Performance Programs and The Measurement Interface

Bryan Carroll
MA-COM

There are times when each of us needs the capabilities of one or more of the performance programs. Programs such as OPT and SOO are used to monitor the performance of individual programs or of the entire system and are widely used in HP3000 environments around the world. These performance programs and others like them are often misunderstood, and, in some cases, information derived from the programs can be inaccurate. This paper will attempt to interpret the data from these programs starting with the MPE Measurement Interface.

The Measurement Interface is a data gathering facility of the HP3000 that has evolved from one operating system release to the next. Its immediate predecessor, MMSTAT, could collect much of the same data but did not make the data available to user programs in a usable format. The only procedure provided by MMSTAT required a dedicated tape drive and a great deal of time to produce meaningful reports. The Measurement Interface that has been released with MPE IV has made data, in the form of event counts, available to priviliged mode programs. MMSTAT had provided a trace of events, each of which could be distinguished from the preceeding and succeeding events in time. Now, with the Measurement Interface, any time dependent relationships must be determined programmatically.

In order to ensure that good data is collected from the system, some design goals must be established. These goals must include a minimum of system resource drain and easy access by a performance program. HP seems to have provided both of these goals in their design of the Measurement Interface as well as a sound base from which to expand. This base takes the form of a system extra data segment which holds the measurement data. This method of implementation could provide a stable base which would not have to change from one operating system release to the next.

The Measurement Interface control information is kept in a system extra data segment that is created and initialized by INITIAL when the system is booted. The data segment is not required to be in bank 0 but is locked and frozen to ensure that it is always in core. The locked and frozen requirement allows the system to access it much more quickly causing much less system overhead. The access time is similar to a load or store instruction since there will never be a segment fault. (A segment fault is caused by accessing an extra data segment that is not in core but out on disc in virtual memory)

The Measurement Interface makes use of some memory locations, other than those in its extra data segments. The Interrupt Control Stack (ICS) has three words reserved for the Measurement Interface, one word for flags and the other two for holding a pause time. Four words in the System Global (SysGlob) area of bank 0 are used to hold some flags and the absolute memory address (bank and offset) of the control extra data segment. The Process Control Block Extention (PCBX) for each process reserves four words to hold times and flags for that processes activity. Each of these places in memory is used to compute pause times or counts to later be stored in the appropriate Measurement Interface extra data segment.

I. Three Types of Statistics Gathered

Prepared by the Southern California Regional User's Group

There are presently three different types of information that can be enabled and disabled selectively or as a group. These three are Global, Input/Output, and Process related statistics. Each type of statistic is kept in a separate extra data segment which is also locked and frozen in place to speed access. The control extra data segment, established by INITIAL, contains pointers to all other extra data segments and is also used to hold the Global statistics when they are enabled. A counter is kept for each type of statistic to determine when to build an extra data segment and when one can be discarded. The counter is incremented each time the Measurement Interface is initiated by a program, such as OPT, and decremented each time the same type of program terminates one or more types of statistic gathering. This will prevent one program from deleting a Measurement Interface extra data segment while another program is still using it. The control extra data segment, also called the Measurement Information Table, contains pointers and work space for the shared clock Interface and space is also reserved for a future HP performance program called TRACER.

Global statistics are futher subdivided into classes, although only one class is currently defined. Each class is then subdivided into subclasses which are divided into groups.

Subclass 0 of the Global statistics is used to hold the counters for system wide CPU pauses, swaps and other dispatcher/memory manager activities. These numbers can be very helpful in determining the overall performance of your machine. These numbers can indicate possible problems such as thrashing, CPU overload, Disc bottle necks and similar problems. All other statistics are limited to specific processes or pieces of hardware.

Subclass 1 Global statistics focus on disc activity. There is one group entry for each disc configured on your system. There are counters for blocked and unblocked reads and writes as well as for memory manager reads and writes. All of the numbers collected in this subclass are in terms of physical disc I/O and no information can be derived about logical I/O's.

Lineprinter and Magnetic Tape activity consume the last two subclasses, subclass 2 and 3 respectively. There is one group in each of the subclasses for each lineprinter or tape drive that is configured into the system. Within each group there are three counters. The numbers of device reads, device writes and control operations are kept for each device for consistency. The counter for lineprinter reads is not used but exists for uniformity between devices.

The Input/Output statistics are kept as class 14 statistics. These statistics include information about all types of I/O. The groups are each 16 words long and contain copies of either the Input/Output Que (IOQ) entry or Disc Request Que (DRQ) entry for disc I/O. The device drivers add and delete entries to this table directly. This type of statistic is very dynamic and would require some analysis before it could be presented in a usable format. Due to the dynamic nature of this type of statistic, the table would have to be frozen while it was either analyzed or copied to another place for analysis. A sampling interval becomes very important during analysis of this statistic and would require some study of the machines hardware and some experimentation to choose correctly. Choosing a sampling interval too long would allow entries to appear on the list and then be removed before they were analyzed. A sampling interval chosen too short may not allow enough activity to happen on the system for proper analysis. The speed of all active I/O devices should be considered in determining a sampling interval.

Class 15 statistics are known as process statistics. This group of statistics can be viewed as a table indexed by the process identification number (PIN) of each active process. The first entry, entry 0, is an overhead entry and contains global information about the operating system release level and the time sampling began. Each entry contains 52 words of process specific information including processing times and some I/O counts. It is important to note here that the counters are initialized to zeros when the extra data segment is created, or when the entry is added to the table. If a program had accumulated some CPU time or disc I/O's prior to enabling process statistics, these times and counters will not be included in the figures presented by the Measurement Interface. It is also important to note that your program that uses the Measurement Interface may not have been the program that actually initiated the statistics gathering. The statistics gathering begins when the first program that requires the Measurement Interface requests it to start and will not terminate until ALL processes using the statistics have disabled access to the Measurement Interface. All the times and counters in the process statistics relate only to a single process and include counters, CPU time, disc I/O's, pause times for terminal reads and disc I/O.

II. Access Routines for The Measurement Interface

There are five (5) intrinsics defined by Hewlett Packard for use with the Measurement Interface. One of these routines is not yet implemented and another routine, although written and included in the MPE Kernels, is not used. The routines as they stand now are not complete enough to use without a user written retrieval routine. The five routines are: STARTSTATISTICS, STOPSTATISTICS, UPDATESTATISTICS, GETSTATISTICS and GETPROCSTATS.

The STARTSTATISTICS routine is used to enable one or more classes of statistics for gathering. The appropriate bits are set in the System Globa y area to communicate to MPE which classes of statistics are enabled for data gathering. If a class of statistic is being enabled that had not previously been enabled, the necessary system extra data segment(s) is built, frozen and locked in place. The Data Segment Table number (DST), the bank number and the offset are put in the Measurement Interface control extra data segment for future references. The enabled counter for each class of statistic being enabled is incremented. If a new data segment is created for a class of statistic, it is initialized to the appropriate values. Starting statistics on a heavily loaded system (one with many active processes) could require a significant amount of overhead. During initialization of process statistics, the initialization routine must obtain a program name, session or job number and the current state of each process. This procedure could cause many disc accesses. Continual starting and stoping of statistics gathering at the process level (class 15) should be minimized because of this overhead.

