Using the Oracle oradebug Utility

Debugging Oracle Applications

Mike Ault

Retail Price $7.95 US/$11.95 Canada
ISBN: 0-9740716-7-6
Copyright © 2003 by Rampant TechPress

Rampant TechPress

Using the Oracle oradebug Utility

Debugging Oracle applications

Mike Ault

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Notice

While the author & Rampant TechPress makes every effort to ensure the information presented in this white paper is accurate and without error, Rampant TechPress, its authors and its affiliates takes no responsibility for the use of the information, tips, techniques or technologies contained in this white paper. The user of this white paper is solely responsible for the consequences of the utilization of the information, tips, techniques or technologies reported herein.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Using the Oracle oradebug Utility

Debugging Oracle applications

By Mike Ault

Copyright © 2003 by Rampant TechPress. All rights reserved.

Published by Rampant TechPress, Kittrell, North Carolina, USA

Series Editor: Don Burleson

Production Editor: Teri Wade

Cover Design: Bryan Hoff

Oracle, Oracle7, Oracle8, Oracle8i, and Oracle9i are trademarks of Oracle Corporation. *Oracle In-Focus* is a registered Trademark of Rampant TechPress.

Many of the designations used by computer vendors to distinguish their products are claimed as Trademarks. All names known to Rampant TechPress to be trademark names appear in this text as initial caps.

The information provided by the authors of this work is believed to be accurate and reliable, but because of the possibility of human error by our authors and staff, Rampant TechPress cannot guarantee the accuracy or completeness of any information included in this work and is not responsible for any errors, omissions, or inaccurate results obtained from the use of information or scripts in this work.

Visit **www.rampant.cc** for information on other *Oracle In-Focus* books.

ISBN: 0-9740716-7-6

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Table Of Contents

Notice iii
Using the Oracle oradebug Utility – Publication Information .. iv
Table Of Contents v
Introduction 1
Invoking Oradebug 1
Using Debug 3
Monitoring the Current Session With the *oradebug setmypid* 4
Using DUMP 5
Commands not requiring SPID to be set 7
Using Oradebug for System Hangs: 10
Getting a PROCESSTATE DUMP 11
Getting a ERRORSTACKS DUMP 12
Using ORADEBUG to Trace A Sessions SQL 12
How to Find the Right PID for oradebug setospid 16
Tracing Errors Using ORADEBUG 17
Using ORADEBUG to Find Semaphore and Memory Segments 19
Finding Parallel SQL Processes Using ORADEBUG 21
Tracking down ORA-04030 errors 22
Using Oradebug to Debug Spinning Processes 22
Oracle 8 IDLM and ORADEBUG 23
How to determine the Events Set in a System 24
Using ORADEBUG to Release DDL locks 24
How to Trace Trigger Actions Using Oradebug 26
Checking on Temporary Segment Usage with Oradebug 26
Suspending a process using Oradebug 27

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Resuming A Process using Oradebug..... 27
Looking at a Processes Statistics using Oradebug..... 28
Looking at a Process Error Stack..... 29
Using Oradebug to Dump the SGA..... 29
Using Oradebug to Look at Latches..... 30
Using ORADEBUG to Look at Library Cache States..... 30
Getting Parallel Server DML Locks Using Oradebug..... 31
Dumping the Control File Contents Using ORADEBUG..... 31
Summary..... 33

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Introduction

Beginning in Oracle7 with ORADBX, the ORADEBUG utility allows DBAs to start and stop tracing for any session, dump SGA and other memory structures, wakeup oracle processes such as SMON or PMON, suspend and resume processing in a SID, debug enqueue services, debug the CGS name service, dump core files and IPC information, in fact many useful operations that aren't usually available. Unfortunately this utility, other than a terse paragraph in the administrator's manuals, is virtually undocumented. In this paper I have attempted to provide as much reference material as can be found or developed about ORADEBUG from various lists, Metalink and the Pipelines at Quest RevealNet labs.

Invoking Oradebug

Oradebug is invoked from SVRMGRL in pre-9i instances and from SQLPLUS in 9i and greater versions. The main commands of oradebug can be displayed by entering *oradebug help* from the svrmgrl or SQLPLUS command line as is shown in Figure 1.

```
SQL> oradebug help
HELP            [command]                 Describe one or all commands
SETMYPID                                  Debug current process
SETOSPID        <ospid>                   Set OS pid of process to debug
SETORAPID       <orapid> ['force']        Set Oracle pid of process to
debug
DUMP            <dump_name> <level>       Invoke named dump
DUMPSGA         [bytes]                   Dump fixed SGA
DUMPLIST                                  Print a list of available dumps
EVENT           <text>                    Set trace event in process
SESSION_EVENT   <text>                    Set trace event in session
DUMPVAR         <p|s|uga> <name> [level]  Print/dump a fixed PGA/SGA/UGA
variable
SETVAR          <p|s|uga> <name> <value>  Modify a fixed PGA/SGA/UGA
variable
PEEK            <addr> <len> [level]      Print/Dump memory
POKE            <addr> <len> <value>      Modify memory
WAKEUP          <orapid>                  Wake up Oracle process
SUSPEND                                   Suspend execution
RESUME                                    Resume execution
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
FLUSH                                    Flush pending writes to trace
file
CLOSE_TRACE                              Close trace file
TRACEFILE_NAME                           Get name of trace file
LKDEBUG                                  Invoke global enqueue service
debugger
NSDBX                                    Invoke CGS name-service debugger
-G              <Inst-List | def | all>  lkdebug cluster database command
prefix
-R              <Inst-List | def | all>  lkdebug cluster database command
prefix
SETINST         <instance# .. | all>     Set instance list in double
quotes
SGATOFILE       <SGA dump dir>         Dump SGA to file; dirname in double
quotes
DMPCOWSGA       <SGA dump dir> Dump & map SGA as COW; dirname in double
quotes
MAPCOWSGA       <SGA dump dir>         Map SGA as COW; dirname in double
quotes
HANGANALYZE     [level]                  Analyze system hang
FFBEGIN                                  Flash Freeze the Instance
FFDEREGISTER                             FF deregister instance from
cluster
FFTERMINST                               Call exit and terminate instance
FFRESUMEINST                             Resume the flash frozen instance
FFSTATUS                                 Flash freeze status of instance
CORE                                     Dump core without crashing
process
IPC                                      Dump ipc information
UNLIMIT                                  Unlimit the size of the trace
file
PROCSTAT                                 Dump process statistics
CALL            <func> [arg1] ... [argn] Invoke function with arguments
```

