Monitoring and Analysis [A]

This appendix describes how to analyze your environment when a guest is running or has failed.

A.1 Status

To get the status of your guest system and the tty number of your console on the IOS, use the following command:

```
guest -v
```

```
gale jqp &lt;27> guest -v
gst-2 guest: Info
   guest Version 9.3.0ed, generated on 09/15/97, at 15:57:03.


              System  CPU       Address Size
Console Ord     Name  (%)       (words) (MW)  Status       Owner  Flags

 zip 0   0     gale   57             0   16  executing     HOST  M
 zip 1   1   galeg1   43      18300928   14  executing      jqp  S

Flag values:
        M (multi-user), S (single-user), F (frozen), H (halt on panic)
        B (breakpointed)

Number of CPUs available to guests:  2
Additional guest tracing is:         ENABLED
```

A.2 CPU monitoring considerations

The `sar`(1) CPU usage output (`-u` and `-p` options) reports the amount of CPU time accumulated in concurrently running systems under a new column titled `guest`. When run in the host, `sar`(8) reports the percentage of CPU time spent in the guest under the `guest` column. When run in the guest, the `guest` column refers to host CPU time.

Across a particular time interval, the values reported in the `guest` column by the host and guest `sar`(8) command should total approximately 100%. The

actual value may be a percentage point higher or lower due to rounding. Also, within a particular system, the sum of all `sar`(8) CPU usage data columns should be within a percentage point of 100.

CPUs assigned to a guest occasionally require some service provided by the host. To invoke such service, the guest CPU uses a mechanism analogous to a user process issuing a system call. The time that a CPU spends in the host kernel in servicing a request from the guest is accumulated as system time within the guest kernel. This time is recorded as `guest` time within the host kernel. The effects of this are as follows:

- System time reported by `sar` in a guest (*%sys column*) will generally be somewhat higher than in a host or stand-alone system running a similar workload. The amount of the increase depends upon the user process mix, with I/O-intensive loads showing the greatest increase.
- System time reported by `timex`(1) generally will be higher in a guest. The size of the increase depends upon the number of host requests the guest kernel must issue during the handling of each user system call issued during the timed interval.
- Wall-clock time (reported as *real* by `timex`) for a particular process may be greater in a guest than in a host or stand-alone system. In general, the amount of increase is proportional to the amount of I/O performed by the process.
- User time values are independent of the kernel's mode of operation (host versus guest). Not only are user CPU times reported by `sar` independent of the kernel mode but user CPU time associated with a particular process also is independent. For example, `timex` reports the same user time for a process when run in a guest as it would for the same process run in the host. User time may vary due to bank conflicts.
- It is possible to monitor the host request activity (host calls issued by an active guest kernel) using a new `sar` option, `-g`. For an explanation of the sample information that follows, refer to the `sar`(1) man page.

```
sn1703% sar -g 10 1000

sn1703a 7.0.0 hst.305 CRAY Y-MP    04/20/94

17:25:59 host call         %time   calls/s   avetime     maxtime   mintime
17:26:10 I/O interrupt         3.4       2.5     115.8     28960.9      26.6
         low speed I/O         2.5       2.9      72.9     16193.2      31.3
         clk interrupt         2.2       1.0     185.3      1466.0      63.5
            return CPU          na       0.4        na          na        na
               SSD I/O         0.1       0.1      52.5      5359.2      31.3
         user exchange        86.6     402.0      18.5    316646.0      12.4
           get cluster         2.5       5.7      38.1      3906.7      31.8
           put cluster         2.7       5.7      41.1      3909.7      32.0

17:26:21 I/O interrupt         7.2       2.7      80.4     28960.9      26.6
         low speed I/O         3.6       1.5      74.2     16193.2      31.3
            return CPU          na       5.5        na          na        na
               SSD I/O         3.3       1.9      52.1      5359.2      31.3
         user exchange        68.5     112.5      18.5    316646.0      12.4
           get cluster         9.1       7.3      37.9      3906.7      31.8
           put cluster         8.4       6.7      37.9      3909.7      32.0
sn1703a sn1703 7.0.0 hst.305 CRAY Y-MP    04/20/94
```

A.3 Dumping

To dump your guest system, use the following command:

```
guest -d
```

It is **not necessary** to stop the guest before dumping it.