STARTSTATISTICS is an integer procedure with one parameter. The routine will return one of four possible values indicating success or failure. A value is not returned to indicate that statistics gathering has been active prior to this call to STARTSTATISTICS. You will be unable to determine from this routine whether or not statistics have just begun or if statistics were being gathered prior to your call to STARTSTATISTICS. The one parameter is a bit mask indicating which class or classes of statistics to start.

STOPSTATISTICS will disable one or more classes of statistics gathering. The class enabled counter is decremented for the requested class. If the counter is then zero, meaning there are zero remaining accessors to this class of the Measurement Interface, the corresponding system extra data segment will be deleted. The bit in the System Global (SysGlob) area corresponding to the class of statistics being disabled will be reset to reflect the termination of statistics gathering for this class. If STOPSTATISTICS is not called prior to process termination, (ie. if the program aborts or the programmer forgets to code a STOPSTATISTICS!) the system will call STOPSTATISTICS for the process. This should prevent the Measurement Interface from being active while no process has it enabled, ensuring a minimum of system overhead.

This routine, like STARTSTATISTICS, has only one parameter but it is not an integer procedure. There is not a return value so the condition code should be checked to determine the success or failure of the routine. The parameter is a bit mask that corresponds to the bit mask used for STARTSTATISTICS.

There is a System Internal Resource number (SIR) reserved for the Measurement Interface. This SIR should not be locked by any user programs. Its use is reserved for the START and STOPSTATISTICS routines only.

The UPDATESTATISTICS routine is written and exists in the KernelC segment of MPE. This routine was written to update the statistics in a given class, subclass and group but is currently not used by MPE. The statistics are currently updated only by inline code throughout the operating system. There is a secondary entry point to this routine that will bypass all parameter checking. This entry point will execute faster bacause of the elimination of paramenter checking but is still much slower than inline code. The inline code is faster but Hewlett Packard will pay the price of speed in supportability. Any changes to the Measurement Interface will require extensive changes to the MPE Kernels updating the data. This routine should never be called by a user program. If a user program was allowed to update the statistics, the data would become meaningless.

Access to class 0 statistics (Global Statistics) is accomplished through the GETSTATISTICS routine. This routine will do parameter checking and validity checking of the class and subclass. There is an entry point, FGETSTATISTICS, that will excute faster by skipping the parameter checking but, it is more likely to cause errors (because of not checking parameters.) This entry point should only be used after a program has been debugged and you are satisfied that your program is working properly. The request for statistics may be as small as one word or as large as the entire table. This is the only routine provided by Hewlett Packard to retrieve data from the Measurement Interface and will only work on class 0 statistics.

The GETPROCSTATS procedure has not yet been implemented. It is planned for this routine to access the process (class 15) and I/O (class 14) statistics in a manner similar to the GETSTATISTICS routine. A secondary entry point, FGETPROCSTATS, is planned to speed execution by eliminating parameter checking. A substitute routine would not be difficult to write and I will provide one upon request.

All five (5) routines used to access the Measurement Interface programmatically require privileged mode to execute and are fairly safe routines to use.

III. Inline Code Updates The Measurement Interface

Currently, all updating of the Measurement Interface data is done through inline code in the operating system. This code appears in many places throughout the MPE source code including KernelC, KernelD, Hardres and the device drivers. Figure 1 shows some examples of Q-MIT KernelC inline code used to update the Measurement Interface statistics.

```
PAGE 0270    KERNELC          DISPATCHER : DSP

01720 3      END;                                                          <<04485>>
01721 2
01721 2   ASMB(ZERO,DZRO);
01722 2   TOS.DISPRUNNINGFLAG:=1;
01723 2   DISPTOAWAKEMSG:=TOS; <<==>DISPATCHER RUNNING,NOT PAUSED>>
01724 2   ABSOLUTE(CPCB):=TOS;
01726 2   TRL(4):=TOS; <<SET QTIME TO 0 - DON'T WANT CLOCK INTERRUPTING>>
01727 2   ENABLE;
01730 2
01730 2
01730 2   <<
01730 2   WHO WAS RUNNING LAST?
01730 2   >>
01730 2
01730 2   ASMB(TEST);
01731 2   IF <> THEN
01732 2      BEGIN <<A PROCESS WAS RUNNING>>
01732 3      IF S0=-1 THEN
01735 3         BEGIN  <<SYSTEM JUST COMING UP>>
01735 4         ASMB(DEL);
01736 4         INITIO(2); <<INITIALIZE SYSTEM DISC>>
01740 4         STARTCLOCK(0,0D); <<GET CLOCK MOVING>>                     <<01770>>
01742 4         END
01742 3      ELSE
01745 3         BEGIN
01745 4         LASTPROCINX:=TOS-SYSBASE;
01750 4         LASTSTKSYSBASEINX:=ICS'STKDST&LSL(2)+DSTSYSBASEINX;
01754 4         IF GCLASSENABLEDMASK.CLASS0 THEN
01761 4            BEGIN  <<MEASURE PROCESS BURST EVENT AND DURATION>>
01761 5            TOS:=MEASSTATXDSBANK;
01764 5            TOS:=MEASSTATXDSBASE;
01767 5            TOS:=TOS+C0SUB0'SEGRELOFF+C'LAUNCH;                     <<RAY.V>>
01772 5            ASMB(LSEA);
01773 5            TOS:=TOS+1;
01774 5            ASMB(SSEA);  << CUM # OF LAUNCHES>>
01775 5            TOS:=TOS-C'LAUNCH+C'CPUPROCESS;                         <<RAY.V>>
01777 5            ASMB(LDEA);
02000 5            ASMB(ZERO;RCLK);
02002 5            ASMB(DADD);
02003 5            ASMB(SDEA;DDEL); <<CUM CPU TIME ON PROCESSES>>
02005 5            END;
02005 4         IF GCLASSENABLEDMASK.CLASS15 THEN                          <<01812>>
02012 4            BEGIN <<CPU TIME & NUMBER OF LAUNCHES>>                 <<01812>>
02012 5            TOS:=MEASPROCXDSBANK;                                   <<01812>>
02015 5            TOS:=MEASPROCXDSBASE;                                   <<01812>>
02020 5            TOS:=TOS+((LASTPROCINX+SYSBASE-ABS(PCBP))/PCBSIZE)*     <<01812>>
02026 5                 CLASS15'SUB0SIZE+CP'LAUNCH;                        <<01812>>
02031 5            ASMB(LSEA);                                             <<01812>>
02032 5            TOS:=TOS+1;                                             <<01812>>
02033 5            ASMB(SSEA);                                             <<01812>>
02034 5            TOS:=TOS-CP'LAUNCH+CP'CPUTIME;                          <<01812>>
02036 5            ASMB(LDEA);                                             <<01812>>
02037 5            ASMB(ZERO;RCLK);                                        <<01812>>
02041 5            ASMB(DADD;SDEA;DDEL);                                   <<01812>>
02044 5            END;                                                    <<01812>>
02044 4         END;
02044 3      END
```

Figure 1

IV. Performance Programs

The Measurement Interface is only one step in a three step procedure to obtain an accurate picture of the performance of any machine. These three steps are the collecting of data (measurement Interface), the logging of the data and the presentation of this data in an understandable format (performance program).

The logging of the data is performed internally by each performance program. Large disc files or tape files are not used to log the data collected by these performance programs. For our discussion, the logging of data will be included in the discussion of the means of presenting the data.