Figure 1: Results of oradebug help for Oracle9i

Once a command is entered into the oradebug utility, almost all of the results are generated into trace files in the udump destination location. The udump location can be identified by reviewing the initialization parameter file for the user_dump_dest parameter value, reviewing initialization parameter settings using Oracle Enterprise Manager (OEM), or, by issuing the *SHOW PARAMETER USER_DUMP_DEST* command in SQLPLUS. In Oradebug the TRACEFILE_NAME command will display the current processes dumpfile name. The trace files generated from the oradebug utility will be named for the spid of the process from where the oradebug is executed.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Using Debug

In order to use oradebug, for most commands, you must first tell oradebug what SPID you wish to execute the debug commands against. You see, the oradebug utility can be run from a privileged user against any other process providing you know the spid of the other process. The SPIDs of the various processes can be found with a simple select, as shown in Figure 2.

```
 SQL> SELECT a.username, a.sid, a.serial#, b.spid
    2 FROM v$session a, v$process b
    3 WHERE a.paddr=b.addr;

USERNAME                         SID    SERIAL# SPID
------------------------------ ---------- ---------- ----
                                     1          1 1588
                                     2          1 1540
                                     3          1 1524
                                     4          1 1528
                                     5          1 1592
                                     6          1 1556
                                     7          1 212
SYS                                  8         70 1964
DBAUTIL                             11         20 620
```

Figure 2: Select to get users and SPIDS

Of course, if you know the username you are looking for, you can modify the select command shown in Figure 2 to reflect this username restriction. If you are in a system where all users use the same username, you can also restrict the command by terminal being used, operating system userid or program being executed as all of this is also stored in the V$SESSION view.

Once you have the SPID you set the value using the command shown in figure 3.

```
$ sqlplus /nolog
SQL> connect sys as sysdba
Password: xxxxx

Connected.
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SQL> oradebug setospid <SPID>
SQL> oradebug unlimit
```

Figure 3: Setting SID for oradebug session

Note in Figure 3 the use of the *unlimit* command, this removes any restriction on trace file size imposed by the initialization parameter settings for the instance for this session.

Monitoring the Current Session With the *oradebug setmypid*

Once the SPID is set, any SID specific oradebug operation can be performed. The SID restricted operations are listed in Figure 4.

```
DUMP            <dump_name> <level>       Invoke named dump
DUMPLIST                                  Print a list of available dumps
EVENT           <text>                    Set trace event in process
SESSION_EVENT   <text>                    Set trace event in session
DUMPVAR         <p|s|uga> <name> [level]  Print/dump a fixed PGA/SGA/UGA
variable
SETVAR          <p|s|uga> <name> <value>  Modify a fixed PGA/SGA/UGA
variable
PEEK            <addr> <len> [level]      Print/Dump memory
POKE            <addr> <len> <value>      Modify memory
SUSPEND                                   Suspend execution
RESUME                                    Resume execution
FLUSH                                     Flush pending writes to trace
file
CLOSE_TRACE                               Close trace file
TRACEFILE_NAME                            Get name of trace file
LKDEBUG                                   Invoke global enqueue service
debugger
NSDBX                                     Invoke CGS name-service debugger
-G              <Inst-List | def | all>   lkdebug cluster database command
prefix
-R              <Inst-List | def | all>   lkdebug cluster database command
prefix
HANGANALYZE     [level]                   Analyze system hang
CORE                                      Dump core without crashing
process
PROCSTAT                                  Dump process statistics
CALL            <func> [arg1] ... [argn]  Invoke function with arguments
```

Figure 4: SID restricted operations

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Using DUMP

The DUMP command allows various structures of a process to be dumped to trace file for examination. The types of dump available for a process are listed in Figure 5. These can be listed at anytime by using the *oradebug dumplist* command.

EVENTS
TRACE_BUFFER_ON
TRACE_BUFFER_OFF
HANGANALYZE
LATCHES
PROCESSSTATE
SYSTEMSTATE
INSTANTIATIONSTATE
REFRESH_OS_STATS
CROSSIC *
CONTEXTAREA
HEAPDUMP
HEAPDUMP_ADDR
POKE_ADDRESS *
POKE_LENGTH *
POKE_VALUE *
POKE_VALUE0 *
GLOBAL_AREA
MEMORY_LOG
REALFREEDUMP
ERRORSTACK
HANGANALYZE_PROC
TEST_STACK_DUMP
BG_MESSAGES
ENQUEUES
SIMULATE_EOV
KSFQP_LIMIT
KSKDUMPTRACE
DBSCHEDULER
GRANULELIST

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

GRANULELISTCHK
SCOREBOARD
GES_STATE
ADJUST_SCN
NEXT_SCN_WRAP
CONTROLF
FULL_DUMPS
BUFFERS
RECOVERY *
SET_TSN_P1 *
BUFFER
PIN_BLOCKS *
BC_SANITY_CHECK
FLUSH_CACHE *
LOGHIST *
ARCHIVE_ERROR
REDOHDR
LOGERROR
OPEN_FILES *
DATA_ERR_ON *
DATA_ERR_OFF *
TR_SET_BLOCK *
TR_SET_ALL_BLOCKS *
TR_SET_SIDE *
TR_CRASH_AFTER_WRITE *
TR_READ_ONE_SIDE *
TR_CORRUPT_ONE_SIDE *
TR_RESET_NORMAL *
TEST_DB_ROBUSTNESS
LOCKS
GC_ELEMENTS
FILE_HDRS
KRB_CORRUPT_INTERVAL
KRB_CORRUPT_SIZE
KRB_PIECE_FAIL
KRB_OPTIONS
KRB_SIMULATE_NODE_AFFINITY

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

KRB_TRACE
KRB_BSET_DAYS
DROP_SEGMENTS *
TREEDUMP <OBJ_ID>
LONGF_CREATE
ROW_CACHE
LIBRARY_CACHE
SHARED_SERVER_STATE
KXFPCLEARSTATS
KXFPDUMPTRACE
KXFPBLATCHTEST
KXFXSLAVESTATE
KXFXCURSORSTATE <cursor_id>
WORKAREATAB_DUMP
OBJECT_CACHE *
SAVEPOINTS *

(* Indicates special arguments required)
All dump options take the standard LEVEL arguments 2, 4, 6, 8, 10, 12 except as noted.

