

Oradebug Utility

Abstract

This is an article about oracle debugger, or oradebug, as it came to be known. Oradebug is an internal, undocumented utility that has been made into a stuff of legend and a utility which was to be used only by the ones consecrated in the deepest secrets of Oracle RDBMS. In fact, it is supposed to be used by the members of Oracle Support only.

This is the right place to make a disclaimer: I do not work nor have I ever worked for oracle support, and so I don't know everything about oradebug utility. My knowledge comes from years of contact with oracle support and conversations with more experienced colleagues. It is a sad fact that the knowledge about oradebug has been spreading mostly by the word of mouth because not many documents are available.

To fend off possible criticism, I must admit that there are many notes on the Oracle Metalink in which certain aspects of oradebug utility are described. To me, the most significant ones are the following : 29786.1, 118570.1, 1058210.6 and 105395.1. All documents are given as Metalink document ids. As its name suggests, oradebug is a debugger and is used primarily for diagnostic purposes, like setting or un-setting events or dumping system or process states. So, what can we do with oradebug and why would we use oradebug if there are other, more supported tools and methods?

For many things, like turning on tracing in another session, there are other, often more convenient, methods, like `DBMS_SUPPORT`, or `DBMS_MONITOR` in Oracle 10. These methods, however, lack the generality of oradebug, but are, in turn, usable without being connected as `SYSDBA`. With `DBMS_SUPPORT`, we can set event 10046 in another session, but we cannot set the event 10053. With oradebug, we can set any event in any session. That can have rather practical use, as I will show a little later.

Oradebug is also used by RDA (Remote Diagnostic Advisor), the tool that can be downloaded from Metalink to produce diagnostic output that can be interpreted by Oracle support personnel only.

While due caution certainly has to be advised, it is also true that this utility has its place among the tools of the trade for any oracle DBA. This article aims to explain the following:

- Invoking oradebug
- Attaching oradebug to an oracle process.
- Taking dumps with oradebug
- Setting events with oradebug
- Suspending and resuming processes with oradebug.
- Generating and interpreting `HANGANALYZE` files.

Invoking oradebug

The oradebug is a part of sqlplus and needs to be invoked from a session connected as SYSDBA. If the connection does not have SYSDBA privilege, it will not be possible to use oradebug. Below is the first look at the oradebug utility, which conveniently supports “help” command.

```
$ sqlplus "/ as sysdba"

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Mar 13 00:21:37 2005

Copyright (c) 1982, 2004, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options

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> <lvl> [addr] 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
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> Parallel oradebug command prefix
-R             <Inst-List | def | all> Parallel oradebug prefix (return output
<instance# .. | all> Set instance list in double quotes
SGATOFILE      <SGA dump dir> Dump SGA to file; dirname in double
quotes
DMP_COW_SGA    <SGA dump dir> Dump & map SGA as COW; dirname in double
quotes
MAP_COW_SGA    <SGA dump dir> Map SGA as COW; dirname in double quotes
HANGANALYZE   [level] [syslevel] 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
SKDSTTPCS     <ifname> <ofname> Helps translate PCs to names
WATCH         <address> <len> <self|exist|all|target> Watch a region of memory
DELETE
```

```

<local|global|target> watchpoint <id> Delete a watchpoint
SHOW <local|global|target> watchpoints Show watchpoints
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
SQL>

```

Each oradebug command must be preceded with the word “oradebug”, so that sqlplus can distinguish among the debugger commands and the commands that need to be sent to the RDBMS for further processing. In other words, oradebug isn't a “mode command” that would set sqlplus in a “oradebug mode”, it is invoked on line by line basis.

Attaching oradebug to Oracle process

As its name implies, oradebug utility is a debugger. As is the case with any debugging utility used for debugging running processes, oradebug has to be *attached* to an oracle process. What exactly does the phrase “attaching to a process” mean? When debugger is attached to a process, it reads all the information about the process from process header and initializes its own routines with the values read from the process header. That enables the debugger to execute commands on behalf of process being debugged and in the context of that process.

In order to attach the debugger to a process, the first thing to know is the process id of the process to which the debugger will be attached. As this is an Oracle utility, this can be found out from the Oracle tables like V\$PROCESS and V\$SESSION. V\$SESSION table has PADDR column which has to be joined to V\$PROCESS table and ADDR column. The system process id is in the SPID column. The query would then look like this:

```

1 SELECT S.USERNAME,S.SID,P.SPID
2 FROM V$SESSION S,V$PROCESS P
3 WHERE S.PADDR=P.ADDR AND
4* S.USERNAME='SCOTT'
SQL> /

```

USERNAME	SID SPID
SCOTT	36 12518

```

SQL>

```

Attaching the debugger to the given session can now be done by the following command:

```

SQL> oradebug setospid 12518
Oracle pid: 19, Unix process pid: 12518, image: oracle10g@medo.noip.com
SQL>

```

Sometimes, we are not interested in debugging particular process, but in doing instance-wide things, like taking dumps. In that case, there is a shortcut:

```
SQL> oradebug setmypid
Statement processed.
```

Now, that we have attached debugger to an oracle process, we can do some useful things.