The performance programs available on the HP3000 are mostly contributed programs whose authors are unknown, whose source is 'unavailable', or for some other reason, are unsupported. These programs are needed and until OPT (On Line Performance Tool) was released, they were the only user runable programs available for system performance monitoring. I was able to find eight (8) separate performance programs for comparison and discussion. I have chosen this group of programs to represent both the most used performance programs and the largest variety of programs. The programs I have chosen are: SOO - four different versions, MOO - a takeoff from SOO, OPT - HP's On Line Performance Tool, Surveyor - another performance program, and Porpoise - a Boeing library program. The source code to OPT was not made available at the time of this writting and will be excluded from some of the discussions.

Each program periodically updates a process display screen, except for Porpoise which displays a single line of CPU statistics. The purpose of this part of the discussion is to compare and contrast the process displays of the various programs from a viewpoint of reliability and accuracy. Some of these programs will use the Measurement Interface and others will not. Each program will use system level routines at different levels. It is hoped that this discussion will give you the information needed to determine the best performance program or programs to meet your needs. I do not intend to critique or recommend performance programs, but to give you the information necessary to allow you to examine them for yourself.

All of the selected programs use privileged mode at some point in their processing. Surveyor runs in privileged mode for the entire span of its run while all the others will execute privileged mode instructions as they need it. A few programs try to improve their response times as well as to provide more timely data by raising their own priority. This is sometimes necessary to compete with high priority system processes or more likely high priority applilcations. MM3000, for example, runs its monitor at priority 100 or higher in the AS que. Table 1 lists the programs which use these intrinsics (GETPRIVMODE or GETPRIORITY). The programs which do not list GETPRIVMODE as an intrinsic, gain privileged mode through a user written procedure.

All programs except for one version of SOO use various MPE undocumented routines. These routines are written for the operating system to use and, for one reason or another, were not released as supported MPE intrinsics. Many of these routines require privileged mode. Documentation for some or all of these routines may be obtained from a 'helpful' HP SE or by attending some system level courses taught by HP SE's. A brief statement about each of these routines may be found in the Appendix. Consult Table 1 for those programs which use undocumented routines.

Program Name	Initial Stack	Max Running Stack	Dangerous Intrinsics	System (Undocumented) Routines
1) SOO (1)	7878	9508	GETPRIVMODE	none
2) SOO (2)	4549	8744	GETPRIVMODE	ATTACHIO
3) SOO (3)	12135	13764	GETPRIVMODE	DMOVE EXTIN' INEXT'
4) SOO (4)			GETPRIORITY	ATTACHIO CHECKLDEV GENMSG GET'DSDEVICE GET'DSXREF

				PROCFILE RESETDB SETSYSDB
5) MOO	2121	7480	GETPRIORITY GETPRIVMODE	ATTACHIO CHECKDISC CHECKLDEV GENMSG GET'DSDEVICE GET'PAGE GET'SIR LOCK'DFS'DATA'S RESETDB SCAN'PAGE SETCRITICAL STARTSTATISTICS STOPSTATISTICS SETSYSDB UNLOCK'DFS'DATA
6) OPT	3639	9660	GETPRIVMODE	ATTACHIO FGETSTATISTICS FINDDEVICES GENMSGU INEXT' PROCFILE STARTSTATISTICS STOPSTATISTICS THISCPU

Program Name	Initial Stack	Max. Running Stack	Dangerous Intrinsics	System (Undocumented) Routines
7) SURVEYOR	1068	8840	GETPRIORITY	DELAY FINDDEVICES GETSTATISTICS STARTSTATISTICS
8) PORPOISE	1131	3827	GETPRIVMODE	GETSTATISTICS STARTSTATISTICS STOPSTATISTICS

Table 1

Figures 2 through 8 show a sample of the process display of each of the performance programs. The first column appearing on most displays is cumulative CPU time. All programs except Surveyor display this statistic. This number is kept in two places in the system, but all of the programs obtain this number from the same location, the PCBX area of the processes stack. This place seems to be the most reliable one since this location is used to update the numbers found in the "Report" command upon process termination. A copy of this number is also put in the logfiles to be used for accounting or billing purposes. There is no reason to suspect any inaccuracy in the other location that supplies this number. This other location is in the Measurement Interface process statistics. The first location where cumulative CPU may be found, in the PCBX area of the stack, is a better choice for this type of display because it supplies a cumulative CPU time since the process began. The Measurement Interface can only provide the number of CPU seconds that have been used since the Measurement Interface has begun.

```
          SON OF OVERLORD              VERSION IV
FILE                            USER                  CPU                   STACK
NAME                            NAME                  TIME    %  Q  J/S#    SIZE  PIN PRI
------------------------------- --------------------- ------ -- - ------   ----- --- ---

C.I.                            TEST     .GNET             5  0 C S25       4920   14 152
C.I.                            MGR      .PAYROLL          2  0 C S26       6008   15 152
QUERY    .PUB      .SYS         MGR      .PAYROLL          2  0 C S27      16732   32 152
TEST     .PUB      .PAYROLL     MGR      .PAYROLL         17  2 C S6        8340   33 154
C.I.                            ACTUAL   .EPS              3  0 C S125      2696   53 152
C.I.                            CONSOLE  .OPERATOR         1  0 C S103      4744   58 152
XXFCS    .EARLE    .EPS         GENERAL  .EPS            252  0 C S16      25852   59 152
QUIC302  .PUB      .QUASAR      CASH     .EPS             10  0 C S138     10896   61 152
XXFCS    .EARLE    .EPS         GROUPX   .EPS            214  0 C S73      25852   63 152
COBOL    .PUB      .SYS         MGR      .DESIGN          24 19 c S140     29940   70 200
MBQ      .UTIL     .SYS         MANAGER  .SYS             26  0 D J19       5904   87 202
XXFCS    .EARLE    .EPS         GROUPM   .EPS            936  0 C S119     26620   98 152
XXFCS    .EARLE    .EPS         GROUPX   .EPS            960  0 C S45      25724  101 152
S001     .TOOLS    .TECH        BRYAN    .TECH             1  2 C S43       8796  114 1
RELATE   .PUB      .CRI         ACTUAL   .EPS             50  0 C S92      19492  119 152
TP3000   .PUB      .CCC         PSR      .PAYROLL         20  1 C S131      8916  131 152
C.I.                            AWAYNE   .LABOR            4  0 C S129      5336  137 152
MAILROOM.HPMAIL    .SYS         MAILROOM.HPOFFICE          1  0 D J41       5212  142 202
C.I.                            MGR      .PAYROLL          1  0 C S7        4752  153 152
TEST     .PUB      .PAYROLL     MGR      .PAYROLL        219  1 C S11       8340  167 152
TEST     .PUB      .PAYROLL     MGR      .PAYROLL         80  2 C S9        8340  173 152
XXFCS    .EARLE    .EPS         SALES    .EPS            401  0 C S80      25596  175 152
TP3000   .PUB      .CCC         ENTRY    .PERSONNL         1  0 C S133      7380  182 152
XXFCS    .EARLE    .EPS         MACOM    .EPS            170  0 C S100     25724  187 152

TIME USED:   4.726 CPU SEC;  17.189 ELAPSED SEC.  27.494% UTILIZATION.

CHANGES TO STATUS LIST

C.I.                            MGR      .PAYROLL     ADDED.
TP3000   .PUB      .CCC         MGR      .PAYROLL     DELETED.
```