Figure 5: DUMP Options

Commands not requiring SPID to be set

There are several commands that do not require a SPID to be set. These refer to system wide type dumps. An example is the IPC command, the results from the IPC command are shown in Figure 6.

```
Result of oradebug IPC on w2k 9.1.0.1.1 goes to trace file:

Dump file C:\oracle\admin\aultdb1\udump\ORA01960.TRC
Thu Mar 21 10:14:36 2002
ORACLE V9.0.1.1.1 - Production vsnsta=0
vsnsql=10 vsnxtr=3
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Instance name: aultdb1
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
Redo thread mounted by this instance: 1

Oracle process number: 11

Windows thread id: 1960, image: ORACLE.EXE


*** 2002-03-21 10:14:36.000
*** SESSION ID:(8.42) 2002-03-21 10:14:36.000
Dump of Windows skgm context
areaflags            0000007f
realmflags           00000001
maxtotalrealmsize    6f21f000
VMpagesize           00001000
VMallocgranularity   00010000
minappaddress         00010000
maxappaddress         7FFEFFFF
stacklimit            07561000
magic                acc01ade
Handle:              02C726B8 `sga_aultdb1'
Dump of Windows realm handle `sga_aultdb1', flags = 00000001
 Area #0 `Fixed Size' containing Subareas 0-0
  Total size 0000000000044f1c Minimum Subarea size 00000000
   Area  Subarea                Start Addr
      0        0                  02C90000
                               Subarea size
                                  00045000
 Area #1 `Variable Size' containing Subareas 2-2
  Total size 0000000009800000 Minimum Subarea size 00800000
   Area  Subarea                Start Addr
      1        2                  64800000
                               Subarea size
                                  09800000
 Area #2 `Redo Buffers' containing Subareas 1-1
  Total size 0000000000013000 Minimum Subarea size 00000000
   Area  Subarea                Start Addr
      2        1                  02CE0000
                               Subarea size
                                  00013000
--------- Process Post/Wait Resource Information ---------
Maximum threads:                 = 7500
Thread SHAD, tid: 1960, Post/Wait Event: 272 = 0
Thread PMON, tid: 1628, Post/Wait Event: 312 = 0
Thread DBW0, tid: 592, Post/Wait Event: 336 = 0
Thread LGWR, tid: 328, Post/Wait Event: 372 = 0
Thread CKPT, tid: 1648, Post/Wait Event: 392 = 0
Thread SMON, tid: 1688, Post/Wait Event: 412 = 0
Thread RECO, tid: 1696, Post/Wait Event: 432 = 0
Thread CJQ0, tid: 452, Post/Wait Event: 452 = 0
Thread S000, tid: 1268, Post/Wait Event: 472 = 0
Thread D000, tid: 440, Post/Wait Event: 492 = 0
```

Figure 6: Results of the oradebug IPC command

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Another command that is generic and requires no SID is the dumplist command. The results of ORADEBUG dumplist command are shown in Figure 7.

```
SQL> oradebug dumplist
EVENTS
TRACE_BUFFER_ON
TRACE_BUFFER_OFF
HANGANALYZE
LATCHES
PROCESSSTATE
SYSTEMSTATE
INSTANTIATIONSTATE
REFRESH_OS_STATS
CROSSIC
CONTEXTAREA
HEAPDUMP
HEAPDUMP_ADDR
POKE_ADDRESS
POKE_LENGTH
POKE_VALUE
POKE_VALUE0
GLOBAL_AREA
MEMORY_LOG
REALFREEDUMP
ERRORSTACK
HANGANALYZE_PROC
TEST_STACK_DUMP
BG_MESSAGES
ENQUEUES
SIMULATE_EOV
KSFQP_LIMIT
KSKDUMPTRACE
DBSCHEDULER
GRANULELIST
GRANULELISTCHK
SCOREBOARD
GES_STATE
ADJUST_SCN
NEXT_SCN_WRAP
CONTROLF
FULL_DUMPS
BUFFERS
RECOVERY
SET_TSN_P1
BUFFER
PIN_BLOCKS
BC_SANITY_CHECK
FLUSH_CACHE
LOGHIST
ARCHIVE_ERROR
REDOHDR
LOGERROR
OPEN_FILES
DATA_ERR_ON
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
DATA_ERR_OFF
TR_SET_BLOCK
TR_SET_ALL_BLOCKS
TR_SET_SIDE
TR_CRASH_AFTER_WRITE
TR_READ_ONE_SIDE
TR_CORRUPT_ONE_SIDE
TR_RESET_NORMAL
TEST_DB_ROBUSTNESS
LOCKS
GC_ELEMENTS
FILE_HDRS
KRB_CORRUPT_INTERVAL
KRB_CORRUPT_SIZE
KRB_PIECE_FAIL
KRB_OPTIONS
KRB_SIMULATE_NODE_AFFINITY
KRB_TRACE
KRB_BSET_DAYS
DROP_SEGMENTS
TREEDUMP
LONGF_CREATE
ROW_CACHE
LIBRARY_CACHE
SHARED_SERVER_STATE
KXFPCLEARSTATS
KXFPDUMPTRACE
KXFPBLATCHTEST
KXFXSLAVESTATE
KXFXCURSORSTATE
WORKAREATAB_DUMP
OBJECT_CACHE
SAVEPOINTS
```

Figure 7: Results from oradebug dumplist command

Using Oradebug for System Hangs:

One area where oradebug is particularly useful is in the diagnosis of system hangs. Usually the system will hang for all users except SYS. If you can log into the SYS user as SYSDBA or as the INTERNAL users in pre-Oracle9i systems, the system hang can be analyzed as is shown in Figure 8.

```
SQL> oradebug setmypid
SQL> oradebug unlimit
SQL> oradebug setinst all
SQL> oradebug hanganalyze 5
SQL> oradebug -g def dump systemstate 10
```

Figure 8: Commands to generate a trace for a system hang.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Notice in the commands in Figure 8 that we get a systemstate dump as well as a hang analyze dump. You don't need a hang to get a systemstate dump, these types of dumps can be obtained anytime using the commands in Figure 9.

