83   5 08AM
(C) HEWLETT-PACKARD CO  1980

                                   ****** PRESENT STACKS ******


                   ******     PCBX AND STACK MARKERS FOR DST 130  (PCB 24 )     ******
                                **** CURRENT  PROCESS ****

SEG REL   SEG REL   JMAT     JPCNT    JOB INPUT  JOB OUTPUT  JOT DST  JIT DST                                             JCUT
DL        DB        INDEX    INDEX    LOG DEV#   LOG DEV #   INDEX    INDEX     JOB TYPE   DUPLICAT  INTERACT  INIT Q     INDEX
000444    000600        1        2      20          20        123       122     #S56          YES       YES    004575         0

ADDRESS   BANK        X      DELTA P   STATUS    DELTA Q     SEGMENT                       OFFSET/PROCEDURE        MOD/PRODUCT

016525     4       000511    004706    140035    000144       35 ABORTDUMP (34)
016361     4       000001    004170    140035    000007       35 ABORTDUMP (34)
[illegible] 4      [illegible] [illegible] [illegible] [illegible] 301  USER SEGMENT
015143     4       000000    001010    140430    000034       30 DIRC (25)
015107     4       000002    001224    142430    000021       30 DIRC (26)
015066     4       000004    001104    140430    000027       30 DIRC (25)
015037     4       000064    000635    142430    000013       30 DIRC (25)
015024     4       000064    004017    140301    000220      301  USER SEGMENT
015604     4       000001    001010    140430    000034       30 DIRC (25)
015550     4       000002    001224    142430    000021       30 DIRC (26)
015527     4       000004    001104    140430    000027       30 DIRC (26)
015500     4       000002    001224    142430    000021       30 DIRC (26)
015457     4       000000    000514    140430    000013       30 DIRC (26)
015444     4       000000    000247    140301    000020      301  USER SEGMENT
015424     4       000000    000000    140043    000004       43 MORGUE (42)
```

Figure 11.

```
PROGRAM FILE BADLABEL PUB GOERTZ

BADLABEL             0
   NAME             STT  CODE ENTRY SEG
   BADLABEL           1     0     0
   FOPEN             14               ?
   XCONTRAP          15               ?
   DIRECSCAN         16               ?
   FCONTROL          17               ?
   PRINTFILEINFO     20               ?
   SORTINITIAL       21               ?
   ASCII             22               ?
   SORTOUTPUT        23               ?
   QUIT              24               ?
   SORTEND           25               ?
   PRINT             26               ?
   FCLOSE            27               ?
   TERMINATE'        30               ?
   NOTAPE             2     0     0
   PROCESS ENTRY      3  1024  1125
   EXCHANGEDB        31               ?
   LUN               32               ?
   ATTACHIO          33               ?
   RELSIR            34               ?
   RESETCRITICAL     35               ?
   DASCII            36               ?
   FWRITE            37               ?
   SETCRITICAL       40               ?
   GET VOLUME'TABL    4  3445  3445
   GETSIR            41               ?
   GET GROUP          5  3634  3634
   MOUNT             42               ?
   DISMOUNT          43               ?
   CHECK'FSPACE'TA    6  4203  4203
   CHECK DFSM         7  4337  4337
   LOAD DSEGS        10  4741  4751
   GETDSEG           44               ?
   LOADPROC          45               ?
   GETUSERMODE       46               ?
   LOAD DSEGS AGAI   11  4741  4757
   CONTROL Y         12  6030  6030
   RESETCONTROL      47               ?
   MINSTACK          13  6043  6043
   SEGMENT LENGTH        6120

PRIMARY DB        130    INITIAL STACK    1440    CAPABILITY        700
SECONDARY DB     4443    INITIAL DL          0    TOTAL CODE       6120
TOTAL DB         4573    MAXIMUM DATA    11610    TOTAL RECORDS      80
ELAPSED TIME   00 00 04 944                   PROCESSOR TIME   00.02.503
```

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

Figure 12.

```
PAGE 0026   BADLABEL        BADLABEL <2 0>      Procedure PROCESS'ENTRY

00254 4                           " LAST FILE IS  "),2,
00256 4          ASSEMBLE(DUP NOP)
00257 4          MOVE * := FILE'NAME,(2*),
00272 4          SCAN * UNTIL " ",1
00274 4          LEN := TOS - @BBUFF
00275 4          PRINT(BUFF,-LEN,%40),
00303 4          ERRORS'FOUND := TRUE;
00305 4       END
00305 3       MOVE FILE'NAME := " ",2; MOVE * := FILE'NAME,(29),
00320 3       MOVE FILE'NAME := BENT,(*)
00324 3       SCAN FILE'NAME UNTIL " ",1,
00327 3       MOVE * := " ",2
00335 3       ASSEMBLE(DUP NOP)
00335 3       MOVE * := GROUP'NAME,(*),
00341 3       SCAN * UNTIL " ",1,
00343 3       MOVE * := " ",2
00351 3       ASSEMBLE(DUP NOP);
00352 3       MOVE * := ACCOUNT'NAME,(*);
00355 3
00355 3       MOVE ADISC'ADDR := ENT(4),(2);
00362 3       VTAB'INDX := DA1 (0:8);
00365 3       IF VTAB'INDX > 255 THEN LDEV2 := 0
00370 3          ELSE LDEV2 := LUN(VTAB'INDX,MVTABX)
00400 3       IF <<NOT>> CHECK'LDEV(LDEV2) THEN BEGIN
00405 4          TOS := SIR
00406 4          RELSIR(*,*)
00407 4        << RESETCRITICAL(0); >>
00407 4          CLEARBUFF
00422 4          MOVE BBUFF := "INVALID VTABINDX FOR ",2,
00443 4          ASSEMBLE(DUP NOP)
00444 4          MOVE * := FILE'NAME,(2*),
00447 4          SCAN * UNTIL " ",1
00451 4          LEN := TOS - @BBUFF
00453 4          PRINT(BUFF,-LEN,%40)
00460 4          ERRORS'FOUND := TRUE
[illegible] 4   <<    GO TO GET'LOST; >>  QUIT(1012);
00463 3       END
00464 3
00464 3       LABEL'LDEV := LDEV2
00466 3       ATTRET := ATTACHIO(LDEV2,0,0,@FLABEL,0,123,DA1,(8:8),DA2,1);
00502 3       IF ATT1 (8:8) = 1 THEN GO TO GOOD'ATTACHIO;
00506 3
00506 3       IF ATT1 (8:8) = %64 THEN BEGIN
00512 4          CLEARBUFF
00527 4          MOVE BBUFF := "INVALID DIRECTORY ADDRESS FOR ",2,
00555 4          ASSEMBLE(DUP NOP)
00556 4          MOVE * := FILE'NAME,(2*)
00561 4          SCAN * UNTIL " ",1
00563 4          LEN := TOS - @BBUFF
00565 4          PRINT(BUFF,-LEN,%40),
00572 4          TOS := SIR
00573 4          RELSIR(*,*)
00574 4          RESETCRITICAL(0)
00575 4          ERRORS'FOUND := TRUE,
00600 4          GO TO GET'LOST,
00601 4       END
00601 3
```

Figure 13.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  50  FIX 20  DUMP TIME  2/08/83   2 28AM
(C) HEWLETT-PACKARD CO  1980

                         ******   REGISTERS   ******

* DATA SEGMENT       * CODE SEGMENT     * MISCELLANEOUS * STATUS = 161301    * ISR = 000300                                  SERIES 30/33 *

  DB BANK = 000005   PB      = 114623   X   = 000000   MODE         = PRIV   RUN/HALT   = HALT   STACK OVR     = OFF
  DB      = 173023   P       = 114640   CIR = 140000   INTERRUPTS   = ON     IRQ        = OFF    BNDS OVR/UNF  = OFF
  S BANK  = 000005   PL      = 114652   NIR = 000000   TRAPS        = ON     CSRQ       = OFF    VIOL CODE     = NONE
  DL      = 172567   PBBANK= 000007                    STACK OP     = LEFT   PARITY     = OFF    DISABLE ATN   = ON
  Q       = 173031   (P-PB)= 000015                    OVERFLOW     = OFF    POWERFAIL  = OFF
  S       = 173031                                     CARRY        = OFF    POWERON    = OFF
  Z       = 176253                                     COND CODE    = CCE    DISP FLAG  = OFF
                                                       SEGMENT #    = [illegible]  ICS FLAG   = OFF

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

(ADDR   0)   CODE SEGMENT TABLE POINTER                    030110
(ADDR   1)   EXTENDED CODE SEGMENT TABLE POINTER           000300
(ADDR   2)   DATA SEGMENT TABLE POINTER                    024110
(ADDR   3)   PROCESS CONTROL BLOCK BASE                    037510
(ADDR   4)   CURRENT PCB POINTER                           041210
(ADDR   5)   INTERRUPT STACK BASE                          041610
(ADDR   6)   INTERRUPT STACK LIMIT                         042606
(ADDR   7)   INTERRUPT MASK                                067600
(ADDR  10)   CRT BANK                                      000000
(ADDR  11)   CRT ADDR                                      000000
```

Figure 14.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/10/83   3 38PM
(C) HEWLETT-PACKARD CO  1980

                            ******    REGISTERS    ******

* DATA SEGMENT      * CODE SEGMENT   * MISCELLANEOUS * STATUS = 141075   * ISR = 000303                               SERIES 30/33 *

  DB BANK = 000000 * PB     = 135114 * X   = 177756 * MODE       = PRIV * RUN/HALT   = HALT   STACK OVR    = OFF
  DB      = 001000 * P      = 140021 * CIR = 030020 * INTERRUPTS = ON   * IRQ        = OFF    BNDS OVR/UNF = OFF
  S BANK  = 000000 * PL     = 163733 * NIR = 000000 * TRAPS      = OFF  * CSRQ       = OFF    VIOL CODE    = NONE
  DL      = 177777 * PBBANK= 000000 *               * STACK OP   = LEFT * PARITY     = OFF    DISABLE ATN  = ON
  Q       = 041554 * (P-PB)= 002705 *               * OVERFLOW   = OFF  * POWERFAIL  = OFF
  S       = 041636 *                *               * CARRY      = OFF  * POWERON    = OFF
  Z       = 042552 *                *               * COND CODE  = CCE  * DISP FLAG  = ON
                   *                *               * SEGMENT #  =  75  * ICS FLAG   = ON

PAUSE INSTRUCTION IN CIR

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

(ADDR %0)	CODE SEGMENT TABLE POINTER	030054
(ADDR %1)	EXTENDED CODE SEGMENT TABLE POINTER	000303
(ADDR %2)	DATA SEGMENT TABLE POINTER	024054
(ADDR %3)	PROCESS CONTROL BLOCK BASE	037454
(ADDR %4)	CURRENT PCB POINTER	000000
(ADDR %5)	INTERRUPT STACK BASE	041554
(ADDR %6)	INTERRUPT STACK LIMIT	042552
(ADDR %7)	INTERRUPT MASK	063610
(ADDR %10)	DRT BANK	000000
(ADDR %11)	DRT ADDR	000000

Figure 15.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83.  8 32AM
(C) HEWLETT-PACKARD CO  1980