Figure 2 S00 version 1

```
          SON OF OVERLORD              VERSION 1D
FILE                            USER                  CPU                   STACK
NAME                            NAME                  TIME    %  Q  J/S#    SIZE  PIN#
------------------------------- --------------------- ------ -- - ------   ----- ----

C.I.                            TEST     .GNET             5  0 E S25       4920    14
C.I.                            ACTUAL   .EPS              3  0 L S125      2696    53
C.I.                            CONSOLE  .OPERATOR         1  0 L S103      4744    58

TIME USED:   0.000 CPU SEC;   2.253 ELAPSED SEC.   0.000% UTILIZATION.
```

Figure 3 S00 version 2

```
 ** SON OF OVERLORD ** Version 1G **   MPE IV  C.C1.A0.   (  30 Sec. delay )
File name                    User name           CPU    CPU %   J/S#   Stack
@.@.@                        @.@                 time     .00 Q @      size  Pin
---------------------------- ------------------- ------ ----- - ------ ----- ---


CI(RUN MIS030.PUB          ) TEST     .GNET          5   .00 C S25     4920  14
CI(RUN QUERY.PUB.SYS       ) MGR      .PAYROLL       2   .00 C S26     5200  15
QUERY.PUB.SYS                MGR      .PAYROLL       2   .00 C S27    16732  32
TEST.PUB.PAYROLL             MGR      .PAYROLL      17   .00 C S6      8212  33
CI(SHOWDEV BANKMUX2        ) ACTUAL   .EPS           3   .00 C S125    2696  53
CI(RUN SPOOK.PUB.SYS       ) CONSOLE  .OPERATOR      1   .00 C S103    4744  58
XXFCS.EARLE.EPS              GENERAL  .EPS         252   .00 C S16    25852  59
XXFCS.EARLE.EPS              GROUPX   .EPS         214   .00 C S73    25852  63
COBOL.PUB.SYS                MGR      .DESIGN       43 10.27 C S140   28916  70
QUIZ104.PUB.QUASAR           AWAYNE   .LABOR         1   .08 C S129    4608  86
MBQ.UTIL.SYS                 MANAGER  .SYS          26   .00 D J19     5904  87
XXFCS.EARLE.EPS              GROUPM   .EPS         996 30.64 C S119   26620  98
XXFCS.EARLE.EPS              GROUPX   .EPS         985 37.03 C S45    25724 101
S003.TOOLS.TECH              BRYAN    .TECH          1  2.19 C S43    13052 11
RELATE.PUB.CRI               ACTUAL   .EPS          50   .00 C S92    19492 119
TP3000.PUB.CCC               PSR      .PAYROLL      21   .00 C S131    8916 131
MAILROOM.HPMAIL.SYS          MAILROOM.HPOFFICE       1   .00 D J41     5212 142
CI(RUN TEST                ) MGR      .PAYROLL       1   .00 C S7      4752 153
TEST.PUB.PAYROLL             MGR      .PAYROLL     220   .00 C S11     8212 167
TEST.PUB.PAYROLL             MGR      .PAYROLL      82  2.58 C S9      8340 173
XXFCS.EARLE.EPS              SALES    .EPS         401   .00 C S80    25596 175
TP3000.PUB.CCC               ENTRY    .PERSONNL      1   .00 C S133    7380 182
XXFCS.EARLE.EPS              MACOM    .EPS         170   .00 C S100   25724 187
<<  23 Displayed                                                     312656     >>
<<   0 Not displayed                                                      0     >>
<<  23 Total active PCBs                                             312656     >>

Time used:  11.274 CPU sec.;  13.638 Elapsed sec.  82.666% Utilization.

Changes:      0 Added      0 Deleted              (  4:18 PM)
```

Figure 4 SOO version 3

```
SON OF OVERLORD IV   VERSION BG.15/SS.V1          MON, NOV 14, 1983,  4:19
U MODE    DELAY = 60                              HIGHLITE= BRYAN    .TECH
:COMMAND OR PROGRAM          USER NAME          CPU  % Q Wc J/S#  STACK   XDS   C
:RUN MIS030.PUB              TEST     .GNET       5    C Bt S25    4920 31568     0
:RUN QUERY.PUB.SYS           MGR      .PAYROLL    2    C Bt S26    5200  3348     0
QUERY    .PUB     .SYS       MGR      .PAYROLL    2    C Bt S27   16732     0 28172
TEST     .PUB     .PAYROLL   MGR      .PAYROLL   17    C Bt S6     8212  2380     0
S004     ?TOOLS   .TECH      BRYAN    .TECH       1    C a  S43    8044  5992  3
:SHOWDEV BANKMUX2            ACTUAL   .EPS        3    C Bt S125   2696 18912  5532
:RUN SPOOK.PUB.SYS           CONSOLE  .OPERATOR   1    C Bt S103   4744  6068     0
XXFCS    .EARLE   .EPS       GENERAL  .EPS      252    C Bt S16   25852  1528     0
XXFCS    .EARLE   .EPS       GROUPX   .EPS      214    C Bt S73   25852  6220     0
MBQ      .UTIL    .SYS       MANAGER  .SYS       26    D T  J19    5904  3252 12892
XXFCS    .EARLE   .EPS       GROUPM   .EPS      996    C Bt S119  26620 15208     0
XXFCS    .EARLE   .EPS       GROUPX   .EPS     1015    C A  S45   25724  9388     0
RELATE   .PUB     .CRI       ACTUAL   .EPS       50    C Bt S92   19492  8452     0
QUIZ104  .PUB     .QUASAR    AWAYNE   .LABOR      0    C Bt S129   4304     0  3124
TP3000   .PUB     .CCC       ENTRY    .PERSONNL   0    C bs S133   5332  1948     0
TP3000   .PUB     .CCC       PSR      .PAYROLL   21    C Bt S131   8916 15972 12520
MAILROOM.HPMAIL   .SYS       MAILROOM.HPOFFICE    1    D Q  J41    5212  1864  7452
TP3000   .PUB     .CCC       MGR      .PAYROLL    0    C bs S7     5140 12180  7812
TEST     .PUB     .PAYROLL   MGR      .PAYROLL  220    C Bt S11    8212 13732 23976
TEST     .PUB     .PAYROLL   MGR      .PAYROLL   84    C Bt S9     8340  2380     0
XXFCS    .EARLE   .EPS       SALES    .EPS      401    C Bt S80   25596 10552     0
```

```
:COBOL PFLSR021.FLSSRCE,U  MGR      .DESIGN      1    C Bt S140  5296  8492      0
XXFCS   .EARLE   .EPS      MACOM    .EPS       170    C Bt S100 25724 31212      0
MIS030  .PUB     .GNET     TEST     .GNET       10    C A  S25  30796   588      0

/S00:e
```