This creates a large trace file in the user_dump_dest (30M or more is not unusual).

Note: the init<sid>.ora parameter MAX_DUMP_FILE_SIZE controls the maximum trace file size. Using Oradebug and setting unlimit will allow a complete dump which we will need.

Do this step for sure if the entire database is frozen or nearly frozen and if this condition came on suddenly and there are no archive errors in the alert log.

Please note: As systemstate dumps are instance specific, they tend to be inconclusive with hanging problems involving Oracle Parallel Server (OPS) unless you get them from each node. You will need 3 system state dumps from each node for OPS.

Do the systemstate dump 3 times in a row.

```
$ svrmgrl
  connect internal
  oradebug setmypid
  oradebug unlimit
  oradebug dump systemstate 10
```

Figure 9: Getting systemstate dumps from Oradebug.

Getting a PROCESSTATE DUMP

If you need to trace certain processes, use oradebug to get a PROCESSSTATE dump.

You should also trace the process from the os level using various os level tools as specific to your os.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

When you get processstate dumps you should get them 3 times. This generates a trace file in your user_dump_dest (from SVRMGRL or sqlplus: show parameter user_dump_dest).

This is demonstrated in Figure 10.

```
$ svrmgrl
      connect internal
      oradebug setospid <process ID>
      oradebug unlimit
      oradebug dump processstate 10
```

Figure 10: Example processstate dumps

Getting a ERRORSTACKS DUMP

You can also get errorstacks from the process. Again, you should usually do this 3 times. This generates a trace file in your user_dump_dest (from svrmgrl or sqplus: show parameter user_dump_dest).

This is demonstrated in Figure 11.

```
$ svrmgrl
      connect internal
      oradebug setospid <process ID>
      oradebug unlimit
              oradebug dump errorstack 3
```

Figure 11: Example errorstack dump

Using ORADEBUG to Trace A Sessions SQL

Why would you want to use oradebug to capture trace information rather than say, DBMS_SYSTEM? If all you are interested in is a level 1 trace to say, capture SQL generated by Discoverer User Edition in order to run the same statements from SQL PLUS then DBMS_SYSTEM is fine. Let's look at this, let me remind you that Discoverer User Edition opens two sessions on the database and you need system privileges to see the trace, i.e. log in as SYS.

First you will need to get SID, SERIAL#, PADDR from V$SESSION, for example:

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SQL> select username, sid, serial#, paddr from v$session where
username='VIDEO31';

USERNAME                          SID   SERIAL#        PADDR
------------------------------------------------------------------------
VIDEO31                          14      13202       820532C8
VIDEO31                          15       4665       82053EC8
```

Once you know the SID and the SERIAL#, you can enable the trace for each session running the command:

```
EXECUTE DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(<SID>, <SERIAL#>, TRUE)
```

Everything the user does will now be traced at trace level 1 until you execute the SQL command again, but replacing 'TRUE' with 'FALSE'.

But what if you want a more detailed trace? Say a level 4? Obviously, since there is no way to set the level using DBMS_SYSTEM, we should use oradegub instead. Let's look at that example next.

Sometimes the trace at level 1 isn't enough, because in the sql statements there are some bind variables. You need their values before you run the query into SQLPLUS. In this case you have to perform trace at level 4, so that you have the value of each bind variable in the .trc file.

Enabling the level 4 trace for a Discoverer user, first get SID, SERIAL#, PADDR from V$SESSION. For example:

```
SQL> select username, sid, serial#, paddr from v$session
where username='VIDEO31';

USERNAME                          SID   SERIAL#        PADDR
------------------------------------------------------------------------
VIDEO31                          14      13202       820532C8
VIDEO31                          15       4665       82053EC8
```

Then you get SPID from the following query:

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SELECT ADDR, PID, SPID FROM V$PROCESS WHERE ADDR = <PADDR from V$SESSION>;
```

For example:

```
SQL> SELECT ADDR, PID, SPID FROM V$PROCESS WHERE ADDR = '820532C8';

ADDR               PID   SPID
---------------------------------------
820532C8            9     5408

SQL> SELECT ADDR, PID, SPID FROM V$PROCESS WHERE ADDR = '82053EC8';

ADDR               PID    SPID
-----------------------------------------
82053EC8           13     5410
```

You can then enable the level 4 trace from a DBA group user using the commands:

```
Sqlplus /nolog
CONNECT / as sysdba
ORADEBUG SETOSPID <SPID from the above query>
ORADEBUG EVENT 10046 TRACE NAME CONTEXT FOREVER, LEVEL 4
```

This should now trace at level 4 everything the Discoverer user does.

Once you have completed gathering the information you need you turn off the SQL trace for the session like so:

```
ORADEBUG EVENT 10046 TRACE NAME CONTEXT OFF
```

An example of a trace session using oradebug is in Figure 12.

```
$ ORACLE_SID=owbdw; export ORACLE_SID;
$ sqlplus /nolog
SQL*Plus: Release 9.0.1.0.0 - Production on Sun Sep 22 13:57:23 2002
© Copyright 2001 Oracle Corporation. All rights reserved.

SQL> connect / as sysdba
Connected.
SQL> ORADEBUG SETOSPID 5408
Oracle pid: 9, Unix process pid: 5408, image: oracle@misun08 (TNS V1-V3)
SQL> ORADEBUG EVENT 10046 TRACE NAME CONTEXT FOREVER, LEVEL 4
Statement processed.
SQL> ORADEBUG SETOSPID 5410
Oracle pid: 13, Unix process pid: 5410, image: oracle@misun08 (TNS V1-V3)
SQL> ORADEBUG EVENT 10046 TRACE NAME CONTEXT FOREVER, LEVEL 4
Statement processed.
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
---DO PROCESSING ON MONITORED PIDS---

SQL> ORADEBUG EVENT 10046 TRACE NAME CONTEXT OFF
Statement processed.
```

As we have seen, another capability of the oradebug program is the ability enable/disable setting the SQL tracing for another user's session. To enable tracing for another session, the Oracle process identifier (PID) or the Operating System processes identifier (SPID) must be identified from v$process. This is an effective way of capturing a SQL trace from a process that is already running. The output can be used to analyze SQL related performance issues.