                  ******** DISC REQUEST TABLE ********  (SUMMARY INFO)
```

TOTAL ENTRIES IN TABLE	100
ENTRY SIZE	20
ENTRIES IN PRIMARY AREA	61
IMPEDED PROCESS PCB'	
TABLE INDEX OF FIRST AVAIL ENTRY	620
TABLE INDEX OF LAST AVAIL ENTRY	2000
MAXIMUM NUMBER OF ENTRIES IN USE	13
CURRENT NUMBER OF ENTRIES IN USE	13
OVERFLOWS	
TOTAL REQUESTS	13354
SYSBASE INDEX OF DISABLED Q HEAD	
SYSBASE INDEX OF DISABLED Q TAIL	

```
                  ******** DISC REQUEST TABLE ********  (ACTIVE LISTS)

        LDEV   1
                                                                STATUS  0 XX -> PENDING
                                                                        1 XX -> SUCCESSFUL
                                                                        2 XX -> END OF FILE
                                                                        3 XX -> UNUSUAL CONDITION
                                                                        4 XX -> IRRECOVERABLE ERROR
```

TABLE INDEX	LDEV	UNIT	PCB	S S	DST/ BANK	OFFSET/ ADDRESS	FUNC	XFER CNT	PARM1	PARM2	MISC	SEG IDENT	SEGDSP	URGCLS	- F L A G S - MAIN	AUX	STATUS	
001120*	1	0	25	S	136	000023	WRITE	1	000000	107240	000000			240	006110	002430	0	1
001460	1	0	3		118	000200	READ	1200	000000	062223	000000			170	006100	002770	0	0
000400	1	0	27	S	140	000023	WRITE	1	000001	004405	000000			240	006100	001710	0	0
001500	1	0	32	S	143	000023	WRITE	1	000001	007633	000000			240	006100	003010	0	0
001040	1	0	22	S	133	000023	WRITE	1	000000	107106	000000			240	006100	002350	0	0
001220	1	0	30	S	141	000023	WRITE	1	000001	007617	000000			240	006100	002530	0	0
000220	1	0	24	S	135	000022	READ	1	000000	107124	000000			242	006100	001530	0	0
001620	1	0	21	S	132	000023	WRITE	1	000000	107070	000000			242	006100	003100	0	0

JOURNAL OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

Figure 16.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83   5 32AM
(C) HEWLETT-PACKARD CO  1980

          (021757)   0    022757      000000 000000 000000 000000 000000 000120 000000 000000
          (021767)   0    022767      000000 000000 000000 000000 000000

DRT NO  41                                                  =  CONTROLLER ERROR STATUS = 000000

          (MAGNETIC TAPE UNIT)              TYPE 24     SUBTYPE 0

          UNIT 0  LOGICAL DEV 7    FLAGS = 002000    NEXT DIT = 000000  DLTP = 177450  ILTP = 054547  IOQP = 000000

          DIT
          (021774)   0    022774      002000 000000 000000 040007 177450 054547 000000 000000
          (022004)   0    023004      000000 000000 000000 000000

          (MAGNETIC TAPE UNIT)              TYPE 24     SUBTYPE 0

          UNIT 1  LOGICAL DEV 8    FLAGS = 002000    NEXT DIT = 000000  DLTP = 177450  ILTP = 054547  IOQP = 000000

          DIT
          (022010)   0    023010      002000 000000 000000 040410 177450 054547 000000 000000
          (022020)   0    023020      000000 000000 000000 000000

          (MAGNETIC TAPE UNIT)              TYPE 24     SUBTYPE 0

          UNIT 2  LOGICAL DEV 9    FLAGS = 002000    NEXT DIT = 000000  DLTP = 177450  ILTP = 054547  IOQP = 000000

          DIT
          (022024)   0    023024      002000 000000 000000 041011 177450 054547 000000 000000
          (022034)   0    023034      000000 000000 000000 000000

          (MAGNETIC TAPE UNIT)              TYPE 24     SUBTYPE 0

          UNIT 3  LOGICAL DEV 10   FLAGS = 002000    NEXT DIT = 000000  DLTP = 177450  ILTP = 054547  IOQP = 000000

          DIT
          (022040)   0    023040      002000 000000 000000 041412 177450 054547 000000 000000
          (022050)   0    023050      000000 000000 000000 000000

DRT NO  49

          (SYSTEM DISC)                     TYPE 0      SUBTYPE 9

          UNIT 0  LOGICAL DEV 1    FLAGS = 040010    NEXT DIT = 000000  DLTP = 177460  ILTP = 054745  IOQP = 044240

          DIT
          (022054)   0    023054      040010 000000 044240 040001 177460 054745 000000 000000
          (022064)   0    023064      044600 044740 000000 100040 000077 001440 034446 000001
          (022074)   0    023074      000001 000000 011400 100043 000077 001437 000000 000000
          (022104)   0    023104      000000 000000 000000 000000 000000 000000 000000 000000
          (022114)   0    023114      000001 000000 103043 000000 000000
```

Figure 17.

```
HP3000 III MEMORY DUMPC 00 05 OF SYS VER C  UPDATE  E0  FIX 20  DUMP TIME  2/09/83   3 18AM
(C) HEWLETT-PACKARD CO  1980

                                 ******    REGISTERS    ******

*********************************************************************************************************************
* DATA SEGMENT      * CODE SEGMENT     * MISCELLANEOUS * STATUS = 142453  * ISR = 000000                 SERIES 30/33 *
*********************************************************************************************************************
*                   *                  *               *                  *                                            *
*  DB BANK = 000000 * PB     = 017423  * X   = 000003  * MODE       = PRIV * RUN/HALT   = RUN   STACK OVR    = OFF       *
*  DB      = 000000 * P      = 037643  * CIR = 001251  * INTERRUPTS = ON   * IRQ        = OFF   BNOS OVR/UNF = OFF       *
*  S BANK  = 000004 * PL     = 040175  * NIR = 000005  * TRAPS      = OFF  * CSRQ       = OFF   VIOL CODE    = NONE      *
*  DL      = 010467 * PSBANK= 000005   *               * STACK OP   = LEFT * PARITY     = OFF   DISABLE ATN  = OFF       *
*  Q       = 012014 * (P-PB)= 020220   *               * OVERFLOW   = OFF  * POWERFAIL  = OFF                            *
*  S       = 012014 *                  *               * CARRY      = ON   * POWERON    = OFF                            *
*  Z       = 021473 *                  *               * COND CODE  = CCL  * DISP FLAG  = OFF                            *
*                   *                  *               * SEGMENT #  =  53  * ICS FLAG   = OFF                            *
*********************************************************************************************************************

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

(ADDR  0)   CODE SEGMENT TABLE POINTER                000000
(ADDR  1)   EXTENDED CODE SEGMENT TABLE POINTER       000000
(ADDR  2)   DATA SEGMENT TABLE POINTER                024110
(ADDR  3)   PROCESS CONTROL BLOCK BASE                037510
(ADDR  4)   CURRENT PCB POINTER                       040070
(ADDR  5)   INTERRUPT STACK BASE                      041610
(ADDR  6)   INTERRUPT STACK LIMIT                     012606
(ADDR  7)   INTERRUPT MASK                            067800
(ADDR 10)   DRT BANK                                  000000
(ADDR 11)   DRT ADDR                                  000000
```

Security Issues: How secure is YOUR system?

Doug Claar, programmer analyst
Hewlett Packard, Computer Systems Div.

Many new computer users implicitly expect that the computer which they are using is private and secure. System managers understand that this is a fallacious assumption, as they are able to to access everything on the system. What system managers often do not understand is that they are not always the only one with access to the entire system. Unless a system manager consistently stays on top of system security, that security quickly evaporates, but if typical security breaching techniques are known and understood, appropriate steps can be taken to foil takeover attempts and restore secure status to the system. It is essential, then, to first understand what a system invader has to go through in order to take over a system to be able to effectively combat security violations. Reformed burglars are said to make the best security experts, and the same might be expected of computer security experts: those who have broken into systems are best able to specify the countermeasures that would have worked against them.

Before discussing some typical security breaching techniques, a disclaimer is in order: neither the author nor Hewlett Packard (especially Hewlett Packard!) condone any unauthorized access of computer systems or of any data thereon. The breakin techniques are described strictly in order to discuss the appropriate counter-measures.

To better understand the techniques described later, it is helpful to present the EDP environment in the Computer Systems (CSY) R&D lab: Although the attitude is perhaps not universal throughout Hewlett Packard, in the Computer Systems (CSY) R&D area everyone is encouraged to utilize the computers as much as possible. Employees using the 3000 (even for personal projects) inevitably benefit the company. People are allowed to use the 3000 (to write papers, do homework or whatever) on their own time, and are encouraged to look for ways to use it on the job. HP benefits from more sophisticated users, and from the programs written by those users. It is a natural outgrowth of this attitude to not have strict security, but as more sensitive information makes its way onto the lab 3000s, this attitude is changing.

A 3000 can be as secure or as insecure as it's management desires. In the past it was argued that lab systems should be low in security, since the machines were strictly R&D—no accounting, payroll, or management functions. It is only recently that the realization has begun to dawn that security is as important for us as it is for our customers. The installation of about fifty dial-in lines has been the impetus for tightening security on the R&D timeshare machines. In addition, as CSY's computer literacy push bears fruit, more people are using the systems for more sensitive data. As an example, several secretaries are beginning to type employee evaluations on the system. EDP has assumed the responsibility of providing the greater degree of security that such users require and expect and of educating them about any requirements placed on them for security of their data. The job is not nearly complete (especially in the area of user re-education), but work is progressing steadily on providing a secure environment for all users.

Although most lab systems have traditionally had low security, there have always been one or two secure lab systems: those whose system manager or other responsible person was individually concerned with security. The system managers on these systems have made it their job to try to get past each other's security schemes, and most of the techniques covered come from that source. Today, their solutions are finally beginning to be put into use lab-wide.

Security solutions must deal with the question of what system security is. From the unauthorized user's point of view, system security is "what is in the way". To get around security, this user must accomplish three key objectives, which are to get onto the system, to work into a position of power, and finally, to leave the system in place to facilitate re-entry. From the System Manager's point of view, security must consist of making each of these objectives as difficult as possible to attain. Let's look at some techniques for achieving the unauthorized user's objectives, and then at ways to block these techniques.

The first requirement for getting "into" a system is to get "onto" it. There are several potential weak spots, with perhaps the most obvious being facility, company or MPE common user IDs. For example, at CSY, almost every system has, in addition to standard MPE IDs, a DS user ID (for DSing through to another system), an I/O utility ID (for transferring spool files to a system with an EPOC on it), and an electronic mail ID (for remote HPMAIL users). It is likely that many if not most computing centers also have some type of common user ID—perhaps even for demonstrations. Although these accounts may not have any special abilities, they provide a foothold (or beach-head) from which to launch an assault.

A second potential weak spot is user IDs from adjacent systems, especially if those systems are DSed to the target system. Often, legitimate users will have the same user IDs and passwords on adjoining systems for their own convenience. Assuming the user's ID and password can be discovered on one system, the adjacent system can also be penetrated.