Performing dumps

Oradebug can help us perform many useful dumps. The list of dumps that can be done from oradebug can be displayed by using DUMPLIST command, like this:

```
SQL> oradebug setmypid
Statement processed.
SQL> oradebug dumplist
EVENTS
TRACE_BUFFER_ON
TRACE_BUFFER_OFF
HANGANALYZE
LATCHES PROCSSTATE
SYSTEMSTATE
INSTANTIATIONSTATE
REFRESH_OS_STATS
CROSSIC
CONTEXTAREA
HEAPDUMP
HEAPDUMP_ADDR
POKE_ADDRESS
.....
SQL>
```

There are many dumps that can be performed, and the list is version dependent. Generally speaking, dumps produce trace files in the directory defined by the USER_DUMP_DEST instance parameter. As many of these dumps are very large in size (system dumps are at least as big as SGA, and SGA areas of several hundred MB are quite common these days) and default limit for trace files is 20MB, the first thing to do is to remove the tracefile size limit, like this:

```
SQL> oradebug unlimit
Statement processed.
```

Now, we can dump system state, taking the level 10 of the dump:

```
SQL> oradebug dump systemstate 10
Statement processed.
```

This has produced a large file in USER_DUMP_DEST directory. As this directory is used for many purposes, including taking a SQL trace of a process or session, it would be nice if we were able to find our newly produced system dump quickly. That is done by the TRACEFILE_NAME command:

```
SQL> oradebug tracefile_name
/oracle/product/10g/admin/oracle/udump/10g_ora_12599.trc
SQL>
```

We now have to flush any pending writes to the file and close it, so that we can manipulate it further. To that end, we can execute the following commands:

```
SQL> oradebug flush
Statement processed.
SQL> oradebug close_trace
Statement processed.
```

Our dump is taken, file `/oracle/product/10g/admin/oracle/udump/10g_ora_12599.trc` is closed and ready for shipping to the Oracle Support. Interpreting system dumps is a job of Oracle Support and should not be done by users. Dumps that can and should be examined by users are HANGANALYZE dumps that will be explained later. Now, let's set some events. Yeee-haw!

Setting events in Oracle process

Our debugger is still attached to the oracle process representing the session by user 'SCOTT'. Let's set the event 10053, one detailing the internal workings of CBO:

```
SQL> oradebug event 10053 trace name context forever, level 1
Statement processed.
SQL>
```

When the session in which the event was set executes a SQL command, the new trace file will be opened by the process 12518:

```
$ pwd
/oracle/product/10g/admin/oracle/udump
$ ls -l
total 424
-rw-r----- 1 oracle dba 26637 Mar 13 01:01 10g_ora_12518.trc
-rw-r----- 1 oracle dba 403735 Mar 13 00:44 10g_ora_12599.trc
$ fuser *.trc
10g_ora_12518.trc: 12518
$
```

We can see that there are two files, one taken by the process 12599 and another one taken by the process 12518. The one taken by the process 12518 is still open and used by the process (fuser command). If you take a look at the system dump section, you'll notice that the other dump is our system dump.

Once again, we will flush the trace and close the trace file:

```
SQL> oradebug tracefile_name
/oracle/product/10g/admin/oracle/udump/10g_ora_12518.trc
SQL> oradebug flush
Statement processed.
SQL> oradebug close_trace
Statement processed.
```

Trace event 10053 is described in detail on Wolfgang Breitling's home page <http://www.centrexcc.com>. Describing it in detail is beyond the scope of this article.

The same methodology can be applied to other events that occasionally need to be set, like the event 10046 which is the most important event for performance evaluation. Why would a DBA want to set event 10046 using oradebug instead of using DBMS_MONITOR or DBMS_SUPPORT? Well, one reason is the convenient TRACEFILE_NAME command which alleviates the need for searching frantically among the gazillion of trace files in the USER_DUMP_DEST directory. True, as of Oracle9i, there is a possibility to set an identifier, but I still like the comfort of oradebug.