Figure 5 S00 version 4

```
MOO  (Mistress of Overlord)       Version  ( RW ) MON, NOV 14, 1983,  4:20
FILE NAME (COMMAND)          USER NAME           CPU  % Q WC J/S# STACK   XDS  CODE
-----------------------------------------------------------------------------------

*:RUN MIS030.PUB             TEST     .GNET        5  0 C BT S25   4920 31568     0
 :RUN QUERY.PUB.SYS          MGR      .PAYROLL     2  0 C BT S26   5200  3348     0
QUERY   .PUB     .SYS        MGR      .PAYROLL     2  0 C BT S27  16732     0 28172
TEST    .PUB     .PAYROLL    MGR      .PAYROLL    17  0 C BT S6    8212  2380     0
 :SHOWDEV BANKMUX2           ACTUAL   .EPS         3  0 C BT S125  2696 18912  5532
 :RUN SPOOK.PUB.SYS          CONSOLE  .OPERATOR    1  0 C BT S103  4744  6068     0
XXFCS   .EARLE   .EPS        GENERAL  .EPS       252  0 C BT S16  25852  1528     0
XXFCS   .EARLE   .EPS        GROUPX   .EPS       214  0 C BT S73  25852  6220     0
MOO     ?TOOLS   .TECH       BRYAN    .TECH        0  1 C A  S43   8160  2904  1
MBQ     .UTIL    .SYS        MANAGER  .SYS        26  0 D T  J19   5904  3252 12892
XXFCS   .EARLE   .EPS        GROUPM   .EPS       996  0 C BT S119 26620 14960     0
XXFCS   .EARLE   .EPS        GROUPX   .EPS      1016  0 C BT S45  25724 10536     0
RELATE  .PUB     .CRI        ACTUAL   .EPS        50  0 C BT S92  19492  8452     0
QUIZ104 .PUB     .QUASAR     AWAYNE   .LABOR       1  5 C bS S129 12672  1408  5608
TP3000  .PUB     .CCC        ENTRY    .PERSONNL    1  0 C BT S133  9044  6384     0
TP3000  .PUB     .CCC        PSR      .PAYROLL    22  1 C BT S131  8916 15972 12520
MAILROOM.HPMAIL  .SYS        MAILROOM.HPOFFICE     1  0 D P  J41   5212  1864  7452
TEST    .PUB     .PAYROLL    MGR      .PAYROLL   221  1 C BT S11   8340 13732 23976
TEST    .PUB     .PAYROLL    MGR      .PAYROLL    85  1 C BT S9    8340  2380     0
XXFCS   .EARLE   .EPS        SALES    .EPS       401  0 C BT S80  25596 10552     0
 :COBOL PFLSR021.FLSSRCE,U MGR      .DESIGN      1  0 C BT S140  5296  8492     0
MIS030  .PUB     .GNET       TEST     .GNET       10  0 C A  S25  30796   588     0
-----------------------------------------------------------------------------------
TP3000  .PUB     .CCC        MGR      .PAYROLL    DELETED.
```

Figure 6 MOO

```
USER SUMMARY REPORT
                                                         WORKING SET INFO
PIN USER.ACCT          PROGRAM NAME (command)       CPU  % PRI CSTSZ STKSZ DSTSZ
-------------------------------------------------------------------------------------

 32 MGR.PAYROLL        user program file           2237  0 152  7360 16732     0
 33 MGR.PAYROLL        user program file          17415  0 152     0  8212  2132
 59 GENERAL.EPS        user program file         252537  0 152     0 25852     0
 87 MANAGER.SYS        user program file          26308  0 202 12176  5904   768
101 GROUPX.EPS         user program file         1032 S 27 152     0 25724 10536
119 ACTUAL.EPS         user program file          50687  0 152     0 19492  2648
125 AWAYNE.LABOR       user program file           7036  0 153  5608  4656  6664
126 ENTRY.PERSONNL     user program file           1735  0 152     0  9044  6328
131 PSR.PAYROLL        user program file          22876  1 152 12520  8916 15844
138 BRYAN.TECH         OPT.TOOLS.TECH               637  1 152  1832 11452     0
142 MAILROOM.HPOFFICE  user program file           1231  0 202     0  5212     0
167 MGR.PAYROLL \      user program file         222861  2 152 23976  8340 13732
173 MGR.PAYROLL        user program file          86510  2 152     0  8212  2132
175 SALES.EPS          user program file         401055  0 152     0 25596  1176
194 TEST.GNET          user program file          10986  0 200     0 30796   588
```

```
CONTINUE EXECUTION?  (YES/NO)  no
```

Figure 7 OPT

```
 4:24 PM  ELAPSED 00:01:25

                 D I S C     W A I T S     Memory       Garbage        Garbage
  Idle    Busy   MAM  User & MAM   User   allocation   collection    allocation
   88%     10%     0%       0%       1%       0%            0%            0

DRIVE-   1   2   3   4
R/sec-   0   0   0   0
W/sec-   0   0   0   0
```

Program	name		J/S#	Q & PIN	%Tot Cpu	......WAIT STATES...... Abs	Dsc	Bio	Trm	Imp	Pre	Disc IO	Swp	Ovr
QUERY	.PUB	.SYS	S27	C32	0%	0%	0%	0%	99*	0%	0%	0	0	0
XXFCS	.EARLE	.EPS	S16	C59	0%	0%	0%	0%	99*	0%	0%	0	0	0
MBQ	.UTIL	.SYS	J19	D87	0%	0%	0%	0%	0%	0%	0%	0	0	0
SURVEYOR	.TOOLS	.TECH	S43	L90	21%	0%	0%	0%	99*	0%	0%	0	0	0
QEDIT	.PUB	.ROBELLE	S25	C91	0%	0%	0%	0%	99*	0%	0%	0	0	0
XXFCS	.EARLE	.EPS	S45	C101	76%	0%	0%	0%	0%	0%	0%	2	0	0
RELATE	.PUB	.CRI	S92	C119	0%	0%	0%	0%	99*	0%	0%	0	0	0
QUIZ104	.PUB	.QUASAR	S129	C125	0%	0%	0%	0%	99*	0%	0%	0	0	0
TP3000	.PUB	.CCC	S133	C126	0%	0%	0%	0%	99*	0%	0%	0	0	0
TP3000	.PUB	.CCC	S131	C131	0%	0%	0%	0%	99*	0%	0%	0	0	0
MAILROOM	.HPMAIL	.SYS	J41	D142	0%	0%	0%	0%	0%	0%	0%	0	0	0
TEST	.PUB	.PAYROLL	S9	C173	0%	0%	0%	0%	99*	0%	0%	0	0	0
XXFCS	.EARLE	.EPS	S80	C175	0%	0%	0%	0%	99*	0%	0%	0	0	0
115														

Figure 8 Surveyor

%Idle	%MMI	%DscI	%Both	%PCPU	%Mam	%Grbg	%Ovhd
93.5	0.0	3.5	0.0	2.7$	0.0	0.0	0.3
97.4	0.0	1.1	0.0	1.3	0.0	0.0	0.2
95.4	0.0	2.1	0.0	2.2	0.0	0.0	0.3
94.9	0.0	1.9	0.0	2.8	0.0	0.0	0.4
61.7	0.0	27.7	0.7	8.1	0.2	0.0	1.8
76.7	0.2	16.8	0.1	4.9	0.1	0.0	1.3
58.8	0.0	22.6	0.2	16.1	0.1	0.0	2.3
15.0	0.0	20.7	2.8	54.0	0.2	0.0	7.5
0.0	1.0	30.2	7.3	53.4	0.5	0.0	8.1
2.2	0.3	31.8	2.2	58.0	0.1	0.0	5.5
1.5	1.5	41.7	2.9	47.6	0.1	0.0	4.8
0.7	0.9	55.7	0.2	38.6	0.0	0.0	3.9
20.7	0.2	51.0	3.9	20.5	0.1	0.0	3.7
40.9	0.0	31.3	0.2	23.9	0.0	0.0	3.7
0.0	0.0	62.2	2.7	28.2	0.1	0.0	6.9
0.2	0.1	65.0	2.6	26.5	0.2	0.0	5.6
2.3	0.0	51.7	1.4	35.4	0.1	0.0	9.2

```
        26.5   0.2  43.6   1.5  23.9   0.2   0.0   4.3
        36.2   0.2  31.1   1.5  26.5   0.1   0.0   4.2 Totals

    [PORPOISE]: e
```