Should both of the aforementioned methods prove fruitless, there are still other techniques available to the determined intruder. One time-honored technique is to look for user IDs among the discarded listings or at unattended terminals. An example of this is when one of the college students we hired for the summer came back the following summer and found all his capabilities gone and his terminal hooked to one of the more secure lab 3000s. After challenging him to get back the capabilities, I was called away from my desk. While I was gone, he simply walked over and upped his capabilities from my terminal. The moral is: people are the weakest link in any security scheme. User carelessness must be combatted by education as well as top level techniques. Users, especially those who require special capabilities, must be convinced that leaving terminals logged on is like leaving a car running, and should be accompanied by close supervision. (If connect time is billed, this point is probably easy to make).

Because any user must first get onto the system in order to do anything, an obvious first administrative step in combatting unauthorized use is to limit access to only those who should be on the system. There are several thing that can be done to limit access: eliminating or severely limiting common user IDs is a good first move. For example, the three common CSY user IDs mentioned earlier can all be restricted so that they are able to perform no other function than their intended one. Looking at a specific example, it can be seen that, in the case of the DS ID (which is intended only for people using DS to get to another system on which they have an account), that systems which are at the end of a DS line do not need this ID, and that those which do require it could limit it to DSLINE, REMOTE, and perhaps FILE commands. The limitations can be accomplished with a UDC that aliases all other MPE commands to either a no-op or a logoff, with operator and system log identification. (If CONSOLE logging is turned on, a simple TELLOP will notify the operator and be recorded in the log).

Other common logons can be analyzed and controlled in the same manner: decide what the logon was designed to accomplish, and disallow everything else. Be aware, however, that some programs and subsystems, such as TDP, allow the user almost full MPE functionality without the restrictions of UDCs. These programs are more troublesome to control, and should thus be disallowed to common (transient) users whenever possible. In general, it is not a good idea to allow any form of the run command in this environment: In the case of electronic mail at CSY, having users type "HPMAIL", a UDC in which the break key won't work, is much simpler—and safer—than having them type "Run HPMAIL.HPMAIL.SYS".

MPE common users need not be a problem either, even if operators are not on duty at all times. At CSY, all .SYS users have a UDC which prevents their use from any location other than certain specified terminals. In addition, the Operator.Sys UDC disallows STORE, RESTORE and many other functions (including SETCATALOG!). Because engineers must be able to bring a system back up on the weekend (instructions are posted on each system), Operator.Sys had no password: its home group has, however, been changed to LOGON, a group created with virtually no capabilities. Because passwords tend to "leak" out, the passwords for Manager.Sys and the pub.sys group are changed frequently. To facilitate this, a program has been written that changes the password, if any, contained in the first record of a group of files specified by the program user. Surprisingly fast, this program makes password changing much less traumatic and time consuming. (This program, named NEWPASS, will be available on the swap tape, but—like everything on the tape—it is not guaranteed).

Assuming that an intruder had been able to log onto the system, their next objective is to move to a position of power. There is no (known) way to bypass security using only standard user capabilities, but that does not mean that the person breaking in needs higher capabilities: only that someone else has "left the keys in the ignition". In fact, the ideal situation for the interloper is to find, or leave behind, a "superman" program (one that gives "super" capabilities) in some innocuous place, and then in the future only log on as a mild-mannered, common user. To plant such a program, (if one can't be found) what capabilities are needed? Obviously, either System Manager (SM) or Privileged Mode (PM) would work quite nicely, but since those capabilities are usually guarded very well, what else might help? System Supervisor (OP) capability will also work, since a user with capability can restore any file anywhere and can also dump the account structure. Account Manager is useful only if the account has OP (system supervisor), SM (system manager) or PM (privileged mode). A .sys logon is useful because files can be restored into .pub from any user.sys, even with only standard capabilities. By the same token, any user within an account can restore to any group in that account, allowing non-privileged users to restore a file to (someone else's) privileged group. Of course, the other user will wonder where the file came from, so it is a good idea eventually leave the program in a group with loads of files, or in pub.sys with a name like "HIOCARD2".

There are several ways to conduct the search for power. The most obvious (and usually the most fruitful) begins with a list and a knowledge of what people tend to call things. It is amazing the number of people who will call a capability program "CAPS", "GETCAPS", "SM", "SUPERMAN", "PRIVS", or the like. In addition, people tend to identify stream jobs with a J or S, and since MPE requires the passwords be in the file, access to the stream job files can be very "helpful" to the intruder. A third group of "useful" files that tend to be named similiarly are UDC files, although these file names can often be found more directly from command.pub.sys. The key point is that meaningful file names can be a two-edged sword: both users and abusers can benefit. Programs that store user and account information on disk are especially dangerous: meaningful file names here can be disasterous if the program accidently (or purposely) leaves the files behind. For example, a programmer at CSY wrote a utility program that read the account structure from one 3000 and formatted it onto a stream tape so that another 3000 could have the same structure. The program worked fine, with one minor flaw: it left three files—TACCT, TUSER and TGROUP—on PUB.SYS with account, user, and group information (including passwords) in them. This program made it all the way to Boise and Fort Collins before its "feature" was realized. Taking advantage of mnemonic names is simply one example of a way to get into a position of power. There are doubtless many others.

System managers must of course be responsible for their own logon, but also ultimately much more: the entire system. The System Manager must administer all data pertaining to the system, all access paths to the system, and all capabilities on the system. The most critical data pertaining to the system can be found on the SYSDUMP tapes: a complete sysdump tape set *is* the system—in terms of everything but physical hardware (which plant security hopefully monitors). Sysdump tapes should not be available to general public: they should be locked up and, if a file needs restoring, EDP should do it.

The access paths to the system should also be controlled as much as possible. Although hardwired terminals may requires monitoring, phone and DS lines are probably more of a concern. Analysis of these paths should include the identification of who uses them, and why. If, for example, a DS line's purpose is to allow the users of one system to access a central resource, then the DS line should be made one-way by eliminating the virtual terminals DS users need in order to log onto the system. If there is occasional two way access, then steps should be taken to insure that communication is limited to those who should be using the line.

System logging, along with some type of data formatter to print appropriate parts of the log, can be used to monitor those who log on to either phone of DS lines. There are several contributed library programs that crunch log files, and if those aren't suitable, the system manager manual provides log file format information for do-it-yourselfers. (A rather inelegant but simple program used on one of the CSY systems will again be available on the swap tape as LISTLOGF (with the standard "no support" proviso).

To *control* those who log on by either phone or DS lines requires some way of knowing what LDEV is being used. A popular way to do this is to set up a UDC that executes every time anyone logs on. This logon UDC, which should not allow the user to break, or to see the UDC definition, can simply execute a security program and log the potential user off if all is not kosher. Because a program has access to MPE intrinsics, it can determine if the user is coming from an LDEV this is defined as DS or dial-up, and can then ask for a password, or just deny access altogether. Besides testing for phone or DS lines, the program can also test for many other conditions: CSY's program also tests Operator.Sys and LDEV 20 (they must occur together). Once again, this program (and associated UDC) will be made available on the swap tape as "STARTUP".

Finally, capabilities of legitimate users of the system *must* be monitored and controlled, as those users will also often see how far they can get on their own system. Thus, after dealing with the outside world, it is time to look inward at protecting users from each other and themselves. Although this is an area in which most system managers have much expertise, it will not hurt to point out several things to watch for. If any users with privileged mode are allowed, they should reside in an account separate from non-privileged users, with user, group and account passwords. Treat privileged mode as if it were radioactive or highly explosive—it is! Remember that OP capability allows unlimited store, restore and sysdump capability. Also, why create .SYS users who can then restore into PUB.SYS? There has to be another place for that user to go. Be constantly on the lookout for new privileged mode programs, user, groups or accounts. Use the list of standard MPE files provided in the communicator to verify which files should be in the SYS account. Use LISTDIR2 to verify that LISTDIR2 has not been released: secure it if it has.

Two programs used at CSY to keep tabs on privileged mode are included on the swap tape. Neither are fantastically elegant, but they both work. The first, LISTFPM, looks for files that require PM capability to run. At CSY, we stream a job, included on the swap tape as LISTFPM.JOB, which runs LISTFPM, lists the secure/released status of LISTDIR2, and runs the log file analyzer program (LISTLOGF). The second program, LISTUSRS, prints a formatted listing of pertinent user and account information, with or without passwords. This program and its output are as dangerous as dynamite, should be handled accordingly. Don't leave the listing on the printer, in a spool file, or on a desk. Don't even throw it away without shredding— remember, people *do* look through discarded listings. If the listing is left unattended, even while waiting in a spooler queue, someone *might* copy it. And no one really wants the visibility of having to tell users that they have to change all their passwords because EDP blew it, or the chore of changing all the passwords EDP is responsible for.

It takes some time and effort to ensure a secure system, but thankfully, there are tools available to help do the job. Although no computer system can ever be one hundred percent secured, the steps outlined here should make the security fence high enough to keep the vast majority of trouble-makers at bay, to trip up the few who get by, and to give users the level of protection they want from all computers.

(The author is interested in exchanging security ideas, horror stories, problems etc. with other 3000 users).

(Reviewer's comment: TDP has the facility to disable the "dangerous" features for all users, or for all users within a specific group or account. The BREAK can be disabled, as can the RUN command, the STREAM command, and/or access to all MPE commands.)

MPE Programming

by Eugene Volokh
VESOFT Consultants

Three Examples of MPE Programming in Action

Recently, in my capacity as systems consultant to a large HP installation, I encountered the following situation:

There is a large system that can operate in one of two modes—ONLINE or BATCH. Which mode it operates in is indicated by the presence or absence of a certain file called HOL100. If this file exists, this means that the system is operating in an ONLINE mode; if it does not, the system is operating in a BATCH mode.

It is desirable to print at logon time which mode the system is running in.

Obviously, since something is to be done at logon time, we should use a logon UDC. The simplest solution is to have one of the form:

```
LOGONUDC
OPTION LOGON
RUN CHKSTAT
```

where CHKSTAT is a program that checks whether HOL100 exists and prints out an appropriate message. However, this is not the best solution. For one, I don't feel like writing a custom SPL program every time a rather simple systems programming task comes around; those who are not familiar with FOPEN will find this even harder to do. Furthermore, even if I did write a custom program for this, either the program or the source file is virtually guaranteed to get lost. And finally, running a program is a rather long and resource-consuming task.

But, if not a program, then what? After all, MPE does not even have a DISPLAY command to print a message, much less a command that will check whether a file exists and display one message if it does and another if it doesn't.

At this point, I must make a confession; despite what I said of the possibilities of MPE as a systems programming language, it was by no means created to be a systems programming language. In fact, you will find that most of the techniques that will be described are actually methods of subverting MPE commands to do tasks that they were never intended to do in the first place. However, they work, and that's what counts.

Returning to the problem at hand, let us attack it one step at a time. For one, as I said, MPE does not provide us with a DISPLAY command. So, we'll make one!

UDCs are permitted to have a number of options. One of these options, LIST, instructs MPE to list out the commands in the UDC as they are executed. Furthermore, there is an MPE command called :COMMENT that does absolutely nothing. So, what do we get when we cross an OPTION LIST and a command that does nothing?

```
DISPLAY !STRING
OPTION LIST
COMMENT !STRING
```

When the above UDC is invoked via a command of the form 'DISPLAY "string"', it will execute the command 'COMMENT string' (which in and of itself will do nothing), but also list this command as it is being executed! Thus, if we don't mind seeing 'COMMENT' on the screen, we now have a way of displaying anything we want to on the terminal from within a UDC.