At any time during the running of your guest, you may dump the system by using the `guest -d` command. If the guest system is not panicked, the host kernel will set the internal freeze (inhibit) flag for the guest system while the dump is in progress. This prevents CPUs from exchanging to user processes in the guest, thus, preserving the integrity of the dump. The freeze flag is cleared when the dump completes successfully.

As noted in the following dump output example, the `guest`(1) command informs the user of how the dump is proceeding and where the dump and binary files will be placed. A guest dump directory contains `crash`(8) and UNICOS binaries for all systems occupying mainframe memory. A *_guest* name extension is added to the guest-related binaries in the dump directory. Guest

system dumps can be much larger than host dumps (up to twice the size), so you may need to increase your dump partition size for mainframe dumps taken from the OWS or SWS. You should set `DUMP_DIRECTORY=`*path* in your `guest.rc` file, where *path* is a file system with enough unused space to handle your particular requirements.

In the dump directory, you start the `crash`(8) command as follows:

```
./crash dump unicos
```

All `crash`(8) commands entered at this point are relative and pertain only to the host's system memory. You may view the guest portion of the dump by entering the `guest`(1) command at the `crash>` prompt. If `crash` finds the appropriate binaries (denoted by guest system name) in the current directory it will ask you if you want to fork a new crash to examine the dump. Unless the system levels are virtually identical, an affirmative response is usually appropriate.

After forking the new crash or allowing the current crash to change its memory bias, all `crash`(8) commands will be relative to the guest's system memory. All `crash` commands (except for some UNICOS under UNICOS specific commands) are available when analyzing a guest dump. For more information, see the `crash`(8) man page.