The ORADEBUG dump produces a trace file in the user_dump_dest that can be formatted with TKPROF. Do the following:

1. Obtain the Oracle process identifier or the Operating System process identifier (SPID) from v$process:

```
SVRMGR> select pid, spid, username from v$process;

            PID    SPID   USERNAME
            ----   -----  --------
            8      25807  oracle
```

2. Attach to the process using ORADEBUG.

   Using the Oracle process identifier:

```
SVRMGR> oradebug setorapid 8

Unix process pid: 25807, image: oracleV804
```

```
 - or -
```

   Using the Operating System process identifier:

```
SVRMGR> oradebug setospid 25807

Oracle pid: 8, Unix process pid: 25807, image: oracleV804
```

3. Turn on SQL Trace for the session.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SVRMGR> oradebug event 10046 trace name context forever, level 12

Statement processed.
```

4. Turn off the SQL trace for the session.

```
SVRMGR> oradebug event 10046 trace name context off
```

5. Format trace file using TKPROF.

How to Find the Right PID for oradebug setospid

There are ways to find the correct os pid from the Oracle database, by querying v$session and v$process, but this method gets the information strictly from the os.

1. At the Unix prompt, type who am i. For example:

```
[tiger5]/app/oracle/product/8.1.6> who am i
 kfarmer    pts/25       Apr 13 10:37    (rociblj-ppp-9.us.oracle.com)
```

   Take note of the terminal, in this case pts/25.

2. Start SQL*Plus and connect as sys as sysdba.

```
[tiger5]/app/oracle/product/9.0.1> sqlplus /nolog

 SQL*Plus: Release 9.0.1.0.0 - Production on Sun Sep 22 13:57:23 2002
 © Copyright 2001 Oracle Corporation. All rights reserved.

 SQL> connect / as sysdba
 Connected.
```

3. Issue this command from SQLPLUS /nolog connected as SYS:

```
 !ps -ef | grep svrmgrl
```

   Your output will look something like this:

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SVRMGR> !ps -ef | grep svrmgrl
kfarmer  3705 25983  0 15:27:52 pts/25   0:00 svrmgrl
kfarmer  2504 24262  0 14:43:57 pts/18   0:00 svrmgrl
kfarmer  3759  3757  0 15:29:52 pts/25   0:00 grep svrmgrl
jharwell 28026 27944  0 16:42:25 pts/17   0:00 svrmgrl
```

Look for the terminal that matches your terminal from the 'who am i' command. Note the pid that goes with it, in this case 3705. There will be another process connected with this terminal, too, but that is the grep, not server manager.

4. Issue this command from SQLPLUS /nolog connected as SYS:

```
!ps -ef | grep 3705

Your output will look something like this:

SVRMGR> !ps -ef | grep 3705
 oracle  3706  3705  0 15:27:52 ?        0:00 oracleV816
 (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
 kfarmer  3705 25983  0 15:27:52 pts/25   0:00 sqlplus
 kfarmer  3856  3705  0 15:33:37 pts/25   0:00 [ sh ]
 kfarmer  3859  3857  0 15:33:37 pts/25   0:00 grep 3705
```

5. Look for a process that has a parent process of 3705. There are 2 here. One is 3706, a sqlnet connection, and 3856, a shell. In this case, I'm searching for the sqlnet connection that is running sqlplus, process 3706. This is the pid I would use for my oradebug setospid command:

```
oradebug setodpid 3706
```

Tracing Errors Using ORADEBUG

What about setting up to trace other errors? It is really quite simple, once you set the SID you are monitoring as shown in previous examples, just use the error code as the input to the event command in oradebug. For example, to trace the occurance of ORA-00942 errors you would enter the command:

```
ORADEBUG EVENT 942 TRACE NAME ERRORSTACK LEVEL 3
```

..which will only produce anything if this session hits an ORA-942 error.

The output from oradebug is a raw trace file. Some experienced DBAs can read these trace files in their raw state, however, I find it much easier to

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

use another Oracle utility, tkprof, to format the output into human readable output.

The 'raw' Trace File is the opposite of the tkprof'd version, in that it shows you the exact sequence in which the various pieces of SQL were run.

Just bear in mind the following: before it is actually executed, any piece of SQL is parsed into the SGA. It is allocated a CURSOR # at this point. This CURSOR # will remain in memory, containing the same piece of SQL code, until another piece of SQL needs to overwrite the memory, at which point the CURSOR # becomes available for re-use as well.

Whenever a piece of SQL is actually executed, an EXEC line is written to the Trace File. Highly simplified, you might see something like this in the 'raw' Trace File:

```
PARSE #1
  SELECT 'x' FROM TABLE1;
PARSE #2
  SELECT 'y' FROM TABLE2;
EXEC #1
EXEC #1
EXEC #2
PARSE #1
  UPDATE TABLE1 SET COLUMN1 = :b0;
EXEC #1
```

Note that CURSOR #1 has now been over-written with a new SQL statement, so any further EXEC statements for that cursor will relate to the 'new' SQL.

Once you have got the hang of this, using Level 4 Trace to extract 'BIND VARIABLES' from the Trace File is straightforward. In the example above, the ':b0' in the penultimate line is a bind variable, and if Level 4 Trace were switched on, there would be an extra set of lines before each EXEC line which would say (in the example):

```
BINDS #1
```

This would be followed by a list of bind variables (:b0, :b1 etc) with some information about each. The last piece of information given against each bind variable is the actual value passed to the SQL statement.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

There are two things to note about this:

1. There will probably be many occurrences of ':b0' type variables in a Level 4 Trace that are not bind variables (for example, the target column_names in a 'SELECT INTO..' statement. However, the only values shown in the BINDS information will be for values in an UPDATE or INSERT statement, or in a WHERE clause.
2. The numbers allocated in the BINDS section do not necessarily bear any relation to the numbers in the PARSE, but rather will always start from zero. That is to say, if the first variable in the WHERE clause is ':b36', it will still appear as ':b0' under BINDS for that cursor.

Using ORADEBUG to Find Semaphore and Memory Segments

We talked about the oradebug utility and non-sid specific commands, in our example we used IPC. What exactly can IPC be used for? You can verify the shared memory segments and semaphores that are attached to the running instances using IPC in oradebug. This is the way to locate which set of semaphores or memory areas are attached to a particular instance.