Thus, we've licked one of our problems—we can now display a message to the terminal. However, this still does not solve the other problem—determining whether a file exists or not and printing one message if it does and another if it doesn't.

Here, we must introduce a very important MPE construct (in fact, its only control structure)—the :IF command. With the :IF command its two sidekicks, :ELSE and :ENDIF, we can, depending on the value of a logical expression, execute one of two sets of commands.

Thus, our task can be expressed as follows:

```
LOGONUDC
OPTION LOGON
Check if HOL100 exists
IF it exists THEN
   DISPLAY "USING THE ONLINE SYSTEM"
ELSE
    DISPLAY "USING THE BATCH SYSTEM"
ENDIF
```

However, even before you start to furiously leaf through your MPE commands manual, you will probably begin to suspect that neither 'Check if HOL100 exists' nor 'it exists' is valid MPE syntax. In fact, there is no check-if-a-file-exists command in MPE. Or is there?

Well, if there is no command that will explicitly check whether a file exists, we ought to look for a command that, as a side effect, yields different results depending on whether a file exists or not. Furthermore, we would be able to differentiate these results using an :IF command.

Let us consider the :LISTF command. If we do a ':LISTF filename', the filename will be listed if the file exists, and a CI error 907 will be generated if it does not. Since we want as little output to the terminal as possible, we actually want to do a ':LISTF filename; $NULL', which will do nothing if the file exists, and print a CI error 907 if it does not. Furthermore, it turns out that the value of the last CI error is stored in a JCW (Job Control Word) called CIERROR, which can be interrogated via the :IF command. Thus, instead of 'Check if HOL100 exists' we should say ':LISTF HOL100; $NULL' and instead of 'it exists' we should say 'CIERROR< >907'. Thus, the solution to our problem is:

```
LOGONUDC
OPTION LOGON
SETJCW CIERROR=0
CONTINUE
LISTF HOL100;$NULL
IF CIERROR<>907 THEN
   DISPLAY "USING THE ONLINE SYSTEM"
ELSE
   DISPLAY "USING THE BATCH SYSTEM"
ENDIF
```

A few comments: 'SETJCW CIERROR = 0' makes sure that CIERROR is cleared before the :LISTF command. Since this is an OPTION LOGON UDC, it is guaranteed to be zero anyway, but in general it is conceivable that it was already 907 before the :LISTF command. More importantly, a :CONTINUE command was added before the :LISTF command to avoid the UDC aborting on the first error; a :CONTINUE (either in a UDC or a job stream) instructs MPE not to abort if the next command fails.

One other point: in addition to the appropriate message, this method leaves some junk on the screen, namely the LISTF command and the error message if the file does not exist (and thus the LISTF command failed) and in either case a 'COMMENT' from the DISPLAY UDC. This is actually rather easy to take care of—merely embed in the DISPLAY string some escape sequences to move the cursor and delete the unwanted lines and characters on the screen. If you're using printing terminals, though, you're out of luck. Thus, we have seen how using MPE alone we can perform some fairly complex tasks easily and efficiently.

So, from this, we can derive a sort of MPE programming methodology:

1. If you see no direct way of performing a given task, try to find a way that yields the desired effect as a side effect, with little or no other direct effects or side effects.

2. If you wish to do two different things depending on some condition that can not be straightforwardly expressed with JCWs, try to find a command or sequence of commands that yields two different JCW values depending on the condition.

Let us take another example:

One of VESOFT's products, MPEX, is an extended MPE user interface that provides many desirable features, and is often "lived in" by its users i.e.—they run it once when they sign on, and stay in it until they are done, when they exit it and immediately sign off.

Some of our users decided to set up an option logon UDC of the form

```
MPEX
OPTION LOGON
RUN MPEX.PUB.VESOFT
BYE
```

This way, they would be automatically dropped into MPEX when they sign on, and will automatically be :BYEd off when they exit it. However, they do not want this to be done for jobs, but rather only for sessions. Thus, the task is to determine within a UDC whether one is in a job or a session.

In my opinion, in addition to the already existing system-defined JCWs such as JCW and CIERROR, HP should have provided us with JCWs such as MODE (to indicate whether we are a session or a job), FSERROR, etc. However, the fact remains that it did not, and we have to determine this for ourselves.

Let us apply our rule #2—is there a command that yields somewhat different results for job mode and session mode? In fact, there is. The :RESUME command, when executed from within session mode (but not from break mode, since the UDC will never be executed from within break mode) yields a CIWARN 1686 (COMMAND ONLY ALLOWED IN BREAK); however, when executed from within job mode, it issues a CIERR 978 (COMMAND NOT ALLOWED IN JOB MODE). Furthermore, since this is an OPTION LOGON UDC and will thus never be executed from break mode, the RESUME command has no other effects! Thus, our solution would be:

```
MPEX
OPTION LOGON
SETJCW CIERROR=0
CONTINUE
RESUME
IF CIERROR<>978 THEN
   RUN MPEX.PUB.VESOFT
   BYE
ENDIF
```

As an additional nicety, we may wish to do something like a 'DISPLAY "PLEASE IGNORE THE FOLLOWING MESSAGE"' before the RESUME command so that the user will not be puzzled by the warning that the RESUME command issues in session mode.

So, score another point for UDC programming.

To round out this section, consider one more example:

Before performing a given task, we wish to find out whether a given file is in use or not. If it is not in use, we should perform the task; if it is in use, we should print an error message.

Solving this problem requires a substantial amount of knowledge file system. What we really want to do is to try to open the file with EXCLUSIVE, INPUT access; if the open succeeds, we want to close the file with SAVE disposition; if it fails, we want to set a flag.

However, we can not explicitly open and close files in

MPE. Rather, we have to find a command to subvert so that it would do this task for us. This command's operation should be essentially similar to our target operation (i.e. it should do an open followed by a close). One command that pops to mind is the :PURGE command. Unfortunately, it opens a file with OUT access and closes it with DEL disposition.

But, via the :FILE command, we can force it to open and close the file with whatever options we please! Thus, our task may be achieved by doing the following:

```
FILE F=filename;EXC;ACC=IN;SAVE
SETJCW CIERROR=0
CONTINUE
PURGE *F
IF CIERROR=384 THEN
  DISPLAY "ERROR: FILE IS IN USE"
ELSE
  the file is not in use; do what is necessary
ENDIF
RESET F
```

Note that any open failure (except 'nonexistent file') during a :PURGE command causes a CIERR 384; furthermore, the last file system error is not accessible as a JCW, so we have to assume that no other open failure will occur.

Advanced MPE Programming

Consider the following problem:

VESOFT distributes its products on a tape along with an installation job stream. When a user wishes to install the products, he :RESTOREs the job stream and streams it. The job stream creates the appropriate accounting structure, and then :RESTOREs all the relevant files off the installation tape. However, it is possible that some files can not be restored; in this case, we want to send an appropriate message to the console.

The obvious thing to do here would be to check CIERROR to see if :RESTORE failed, and if so, do a :TELLOP. But, :RESTORE does NOT set CIERROR if not all files were restored! It merely prints the filenames and the count of the files that were not restored to its list file, and terminates just like all files were restored.

We have run into a problem that we can't really solve with the techniques outlined above because no MPE command can examine the contents of a file for us. However, there is one HP utility that is made explicitly for examining the contents of files—EDITOR!

Our plan of attack will be as follows: we will redirect the listing of the :STORE command to a disc file (by setting a file equation for SYSLIST), massage it with EDITOR, somehow cause EDITOR to set a JCW depending on the number of files not stored, and then, when we're back in MPE, examine that JCW.

So, our "program" will look like this:

```
:FILE SYSLIST,NEW; DEV=DISC; REC=-80,16,F,ASCII; NOCCTL; TEMP
:RESTORE *VESOFT; @.@.VESOFT, @.@.SECURITY; SHOW; OLDDATE
:RESET SYSLIST
:SETJCW FILESNOTRESTORED=0
:EDITOR
TEXT SYSLIST
LIST ALL
CHANGEQ "FILES NOT RESTORED", ":SETJCW FILESNOTRESTORED" IN ALL
DELETEQ 1/*-1,*+1/LAST
KEEP $NEWPASS,JNNUMBERED
USE $OLDPASS
EXIT
:IF FILESNOTRESTORED<>0 THEN
: TELLOP SOME FILES NOT RESTORED, CHECK SPOOL FILE!
:ENDIF
```

What does this mess do? Well, the first three lines do a :RESTORE, redirecting this listing to a disc file. Then, we enter EDITOR and text in the list file. Now, we have to make EDITOR set a JCW depending on the number of files not restored. The way that we do this is by changing the 'FILES NOT RESTORED = xxx' line to ':SETJCW FILESNOTRESTORED = xxx' with the CHANGE statement, deleting all the other lines in the file, keeping this as a temporary file, and USEing this file! The USE command will read the file and execute the :SETJCW command that we put in it; now, when we exit EDITOR, the FILESNOTRESTORED JCW is equal to the number of files not restored, and can now be interrogated.

This kind of trick is a very valuable one, and should be added to our methodology:

3. If the parameters of an MPE command (in this case :SETJCW) depend on the result of another MPE command (in this case :STORE), redirect the listing of the latter into a disc file, and use EDITOR to create and execute the former. Similarly, if the input of a program depends on the result of another program or command, redirect the listing of the latter into a disc file, and use EDITOR to create the input file for the former.

This point is best explained by another example:

VINIT, an HP utility, has a '>PDTRACK ldev' command, which prints the addresses of all the defective disc tracks on the disc device indicated by ldev. However, VINIT has no '>PDTRACK ALL' command. Implement it.

Applying our methodology, our strategy should be:

A. Find a command that lists all the disc devices in the system, and redirect its output to a disc file.

B. Using :EDITOR convert this output into input for VINIT.

C. Run VINIT using this newly-generated input file.

For step A, one command that seems to fit the bill is :DSTAT ALL. This little-known command produces output of the form:

```
LDEV-TYPE STATUS   VOLUME (VOLUME SET-GEN)

1-7925    SYSTEM   MH7925U0
2-7925    SYSTEM   MH7925U1
3-7925    SYSTEM   MH7925U2
```

As you see, this command displays, among other things, the logical device numbers of all the discs in the system. However, one problem comes up immediately: unlike the :STORE command, whose output can easily be redirected to a disc file, :DSTAT ALL's output always goes to $STDLIST.

So, how are we to redirect the output of a command that can only send its output to $STDLIST? The answer is simple: redirect $STDLIST! Although we can not redirect the $STDLIST of a job or of a command, we can redirect the $STDLIST of a program. So, all we need to do is to issue the following commands:

```
:FILE LISTFILE,NEW; REC=-80,,F,ASCII; NOCCTL; TEMP
:RUN FCOPY.PUB.SYS; STDLIST=*LISTFILE
:DSTAT ALL
EXIT
:RESET LISTFILE
```

What we do is run FCOPY with its $STDLIST redirected to a disc file, and cause it to do a :DSTAT ALL. :DSTAT ALL will obediently print its output to $STDLIST, which has been redirected!