An example of the `-d` option of the `guest`(1) command follows:

```
gale jqp <34> guest -d
Enter a dump comment (up to 80 characters):
tpconfig process hung in guest
gst-23 guest: Info
   Guest dump files will be located below the following directory:
        ./dumps/04050534

Dump segment address:    030412000   cumulative words:    012110000
gst-25 guest: Info
   Successfully completed a guest dump of the system named: gale

Dump segment address:    077610700   cumulative words:    022703000
gst-25 guest: Info

   Successfully completed a guest dump of the system named: galeg1

gale jqp <35> cd ./dumps/04050534
/usr/guest/jqp/dumps/04050534

gale 04050534 <36> ls
crash          crash_galeg1   dump           unicos         unicos_galeg1

gale 04050534 <37> ./crash dump unicos
> guest
Use ./crash_galeg1 (y|n)? > y
Running ./crash_galeg1 -s -g galeg1 dump ./unicos_galeg1
> ut
+          0000000000000000010 0000000000000000000 0000000000000000000
host2->    0030701403117621106355 0000000000000000001 0000000000000000000
+          0000000000000000003 0000000000000000000 0000000000000000000
host1->    0030701403117621106160 0000000000000000000 0000000000000000010
host->     0030701403117621106025 0000000000500000000 0000000000000000000
+          0000000000000000010 0000000000000000000 0000000000000000000
->host2    0030701403117602533114 0000000000000000001 0000000000000000000
+          0000000000000000003 0000000000000000000 0000000000000000000
->host1    0030701403117602532722 0000000000000000000 0000000000000000010
->host     0030701403117602532565 0000000000500000000 0000701403117621511220
clockown   0030701403117602531633 0000000000000000004 0000000000000000000
+          0000000000000000017 0000000000000000010 0000000000000000000
routcp5    0030701403117602530357 0000000000000000010 0000000000000000003
+          0000000000000000003 0000000000000000010 0000000000000000000
```

```
routcp4    00307014031176025300070000000000000000000000000000000010
+          00000000000000000010000000000000000000040000000000001040
host2->    00307014031176025267170000000000000000000000000000000000
+          00000000000000000003000000000000000000100000000000000010
host1->    00307014031176025265430000000000000000000000000000000010
host->     00307014031176025264100000000020000000000000000000000000
+          00000000000000000010000000000000000000040000000000000000
->host2    00307014031176021412110000000000000000000000000000000000
+          00000000000000000003000000000000000000100000000000000010
->host1    00307014031176021410140000000000000000000000000000000010
->host     00307014031176021406650000000002000000010400000000000000
swtchc     00307014031176021371240000226203000075710000645443306240000000000
swtchg1    00307014031176021341760000000000000000000100000000000000001
swtchd     00307014031176021305600000226203000075710000645443306240000000000
+          00000000000000000017000000000000000000100000000000000000000
routcp5    00307014031176021275570000000000000000000100000000000000003
+          00000000000000000003000000000000000000100000000000000000000
routcp4    00307014031176021272030000000000000000000000000000000010


> p
SLT ST  PID  PPID  PGRP   UID  EUID  PRI CPU   EVENT  NAME     FLAGS
  0 s     0     0     0     0     0    0   - 00223731 sched    Ld Sys
  1 s     1     0     1     0     0   39   - 00205552 init     Ld rwt
  2 r     2     0     0     0     0  999   0     -    idle     Ld Sys Idl conn
  3 r     3     0     0     0     0  999   1     -    idle     Ld Sys Idl conn
  4 r     4     0     0     0     0  999   2     -    idle     Ld Sys Idl conn
  5 r     5     0     0     0     0  999   3     -    idle     Ld Sys Idl conn
  6 s  1127     1  1127   526   526   39   - 00205552 csh      Ld Nocore rwt
  6                                                           oldmask
  7 s  1129     1  1129     0     0   28   - 01013567 getty    Ld
  8 s  7260  7252  7260     0     0   39   - 00205552 csh      Ld rwt oldmask
  9 s   380     1   380     0     0   26   - 02322001 slogdemo Ld Plock
 10 s  1128     1  1128     0     0   28   - 01013521 getty    Ld
 11 s   621     1   621     0     0   26   - 00730374 cron     Ld
 12 s   608     1   301     0     0   39   - 00205552 shrdaemo Ld rwt oldmask
 13 r   634     1   634     0     0   39   -     -    fsdaemon Ld rwt oldmask
 14 s   626     1   626     0     0   26   - 01232235 msgdaemo Ld
 15 s   777     1   777     0     0   26   - 01232235 inetd    Ld
 17 s   792     1   792     0     0   26   - 01232235 lpd      Ld
 18 s   782     1   301     0     0   26   - 05264014 sendmail Ld
 19 s   984     1   984     0     0   40   - 07300071 tpdaemon Ld
 20 s   959     1   301     0     0   26   - 01232235 errdemon Ld
```

```
 21 s   799     1   301     0     0   26   - 01232235 snmpd    Ld