Figure 9 Porpoise

The next item on most of the performance programs is the percent of the CPU a process used during the last interval. An interval is defined in each of the performance programs as either the time between characters input from the keyboard or the configurable time in seconds known as the delay time, whichever is shorter. The delay time is used as a terminal read timeout and will allow the program to wait only the specified amount of time before automatically issuing a carriage return to the pending read. SURVEYOR computes this number by holding the cumulative CPU milliseconds used on the last Measurement and subtracts that number from the cumulative number of CPU milliseconds up to the most recent reading. Both of these numbers come from the Measurement Interface. The difference of these numbers is then divided by the total amount of CPU milliseconds expended on all processes during the interval. This difference is then multiplied by 100 to obtain the percent of total time allocated to processes taken by this process. The other performance programs differ in their division. The others divide the difference of cumulative CPU times by the elapsed clock time during the interval. This procedure gives a substantially different number which is the total amount of time during the interval that the process was being serviced by the CPU. Another difference between SURVEYOR and the others is that the other programs obtain their cumulative CPU times from the PCBX area of each processes stack. The sum of the percent of CPU column in SURVEYOR should always be at or very near 100% while the same sum from the other programs would never be 100% and most times not close.

```
    SURVEYOR:
 %CPU = ((Cumulative CPU now - Cumulative CPU last interval) /
          Total CPU spent on processes) * 100.0)
    Other performance programs:
 %CPU = ((Cumulative CPU now - Cumulative CPU last interval) /
          Total elapsed time since last interval) * 100.0)
```

The que and priority columns should be discussed together. A general understanding of the MPE que structure is required to understand what is presented in the QUE and Priority columns in the performance programs. The MPE scheduling que allows the System Manager to assign a minimum and maximum priority to the three circular ques, C, D and E. The two linear ques have a fixed priority range of 1 to 100 for A and 100 to 150 for B. (see Figure 10) A low priority number indicates a high priority.

The highest priority process requesting CPU will be serviced the next time the dispatcher reviews the scheduling que. Before a process may request the CPU, all of the resources required by that process must be available. This means that any data or code segments needed must be in core. Each time a process uses its slice of CPU time, its priority is decremented until it is at the bottom of its que or until a terminal read is issued. When a terminal read is issued, the process jumps to the top of its que.

A process may be pushed out of its que to a higher priority when a higher priority process is impeded on a resource that your process has locked, most likely a SIR. Assume you are running in the CS que say at priority 160 and have the system directory SIR locked. If you are being serviced by the CPU and a system process running at say priority 50, wants to lock the system directory SIR, the system will see this conflict and push your priority up to priority 50 until you unlock the system directory SIR. A process with privileged mode capability may also call the GETPRIORITY intrinsic and move itself or one of its son processes to a priority out of its que. These are the only two cases that I know of where a process may be running out of its assigned que. This may explain some situations where

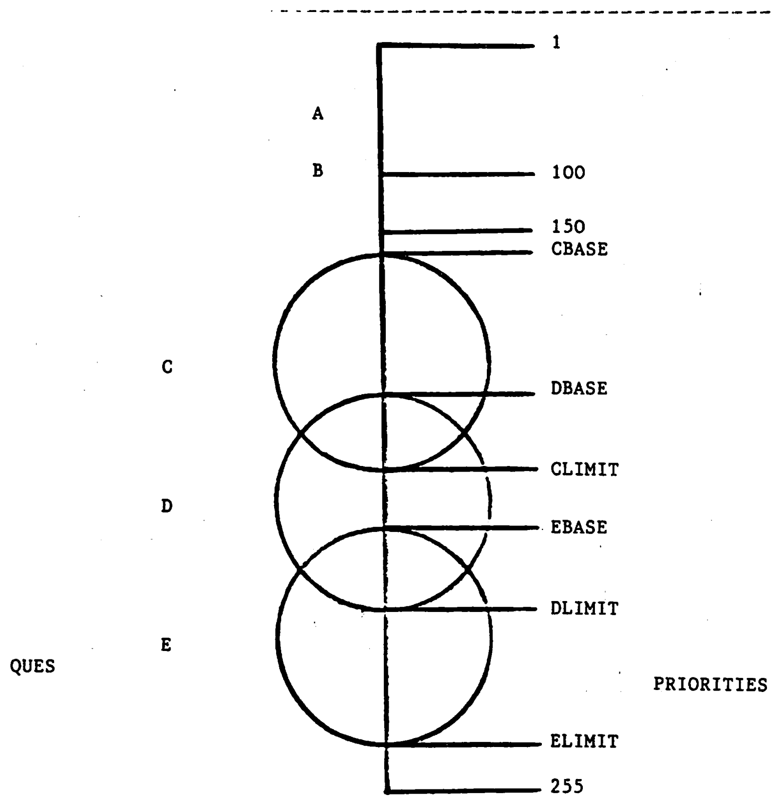

Figure 10

processes in the CS que are running at priority 50.

The que data is kept in the PCB and also in the Measurement Interface. The priority is only kept in the PCB. Surveyor gets its que information from the Measurement Interface while all others obtain their information from the PCB. One version of SOO, version 1, could misrepresent itself with its que data. This author apparently realized the potential of a process running out of its assigned que and decided to assign a que based on prioirty. In doing so he fixed the que limits by hard coding them into his program. The limits he decided on are as follows:

Priority	Que
< 150	B (No A que)
151 - 176	C
177 - 201	Sub C
202 - 220	D
221 - 239	Sub D
240 - 249	E
> 250	Sub E

These assignments will most likely be different from the limits set on your machine and consequently the readings from this program will be inaccurate.

The stack size of each process is only found by using the PCB and the DST tables. First the PCB is searched for the stack DST number (word 3 bits 1:10). The DST table is then searched for this entry. The first word of the DST entry contains the size of the data segment divided by four (4) to save space. The value for stack size, or any segment size, should always be a multiple of four (4).

The current code segment size and the extra data segment size found in SOO and MOO are related. The Segment Locallity List (SLL) is a list of the resources required to be available (in core) for the process to request the CPU. One version of SOO and MOO check the SLL for each process to determine the size of all code segments or extra data segments listed in the SLL. The SLL is a very dynamic table changing every time a process switches code segments, builds a new extra data segment or opens a file requiring a new extra data segment. The data collected from the programs SOO, and MOO from the SLL can be very valuable but it must be understood that these figures will not necessarily include all code segments or extra data segments used by a program, only those required for the process to run.

The program name/command name column found in a version of SOO and MOO requires some knowledge of the command interperter stack. The program name may be obtained from a system routine named PROCFILE. The currently active PIN number is required as input to the routine. The most recently executed command in a command interperter process is a little more difficult to obtain. This command name is stored at DB+1 in the command interpreter stack and may be changed by future releases of the operating system. It is a helpful bit of information for those programs that wish to find it.

Disc I/O information is only provided at the process level by Surveyor and can be very valuable. This information is collected from the Measurement Interface and is persented in terms of disc I/O's per elapsed second.