So, we have the :DSTAT ALL listing (along with some other stuff printed by FCOPY) in a temporary disc file called LISTFILE. Now, it is time for step B—converting this :DSTAT ALL list file to a VINIT input file:

```
:FILE INFILE;TEMP
:EDITOR
TEXT LISTFILE
DELETE 1/6,LAST << delete the various headers >>
FIND FIRST
WHILE
  FIND "-" <<delete everything after the "-", >>
  DELETE *(*)/*(LAST) <<leaving only the ldev >>
CHANGE 1,"PDTRACK",ALL << insert PDTRACKs before the ldevs >>
ADD << add an EXIT command >>
  EXIT
//
KEEP *INFILE
EXIT
```

We now have the VINIT input file; all we need to do is

```
:FILE INFILE,OLDTEMP :RUN PVINIT.PUB.SYS;STDIN=*INFILE
```

and we're done!

We finish off this section with one more example:

VESOFT's installation stream signs on as MANAGER. SYS, builds the VESOFT and SECURITY accounts and streams two jobs, which sign on as MANAGER. VESOFT and MANAGER.SECURITY and build the VESOFT and SECURITY accounts. It is also the duty of the MANAGER.SYS job stream to restore the VESOFT and SECURITY files. However, it can not do this until the other two jobs finish. How can we make the MANAGER.SYS job stream wait for the others to terminate?

The key word in this problem is "wait". Again, on the surface it seems that MPE has no comand that permits one to wait for a certain event to occur. Again, however, a trick exists that saves the day. This trick uses MESSAGE FILES.

Message files are a kind of file (introduced in MPE IV) that have the property that if a reader tries to read an empty message file, he does not get an immediate end of file, but rather suspends until the message file is no longer empty.

So, even before the two job streams are streamed, we build two message files in PUB:SYS: MSGVESOF and MSGSECUR. Furthermore, in each of the two internally streamed job streams, after we are all done, we write a record (via FCOPY) to the appropriate message file. And, in the main (MANAGER.SYS) job stream, right after we stream the two other job streams but before we do the :RESTORE, we read the two message files (again, via FCOPY). The resultant job stream goes like this:

```
!JOB MANAGER.SYS
!NEWACCT VESOFT
!NEWACCT SECURITY
!BUILD MSGVESOF
!RELEASE MSGVESOF <<so the job stream can write to it >>
!BUILD MSGSECUR
!RELEASE MSGSECUR
!STREAM ,# << stream the two other job streams >>
#JOB MANAGER.VESOFT
...
#FCOPY FROM;TO=MSGVESOF.PUB.SYS
VESOFT ACCOUNTING STRUCTURE BUILT! << any message will do >>
#EOJ
#JOB MANAGER.SECURITY
...
#FCOPY FROM;TO=MSGSECUR.PUB.SYS
SECURITY ACCOUNTING STRUCTURE BUILT!
#EOJ
!FCOPY FROM=MSGVESOF;TO << wait for the VESOFT stream >>
!FCOPY FROM=MSGSECUR;TO << wait for the SECURITY stream >>
!RESTORE ...
!EOJ
```

The message file reads cause the job stream to suspend until the message files are non-empty, i.e. until the other job streams have written something to them. Thus, when the :RESTORE is executed, we are assured that the VESOFT and SECURITY accounting structures have been built.

Conclusion

I have presented some examples and some guidelines that should give the reader an idea of what MPE programming can do and how it can do it. It is my belief that with this knowledge and some ingenuity, the reader can use the art of MPE programming to advantage.

15 Ideas on Improving MPE Security

Norman B. Wright

A few years ago, when the number of Hewlett Packard 3000 sites was somewhat less than one thousand, it used to be sufficient to put a few passwords and lockwords on key accounts and files. We could then take refuge from our worried management behind the mythical "technically knowledgeable user". "The system is secure," we would say, "except from the technically knowledgeable user who is intent upon breaking its security". For many installations, this provided a moderate degree of safety. We could be relatively certain who the few technically knowledgeable users were who would be capable of breaking security. We could also take steps to assure that these users were not maliciously intent on circumventing security. At worst, we could keep a very close eye on them.

No more! The user community at most Hewlett Packard 3000 sites has outgrown the ability of one system or security manager to be personally in touch with each member. Furthermore the sophistication and knowledge of even casual users has grown to such a point that very few of us can take refuge in the myth of the "knowledgeable user". Most users can be assumed to have had previous exposure to computers, and to be in some degree aquainted with operating systems and utilities. The widespread use of microcomputers is proliferating this knowledge to a point where most of us have users who are not professional programmers, but who nonetheless know enough to attempt disk dumps, system crashes, and security breaches of considerable ingenuity. Since the movie *TRON*, every system can be said to be fair game for this sort of attempt.

The following ideas are offered, not as an exhaustive checklist of security measures, but as a list of workable ideas which you may wish to consider in setting up or improving the security of your HP3000 installation.

1. Establish control over the physical security of the computer itself. While the advent of the minicomputer brought a breath of fresh air to the large "closed shop" environment, the growth to "super" minis has brought us back full circle. We have met the enemy and he is us. Most HP3000 installations now deal with information which is far too valuable or sensitive to afford the luxury of the "open shop". At a minimum the computer and its tape and disk library should be in a secured environment with only those persons absolutely required for its operation able to enter.

2. Appoint a security manager. Have this person spend a certain amount of time thinking about security each month, in proportion to the amount of potential loss at stake. One of the key points in your security program should be that it is always changing, and continually improving. The security manager should carry on a continuing risk analysis, pinpointing current vulnerabilities of the installation. He or she should be inventive enough to consider all potential motivations: financial gain, malicious sabotage, corporate embarassment, and mischievous fun. Your best source of what is vulnerable on your system will always be your own in-house technically knowlegeable users. Keep them thinking regularly about security problems on your system. There are always going to be holes and weak points. Concentrate on the ones that are the most obvious with the highest potential loss to the installation.

3. Make your personnel security conscious. Make certain that they understand the sensitivity of certain data and are following established procedures in dealing with it. The greatest security risks, of course, involve your own personnel who must have day-to-day contact with sensitive or valuable information. Fortunately, this is also your first line of defense. Make certain that the need for security is known and understood by all employees. Check frequently to see that established procedures are being followed, not being pushed aside in the crush of day-to-day business. Make sure that your employees feel free to report even accidental or casual security violations to the security manager.

4. Establish manual or automated cross-checking procedures for information which is particularly valuable or sensitive. As with money, it is usually better to have at least two people involved in the handling of sensitive data so that collusion between them would be necessary for fraud or theft to be perpetrated.

5. Pay particular attention to the movement of magnetic tape, disk, and other media. Regardless of how elegant and effective your online security techniques might be, they could always be rendered useless by the theft of a single system dump or backup tape from your installation. The only way to protect against this (short of data encryption) is to establish very tight controls on the removal of such media. Dump tapes in particular may need to be kept under lock and key and bulk erased after they have expired. If you have tape or disk media which are routinely shipped or taken from the site, you may want to establish a program of cross checking their contents. At any rate, insist upon accurate logs for all information on magnetic media which leaves your computer room, including a record of what was taken, who took it, where it went, and for what purpose.

6. Store sensitive data separately. Due to the storage and handling problems with dump tapes, you may wish to consider backing up and storing particularly sensitive or critical data separately from your

SYSDUMP procedures. Backup media for the sensitive data can then be subjected to additional cross checking, perhaps even placed in custody of someone who will take overall responsibility for its security. Since it is on independent media, it can be placed under seperate lock and key, and purged from the system prior to all SYSDUMP procedures. If you wish to make doubly sure the data is destroyed from the disk, overwrite it instead of purging it. The program "BLATFILE" in the User's Group Library performs this function.

7. Lock up the key capabilities of the system and check them frequently. It is well known on the Hewlett Packard 3000 system that users with privileged mode capability (PM) or with system manager (SM) can easily break almost all security mechanisms. Reserve the use of these capabilities to a few users and make certain that extra precautions are exercised over them. If your system account structure is highly volatile, you may wish to set up auditing procedures to check, at periodic intervals, to make certain that these capabilities have not "leaked" out to other users. Privileged mode is notorious for doing this since it is also, by some quirk of MPE, required for restoring data bases. The CS capability for using the distributed systems lines, is another one which you should consider restricting if your installation uses this facility.

8. Use the MPE password system or a good alternative. MPE password protection at both the account and user levels has some excellent advantages, if used correctly. Making your passwords randomly generated strings of letters and numbers affords a measure of increased security which is highly recommended. You should plan on changing passwords periodically, at irregular intervals, perhaps to coincide with the departure of key personnel such as programmers or operators. Remember that these personnel frequently gain privity to passwords other than those authorized to them. Using MPE's double password system allows you to change the global account passwords and leave the user passwords the same. Changing passwords has the twofold advantage of requiring the security manager to keep up-to-date records of the user population and requiring the user population to keep in close touch with the security manager. Users who no longer have current need for access to the system, but who have failed to notify the security manager, will be automatically excluded by these periodic changes.

9. Get the passwords out of your streams. In order to change passwords easily and painlessly, you must develop methods for removing them from job stream files used as a regular part of development and production. There are a variety of packaged programs and utilities available to help with accomplishing this. They include the extended stream facility of Vesoft's MPEX, and at least two programs available free in the Users Group Library—JES and STREAMER. All of these programs depend upon programmatic insertion of the passwords into the job stream file before it is streamed. The password is usually obtained from a password file or from the system itself at run time. If one of these packages does not have enough flexibility for your installation, it will pay you to write a simple one yourself. The requirement for passwords in the job stream file is a major security problem in MPE which will also make changing passwords regularly a forbiddingly burdensome task.

10. Use the LOGON,NOBREAK UDC to control users. This is particularly applicable for users who are dedicated to only a few different functions on the system, such as payroll or inventory clerks. A properly constructed UDC can tightly restrict what such a user could do on the system, allowing him or her to access only those functions which are authorized. The UDC can be set to automatically log off the user upon completion of the specific function. Setting the UDC on an account-wide basis (;ACCOUNT) will alleviate the time-consuming task of having to log on to each newly created user id. For those few users whom you wish to allow privileged access on the account, you can set UDC's with an overriding commanConsider writing your own security screening program. This program could be used in conjunction with a system or account-wide UDC to check for a variety of user defined security violations. Many installations may wish to restrict certain users to specific terminals (logical devices), or to specific time periods during the day or week. In designing such a program you may find the seldom-used LOCATTR attribute of the user id useful for further screening and restricting user capabilities. A user-written security screening program can also do additional password or protection prompting, and logging for installations where several users are using the same user id. An interesting example of this type of program is found in the KMGR program in the Users Group Library which provides extra logon security for privileged accounts A security screening program, coupled with the LOGON UDC provides good capability for customized sign-on protection.

12. Consider disabling the :LISTF command entirely. MPE's :LISTF command has been faulted frequently because it gives all users the capability of listing the file directory contents of the entire system. Thus, for example, when the user sees the program DISKED2. PUB.SYS, the temptation to experiment can prove almost overpowering. Particularly on college campuses, where some of the most severe security problems of this kind exist. locking up the :LISTF capability and a variety of other capabilities based on it (LISTDIR2, PURGEFILE, etc) seems to be a good precaution. Alternative commands for listing the user's group and account fileset can, of course, be provided. Almost any use of the UDC in a security program, it should be added, will mean that the :SET-

CATALOG command itself will also have to be disabled at the user level. Otherwise the user will easily learn to circumvent the UDC commands with an overriding UDC. A good program of system and account UDC's can frequently alleviate the need for numerous user UDC's anyway. You may wish to abandon the many problems of MPE's UDC facility entirely, in favor of a programmatic capability designed as an integral part of the security screening program.