Another example of setting event comes from the situation when there is a need to debug a 3rd party application for which we have no source. How many times did it happen to you that a 3rd party utility was reporting ORA-00942: Table or view does not exist, without telling us which table is it looking for? It happened to me more times than I'd like. We cannot turn on the tracing by modifying the source because we don't have the source. The oracle error 942 is just an example, we can use it for almost any SQL error. Here is how this problem is addressed:

```
SQL> oradebug event 942 trace name errorstack forever , level 3
Statement processed.
SQL>
```

Now, we execute the following SQL command in the session that our debugger is attached to:

```
SQL> select count(*) from NONEXISTING_TABLE;
select count(*) from NONEXISTING_TABLE
*
ERROR at line 1:
ORA-00942: table or view does not exist
```

Our SELECT gave us predictable and somewhat expected result. Now, let's see the contents of our trace file:

```
/oracle/product/10g/admin/oracle/udump/10g_ora_19174.trc
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options
ORACLE_HOME = /oracle/product/10g
System name: Linux
Node name: medo.noip.com
Release: 2.4.27
Version: #1 Sun Aug 8 01:31:49 EDT 2004
Machine: i686
Instance name: 10g
Redo thread mounted by this instance: 1
Oracle process number: 20
Unix process pid: 19174, image: oracle10g@medo.noip.com

*** 2004-08-22 21:32:11.471
ksedmp: internal or fatal error
ORA-00942: table or view does not exist
Current SQL statement for this session:
select count(*) from NONEXISTING_TABLE
```

```
----- Call Stack Trace -----
calling      call  entry      argument values in hex
location     type  point      (? means dubious value)
-----
```

The important part is shown in the bold and underlined font: our trace file contains the SQL statement that caused the error, thus enabling us to find out which table is missing, even without the source code for the utility. How are events turned off? Here is how we turn both the event 942 and 10053 off:

```
SQL> oradebug event 10053 trace name context off
Statement processed.
SQL> oradebug event 942 trace name context off
Statement processed.
```

Suspending and resuming sessions

This example is given only for completeness, In reality, need for suspending user sessions arises very infrequently, and even then, we should look for an alternative solution, if possible. Suspending and resuming a session is an internal operation and, as such, should be used only when truly necessary or requested by oracle support.

We can suspend the session that we have attached our debugger to, by using the following command:

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

The process that debugger is attached to will be suspended. It is important not to try that with the sessions that are connected through the dispatcher (MTS) because, the session will find another free shared server process to execute its SQL command. This is well suited for dedicated server sessions only. The dedicated server session that I attached my process to will appear to be hanging and will not react to SQL. Reaction will come only when I resume the session with the following command:

```
SQL> oradebug resume
Statement processed.
SQL>
```

The logical question is why would a DBA want to suspend a session? For me, the suspension was motivated by the need to have a high-priority job proceeding without killing a long running batch job which was consuming lots of resources. Suspending and resuming is done only in exceptional situations like that.

Using HANGANALYZE

The HANGANALYZE utility is new in oradebug, it was introduced with Oracle8i. It is a powerful utility which helps in situations when database instance or session is, well, hung. The HANGANALYZE is not a dump. It produces an output in the user dump directory, but it is not a dump, but a human readable file, intended to be read by an eager DBA. General syntax for the HANGANALYZE utility is the following:

```
SQL> oradebug hanganalyze 3
Hang Analysis in /oracle/product/10g/admin/oracle/udump/10g_ora_12655.trc
SQL>
```

The number “3” in the command is called “level”. Incidentally, Oracle Corporation sanctions taking HANGANALYZE only up to level 3. Higher levels should be taken only on the request of Oracle Support person. Here is definition of all the levels:

- 10 Dump all processes (IGN state)
- 5 Level 4 + Dump all processes involved in wait chains (NLEAF state)
- 4 Level 3 + Dump leaf nodes (blockers) in wait chains (LEAF,LEAF_NW,IGN_DMP state)
- 3 Level 2 + Dump only processes thought to be in a hang (IN_HANG state)
- 1-2 Only HANGANALYZE output, no process dump at all