V. Subjective Comments About Performance Programs

SOO - Version 1

Version One of SOO is a well written program. Block comments are used prior to most routines and do a satisfactory job of explaining the objectives of the routines. The program is very readable and could probably be supported fairly well by someone familiar with SPL. The program is clean in its design and runs fairly efficiently except for the Que column which was described previously. The program tries to derive the Que by obtaining the current absolute priority and trying to fit it to a previously defined Que structure. This could result in substantially inaccurate results in the Que column. The dangerous code is isolated and for the most part is used only when absolutly needed. A good DST retrieval routine is needed to make any of the performance programs work well, and this program has one of the better ones. With the understanding of what the program is trying to accomplish, I feel I could rely on the output of this program with the exception of the Que column.

SOO - Version 2

Version Two of SOO seems to be a predecessor to both Version One and three. This program has many of the same block comments in many of the same named routines but seems to have been written for pre-MPE IV systems. The program only allows for a maximum of 128 process identification numbers (PIN's). The DST retrieval routine is also much longer and more complicated. The routine makes an attempt to go out to virtual memory on its own to retrive data from an absent data segment. I am not sure if this method is still required or not. I have crashed the system several times with a System Failure #16 (DST Violation/Interval Interrupt) trying to access an extra data segment in virtual memory without using disc I/O calls, but there are versions of SOO that seem to work without these calls. I suspect there is an error on my part somewhere, but I havn't found it yet. This program is as reliable as Version One except for the handling of the Que column. This version takes the que data straight from the PCB entry and does not try to make it fit a predefined Que structure. For pre-MPE IV machines this version would be the best and maybe the only choice.

SOO - Version 3

Version Three of SOO seems to be an enhanced Version One. It contains many of the same routine names and comments but goes a little farther than Version One. This routine is better documented and therefore would be easier and quicker to support. There are some warnings in the compiled version that I obtained but they don't seem to affect the performance of the program. The program replaces the DST retrieval routine used in Version One with a call to another procedure which then calls a system level (undocumented) routine named DMOVE to transfer data from extra data segments to the users stack. I assume that this routine would be more reliable being a system routine but I have no prior experience with the routine. The way in which the programmer implemented the new routine is a little costly in terms of procedure calls but may have been left that way for history. This version seems to be very reliable, and it seems to be remarkably similar to the version of SOO included on the Robelle contributed library.

SOO - Version 4

Version Four of SOO represents a substantially different version of SOO. The program seems to be an entirely new program, and not a clone from some original SOO like the previous three versions seem to be. The program does not jump into a process display but prompts with 'SOO:'. The commands to get the process display are not apparent and it will be initially frustrating for the user expecting the familiar SOO process display. This version provides many more options than are available on other versions of SOO and is aimed at the more involved and sophisticated user. The code is also more complex and harder to read and support. Many higher level constructs, such as using split stack mode to switch to the system globals area, are used making it all but unsupportable by a "mere mortal" programmer. The comments are also rather sparse, but it is a very efficient program. The DST retrieval routine is taken directly from Version One of SOO and seems to be reliable. The program will allow you to do almost anything including running programs like FREE2.PUB.SYS, changing your own priority, and initiating the SEGMENTER? The program has a help (?) facility and each option appears to be safe enough for the average user to try. The program is more difficult to use but allows more flexibility and provides about the same reliability.

MOO

The program MOO claims to be a modified version of SOO and has some similarities. This version is more an extention of SOO than a modified version. The Measurement Interface was incorporated for some I/O statistics and routines were added to allow the running of other programs like Spook. New routines have been added to do things like a FREE2 display or LISTEQ2. The program is fairly well commented, but like Version Four of SOO, the constructs are more difficult to follow. The code is very sharp and efficient, and in my experience it has been very reliable. The program would be difficult to support but it can be done by most mortals. The program only stays inside privileged mode long enough to get what it needs which makes it fairly safe.

Surveyor

Surveyor could have been written as a demonstration program for the Measurement Interface. It follows the constructs of the Measurement Interface very closely. The program is initialized in privileged mode and remains there during its entire execution. This has the potential to cause problems although I have never seen it happen. The program has two displays, either a tables display, similar to Tuner4, or a process display that is very different from SOO, OPT or MOO. The process display is the only display that I have seen which displays both CPU statistics and I/O statistics on the same screen. The program presents the data necessary to determine resource hogs in a fairly concise and readable format. The program uses a large section of the stack in the DL to DB area to swallow all of the process statistics in one bite while keeping its stack size reasonable (under 10K). The code is a little short on comments but is fairly well structured, and seems to be efficient enough. A modification to the program using only privileged mode when it was required would make me feel more comfortable, but it has not caused us any problems....yet. Since all of the process related data comes directly from the Measurement Interface, I feel I can rely on the data from the program as much as I can rely on the Measurement Interface. I have no reason to doubt the data coming from the Measurement Interface.

Porpoise

Porpoise is another program that could have been written to demonstrate the Global Statistics gathering of the Measurement Interface. The program has a fancy driver but really has only one display and that is global CPU related statistics. The program is excellent for gathering CPU activity to be later consolidated and perhaps graphed. All of the data comes directly from the Global Statistics (Class 0) and is as reliable as the Measurement Interface. The code is clean and short enough to be easily read. Many of the routines used to make the driver fancy are part of the Boeing account library allowing the code to be shorter. The code is very well documented and should be easy to support.

OPT

OPT is Hewlett Packard's On Line Performance Tool written to supply a long needed supported version of a performance monitoring program. OPT is slowly gaining acceptance with the user community as enhancements are made to make it more usable. The Q-MIT version of OPT, version 10, has about equaled what has been available with SOO, MOO and Surveyor all along. Futher comments, analogous to the ones made of the other performance programs, are unavailable at this time because the source has been unavailable.

Summary

All of the programs do a good job at what they were designed to do. I don't think the value of any one of them can be raised above the others without describing specific circumstances. All of the programs with the exception of Porpoise, have no distinguishing marks that could indentify either a company or an individual as the author. Our company has made use of Surveyor, OPT, and a supply of our own home grown utilities to monitor our systems. I trust I have related enough information for the reader to determine which programs are best suited for each situation in his or her shop.

Appendix

*** ATTACHIO ***

The ATTACHIO routine is the primary I/O routine used by the I/O system.

```
DOUBLE PROCEDURE ATTACHIO(I'LDEV, I'QMISC, I'DSTX, I'ADDR,
                          I'FUNC, I'COUNT, I'P1, I'P2, I'FLAG);
     VALUE I'LDEV, I'QMISC, I'DSTX, I'ADDR,
           I'FUNC, I'COUNT, I'P1, I'P2, I'FLAG;
     INTEGER I'LDEV, I'QMISC, I'DSTX, I'ADDR,
             I'FUNC, I'COUNT, I'P1, I'P2, I'FLAG;
     OPTION PRIVILEGED, UNCALLABLE, EXTERNAL;
```

*** CHECKLDEV ***

The CHECKLDEV routine does checking on the type of device of a given logical device. (ie. DS Device.)

```
PROCEDURE CHECKLDEV (DEV);
      VALUE DEV;
      INTEGER DEV;
      OPTION EXTERNAL;
```

*** CHECKDISC ***

The CHECKLDEV routine will return global information about the requested disc.

```
PROCEDURE CHECKDISC ( LDEV, INFO );
      VALUE LDEV;
      INTEGER LDEV, INFO;
      OPTION EXTERNAL;
```

Appendix

*** DELAY ***

The DELAY routine is identical to the PAUSE intrinsic except the parameter is a double word millisecond.

```
PROCEDURE DELAY(TIME);
      VALUE TIME;
      DOUBLE TIME;
      OPTION EXTERNAL;
```