13. Consider using private volumes to enhance your security. Much has been written about the use of the private volume capability in terms of increased data handling flexibility and backup. However, private volumes also provide one of the best methods for tightly restricted access to key data on the system. Data such as a payroll account or other critical information can be physically removed from the system when not in use on line. When the information is required on line, it is protected from unauthorized tampering by the necessity of UV (use private volumes) capability which can be granted only to the authorized users.

14. Program security into your applications. Regardless of what measures you take to restrict access to the system, you are also going to have to protect against the inside job—the authorized user who uses the data in unauthorized ways. As we mentioned earlier, your best line of defense will always be other personnel and system cross checks designed to prevent this. However, most users find they must also look at the applications programs or packages themselves to further identify restrictive mechanism which can be implemented. A key feature to all sensitive applications should be some form of logging facility to track what transactions were made and by whom. The logging capability could be a built-in one, such as IMAGE's transaction logging, or it might be one which is designed into the application. Logging capabilities frequently serve a variety of useful functions in addition to the security function.

15. Establish a vigorous random auditing program. Your entire security edifice will collapse without constant monitoring to determine if and when security breaches are being attempted. The security manager should bring into play everything that he knows about the system to periodically monitor activity. Use the log files and programs which manipulate them (LISTLOG2, READLOG, CENSOR, etc.); use online monitors (OPT3000, S00IV); use programmatic or manual checks on other logs such as DBAUDIT for IMAGE data base logging, or monitoring your own set of logs from security or transaction screening programs. The auditing program should check for application-defined "unusual", "excessive", or "special" conditions. Try to make the programmatic definitions of these terms parameterized so that they can be varied. One crucial element of the auditing program is that it must be constantly changing and improving. As quickly as one check or audit is permanently installed and performed on a regular basis it can be assumed that another way will be found to circumvent the existing checks. Only by constantly changing and improving the security system faster than the sophistication of your average user—a sophistication which is itself constantly increasing—can you hope to offer any assurance of a secure system.

JOURNAL
OF THE HP 3000 INTERNATIONAL USERS GROUP, INCORPORATED

Advanced Techniques Using VPLUS

Michael A. Casteel
Vice President
Computing Capabilities Corporation
Mountain View, California

Introduction

This paper is intended to cover two particular topics of interest to a number of VPLUS users. It will cover procedures for alternating between VPLUS block mode, using formatted screens, and conversational mode such as used with MPE and utilities. This technique may be used to integrate existing conversation mode dialogues with new VPLUS applications. Perhaps more generally, it can be of tremendous value when debugging VPLUS applications, especially when two terminals are not available within arm's reach.

Also presented are procedures for printing the screen contents, either to an attached or integral printer or to the system printer. Of course, these procedures are not specific to VPLUS applications and as such may be of even broader utility.

Both of the particular topics of this paper involve communication with and control of the terminal. It is important, therefore, to cover some background material first, in order to understand the terminal configuration and communications protocol in effect in the VPLUS environment. Printing the screen or switching from block/format to conversational mode are not particularly difficult, once the VPLUS operating mode is understood. Happily, all VPLUS-supported CRT terminals are basically compatible. There are some significant differences between point-to-point and multipoint (MTS) operation, which will be covered in the discussion.

Terminal and I/O Configuration

The information in this section has been deduced from assorted terminal and software manuals, observations and conversations. It can't all be guaranteed correct, but has so far proven out in those cases where it was needed.

Basic Terminal Configuration

A few of the Keyboard Interface straps (optional) must be set in a particular way for VPLUS operation. For 2640B or 2644 terminals you must do this manually, by opening the terminal and setting switches. VPLUS sets the others automatically during VOPENTERM or VGETNEXTFORM with $REFRESH. Special point-to-point options are:

D - open (Line/Page) Establishes Page mode, whereby the ENTER key transmits the entire screen instead of only a single field or line.

E - open (2640B) Allows the terminal function keys (f1-f8) to be used without holding the CNTL key. (Optional)

F - open (2640B) Provides for 2645-compatible handshake protocol (DC1/DC2/DC1).

G - open (InhHndShk) Provides that computer-requested block transfers (e.g. terminal status, cursor sense, printer command response) observe the DC1/DC2/DC1 protocol.

The main strap of interest in multipoint (MTS) is:

J - closed (Auto Term) MTS opens this strap, which has the effect of limiting data transmission to data on the screen above the cursor position at the time the ENTER key is pressed. VPLUS closes this strap to allow transmission of all data in the form.

There are other straps which are important to VPLUS operation, but the "normal" setting is the appropriate one.

In addition, VPLUS requires that the terminal be set for block mode. This is the normal mode in MTS. In point-to-point, VPLUS will set the terminal in block mode automatically (except 2640/44).

Terminal File Configuration

VPLUS controls and communicates with the terminal through the MPE file system using the standard set of intrinsics. Some special file system parameters are set for the point-to-point and multipoint device drivers under VPLUS, generally via the FCONTROL intrinsic. Significant FCONTROL functions for point-to-point operation are:

13 - Echo is turned off (if it isn't already)
25 - Set alternate terminator to RS
31 - Enable VPLUS driver control
38 - Set terminal type to 10 (if it isn't already).

Character Echo (FCONTROL 12/13)

Normal full-duplex point-to-point operation requires the HP3000 to echo each character it receives back to the terminal to be displayed. In block mode, under VPLUS, each character is displayed on the screen as you type it, and nothing is sent to the computer until you press ENTER. Since everything you type is already shown on the screen, a computer echo of the block transmission would only confuse matters. Therefore, FCONTROL 13 (disable echo) is used.

Alternate Terminator (FCONTROL 25)

VPLUS uses the "ultimate" form of block mode on HP terminals, i.e. Block/Page mode. This allows the terminal to send the whole screen at once, the most efficient form of block transmission. However, full page, block mode inputs do not end in the usual carriage return (CR) code which terminates other inputs; in fact, there may be a number of carriage returns in the midst of the input. Instead, HP terminals send a control code called "Record Separator" (RS) to signal the end of the block. VPLUS accommodates this by setting the RS code as the "alternate terminator" in point-to-point operation, using FCONTROL 25.

Terminal Type (FCONTROL 38)

Terminal ports used by HP CRT terminals are usually configured as Terminal Type 10, which signals the I/O driver to observe certain protocols which are appropriate to such terminals. For example, the driver will send a control code, ENQ, after every 80 characters output. HP terminals will answer with an ACK when they are ready for the next 80 characters. This is the famous ENQ/ACK handshake. When you use VPLUS you must be using an HP compatible terminal; so VPLUS sets the Terminal Type to 10 using FCONTROL 38, in order to activate these protocols.

VPLUS Driver Mode (FCONTROL 30/31)

FCONTROL 31 is more puzzling, since HP has so far neglected to document it. When reading from the terminal under this option, a DC2 code as the first character received causes the MPE device driver to set up a block read. This is necessary since, when you press ENTER, the terminal doesn't just send the screen contents as a big block of data. It only sends the single control code, DC2, and waits for a DC1. VPLUS used to handle the DC2 itself but now uses the new Driver Mode. Now, on receipt of the DC2, the MPE device driver (not the program, not VPLUS) responds with:

<esc>c<esc>H<DC1>

where <esc> stands for the ASCII 'ESCAPE' control code, and <DC1> for the DC1 code. These control codes lock the keyboard, home the cursor, and trigger the block data transfer. The driver times the block read, in case the terminator character is lost. The read terminates by character count, timeout or receipt of an RS code (the VPLUS alternate terminator, i.e. the Block/Page Mode terminator character). It is natural to assume that the block read functions armed with FCONTROL 31 have been moved into firmware on the ATP.

MTS Unedited Input (FCONTROL 41)

The above functions are applicable to point-to-point terminals. For multipoint terminals, the MTS driver can handle block transmissions pretty much as usual. The only important difference to the driver is one FCONTROL:

41 - Set unedited mode (with parameter %137)

This option is used to stop MTS from placing a block delimiter everywhere VPLUS puts the cursor. Block delimiters set by MTS (or the ENTER key with the terminal's strap J open) would prevent the transmission of the full screen to the computer.

User Read Time-out (FCONTROL 4)

For point-to-point or multipoint, VPLUS will also use FCONTROL:

4 - Enable read time-out

when OPTIONS (word 56 of the Comarea) has bits 9-10 set to 01. USER'TIME (word 58) gives the number of seconds to allow for input.

Conversation vs. VPLUS Mode

There are occasions in many applications where it is desirable to remove the terminal from VPLUS operation for a while, then resume VPLUS. For example, you may wish to run another program which doesn't use VPLUS, or just engage in a conversational dialogue. Debugging, with DISPLAYs and ACCEPTs or PRINTs and READs, or using MPE Debug or Toolset, is an almost universal occasion for conversation mode.

Normally, a program resumes conversation mode with a call to VCLOSETERM when it terminates. This suggests an approach to switching modes: To change from VPLUS mode to conversation mode, call VCLOSETERM. To switch back, call VOPENTERM. Since VCLOSEFORMF isn't called, a lot of valuable information is preserved:

Current form name
Data buffer contents
Next form name
Repeat/Next form options
Screen label settings
Save field contents

This may be the best approach to use when you wish to switch modes in order to run another program. There's no telling in what state the other program will leave the terminal and I/O configuration, but VOPENTERM should be able to sort it out. In fact, the undocumented intrinsics VTURNOFF and VTURNON should have about the same effect, without taking quite such drastic steps as closing and re-opening the terminal file, clearing the screen, and so on. Parameters are the same as VCLOSETERM and VOPENTERM, respectively.

If you take this approach, you will discover a few complications which may (or may not) affect your application:

1. **Your screen remains empty.**

VOPENTERM clears the display, but VSHOWFORM doesn't know it. This means that your next call to VSHOWFORM may not write any data to the screen, since VPLUS believes the last form to be still there. This is a result of VSHOWFORM optimization, which you can correct by moving 7 to word 34 of the Communications Area (SHOWCONTROL) before calling VSHOWFORM. Don't forget to reset SHOWCONTROL afterward (New requirement with the Q MIT). Or, you may call VGETNEXTFORM with Next Form name $REFRESH.

2. **Your screen is still empty.**

VOPENTERM reconfigures workspaces on a 2626 using local form storage, but VSHOWFORM doesn't know it. In this case, moving the 7 to SHOWCONTROL might just cause VSHOWFORM to try to redisplay the form from the workspace where it used to be. You need to use $REFRESH in this case, or perhaps VLOADFORMS with SHOWCONTROL. As of this writing, it is too early to tell the effect on 2624B Local Forms Storage.