Level 4 and above can produce very large and resource intensive output, which can seriously impact the instance performance. Here is the output of the HANGANALYZE utility, taken at the level 3:

```
/oracle/product/10g/admin/oracle/udump/10g_ora_12655.trc
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options
ORACLE_HOME = /oracle/product/10g
System name: Linux
Node name: medo.noip.com
Release: 2.6.10-1.770_14.rhfc3.at
Version: #1 Fri Mar 4 11:34:31 EST 2005
Machine: i686
Instance name: 10g
Redo thread mounted by this instance: 1
Oracle process number: 14
Unix process pid: 12655, image: oracle@medo.noip.com (TNS V1-V3)

*** SERVICE NAME:(SYS$USERS) 2005-03-13 01:22:24.704
*** SESSION ID:(32.21) 2005-03-13 01:22:24.704
*** 2005-03-13 01:22:24.704
=====
HANG ANALYSIS:
=====
Open chains found:
Other chains found:
Chain 1 : <cnode/sid/sess_srno/proc_ptr/ospid/wait_event> :
         <0/32/21/0x5bda30d4/12655/No Wait>
Chain 2 : <cnode/sid/sess_srno/proc_ptr/ospid/wait_event> :
```

```

<0/41/7/0x5bda35d0/12455/Queue Monitor Wait>
Chain 3 : <cnode/sid/sess_srno/proc_ptr/ospid/wait_event> :
  <0/43/20/0x5bda4ebc/12540/wakeup time manager>
Extra information that will be dumped at higher levels:
[level 5] : 3 node dumps -- [SINGLE_NODE] [SINGLE_NODE_NW] [IGN_DMP]
[level 10] : 11 node dumps -- [IGN]

State of nodes
([nodenum]/cnode/sid/sess_srno/session/ospid/state/start/finish/[adjlist]/predecessor):
[31]/0/32/21/0x5bdd4c68/12655/SINGLE_NODE_NW/1/2//none
[35]/0/36/19/0x5bdd9408/12518/IGN/3/4//none
[37]/0/38/1/0x5bdbb7d8/12459/IGN/5/6//none
[39]/0/40/1/0x5bdddba8/12457/IGN/7/8//none
[40]/0/41/7/0x5bdded90/12455/SINGLE_NODE/9/10//none
[42]/0/43/20/0x5bde1160/12540/SINGLE_NODE/11/12//none
[43]/0/44/1/0x5bde2348/12441/IGN/13/14//none
[44]/0/45/1/0x5bde3530/12439/IGN/15/16//none
[45]/0/46/1/0x5bde4718/12437/IGN/17/18//none
[46]/0/47/1/0x5bde5900/12435/IGN/19/20//none
[47]/0/48/1/0x5bde6ae8/12433/IGN/21/22//none
[48]/0/49/1/0x5bde7cd0/12431/IGN/23/24//none
[49]/0/50/1/0x5bde8eb8/12429/IGN/25/26//none
[50]/0/51/1/0x5bdea0a0/12427/IGN/27/28//none
=====
END OF HANG ANALYSIS

```

The reference document which explains the HANGANALYZE output is the Metalink note 215858.1. Here, I'll only explain the purpose and terminology:

Chains: chains are formed by the processes competing for a resource. Cyclical or closed chains are deadlocks and are considered true hangs. Open chains consist of processes waiting for a resource held by a blocker. These are not true hangs and can be resolved by killing the holding session and allowing the blocked processes to proceed. This can be useful when a process is holding a latch, frequent example is the library latch, and other processes are waiting for the latch. Be careful, however, not to kill SMON or PMON as that would usually bring down the entire instance. The format of each chain entry is the following:

```
Chain 1 : <cnode/sid/sess_srno/proc_ptr/ospid/wait_event> :
```

These abbreviations have the following meanings:

CNODE=Cluster Node, available as of 9i
 SID = Session Id, the same as SID in V\$SESSION
 SESS_SRNO = Serial number, the same as SERIAL# in V\$SESSION.
 OSPID=Operating System PID (Process ID)

Nodes: Nodes are sessions. The first line explains the session listing:

```
([nodenum]/cnode/sid/sess_srno/session/ospid/state/start/finish/[adjlist]/predecessor):
```

The meaning of the abbreviations is the same as for the chains.

What we are interested in is the STATE information. Obviously, if the state is IN_HANG, then there is a problem. If the state is LEAF or LEAF_NW, then the session is blocking others and is probably on the top of an open chain. Sessions in the NLEAF state are waiting sessions and appear to be hanging. If the state is IGN or IGN_DMP, the session is idle. For more information, please see the reference document on Metalink.

Conclusion

That would conclude my introductory description of the oradebug utility. There are many more things that can be done by oradebug. One of them is calling any kernel function by using the CALL command. This is a very dangerous capability and I must recommend not using it, unless you are a senior Oracle Support person in which case this article probably isn't for you. Using such a powerful utility as oradebug should always be the last resort, not the first choice or regular method. To use oradebug, one needs to connect as SYSDBA and have the full control of the database. That implies that the only people who should attempt to use the utility are database administrators. While oradebug is not the most important part of the DBA tool chest it certainly is very helpful in certain vitally important situations. I hope that by writing this article, I've made life to my DBA colleagues easier.

Mladen Gogala,
Oracle DBA.