*** DISKSPACE ***

The DISKSPACE routine will return disc free space information about the requested disc.

```
INTEGER PROCEDURE DISKSPACE(LDEV,NSECT,PDISKADR);
      VALUE LDEV,NSECT;
      INTEGER LDEV;
      DOUBLE NSECT,PDISKADR;
      OPTION EXTERNAL;
```

*** DMOVE ***

The DMOVE routine will move data from a users stack to a system extra data segment or from a system extra data segment to the users stack.

```
LOGICAL PROCEDURE DMOVE(DSTTABNUM,TABSTARTPTR,NUMWRDS,USERTABPTR,
      TOFROM,PARMNEQ8);
      VALUE DSTTABNUM,NUMWRDS,PARMNEQ8,TABSTARTPTR,TOFROM,USERTABPTR;
```

```
        LOGICAL DSTTABNUM,TOFROM;
        INTEGER NUMWRDS,PARMNEQ8,TABSTARTPTR,USERTABPTR;
        OPTION EXTERNAL,UNCALLABLE,PRIVILEGED;
```

Appendix

*** FINDDEVICES ***

The FINDDEVICES routine will return all the logical device numbers of a particular type. (ie. Type = 1 will return all logical devices that are discs.)

```
PROCEDURE FINDDEVICES(TYPE,TARGET'ARRAY);
        VALUE TYPE;
        INTEGER TYPE;
        INTEGER ARRAY TARGET'ARRAY;
        OPTION EXTERNAL;
```

*** GENMSG ***

The GENMSG routine will format and display an error message on $STDLIST. Parameter substitution is allow in the message.

```
INTEGER PROCEDURE GENMSG(MSET,MNUM,MASK,P1,P2,P3,P4,P5,
                         DEST,REPLY,OFFSET,DST,CNTRL);
        VALUE   MSET,MNUM,MASK,P1,P2,P3,P4,P5,DEST,REPLY,OFFSET,DST,CNTRL;
        LOGICAL MSET,MNUM,MASK,P1,P2,P3,P4,P5,DEST,REPLY,OFFSET,DST,CNTRL;
        OPTION VARIABLE,EXTERNAL;
```

*** GETSIR ***

The GETSIR routine will lock the requested System Integrity Resource number. The routine will wait for the SIR to become free if it is locked.

```
LOGICAL PROCEDURE GETSIR(SIR);
        VALUE SIR;
        LOGICAL SIR;
        OPTION EXTERNAL;
```

Appendix

*** GETSTATISTICS ***

The GETSTATISTICS routine will return Global statistics information from the Measurement Interface.

```
INTEGER PROCEDURE GETSTATISTICS(CLASS,SUBCLASS,STARTINGITEM,
                                WORDCOUNT,TARGET'ARRAY);
        VALUE CLASS, SUBCLASS,STARTINGITEM,WORDCOUNT;
        INTEGER CLASS, SUBCLASS,STARTINGITEM,WORDCOUNT;
        INTEGER ARRAY TARGET'ARRAY;
        OPTION EXTERNAL;
```

*** GET'PAGE ***

The GET'PAGE routine is a disc paging routine.

```
INTEGER PROCEDURE GET'PAGE(PAGE);
      VALUE PAGE;
      INTEGER PAGE;
      OPTION EXTERNAL;
```

*** GET'DSDEVICE ***

The GET'DSDEVICE routine will return DS information on a given logical device.

```
INTEGER PROCEDURE GET'DSDEVICE (LDEV);
      VALUE LDEV;
      INTEGER LDEV;
      OPTION EXTERNAL;
```

*** GET'DSXREF ***

The GET'DSXREF routine will return a DS device cross reference for a given active process.

```
INTEGER PROCEDURE GET'DSXREF (PIN);
      VALUE PIN;
      INTEGER PIN;
      OPTION EXTERNAL;
```

Appendix

*** LOCK'DFS'DATA'SEGMENT ***

The LOCK'DFS'DATA'SEGMENT routine will lock the Disc Free Space table.

```
INTEGER PROCEDURE LOCK'DFS'DATA'SEG ( LDEV );
      VALUE LDEV;
      INTEGER LDEV;
      OPTION EXTERNAL;
```

*** PROCFILE ***

The PROCFILE routine will return the fully qualified filename of the executing process.

```
PROCEDURE PROCFILE (PIN, NAME);
      VALUE PIN;
      INTEGER PIN;
      BYTE ARRAY NAME;
      OPTION EXTERNAL;
```

*** RELSIR ***

The RELSIR routine will release a previously locked System Intregrity Number.

```
PROCEDURE RELSIR(SIR,FLAG);
      VALUE SIR,FLAG;
      LOGICAL SIR,FLAG;
      OPTION EXTERNAL;
```

*** RESETCRITICAL ***

The RESETCRITICAL routine will enable the process to abort without causing a system failure should an abnormal situation occur.

```
PROCEDURE RESETCRITICAL(C);
      VALUE C;
      LOGICAL C;
      OPTION EXTERNAL;
```


*** RESETDB ***

The RESETDB routine will reset the DB pointer for your process back to your processes stack.

```
PROCEDURE RESETDB (DBX);
      VALUE DBX;
      INTEGER DBX;
      OPTION EXTERNAL;
```

*** SCAN'PAGE ***

The SCAN'PAGE routine will scan a disc page obtained with GET'PAGE.

```
INTEGER PROCEDURE SCAN'PAGE;
      OPTION EXTERNAL;
```

*** SETCRITICAL ***

The SETCRITICAL routine will cause the system to fail if the process meets an unexpected situation which would normally cause the process to abort. This routine is used to ensure all items in a list are completed.

```
LOGICAL PROCEDURE SETCRITICAL;
      OPTION EXTERNAL;
```

*** SETSYSDB ***

The SETSYSDB routine will set the DB pointer for your stack to point to the System Globals area.

```
INTEGER PROCEDURE SETSYSDB;
      OPTION EXTERNAL;
```

Appendix

*** STARTSTATISTICS ***

The STARTSTATISTICS routine will enable any one or all of the statistics gathering types of the Measurement Interface.

```
INTEGER PROCEDURE STARTSTATISTICS(CLASSMASK);
      VALUE CLASSMASK;
      LOGICAL CLASSMASK;
      OPTION EXTERNAL;
```

*** STOPSTATISTICS ***

The STOPSTATISTICS routine will disable any or all of the statistics gathering types of the Measurement Interface.

```
PROCEDURE STOPSTATISTICS(A);
      VALUE A;
      LOGICAL A;
      OPTION EXTERNAL;
```

*** UNLOCK'DSF'DATA'SEG ***

The UNLOCK'DFS'DATA'SEG routine will unlock the Disc Free Space table locked with LOCK'DFS'DATA'SEG.

```
PROCEDURE UNLOCK'DFS'DATA'SEG;
      OPTION EXTERNAL;
```

Bryan Carroll is currently a Hewlett Packard Support Specialist at MA-COM in Burlington, Mass. He graduated from Abilene Christian University with a B.B.A. in Business Computer Science in 1982 and received the Certificated Data Processing (CDP) certificate in October of 1982. Mr. Carroll is also a member of the Association for Computing Machinery and was a member of the Abilene Christian University Programming Team in 1982 winning first place honors at the regional level. His experience includes four years with the HP3000, two of which were at the systems level.