First, run the "ipcs -b" command to show the memory and semaphore listings for the Unix box. An example output from the ipcs -b command on an HPUX system is shown in Figure 13.

```
gpsp083:/dwqpkg/orasw > ipcs -b
IPC status from /dev/kmem as of Sun Sep 22 12:54:16 2002
T      ID     KEY        MODE        OWNER     GROUP QBYTES
Message Queues:
q       0 0x3c1827b6 -Rrw--w--w-      root      root  16384
q       1 0x3e1827b6 --rw-r--r--      root      root    264
T      ID     KEY        MODE        OWNER     GROUP  SEGSZ
Shared Memory:
m    7176 0x80bfe0c4 --rw-r-----    oracle       dba 1064779776
m   18953 0x94644538 --rw-r-----    oracle       dba 247726080
m   22026 0x00000000 --rw-r-----    oracle       dba 3221225472
m   14347 0x00000000 --rw-r-----    oracle       dba 3221225472
T      ID     KEY        MODE        OWNER     GROUP NSEMS
Semaphores:
s 3304021 0x8d6f79a0 --ra-r-----    oracle       dba  2048
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
s 2384022 0x8d7059e0 --ra-r-----    oracle       dba  2048
s 5312023 0x0b148384 --ra-r-----    oracle       dba   159
s  184024 0xaaa24ea0 --ra-r-----    oracle       dba   154
gpsp083:/dwqpkg/orasw >
```

Figure 13: Example ipcs -b command output

As you can see, the output is not very user friendly, it doesn't give the database sid information but only an internal id for semaphores or memory segments.

To determine the actual instance that is connected to a set of semaphores or memory areas, perform these steps for each instance that is up and running(Note: the documents in Metalink state the SPID doesn't have to be set, I found at least on SuSE Linux and Oracle9i 9.0.1 you had to set it.):

```
>sqlplus /nolog

SQL> connect sys as sysdba
Password: xxxxxx

Connected.

SQL> oradebug setmypid
Statement processed.
SQL> oradebug ipc
Information written to trace file.
SQL> oradebug tracefile_name
/var/oracle/OraHome2/admin/galinux1/udump/ora_19134.trc
```

This will show the shared memory segment and semaphore that each instance has attached/in use.

An example output from the summary section of the oradebug ipc command is shown in Figure 13.

```
-------------- Shared memory --------------
Seg Id     Address   Size
22026      c1eaf000  3221225472
Total: # of segments = 1, size = 3221225472
-------------- Semaphores ----------------
Total number of semaphores = 159
Number of semaphores per set = 159
Number of semaphore sets = 1
Semaphore identifiers:
5312023
```

Figure 13: Example output from oradebug IPC command.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

The Seg Id shows 22026 for the shared memory that is attached to the RUNNING instance. If you were looking to remove a memory segment from a crashed instance (for example from a broken pipe in Unix) you would now know that this shared memory segment is not the one to remove.

The Semaphore identifier shows 5312023 for the semaphore that is attached to the RUNNING instance. Again, a broken pipe in Unix could result in an instance crash where the memory and semaphore assignments are left hanging, this would show that this semaphore set was not the one to remove.

Once you have noted ALL of the identifiers for ALL of the instances which are up and running, compare these id numbers to those in the "ipcs -b" listing. The entry that does not have a running instance to match is the orphaned entry.

The ipc command used to remove these entries is:

```
ipcrm <option> <id#>
```

NOTE: The option differs for shared memory and semaphores.

```
ipcrm -m <shm_id#>      <== Use for the Shared Memory entry

ipcrm -s <sem_id#>      <== Use for the Semaphore entry
```

Finding Parallel SQL Processes Using ORADEBUG

Set parallel_min_servers = 4 (or any other number you like) to prespawn a number of slaves on instance startup, look them up in your process list by issuing the command ps -ef | grep p00 on your HP Unix system and note the process numbers, then fire up svrmgrl, connect internal and for each process do:

```
SVRMGR> oradebug setospid <process id>
SVRMGR> oradebug unlimit
SVRMGR> oradebug event 10046 trace name context forever, level 4
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Then do whatever you want to trace that uses the parallel query slaves and the sql trace info will appear in files in background_dump_dest,

Tracking down ORA-04030 errors

If a process size keeps growing, then it may eventually fail with an ORA-4030 "out of process memory when trying to allocate %s bytes" error if the operating system is exhausted of memory, or the memory size hits some operating system defined limit (such as maxdsiz on HP-UX).

To start diagnosing a problem with the process size, such as a suspected memory leak, a heapdump of the offending process is required:

```
$ sqlplus /nolog
SQL> connect sys as sysdba
Password: xxxxxxxxx
Connected.

SQL> oradebug setospid <pid>
SQL> oradebug unlimit
SQL> oradebug dump heapdump 5  <--this dumps PGA and UGA heaps
```

Using Oradebug to Debug Spinning Processes

In the case of a Spin situation the session events may or may not be static depending on where in the code the spinning is taking place. It would be expected that the session would be utilizing resources heavily such as CPU and memory.

For a Spin situation it is important to determine which area of the code the session is spinning in. Some indication of this may be derived from the event however it is usually necessary to produce an errorstack of the process a few times for analysis by support:

```
>sqlplus /nolog
SQL> connect sys/sys as sysdba
Password: xxxxxxx
Connected.

SQL> oradebug setospid <SPID>
SQL> oradebug unlimit
SQL> oradebug dump errorstack 3
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Where the SPID is the operating system process identifier, you can get it from v$process.

Note that more detailed information can be found by dumping systemstate information for the instance:

```
ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME SYSTEMSTATE LEVEL 10';
```

The systemstate tracefile will be created in your USER_DUMP_DEST directory. Get the Process ID of the problem session from the V$PROCESS:

```
SELECT PID FROM V$PROCESS WHERE ADDR=
       (SELECT PADDR FROM V$SESSION
        WHERE SID=sid_of_problem_session);
```

The systemstate dump includes information for each process, search for 'PROCESS id' and look up the wait event by doing a search on 'waiting for'.

Oracle 8 IDLM and ORADEBUG

It is possible to dump the IDLM using lkdebug in oradebug. After you connect internal you can type 'oradebug lkdebug help' to see a list of options. This is similar to lkdump and lkdbx in Oracle7 with the following important exceptions.