 22 s   812     1   812     0     0   26   - 01232235 portmap  Ld
 23 s   850     1   850     0     0   26   - 01232235 ypserv   Ld
 24 s   968     1   968     0     0   26   - 01232235 syslogd  Ld
 26 r   975     1   975     0     0   26   -     -    crayperf Ld
 27 r  1135   984   984     0     0   39   -     -    slnet    Ld rwt oldmask
 28 s   859     1   859     0     0   26   - 01232235 ypbind   Ld
 29 s  1304     1  1304     0     0   26   - 05463020 nfsd     Ld
 30 s  1092  1014  1014     0     0   39   - 00205552 qfdaemon Ld rwt
 31 s  1088  1014  1014     0     0   26   - 05325614 netdaemo Ld
 32 s  1305  1304  1304     0     0   26   - 05463020 nfsd     Ld
 33 s  1013     1  1013     0     0   26   - 00671340 logdaemo Ld
 34 s  1014     1  1014     0     0   26   - 00672324 nqsdaemo Ld
 35 s  1117     1  1113     0     0   26   - 05331414 rtidaemo Ld
 36 s  1144   984   984     0     0   39   - 00205552 avrproc  Ld rwt
 37 s  1306  1304  1304     0     0   26   - 05463020 nfsd     Ld
 38 s  1307  1304  1304     0     0   26   - 05463020 nfsd     Ld
 39 s  1313  1312  1312     0     0   26   - 05466620 cnfsd    Ld
 40 s  1312     1  1312     0     0   26   - 05466620 cnfsd    Ld
 41 s  1314  1312  1312     0     0   26   - 05466620 cnfsd    Ld
 42 s  1315  1312  1312     0     0   26   - 05466620 cnfsd    Ld

 44 s  1324     1  1324     0     0   26   - 01232235 mountd   Ld
 45 s  1329     1  1329     0     0   26   - 02246673 biod     Ld
 46 s  1330     1  1330     0     0   26   - 02246673 biod     Ld
 47 s  1331     1  1331     0     0   26   - 02246673 biod     Ld
 48 s  1332     1  1332     0     0   26   - 02246673 biod     Ld
 49 s  7250     1  7250     0     0   26   - 01232235 automoun Ld
 51 s  7271  7260  7271     0     0   26   - 00704723 tpconfig Ld
 53 s  7248     1  7248     0     0   26   - 01232235 automoun Ld
 54 s  7252  1127  7252   526     0   30   - 00253176 zup      Ld Nocore rwt
 55 s  7255     1  7255     0     0   26   - 01232235 automoun Ld
 56 s  7253     1  7253     0     0   26   - 01232235 automoun Ld
 57 s  7257     1  7257     0     0   26   - 01232235 automoun Ld
 58 s  7259     1  7259     0     0   26   - 01232235 automoun Ld
> stack 51
u_save[0].B01=01735147, u_save[0].B02=010122231, &u_stack[0]=010122057
```

```
KERNEL STACK TRACE for proc slot 51:

 routine name         entry point  return addr  line #   arguments

 swtch                01737312c    01734626b    *** arg list ptr out of bounds
 sleep                01734353c    01400026b    315      0000000000000000704723
                                                         0000000000000000004032
 nc1pread             01377655c    01741022a    162      0000000000000000602250
                                                         0000000000000000000000
 rdwr                 01740534c    01740517a    63       0000000000000000000001
 read                 01740506c    01606140a    *** arg list ptr out of bounds
 umain                01670606c    01607612b
> stack 19
u_save[0].B01=01735147, u_save[0].B02=020220305, &u_stack[0]=020220057
KERNEL STACK TRACE for proc slot 19:

 routine name         entry point  return addr  line #   arguments

 swtch                01737312c    01734626b    *** arg list ptr out of bounds
 sleep                01734353c    01536700d    2866     0000000000000007300071
                                                         0000000000000000001724
 bmxsleep             01536641c    01536332d    2693     0000000000000007300071
                                                         0000000000000000001724
 bmxaux               01536203c    01527513b    604      0000000000000007300071
                                                         0000000000000000000005
                                                         0000000000000000000020
                                                         0000000000000000000001
 bmxclose             01526767c    01407254a    693      0000000000000000600112
                                                         0000000000000000000001
 nc1closei            01407050c    01645755a    110      0000000000000000600112
                                                         0000000000000000000001
                                                         0000000000000000000001
                                                         0000000000000000000000
                                                         0000000000000000511011
 closef               01645574c    01745033c    1238     0000000000000000511011
                                                         0000000000000000000013
 close                01745004c    01606140a    *** arg list ptr out of bounds
 umain                01670606c    01607612b
> q
Back in host crash.
> q
```