3. **Your screen labels are missing.**

VCLOSETERM removes your screen labels from the terminal, but VOPENTERM doesn't put them back. It is anticipated that you can use $REFRESH to correct this problem in the Q MIT. We can only hope that VOPENTERM will also be corrected. To solve this problem prior to the Q MIT, follow VOPENTERM with VSETKEYLABELS, for global or form labels, whichever is appropriate. I use VGETKEYLABELS to retrieve the needed values, so they don't need to be coded into the program.

You may actually have to do two VSETKEYLABELS, due to VPLUS optimization. If you simply set them to the same value they had, VSHOWFORM will not bother to put them on the screen. I get around this by first setting the labels to a different value, then setting them back.

Note: As of VPLUS B.02.02, you must do your VGETKEYLABELS before calling VCLOSETERM, since the latter destroys the values in the VPLUS label buffer.

Switching Modes Yourself

Most often all you want to do is switch modes quickly in order to display a message or allow some form of debugging dialogue. For example, the VPLUS debugging facilities in INSIGHT II and RADAR (two Computing Capabilities Corporation products) keep the terminal in conversation mode except during VPLUS output or input. As a result, Debug breakpoints, abort messages and VPLUS screens are all accommodated on a single terminal.

This is not a difficult task, a it turns out, made easier since the addition of FCONTROL 30/31 (VPLUS Driver Mode) for point-to-point terminals. Appendix 1 includes a listing of a short SPL procedure, VSETMODE, which performs the necessary functions. This procedure has two parameters: The VPLUS Communications Area (just like VPLUS intrinsics) and an integer mode. Mode zero calls for VPLUS operation, and one calls for conversational mode.

This routine operates outside the bounds of documented VPLUS operations. This means it is possible that HP could change things around in such a way that it won't work. This is not highly likely, since the operations performed are so fundamental.

VSETMODE Comarea Usage

You will see that VSETMODE uses three words in the VPLUS Comarea. The first two are the terminal file number and the terminal model as determined by VPLUS. Both are documented in the current edition of the VPLUS Reference Manual.

The third word used is not doucmented in the VPLUS manual. Bits 4–9 of this word contain the original MPE Terminal Type. We need this to learn if the terminal is an MTS terminal, designated by Terminal Type 14. This (MTS) information could also be obtained using FCONTROL 39 on the terminal file, without reference to this word.

Bit 1 is used to determine whether character echo was on before VOPENTERM. Half-duplex connections, for example, should not have echo turned on. The essential information could be obtained without reference to this word by using FGETINFO to check for a half-duplex subtype.

VSETMODE Operation

To enter conversational mode, VSETMODE first conditions the terminal by writing a sequence of control commands. This begins by turning off Format Mode, moving the cursor to the bottom of memory, and unlocking the keyboard:

<esc>X<esc>F<esc>b

On terminals which support this operation, VSETMODE also turns off block mode:

<esc>&kOB

On the new line of terminals, the aids and modes keys will be unlocked:

<esc>&jR

When returning to VPLUS mode, these operations are reversed, although the keyboard, aids and modes keys are not locked.

If character echo was on before VOPENTERM, it will be turned on again by VSETMODE when entering conversation mode and off when resuming VPLUS mode (FCONTROL 12 and 13, respectively).

Since the introduction of VPLUS Driver Mode for point-to-point terminals, it is no longer necessary to change other MPE file parameters. Your terminal will operate normally unless you both:

1) Read input using the terminal file number in the VPLUS Comarea.

2) Send a DC2 code to the computer (for example, press ENTER or a function key).

If you do both these things you may find that your terminal locks up. This is because the MPE device driver locks your keyboard when it receives the DC2. To free up the terminal you will have to:

1) Unlock the keyboard, e.g. soft reset.

2) Type an RS control code (control-^) to end the read.

If your terminal is connected via MTS rather than point-to-point, VSETMODE has to change another file parameter. In order to enter conversational mode, Unedited Mode (FCONTROL 41) set by VOPENTERM must be turned off. This is done by FCONTROL 41 with a parameter of zero. When resuming VPLUS mode, Unedited Mode must be turned on (parameter octal 137), and the terminal's J strap must be reset.

Printing the Screen Contents

There are many occasions when it would be useful to print the screen contents. There is a VPLUS intrinsic, VPRINTFORM, whose purpose is to print a copy of the current form and its contents. However, since this intrinsic only prints to a file, it is mainly useful for obtaining listings on a system printer.

If the screen to be printed consists of several forms, one call to VPRINTFORM will not print the entire screen. Instead, you must have the foresight to call VPRINTFORM as each form is displayed, thus piecing together a screen image in the print file at the same time it is being assembled on the terminal.

Printing to a Terminal Printer

A more direct approach is to perform a full screen print operation. The simplest way is to equip your terminals with local printers, either integrated or attached. It then becomes a matter of commanding the terminal to copy the screen to the printer. This produces a true hard copy of what was actually displayed. Also, the only foresight you need is to provide a routine which will issue the right command when needed. Appendix 2 includes a listing of an SPL procedure, VPRINTLOCAL, which will print the terminal screen to a terminal printer. This procedure requires only one parameter, the VPLUS Comarea.

My usual practice is to assign one of the terminal function keys as a PRINT key. Whenever the program accepts input through VREADFIELDS, I check word 6 (LASTKEY) in the VPLUS Communications Area. If the user pressed the PRINT key, call the Print routine, and when finished loop back to call VSHOWFORM and VREADFIELDS once more. VSHOWFORM is called to unlock the terminal keyboard which was left locked by VREADFIELDS. It also serves to display any message sent to the window by the Print routine.

VPRINTLOCAL Operation

VPRINTLOCAL uses two different approaches to performing its function. This is because there are two forms of print command recognized by HP terminals: a simple "dump the screen" available on 2640B and 262X terminals, and the more complicated device control sequence needed for the 2645-based terminals. Except on the latter, all it takes to "dump the screen" is to command the terminal to:

1) Turn off Format Mode
2) Copy the screen to the printer
3) Turn on Format Mode

This is a matter of sending six characters to the terminal:

<esc>X<esc>0<esc>W

The actual "screen print" command is the <esc>0 (that's the numeral zero). You need to turn Format Mode off in order to copy the protected areas of the form. If Format Mode is on when using an attached printer, the terminal assumes that the printer contains a preprinted form matching the screen. Only the data in unprotected fields will be printed.

VPRINTLOCAL sends this command using FWRITE to the terminal file number in the Comarea (word 49, FILEN). This takes advantage of the VPLUS Driver Mode in effect on this file, and will work even if the terminal is not the job/session logon device. This FWRITE includes carriage control code %320 (octal 320, decimal 208) as its fourth parameter. This value suppresses the carriage return and line feed which normally follow every output.

Printing on a 2645 is more complicated in two ways: first, the command sequence is longer; worse, the terminal insists on talking back. You have to program a dialogue with the terminal. VPRINTLOCAL starts by sending the operative command:

<esc>X<esc>H<esc>b<esc>&p3s4dM

This means,

<esc>X Turn off Format Mode

<esc>H Home the cursor

<esc>b Unlock the keyboard. This allows the user to cancel the print operation by pressing the Return key.

<esc>&p3s4dM Copy everything (M) from the display (3s) to the printer (4d) starting from the cursor position.

Note: This command will also work on the 262X terminal family. The '4d' code normally specifies the external (attached) printer. To use the integral printer, use '6d' or specify 'DeviceCode4' on the terminal's Configuration Menu as the INT printer.

This starts the printing process, if the terminal has an attached printer and the firmware to support it. Once the terminal has completed printing, or determined that it couldn't print, it will want to send a single character response:

S - Print operation completed.
U - Operation aborted by the user.
(User pressed Return)
F - Print not completed (out of paper, etc.).

VPRINTLOCAL must read this response, or the next VREADFIELDS will read it as if the user sent it with the ENTER key. This is done in the subroutine RESPONSE, using a timed FREAD (FCONTROL 4 before FREAD) in case something goes wrong. 60 seconds are allowed, which should be enough to print a screen unless you're using a very slow printer.

Note: Under VPLUS Driver Mode (FCONTROL 31), the terminal's response will cause the MPE device driver to lock the keyboard. You will need to call VSHOWFORM in order to unlock it before the next VREADFIELDS.

Finally, after receiving the terminal's response, turn format mode back on:

<esc>W

In order to determine which type of terminal it is dealing with, VPRINTLOCAL looks at VPLUS Communications Area word 59 (IDENTIFIER). Values of 1,8,9,11 or 13 signify 2640B or 262X terminals.

Screen Copy on 2626

Users with 2626 terminals may wish to use the 'Screen Copy' device control operation. This function is performed using the device control sequence:

<esc>&p4dE

Like the device control sequence on a 2645, this command produces a response which must be processed by the computer. The 2626 Screen Copy includes the screen labels on the printed output.

Application Notes

VPRINTLOCAL uses the more complicated device control sequence on everything but a 2640, which doesn't support it. This is because I want the printer to do a page eject after printing the screen, but this doesn't happen on my 2624 terminal if I just give it an <esc>0. The page eject results from the command <esc>&p4u5C.

My Direct 825 terminal pretends that it is a 2622, but does not recognize the device control sequence. Since VPRINTLOCAL sends the device control sequence to 2622s, the 825 gets confused. It's easy to command the terminal PRINT function from the keyboard on the 825, though: just Function/Print (9 on the numeric keypad). No need to take the terminal out of format mode, and it does a page eject, too.

Printing to a System Printer

It takes more work than printing to a terminal printer, but you can copy the screen to a file or system printer if you wish. The well-known program PSCREEN performs such a function. Appendix 3 includes a listing of a procedure, VPRINTSCREEN, which has been tailored to perform this function in the VPLUS environment, for both point-to-point and MTS terminals. This procedure accepts two parameters: The VPLUS Comarea and an integer carriage control code, which will be written after the screen has been printed. The value 49, for example, will produce a form feed.

Like VPRINTFORM, VPRINTSCREEN will print to any file if you place its MPE file number in word 36 of the VPLUS Comarea (PRINTFILNUM). If this word is zero, VPRINTSCREEN will open a file named FORMLIST on device LP, print the screen and close the file.

VPRINTSCREEN Operation

There are two operations of special interest in VPRINTSCREEN: Reading the screen contents and stripping control codes from the data before printing. These operations are performed in the subroutines READ'SCREEN and PRINT'LINE, respectively.

Reading the Screen

Reading the screen is a simple matter since the addition of VPLUS Driver Mode for point-to-point terminals. In order to read the entire screen, the terminal must first be removed from format mode. All that is required is the command string:

<esc>X<esc>d

After format mode has been turned off, the <esc>d commands the terminal to transmit the contents of its memory to the computer. The data is then read using FREAD. Note that VPRINTSCREEN uses the terminal file number in the VPLUS Comarea. This takes advantage of the VPLUS Driver Mode in effect on this file, and will support terminals which are not the job/session logon device.

The last character read is the block terminator character, RS for point-to-point or GS for MTS. The subroutine replaces this by a Carriage Return followed by RS for consistency.

Stripping Control Codes

The characters received from the terminal include not only form and field data but display control codes, such as:

<so> Shift to alternate character set
<esc>[Start field
<esc>&d<letters> Display enhancement

Each screen line is terminated by a Carriage Return.