1. Some lkdump commands take a database name parameter for eg. lkdump -O 0x0 0x0 ST V733

The database name parameter is not now required as you will already be connected to the database in question in svrmgrl when running:

```
oradebug lkdebug -O 0x0 0x0 ST 2
```

The output does not go to the screen but rather to the user_dump_dest location for your svrmgrl session. This means that the best tip for using lkdebug is to tail the trace file in another window.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

How to determine the Events Set in a System

One way to find events set is to use DBS_SYSTME. However, you will find that the read_ev call used in dbms_system works only for event name CONTEXT. An alternative approach is to invoke ORADEBUG dump events N where N is:

1 for session
2 for process
4 for system

Use svrmgrl, or sqlplus with connect internal os sys as sysdba in your version of Oracle.

```
$ sqlplus /nolog
SQL> connect / as sysdba
Connected
SQL> oradebug setmypid
Statement processed.
SQL> oradebug dump events 4
Statement processed.
SQL> oradebug tracefile_name
/var/oracle/OraHome2/admin/galinux1/udump/ora_19206.trc
SQL>!vi /var/oracle/OraHome2/admin/galinux1/udump/ora_19206.trc
```

The output will give a typical trace file header and then will list the information on all events that have been set at the particular level in the instance you asked for, in this case, system wide.

Using ORADEBUG to Release DDL locks

First, you must kill the sessions through alter system kill session, OEM or some other Oracleprocess killer. Then make sure the server processes are really dead (in the OS, I know how to do this on Unix, but can't help you with NT other than to suggest getting the MKS toolkit from Microsoft which has a monitor session to break threads into their component processes.)

Eventually, PMON will wakeup and notice that it needs to clean up those locks and then the resources will be freed. However, using oradebug, you

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

can wakeup PMON immediately. To do that, connect either svrmgrl or sqlplus as internal (or as sys as sysdba in Oracle9i.) Then:

```
select pid from v$process p, v$bgprocess b
where b.paddr = p.addr
and name='PMON'
```

That tells you the Oracle pid of the PMON process. Then, just:

```
oradebug wakeup <pid>
```

The oradebug command requires connect internal or sys as sysdba.

Why this works:

What actually being held on a stored pacakage/procedure/function when a user process is running it is a library cache pin. It remains in force till the execution completes. If the running process is interupted (gets ORA-1013), then the process will rollback any changes (this could take a very long time) and then release the pin. During the rollback, the library cache pin is being held. If you kill the session (alter system kill session), the corresponding server process is not necessarily killed immediately. Until it's actually gone, PMON won't start cleaning up. Time wise, this alternative could leave things tied up as long as the previous scenario.

So, we come to our scenario: Alter system kill session, then kill server process in the OS, (SIGKILL is effective), now, the process has stopped executing, and PMON can start clean up immediately. It will free the library cache pin immediately, and take care of rollback after that, so, your stored object is freed. One other point: PMON wakes up every 3 seconds, but, it only checks for dead processes every 20th wake up, or, every 60 seconds. I have demonstrated this by running truss on the PMON process.

So, if you alter system kill session, but you don't go after the server process by killing via OS mechanism, if could take several minutes to clean up, and then PMON would take an additional minute to wake up and clean things up.

By doing alter system kill session, followed by zapping processes, followed by PMON wakeup, you get nearly instant results.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

How to Trace Trigger Actions Using Oradebug

```
Tracing trigger actions is easy to do in oradebug, you simply set the
10309 event using a level 1 trace.

$ sqlplus -s /nolog
SQL> connect / as sysdba
Connected.
SQL> oradebug event 10309 trace name context forever,  level 1
Statement Porcessed.
```

Don't forget to turn it off using context off when you are done.

Checking on Temporary Segment Usage with Oradebug

Temporary segments are created by sorts, hashes, temporary table usage and DDL statements like "create table as select .." or "alter index ... rebuild;".

When a temporary segment involves a DDL statement, the new object is originally created as a TEMPORARY segment in the target tablespace and is changed to the appropriate object type (table, index) when DDL action is finished.

You can check whether such action is active using ORADEBUG in the following manner:

1. connect internal or in Oracle9i sys as sysdba
2. SELECT owner FROM dba_segments WHERE segment_name=<temporary_segment>;
3. SELECT pid FROM v$process WHERE username=<owner of segment>;
4. oradebug setorapid <pid>
5. oradebug dump errorstack 3
6. This will generate trace files, which contain the "current sql statement".
7. If this is a CTAS or alter index ... rebuild, then you will have to wait for the DDL action to finish.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

If you do not find active processes for the segment owner, or none of them has a DDL action, it is possible that someone started this DDL statement and the session died for some reason. The process PMON will clean up stray processes after a crashed session and call SMON to remove this temporary segment. But temporary segments can be very large and removing them can take a long time, in some cases days. The temporary segment can also be removed by a database shutdown, or shutdown immediate. You can also force SMON to remove it by:

```
ALTER TABLESPACE <permanent tablespace> coalesce;
```

Alternatively see the section on waking up processes using ORADEBUG.

Suspending a process using Oradebug

Oradebug also allows you to suspend a process. First you need to identify the shadow process that you want to suspend. Then set your debug session to point to that process:

```
>sqlplus /nolog
SQL> connect sys as sysdba
Password: xxxxxxx

Connected.

SQL> oradebug setospid 21335
Oracle pid: 13, Unix process pid: 21335, image: oracleKLF
```

And then you can suspend its execution using suspend:

```
SQL> oradebug suspend
Statement processed.
```

This stops a process dead in its tracks, examining v$session_wait shows that it is waiting on the debugger:

```
8 GTX2       INACTIVE debugger
```

Resuming A Process using Oradebug

To start a suspended process off again use the resume command:

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SQLPLUS> oradebug resume
Statement processed.
SQLPLUS>
```

Looking at a Processes Statistics using Oradebug

You can also examine process stats using the procstat command:

```
SQLPLUS> oradebug procstat
```

The output in the trace file area for udump gives various details:

```
Dump file /opt/oracle/oradata10/dumparea/KLF/udump/ora_21335.trc
Oracle7 Server Release 7.3.3.5.0 - Production Release
With the distributed, replication and parallel query options
PL/SQL Release 2.3.3.5.0 - Production
ORACLE_HOME = /opt/oracle/app/oracle/product/7.3.3
System name:    DYNIX/ptx
Node name:      fes4
Release:        4.0
Version:        V4.4.2
Machine:        i386
Instance name: KLF
Redo thread mounted by this instance: 1
Oracle process number: 13
Unix process pid: 21335, image: oracleKLF

Mon Aug  3 08:48:35 1998
*** SESSION ID:(8.2121) 1998.08.03.08.48.35.000
----- Dump of Process Statistics -----
User time used = 6
System time used = 27
Maximum resident set size = 1431
Page faults = 8
Page reclaims = 0
Zero-Fill Pages = 1393
Resident Set Size increases = 0
Resident Set Size decreases = 86
Swaps = 0
System calls = 0
Voluntary context switches = 0
Involuntary context switches = 0
Signals received = 0
Logical Reads = 0
Logical writes = 0
Disk Reads = 2960
Disk writes = 0
Bytes from Logical reads = 631
Bytes from Logical writes = 0
Dumping the process:
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Looking at a Process Error Stack

You can also view a processes error stack using ORADEBUG.

```
SVRMGR> oradebug dump errorstack 1

Mon Aug  3 08:54:18 1998
ksedmp: internal or fatal error
Current SQL statement for this session:
select * from cust_applic
----- Call Stack Trace -----
calling              call     entry               argument values in hex
location             type     point               (? means dubious value)
-------------------- -------- ------------------- ----------------------
-----
ksedmp+133           CALL     ksedst              0
ksdxcb+908           CALLp    00000000            8045560 11 3 8045600
80455B0
ksdxsus+216          CALL     ksdxcb              1
sspuser+174          CALLr    00000000            1
nsprecv+4892         CALLr    00000000            881DAE0 881FE02 881DAB8
0
```

Notice how the current sql statement is displayed. This is useful for identifying what a hanging/long running program is doing. By suspending the process and dumping it, you can see what it is doing.

```
oradebug dump errorstack 2
```

Using Oradebug to Dump the SGA

For a greater level of detail, you can also dump the sga (be careful for large SGAs you may exceed the limits for your dump file, you may want to use oradebug unlimit first):

```
SVRMGR> oradebug dumpsga
```

Finding Out What can be Dumped

If you aren't sure what you want to dump, you can get a list of things to dump:

```
SVRMGR> oradebug dumplist
```

This gives the following list (the full list is found elsewhere):

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
LATCHES
PROCESSSTATE
SYSTEMSTATE
INSTANTIATIONSTATE
CROSSIC
```

Using Oradebug to Look at Latches

You can dump a plethora of things about a user processes. For example, let's look at the latches for a specified process (you must have used setmypid or setopid <pid>):

```
SQLPLUS> oradebug dump LATCHES 1
Statement processed.

*** SESSION ID:(7.1446) 1998.08.03.09.43.24.000
LATCH STATUS
============
 20000a48 latch wait list level=9 state=free
 20000ae0 process allocation level=0 state=free
 20000b3c session allocation level=5 state=free
 20000b98 session switching level=0 state=free
```

Using ORADEBUG to Look at Library Cache States

Again, once the SPID is set, you can look at many session specific statistics, now, let's look at library cache stats (remember that the actual output is placed in a trace file in the udump location) :

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

```
SQLPLUS> oradebug dump LIBRARY_CACHE 1
Statement processed.

..
LIBRARY CACHE STATISTICS:
      gets hit ratio       pins hit ratio    reloads   invalids namespace
---------- --------- ---------- --------- ---------- ---------- ---------
     51513 0.9595830     181042 0.9733432        700       1454 CRSR
      8425 0.8227893      15429 0.8210512        987          0 TABL/PRCD
       260 0.9730769        260 0.9730769          0          0 BODY
       307 0.8469055        333 0.7717718          4          0 TRGR
       609 0.0098522        642 0.0514019          6          0 INDX
        27 0.4444444         15 0.3333333          0          0 CLST
         0 0.0000000          0 0.0000000          0          0 OBJE
         0 0.0000000          0 0.0000000          0          0 PIPE
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
         0 0.0000000          0 0.0000000          0          0 ?
     61141 0.9305376     197721 0.9580773       1697       1454 CUMULATIVE
```

Getting Parallel Server DML Locks Using Oradebug

To get Parallel Server DML lock information you must be connected as a DBA level user and then use the various lkdebug commands to dump various information about the parallel DML locking data.

```
SQLPLUS> connect / as sysdba;
Connected.

SVRMGR> REM Dump Parallel Server DLM locks
SVRMGR> oradebug lkdebug -a convlock
SVRMGR> oradebug lkdebug -a convres
SVRMGR> oradebug lkdebug -r <resource handle> (i.e 0x8066d338 from convres
dump)
```

Dumping the Control File Contents Using ORADEBUG

The contents of the current controlfile can be dumped in text form to a process trace file in the user_dump_dest directory using the CONTROLF dump. The levels for this dump are as follows.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Dump Level Dump Contains:

1 -- only the file header
2 -- just the file header, the database info record, and checkpoint progress records
3 -- all record types, but just the earliest and latest records for circular reuse record types
4 -- as above, but includes the 4 most recent records for circular reuse record types
5+ -- as above, but the number of circular reuse records included doubles with each level

For example, the following syntax could be used to get a text dump on the controlfile in the trace file of the current process showing all the controlfile record types but only the oldest and most recent of the circular reuse records.

```
oradebug setmypid
oradebug dump controlf 3
```

Of course, the session must be connected AS SYSDBA to use the ORADEBUG facility. However, any session with the ALTER SESSION priviledge can use the following event syntax to take the same dump.

```
alter session set events 'immediate trace name controlf level 3';
```

If you would like to play around with this some more, the commands to dump the controlfile and file headers to your process trace file are as follows.

```
oradebug setmypid
oradebug dump controlf 10
oradebug dump file_hdrs 10
```


COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.

Summary

The ORADEBUG utility is a powerful and complex utility which allows you to easily control the dumping and tracing of virtually any database information relating processes. We have only touched the surface in this paper. Whenever using ORADEBUG be sure to try the options on a test environment before attempting them in a production situation.

COPYRIGHT © 2002 RAMPANT TECHPRESS. ALL RIGHTS RESERVED.