The PRINT'LINE subroutine scans the characters in the line, skipping control codes, such as <so>, and escape sequences (starting with <esc>). There are two kinds of escape sequences: Standard (two characters), such as <esc>[, and generic, such as <esc>&d<letters>. If the character following <esc> signifies a generic escape sequence, PRINT'LINE skips to the terminator character, which is either an '@' or uppercase letter. During the scan, PRINT'LINE moves the printable characters to the start of the buffer, to ensure that the print line begins on a word boundary for FWRITE.

VPRINTSCREEN does not suppress printing of Security Video fields, which do not display on the screen. If you wish, you may add the logic yourself (send me a copy). In general, characters displayed in an alternate character set (i.e. following <so>) should not be printed either.

Acknowledgments

Thanks to Ross Scroggs for first documenting the precise effects of FCONTROL 30/31, and to Mark Cousins of CCC for the original version of VPRINTSCREEN. VSETMODE and VPRINTLOCAL have been lifted, almost bodily, from INSIGHT II, a VPLUS transaction processor.

Should the reader discover any improvements or corrections to any of these routines, in any environment (DS/1000?), the author would appreciate hearing from you.

```
Appendix 1. VSETMODE Procedure.

The SPL procedure will switch the user's terminal between
VPLUS block/format mode and MPE conversational mode.

PROCEDURE VSETMODE(COMAREA, MODE); VALUE MODE;

INTEGER MODE;                    << 0=VPLUS ; 1=CONVERSATION >>
INTEGER ARRAY COMAREA;
BEGIN

ARRAY BUF(0:19);                 << LOCAL OUTPUT BUFFER >>
BYTE ARRAY BBUF(*)=BUF;
INTEGER LEN;

DEFINE CONV      = (MODE=1)#,
       COM'TERM  = COMAREA(48)#,    << TERMINAL FILE NUMBER >>
       COM'IDENT= COMAREA(58)#,        << TERMINAL MODEL >>
       COM'TYPE  = COMAREA(49).(4:6)#,   << MPE TERM TYPE >>
       COM'ECHO  = COMAREA(49).(1:1)#;     << WAS ECHO ON >>

INTRINSIC FWRITE, FCONTROL;

<< FIRST, CONDITION THE TERMINAL >>

IF COM'IDENT > 2 THEN               << PROGRAMMABLE STRAPS >>
   IF CONV THEN MOVE BBUF := (27,"X",27,"F",27,"b",
                              27,"&k0B"),2
   ELSE MOVE BBUF := (27,"W",27,"&k1B"),2
ELSE                                          << 2640/44 >>
   IF CONV THEN MOVE BBUF := (27,"X",27,"F",10,
                              27,"bUNLATCH BLOCK MODE"),2
   ELSE MOVE BBUF := ("LATCH BLOCK MODE",27,"W"),2;
IF CONV AND COM'IDENT >= 8 THEN MOVE * := (27,"&jR"),2;
LEN := TOS - LOGICAL(@BBUF);
FWRITE(COM'TERM, BUF, -LEN, %320);

<< NEXT, RESET CHARACTER ECHO >>

IF COM'ECHO = 0 THEN
   FCONTROL(COM'TERM, (IF CONV THEN 12 ELSE 13), LEN);

<< FINALLY, TAKE CARE OF MTS >>

IF COM'TYPE = 14 THEN BEGIN                    << MTS >>
   LEN := IF CONV THEN 0 ELSE %137;
   FCONTROL(COM'TERM, 41, LEN);
   MOVE BBUF := (27,"&s0J"),2;
   LEN := TOS - LOGICAL(@BBUF);
   IF NOT CONV THEN FWRITE(COM'TERM, BUF, -LEN, %320);
   END;
END;
```

```
APPENDIX 2. VPRINTLOCAL Procedure.

This procedure prints the screen contents by [illegible]
CRT terminal to copy the screen to [illegible] attached [illegible]
printer.

PROCEDURE VPRINTLOCAL(COMAREA);
INTEGER ARRAY COMAREA;
BEGIN

ARRAY BUF(0:1[illegible]);                [illegible]
BYTE ARRAY BBUF(*)=BUF;
INTEGER TIME, LEN;

DEFINE COM'STATUS     = COMAREA#,          [illegible]
       COM'FILERRNUM  = COMAREA([illegible])#,     [illegible]
       COM'TERM       = COMAREA([illegible])#,     [illegible]
       COM'IDENT      = COMAREA([illegible])#;     [illegible]

INTRINSIC FCONTROL, FWRITE, FREAD, [illegible];

LOGICAL SUBROUTINE RESPONSE;        [illegible]
BEGIN
TIME := [illegible];
FCONTROL(COM'TERM, 4, TIME);
FREAD(COM'TERM, BUF, -[illegible]);
IF <> THEN
   BEGIN
   FCHECK(COM'TERM, COM'FILERRNUM);
   IF COM'FILERRNUM <> [illegible] THEN COM'STATUS := 1[illegible];
   END;
IF COM'STATUS = [illegible] THEN
   IF BBUF [illegible] "[illegible]" AND BBUF [illegible] "[illegible]" THEN COM'STATUS [illegible]
RESPONSE := COM'STATUS=0;
END;

IF COM'STATUS [illegible] THEN RETURN;         [illegible]
IF COM'IDENT [illegible] THEN                  [illegible]
   BEGIN
   MOVE BBUF := ([illegible],"X",[illegible],"[illegible]",[illegible],"[illegible]"),2;
   LEN := TOS - LOGICAL(@BBUF);
   FWRITE(COM'TERM, BUF, -LEN, %[illegible]);
   END
ELSE                                          [illegible]
   BEGIN
   MOVE BBUF := ([illegible],"X",[illegible],"[illegible]",[illegible],"[illegible]",[illegible],"[illegible]"),2;
   LEN := TOS - LOGICAL(@BBUF);
   FWRITE(COM'TERM, BUF, -LEN, %[illegible]);
   IF NOT RESPONSE THEN RETURN;
   MOVE BBUF := (27,"&p4u[illegible]"),2;       [illegible]
   LEN := TOS - LOGICAL(@BBUF);
   FWRITE(COM'TERM, BUF, -LEN, %32[illegible]);
   RESPONSE;
   END;
END;
```

APPENDIX 3. VPRINTSCREEN Procedure.

This procedure prints the screen contents to a file named FORMLIST on a device LP. Escape sequences and control characters are stripped out of the data before printing.

```
PROCEDURE VPRINTSCREEN(COMAREA, PAGECTL);
INTEGER ARRAY COMAREA;
INTEGER PAGECTL;
BEGIN

ARRAY BUF(0:2047);                          << LOCAL I/O BUFFER >>
BYTE ARRAY BBUF(*) = BUF;
LOGICAL LOCAL'FILE := FALSE;
BYTE POINTER BUFCHAR,
             LINECHAR;
INTEGER LEN;

EQUATE LF  = 10,
       CR  = 13,
       RS  = 18,
       ESC = 27;

DEFINE COM'STATUS       = COMAREA#,
       COM'PRINTFILNUM = COMAREA(35)#,
       COM'FILERRNUM   = COMAREA(36)#,
       COM'TERM        = COMAREA(48)#;

INTRINSIC FOPEN, FREAD, FWRITE, FCHECK, FCLOSE;

SUBROUTINE READ'SCREEN;                 << READ SCREEN CONTENTS >>
BEGIN
MOVE BBUF := (27,"X",27,"d"),2;
LEN := TOS - LOGICAL(@BBUF);
FWRITE(COM'TERM, BUF, -LEN, %320);
LEN := FREAD(COM'TERM, BUF, -4094);
IF <> THEN
   BEGIN
   FCHECK(COM'TERM, COM'FILERRNUM);
   IF COM'FILERRNUM <> 31 THEN COM'STATUS := 160;
   END;
IF LEN > 0 THEN LEN := LEN - 1;            << STRIP TERMINATOR >>
BUF(2047) := [8/27,8/"W"];              << RESTORE FORMAT MODE >>
FWRITE(COM'TERM, BUF(2047), -2, %320);
MOVE BBUF(LEN) := (CR, RS);     << SET STANDARD TERMINATOR >>
END;

SUBROUTINE FILE'ERROR;
BEGIN
IF COM'STATUS <> 0 THEN RETURN;
FCHECK(COM'PRINTFILNUM, COM'FILERRNUM);
COM'STATUS := IF COM'PRINTFILNUM = 0 THEN 190 ELSE 191;
END;

SUBROUTINE START'FILE;                 << OPEN COM'PRINTFILNUM >>
BEGIN
IF COM'PRINTFILNUM <> 0 THEN RETURN;
MOVE BBUF := "FORMLIST LP ";
COM'PRINTFILNUM := FOPEN(BBUF, %507, %4, -251, BBUF(9));
IF COM'PRINTFILNUM = 0 THEN FILE'ERROR
ELSE LOCAL'FILE := TRUE;
END;

SUBROUTINE PRINT'LINE;            << STRIP CONTROLS AND PRINT >>
BEGIN
@BUFCHAR := @BBUF;
WHILE LINECHAR <> CR DO
   IF LINECHAR = ESC THEN                  << ESCAPE SEQUENCE >>
      IF "&" <= INTEGER(LINECHAR(1)) <= "*" THEN
         DO @LINECHAR := @LINECHAR(1)
         UNTIL "@" <= INTEGER(LINECHAR(-1)) <= "Z" OR
               LINECHAR = CR
      ELSE @LINECHAR := @LINECHAR(2)
   ELSE
      IF LINECHAR < " " THEN @LINECHAR := @LINECHAR(1)
      ELSE
         BEGIN
         MOVE BUFCHAR := LINECHAR,(1),1;
         @LINECHAR := TOS;
         @BUFCHAR := TOS;
         END;
@LINECHAR := @LINECHAR(1);                           << SKIP CR >>
LEN := LOGICAL(@BUFCHAR) - LOGICAL(@BBUF);
FWRITE(COM'PRINTFILNUM, BUF, -LEN, %40);
IF <> THEN FILE'ERROR;
END;

SUBROUTINE END'FILE;
BEGIN
FWRITE(COM'PRINTFILNUM, BUF, 0, PAGECTL);     << DO PAGECTL >>
IF <> THEN FILE'ERROR;
IF LOCAL'FILE THEN                       << CLOSE IF OPENED HERE >>
   BEGIN
   FCLOSE(COM'PRINTFILNUM, 0, 0);
   COM'PRINTFILNUM := 0;
   END;
END;

<<
   HERE IS THE MAIN LOGIC OF VPRINTSCREEN
>>

IF COM'STATUS <> 0 THEN RETURN;           << JUST LIKE VPLUS >>

START'FILE;                              << OPEN COM'PRINTFILNUM >>
IF COM'STATUS <> 0 THEN RETURN;
READ'SCREEN;                             << READ SCREEN CONTENTS >>
IF COM'STATUS <> 0 THEN RETURN;

@LINECHAR := @BBUF;

DO PRINT'LINE                        << CLEAN AND PRINT LINES >>
UNTIL COM'STATUS <> 0 OR LINECHAR = RS;

END'FILE;                        << CLOSE FILE IF OPENED HERE >>
END;

END.
```

Journal
Administrative Offices
HP 3000 International Users Group, Inc.
289 South San Antonio Road
Los Altos, California 94022
U.S.A.

BULK RATE
U.S. POSTAGE
PAID
Permit No. 656
Los Altos, Calif. 94022